UNIVERSIDAD DE GRANADA
Departamento de Ciencias de la Computación
e Inteligencia Artificial

NEW APPROACHES IN TIME SERIES FORECASTING: METHODS, SOFTWARE, AND EVALUATION PROCEDURES

TESIS DOCTORAL
presentada para la obtención del
GRADO DE DOCTOR
por

Christoph Norbert Bergmeir

Granada, enero de 2013

Editor: Editorial de la Universidad de Granada
Autor: Christoph Norbert Bergmeir
D.L.: GR 1923-2013
ISBN: 978-84-9028-610-4

New Approaches in Time Series Forecasting:
Methods, Software, and Evaluation Procedures

Esta memoria, titulada **New Approaches in Time Series Forecasting: Methods, Software, and Evaluation Procedures** (es decir, **Nuevos Enfoques para Predicción de Series Temporales: Métodos, Software y Procedimientos de Evaluación**) es presentada por D. Christoph Norbert Bergmeir con el fin de optar al grado de Doctor y ha sido realizada en el Departamento de Ciencias de la Computación e Inteligencia Artificial de la Universidad de Granada bajo la dirección del Doctor D. José Manuel Benítez Sánchez. El doctorando y el director de la tesis garantizamos, al firmar esta tesis doctoral, que el trabajo ha sido realizado por el doctorando bajo la dirección del director de la tesis y hasta donde nuestro conocimiento alcanza, en la realización del trabajo, se han respetado los derechos de otros autores a ser citados, cuando se han utilizado sus resultados o publicaciones.

Lo cual, para que conste, firmaron ambos en Granada,
siendo de 2013 el día 15 del mes de enero.

El Director

Fdo. J.M. Benítez

El Doctorando

Fdo. C. Bergmeir

Agradecimientos

This work was made possible due to a scholarship from the Spanish Ministry of Education (MEC) of the "Programa de Formación del Profesorado Universitario (FPU)." Furthermore, I'd like to thank my supervisor, José Manuel Benítez, who – naturally – had great influence on this work. I also want to thank Mauro Costantini and Robert Kunst, who made possible a successful research stay at the Department of Economics, University of Vienna. I also thank my other coauthors Isaac Triguero, Daniel Molina, and José Luis Aznarte for the good collaboration. The not-so-short four years have finally come to an end...

Índice / Contents

1. Introducción y Conclusiones

En este capítulo, se presenta una introducción al análisis y la predicción de las series temporales y las principales conclusiones de la tesis en castellano. La introducción muestra las posibilidades y limitaciones de la predicción, así como los conceptos generales de predicción autorregresiva y evaluación de predictores. Además, se muestran los objetivos propuestos para este trabajo y las conclusiones alcanzadas.

1.1. Análisis de series temporales

Una serie temporal es un tipo de dato especial, que toma en cuenta el tiempo explícitamente. Típicamente es una realización de las mismas mediciones a lo largo de diferentes puntos en el tiempo. Las series temporales son ubicuas en la vida cotidiana, desde la predicción meteorológica, al procesamiento de señales (audio), las finanzas, la Economía, el tráfico, la Física, la Biología, la Medicina y la Psicología. A lo largo de los últimos 40 años, el análisis de series temporales se ha convertido en una importante área de investigación, cuyos principales campos son la Estadística, la Econometría, la Física y las Ciencias de la Computación.

Los objetivos y métodos para el análisis de series temporales son variados. Tradicionalmente, se agrupan en tareas de modelado, predicción y/o caracterización de series temporales. Además, con la disponibilidad hoy en día de enormes cantidades de datos, nuevas tareas adquieren importancia en las Ciencias de la Computación, en su mayoría relacionadas con la minería de datos de series temporales, como la segmentación y el indexado de estas series.

Aunque todas estas aplicaciones tengan propósitos diferentes, comparten algunos conceptos generales y están relacionadas entre sí. La caracterización de series temporales implica encontrar descripciones estadísticas generales de las series. Puede usarse para complementar el modelado o la predicción, en el sentido de que se utiliza como un primer paso para encontrar buenos métodos en posteriores etapas, y para encontrar limitaciones globales. El modelado de series temporales busca entender el sistema subyacente que provoca la dinámica de las series. Por su parte, la predicción busca predecir futuros valores de las series, habitualmente con un horizonte temporal bastante corto. Aunque el modelado y la predicción están relacionados, se trata de conceptos diferentes. El entendimiento del comportamiento global del sistema no implica necesariamente las mejores predicciones a corto plazo, y para obtener un buen rendimiento en estas predicciones a menudo no es necesario conocer y entender el sistema subyacente. Esto puede validarse con experimentos relativamente sencillos, como en [42].

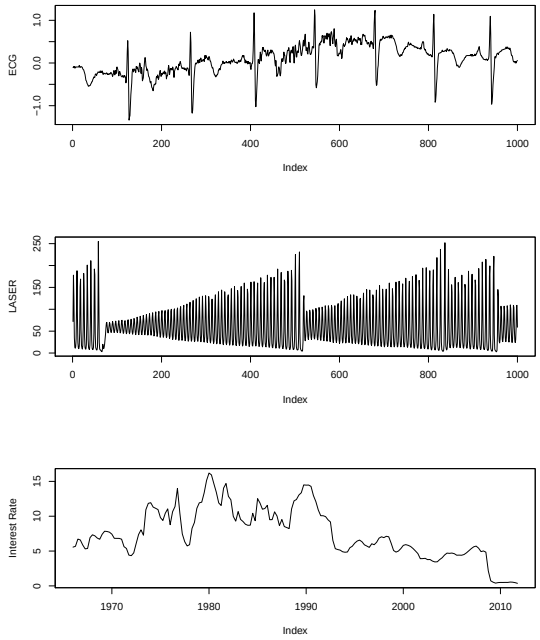

Figura 1.1.: Tres ejemplos de series temporales de distintos contextos de aplicación. La primera serie es un electrocardiograma de un ritmo cardiaco humano normal [61]. La segunda serie es una medición de un *Far-Infrared-Laser* en estado caótico [78]. La tercera serie es la tasa de interés mensual del Reino Unido [6].

Algunos ejemplos típicos de series temporales se presentan en la Figura 1.1. Un electrocardiograma (ECG) muestra la actividad eléctrica periódica del corazón. Esta actividad refleja las distintas contracciones que tienen lugar en un corazón en funcionamiento para bombear la sangre al resto del cuerpo. En Medicina y Biología, cada uno de estos ciclos se divide comúnmente en ondas P, Q, R, S y T. Dependiendo de las características de estas ondas, los médicos pueden encontrar anormalidades y problemas. Como el sistema subyacente se comprende razonablemente bien, la serie temporal puede modelarse de esta manera, y el modelado tiene aplicaciones importantes en Medicina y Biología. Sin embargo, en este ejemplo, la realización de predicciones probablemente no sea útil. La segunda figura muestra una medición de un *Far-Infrared-Laser* en un estado caótico. También en este caso, el modelado de la serie temporal y la comprensión del sistema subyacente son las tareas principales, aunque esta serie ha sido usada en competiciones de predicción en el pasado. El tercer ejemplo de la figura es la tasa de interés mensual en el Reino Unido. Aquí, la predicción es claramente una tarea valiosa, dado que una predicción buena o mala puede te-

ner un gran impacto económico. Sin embargo, la dinámica subyacente es difícil de analizar y de comprender.

Este trabajo se centra en la predicción de series temporales, donde valores individuales del futuro cercano de las series son extrapolados. Sin embargo, a través de las relaciones entre el modelado y la predicción de series temporales, se abarcarán también algunos aspectos del modelado.

1.2. Predicción: posibilidades y limitaciones

En la predicción, la suposición fundamental es que el futuro depende del pasado. Dos conceptos importantes son la predicción univariante y multivariante. En la predicción univariante, la suposición es que el futuro de una serie temporal depende (principalmente) de su propio pasado, de forma que el conocimiento sobre este pasado es suficiente para realizar una predicción útil. En predicción multivariante, también se tienen en cuenta variables exógenas (por ejemplo, mediciones de otras fuentes).

Un requisito importante para el análisis y la predicción de series temporales son los datos, que a menudo se encuentran abundantemente disponibles hoy en día. Además, los métodos de predicción son genéricos y fáciles de aplicar, de modo que las predicciones pueden realizarse y son realizadas para prácticamente todo. Sin embargo, una capacidad fundamental para un buen predictor es entender las posibilidades y limitaciones de la predicción, para determinar la utilidad de los resultados.

La previsibilidad de los fenómenos del mundo real varía en gran medida. Por ejemplo, la hora del crepúsculo del día de mañana (o de un día dentro de 7 años) o la programación de televisión de mañana se pueden predecir de forma bastante precisa. En cambio, predecir los números de lotería del día de mañana es imposible, aunque están disponibles muchos datos acerca de los números de lotería del pasado. Hay muchos fenómenos del mundo real que no son predecibles de ningún modo significativo, o al menos no son predecibles con la precisión suficiente para que las predicciones sean útiles. Es en gran medida responsabilidad de los predictores distinguir entre los casos donde la metodología aplicada funcionará, y los casos en los que no.

Otro problema es que la incertidumbre real puede no estar bien representada en los datos. Éstos pueden parecer regulares y predecibles, hasta que un evento único e impredecible ocurre. Entonces puede ser importante si el evento se considera como un *outlier* (es decir, sin importancia para la serie en general y sus aplicaciones), o si el evento es precisamente la parte más importante del fenómeno y la aplicación. La predicción tal y como se entiende en este trabajo no trata sobre prever cambios drásticos e inesperados. Se trata de los procedimientos cotidianos en los que se basan muchos sistemas informáticos para la planificación, la facturación, el control y la administración.

Según Hyndman y Athanasopoulos [34], tres factores contribuyen a la previsibilidad: (1) el entendimiento de los factores implicados en la creación de los fenómenos que se quieren predecir, (2) de cuántos datos se dispone, y (3) si la predicción realizada afectará al futuro.

Por tanto, para realizar predicciones es importante disponer de datos. Además, es importante

entender al menos parcialmente el sistema subyacente, para determinar si los datos son representativos y suficientes para la tarea de predicción a realizar. Y finalmente, después de realizar la predicción, ésta no debería alterar el comportamiento real de la variable objetivo.

El objetivo de utilizar métodos matemáticos para la predicción, como es el caso de este trabajo, es modelar adecuadamente patrones repetitivos y características centrales de datos pasados. El ruido y los eventos únicos e impredecibles no son útiles en tal entorno de predicción. Por tanto, la idea central es separar la serie temporal en una parte de valor predictivo, y otra parte de ruido. La parte de valor predictivo debe ser modelada adecuadamente, sin las influencias causadas por el ruido.

Hay muchas aplicaciones en las que la predicción proporciona muy buenos resultados y se utiliza con éxito, por ejemplo cuando se predice el consumo eléctrico, la retirada de efectivo de cajeros automáticos, los datos relacionados con el tráfico como el caudal de coches de ciertas calles o intersecciones, la carga de trabajo de centros de llamada, la carga de almacenes y las reservas de hotel o vuelos, entre otros. Pero ciertamente hay otros ejemplos, especialmente los relacionados con la crisis financiera global de 2008, en los cuales la predicción falló estrepitosamente. Las predicciónes en Economía de precios, tasas de intercambio, o cotización de acciones, por ejemplo, tienen varias dificultades. Los datos suelen ser abundantes, pero los sistemas subyacentes no se entienden correctamente y las predicciones pueden afectar al futuro en gran medida, pues estas series temporales no suelen ser funciones de su propio pasado, sino de las expectativas de la gente acerca de su futuro.

A continuación se presenta un ejemplo que ilustra algunos aspectos básicos de la predicción y la previsibilidad. La primera serie de la Figura 1.2 muestra la temperatura media entre septiembre de 2009 y marzo de 2012 en París, Francia. El pronóstico meteorológico es probablemente la aplicación más conocida de las técnicas de predicción. La serie tiene un fuerte patrón estacional, que podría utilizarse para la predicción (es fácil predecir que el próximo invierno será más frío que el próximo verano, y que la tempreatura será aproximadamente la misma que este invierno). Además, la serie es estable en el sentido de que la predicción ingenua (*naïve*), que consiste simplemente en tomar como predicción el último valor conocido, es un resultado a corto plazo básico pero útil. Sin embargo, para la predicción meteorológica existen disciplinas de investigación especializadas que intentan comprender el fenómeno subyacente, identificar los factores que influyen en él como la temperatura (la temperatura se ve mayoritariamente influenciada por el número de horas diarias de luz solar, la advección térmica, la velocidad del viento y la nubosidad), y tomar mediciones periódicas de estas variables, para poder realizar el pronóstico.

La segunda serie temporal de la Figura 1.2 presenta el consumo eléctrico diario en Francia en el mismo periodo. Se ven fuertes patrones semanales así como un fuerte patrón estacional, que tiene una correlación negativa respecto a la temperatura: cuanto más frío es el tiempo, más energía se consume. Toda esta información puede utilizarse para la predicción, bien usando solamente los patrones semanales y estacionales del pasado de las series (predicción univariante), o bien utilizando los patrones en combinación con predicciones de temperatura específicas, lo cual es claramente mejor. Esto permite predicciones razonablemente precisas para el consumo de electricidad.

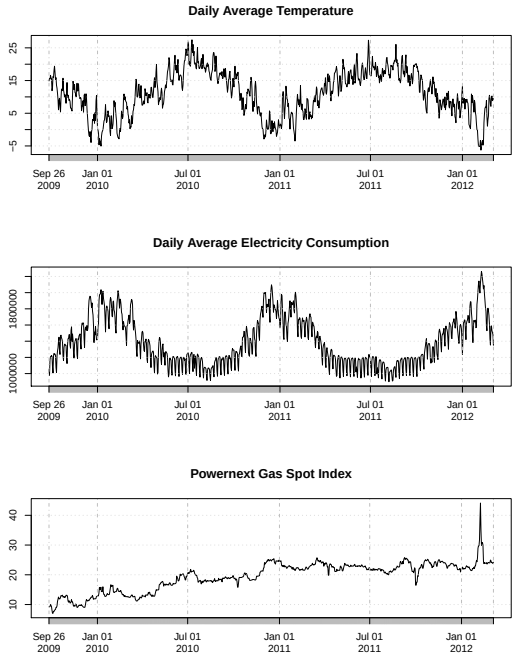

Figura 1.2.: La primera serie es la temperatura media en París (obtenida de: http://academic.udayton.edu/kissock/http/Weather/). La segunda serie es el consumo diario de energía en la Francia metropolitana, exceptuando Córcega (fuente: http://www.rte-france.com). La tercera serie es el índice "Powernext gas spot", que representa el precio del gas en Francia (fuente: http://www.powernext.fr).

Un hecho a resaltar es que hacia el final de la serie de temperaturas, hay un periodo en febrero de 2012 de tiempo muy frío. Durante ese tiempo, hubo una ola muy fría en toda Europa central. Se aprecia este periodo de muy bajas temperaturas directamente reflejado en un pico de consumo eléctrico esos días.

Con esto en mente, se trata ahora de observar una tarea de predicción distinta. La tercera serie temporal es el índice "Powernext gas spot", que refleja el precio del gas en Francia, para el mismo periodo de tiempo. Hay que notar que, durante la ola fría de febrero de 2012, los precios del gas tienen un pico muy acusado. Es razonable asumir que este pico está causado por el periodo de tiempo frío, probablemente debido a carencias de suministro o incidentes similares. Teniendo en cuenta solamente el pasado del precio del gas, el evento era claramente impredecible, dado que no se había comportado de tal manera anteriormente. Y, aunque en este caso particular la serie está relacionada con la temperatura, esta relación es mucho más indirecta que en el caso

del consumo eléctrico, y en los datos previos antes de este evento extremo, los precios del gas no parecen tener ninguna conexión con la temperatura. Por consiguiente, probablemente no se trate de que el tiempo frío en sí mismo llevó a la subida del precio del gas, sino más bien que el hecho de que la intensidad y la duración del frío fueron *inesperados*. En resumen, predecir el precio del gas en este ejemplo es mucho más difícil que predecir el consumo eléctrico.

1.3. Un entorno general de predicción

Un concepto importante relacionado con la previsibilidad es la estacionaridad. La idea fundamental de la estacionaridad es que las características estadísticas principales de la serie temporal no cambian a lo largo del tiempo. La estacionaridad puede definirse formalmente como sigue (ver Cryer y Chan [18]). Una serie temporal x_t es estrictamente estacionaria, si para cualesquiera puntos en el tiempo $t_1,\ldots,\ t_n$ y cualquier parámetro de desplazamiento k, la distribución conjunta de $x_{t_1},\ldots,\ x_{t_n}$ y $x_{t_{1-k}},\ldots,\ x_{t_{n-k}}$ es la misma.

Una serie temporal x_t es estacionaria débil o de segundo orden, si los momentos de primer y segundo orden no cambian en el tiempo. Es decir, la función media es constante en el tiempo, y la covarianza sólo depende del desplazamiento temporal entre dos valores de la serie temporal (de aquí se extrae directamente que la varianza también es constante en el tiempo):

$$E(x_{t_1}) = E(x_{t_{1-k}})$$

y

$$Cov(x_{t_1}, x_{t_2}) = Cov(x_{t_{1-k}}, x_{t_{2-k}})$$

para cualesquiera puntos temporales $x_{t_1},\ x_{t_2}$, y cualquier desplazamiento k. Si la media y la covarianza existen, la estacionaridad estricta implica estacionaridad débil.

Es una práctica común en predicción limitar las consideraciones a series temporales estacionarias, y a casos especiales de no-estacionaridad. La no-estacionaridad se maneja típicamente mediante pasos de preprocesamiento. La des-tendencia y des-estacionalización se utilizan para eliminar cambios determinísticos en la media. Otros métodos comunes son, por ejemplo, diferenciar o logaritmizar la serie para obtener estacionaridad. Si no puede hallarse ningún preprocesamiento adecuado para eliminar la no-estacionaridad, puede ser apropiado no utilizar el pasado completo de la serie para la predicción, sino solamente una parte del final [20], o la predicción puede no ser posible [20, 39].

Dado que en una serie temporal estacionaria las correlaciones son constantes en el tiempo, en el sentido de que dependen solamente del retraso pero no del punto exacto en el tiempo, una función de autocorrelación significativa puede calcularse. La función de autocorrelación determina los retrasos en la serie temporal que muestran una correlación significante. En la Figura 1.3 se muestra un ejemplo.

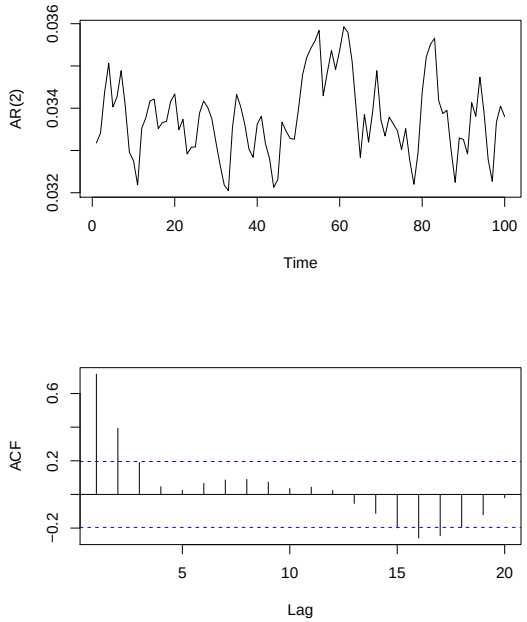

Figura 1.3.: La imagen superior muestra una serie temporal, generada por un proceso lineal autorregresivo. La imagen inferior representa su función de autocorrelación.

De este modo, es posible identificar los retrasos temporales relevantes para la predicción, construir una matriz cuyas columnas son las versiones retrasadas de la serie temporal (ver la Tabla 1.1 para una ilustración), y utilizar esa matriz como entrada para un método de regresión. Esta arquitectura de modelo se llama modelo autorregresivo de orden fijo, y el paso de preprocesamiento consistente en construir la matriz se llama empotrado (*embedding*) de la serie temporal. Hay que señalar que al igual que la estacionaridad justifica directamente el uso del modelo autorregresivo de orden fijo, si la serie no es estacionaria el uso de esta arquitectura de modelo puede no dar buenos resultados. Sea $x(t)$ una serie temporal, y sea d el parámetro de desplazamiento del máximo retraso que se pretende usar para la predicción. Encontrar un modelo de predicción puede definirse como encontrar una función F (lineal o no-lineal), tal que

$$x(t) = F(x(t-1),\ \ldots,\ x(t-d)) + e(t), \tag{1.1}$$

donde $e(t)$ es una serie temporal de términos de error independientes e idénticamente distribuidos (i.i.d.). Esta definición refleja directamente la separación de información útil y ruido ya mencionada en la Sección 1.2. Por tanto, el concepto clave de esta definición no es sólo encon-

índice	retraso 4	retraso 3	retraso 2	retraso 1	objetivo
		...	...		
7	14	10	26	11	-13
8	10	26	11	-13	-15
9	26	11	-13	-15	-8
10	11	-13	-15	-8	35
11	-13	-15	-8	35	40
12	-15	-8	35	40	-8
13	-8	35	40	-8	-16
14	35	40	-8	-16	7
15	40	-8	-16	7	17
		...	...		

Cuadro 1.1.: Ejemplo de una serie temporal preprocesada para la regresión, usando los últimos cuatro valores (valores retrasados) para predecir el valor actual.

trar una separación, sino encontrarla de modo que se produzcan errores i.i.d., para que F capture adecuadamente toda la información valiosa para la predicción.

Usando el entorno de la Ecuación 1.1, un modelo autorregresivo lineal (como en Box y Jenkins [13]) se define escogiendo F de forma que calcule una combinación lineal de los valores pasados de la serie:

$$x(t) = a_0 + \sum_{i=1}^{d} a_i x(t-i) + e(t). \tag{1.2}$$

Aquí, $a_0, \ldots, a_d$ son los coeficientes del modelo lineal (que deben ser estimados durante la construcción del modelo). Con $\mathbf{z}(t) = (1,\ x(t-1),\ x(t-2), \ldots,\ x(t-d))^T$ y $\mathbf{a} = (a_0, \ldots, a_d)$, la Ecuación 1.2 se presenta en notación vectorial:

$$x(t) = \mathbf{a}\mathbf{z}(t) + e(t). \tag{1.3}$$

La Figura 1.4 muestra una gráfica dispersa tridimensional de la serie de la Figura 1.3, donde los valores de la serie temporal en el tiempo t se representan junto a los valores en los tiempos $t-1$ y $t-2$. En esta gráfica pueden verse claramente las relaciones lineales entre los valores en un tiempo dado, y los valores de retraso uno y dos.

1.4. Evaluación de predictores

Después de haber especificado un predictor y estimado sus parámetros, es importante establecer su calidad. Una crítica de Taleb [73] es que en Economía, los científicos realizan predicciones sin

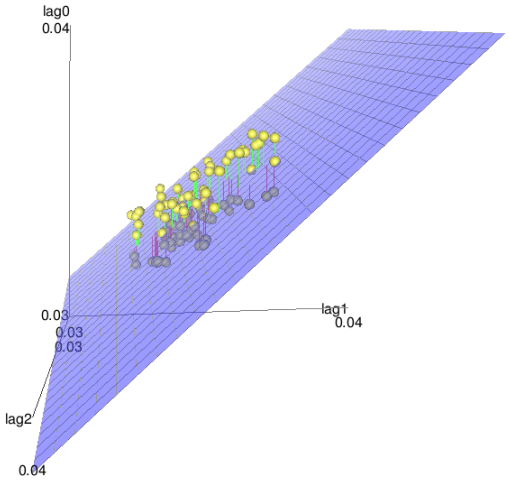

Figura 1.4.: Gráfica dispersa tridimensional de la serie de la Figura 1.3. Los valores de la serie temporal en el tiempo t se representan junto a los valores en los tiempos $t-1$ y $t-2$. Se puede identificar una relación lineal entre los tres retrasos temporales, y el plano azul muestra una regresión lineal de los datos.

ningún *feedback* acerca de cuán buenas fueron las predicciones. Otras críticas son que ni los expertos humanos ni los métodos automáticos de predicción son capaces de predecir las variables más importantes [50, 51], y que las personas tienden a subestimar en gran medida la incertidumbre de las predicciones. Teniendo esto en cuenta, la evaluación es un aspecto central de la predicción, dado que éstas siempre pueden realizarse, pero es necesario medir su calidad para saber si serán útiles o incluso si pueden ser peligrosas o dañinas.

Uno de los primeros en medir sistemáticamente la calidad de los métodos de predicción fue Makridakis en las llamadas competiciones M, M [47], M2 [48], y M3 [49]. En estas competiciones, se pedía a los participantes que proporcionaran predicciones para muchas series temporales diferentes. Las predicciones eran entonces evaluadas en conjuntos de test (que no se proporcionaban previamente a los participantes) consistentes en los valores posteriores de las respectivas series, y los errores se calcularon utilizando diferentes medidas. Desde los inicios del trabajo en evaluación de predicciones, este campo aún es hoy en día un área activa de investigación, y una parte importante del trabajo de esta tesis se ha llevado a cabo en este campo.

Además de diferentes tipos de predicciones, medidas de error, y vías para el particionado de datos, otra pregunta importante es qué consecuencias tienen las predicciones incorrectas. Las predicciones no se suelen hacer *per-se*, sino con el objetivo de tomar decisiones informadas. Dependiendo de la predicción la decisión puede variar, y el error debería tomar en cuenta de qué manera resultarán de las predicciones decisiones equivocadas. Un ejemplo de este caso son las medidas de

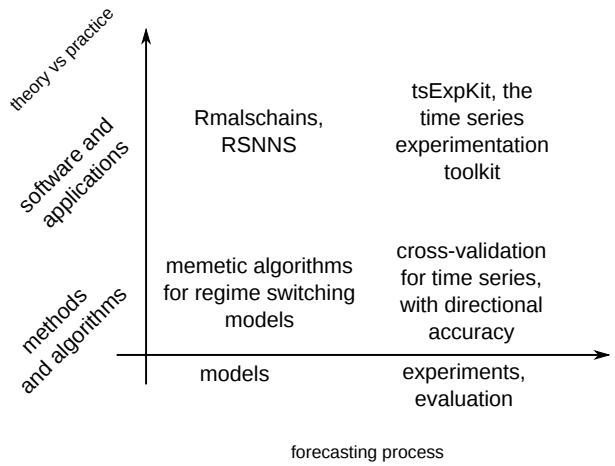

Figura 1.5.: Clasificación esquemática de las partes principales de esta tesis.

precisión direccional que miden si la predicción es capaz de anticipar correctamente la dirección (es decir, si una variable sube o baja), dado que éste suele ser un criterio importante en el que se basan las decisiones.

1.5. Objetivos de esta tesis

El objetivo de este trabajo es explorar los diferentes aspectos del proceso de predicción en su conjunto, con el fin de lograr un buen rendimiento de predicción en problemas prácticos. La idea es que se deben tener en cuenta la identificación, la estimación y la evaluación del modelo, para encontrar y poder utilizar los modelos más adecuados para una tarea concreta. Con respecto a los modelos y su evaluación, el objetivo de este trabajo es considerar problemas tanto teóricos como prácticos para mejorar diversos aspectos de los algoritmos, los métodos, las prácticas y el software del estado del arte. Se lleva a cabo una investigación sobre nuevos procedimientos y se presentan implementaciones de software de calidad, que hacen reproducible la investigación y ayudan en la difusión y la transferencia de tecnología. La Figura 1.5 ilustra esto con una clasificación esquemática de las partes principales de la tesis.

Concretamente, con respecto a los modelos de predicción, se propone una nueva técnica de predicción de series temporales, que combina el procedimiento NCSTAR con un potente algoritmo memético de optimización, y se muestra que es competitiva respecto a otras técnicas utilizadas en el área, manteniendo las ventajas de un procedimiento de construcción estadísticamente bien fundamentado y la posibilidad de ser interpretada por sistemas basados en reglas difusas. En relación con este trabajo, se proporcionan implementaciones para el lenguaje de programación R de

la clase de algoritmos meméticos utilizada. Además, se implementa un paquete completo de redes neuronales para R, que incluye muchos modelos de redes neuronales que pueden ser utilizados para la predicción, como perceptrones multicapa, redes recurrentes de Elman y Jordan, redes de función base-radial, etc.

Con respecto a la evaluación de los predictores, nuestro objetivo es el estudio de la materia desde una perspectiva de las Ciencias de la Computación, así como establecer buenas prácticas para la evaluación de predictores que empleen métodos de Inteligencia Computacional. Se lleva a cabo una revisión de los métodos existentes, para luego poner énfasis en la validación cruzada para series temporales, donde se propone el uso de un esquema de evaluación de predictores mediante validación cruzada por bloques, aprovechando así las ventajas de la validación cruzada y eludiendo sus problemas cuando se emplea en la predicción de series temporales. Además, se estudia la validación cruzada en el caso de uso especial de las medidas de precisión de la predicción direccional. Una implementación de toda la metodología de evaluación de predictores se presenta en el paquete `tsExpKit` para R. Además de la evaluación de predictores, el paquete también estandariza los formatos de datos, las descripciones de métodos, y la configuración experimental, con el fin de facilitar la realización de experimentos bien documentados y reproducibles.

1.6. Conclusiones

Hemos desarrollado nuevos métodos, nuevas normas de evaluación, e implementaciones fácilmente utilizables para la predicción de series temporales. Por lo tanto, la tesis pretende resolver problemas en todas las fases del procedimiento de predicción, y pretende ser de utilidad práctica, tanto por su teoría como por las implementaciones de software.

Los modelos de transición entre regímenes entrenados con algoritmos meméticos son procedimientos eficientes de modelado y predicción de series temporales. Son métodos híbridos que aúnan modelos estadísticos con procedimientos de optimización basados en Inteligencia Computacional (IC). Tienen las ventajas de solidez matemática y de fácil interpretación, y pueden ser predictores precisos, como sugiere el estudio realizado.

La familia de algoritmos meméticos con cadenas de búsqueda local es un procedimiento de optimización de IC del estado del arte. No sólo se ha utilizado para ajustar los parámetros de los modelos de transición entre regímenes, sino que también se ha implementado como un paquete de R, de modo que pueda utilizarse para optimización global por la comunidad de usuarios de R. Se ha demostrado en un estudio que es competitivo y a menudo mejor que muchas otras implementaciones en R de algoritmos de optimización. Además, se han implementado dos nuevos paquetes de software para el lenguaje de programación R. `tsExpKit` facilita la realización de experimentos estructurados y reproducibles para la predicción de series temporales, y `RSNNS` es un paquete que proporciona un conjunto de herramientas de redes neuronales para la comunidad de R. Este último contiene muchas implementaciones estándar de arquitecturas de red de las que carecía R, y es bastante exitoso, dado que ya tiene una cantidad considerable de usuarios.

En nuestros estudios de evaluación de los procedimientos de predicción, se realizó un extenso análisis del estado del arte, y se señaló que en el caso de los modelos autorregresivos puros para el pronóstico de series temporales estacionarias, la validación cruzada por bloques se puede utilizar sin problemas teóricos ni prácticos. Al utilizar los métodos de IC para la predicción, éste es con mucho el caso de uso más común, por lo que la validación cruzada por bloques podría convertirse en un procedimiento estándar para la evaluación de métodos de IC para la predicción de series temporales. Además, la validación cruzada es de particular interés cuando se utilizan medidas de precisión direccional, ya que las medidas de este tipo realizan una binarización y con ello pierden información, de modo que es posible que en las aplicaciones prácticas el procedimiento de evaluación tradicional no sea capaz de diferenciar los modelos. La validación cruzada puede superar este problema, ya que usa más datos para calcular las medidas de error.

Part I.

PhD Dissertation

2. Introduction

This chapter gives a general introduction into time series analysis and forecasting. Then, it shows possibilities and limitations of forecasting, and presents the general frameworks for autoregressive forecasting and for forecast evaluation. Furthermore, it presents the objectives of this work and its structure.

2.1. Time series analysis

A time series is a special data type, which explicitly takes time into account. It is typically a realization of the same measurements at different points in time. Time series are ubiquitous in everyday life, from weather forecasting, to signal processing (audio), to finance and economics, to traffic sciences, to physics, biology, medicine, and psychology. Throughout the last 40 years, time series analysis has grown into an important area of research, getting important input mainly from Statistics, Econometrics, Physics, and Computer Science.

The objectives and methods of time series analysis may be various. They are traditionally grouped into modeling, prediction, and/or characterization tasks of the time series. Furthermore, with the availability of huge amounts of data nowadays, in Computer Science new tasks become important, mostly connected with mining of time series data, like segmentation and indexing of time series.

Though all of these applications may have different purposes, they share some general concepts and are related to each other. Time series characterization means to find general statistical descriptions of the series. It may be used to complement modeling or forecasting, in the sense that it is used as a first step to find good methods for the next step, and to find overall limitations. Time series modeling aims to understand the underlying system that causes the dynamics in the series. Forecasting aims to predict future values of the series, usually with a rather short time horizon. Though modeling and prediction are related, they are not the same. Understanding the overall behavior of the system does not necessarily yield the best short-term forecasts, and in order to perform well in (short-term) forecasting, it is often not necessary to know and understand the underlying system (this can be validated with relatively easy experiments, as in [42]).

Some typical examples of time series analysis are given in Figure 2.1. An electrocardiagram (ECG) shows the periodic electrical activity of the heart. This activity reflects the different contractions taking place in a functioning heart in order to pump blood through the body. In medicine and biology, one such cycle is commonly divided into P, Q, R, S, and T wave. Depending on the

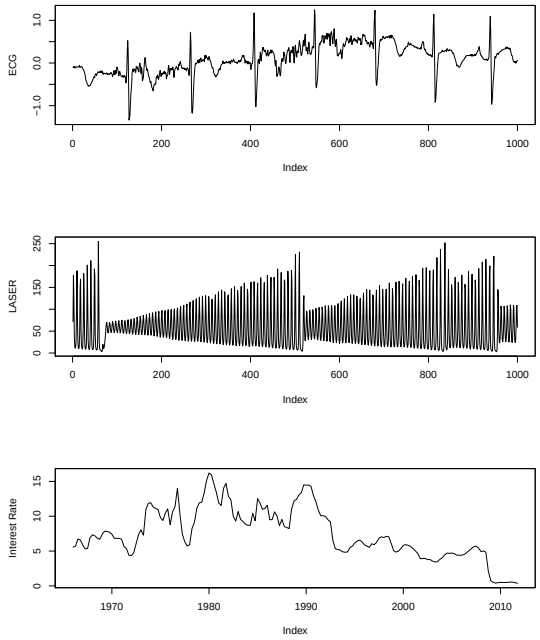

Figure 2.1.: Three examples of time series from different application contexts. The first series is an electrocardiogram of a normal human cardiac rhythm [61]. The second series is a measurement from a Far-Infrared-Laser in a chaotic state [78]. The third series is the monthly UK interest rate [6].

characteristics of these waves, physicians can find abnormalities and problems. As the underlying system is understood pretty well, the time series can be modeled in this way, and the modeling has important applications in medicine and biology. However, in this example, making forecasts is probably not very useful. The second figure shows a measurement from a Far-Infrared-Laser in a chaotic state. Also in this case, modeling the time series and understanding the underlying system are the primary tasks, though this series was also used in forecasting competitions in the past. The third example from the figure is the monthly UK interest rate. Here, forecasting clearly is a valuable task, as a good/bad forecast may have great economic impact. However, the underlying dynamics are difficult to analyze and understand.

The focus of this work is time series forecasting, where single values in the near future of the time series are to be extrapolated. However, through the connections of time series modeling and forecasting, we will also touch some aspects of modeling.

2.2. Forecasting: possibilities and limitations

In prediction, the fundamental assumption is that the future depends on the past. Important concepts are univariate and multivariate forecasting. In univariate forecasting, the assumption is that the future of a time series depends (mostly) on its own past, so that knowledge only about the time series' past is sufficient to perform a useful forecast. In multivariate forecasting, exogeneous variables, i.e., measurements from other sources, are also taken into account.

An important prerequisite for time series analysis and forecasting are data, which are often abundantly available nowadays. Furthermore, forecasting methods are generic and easy to apply, so that forecasts can be performed and are performed for nearly everything. However, a fundamental ability of a good forecaster is to understand the possibilities and limitations of forecasting, in order to assess the usefulness of a forecast.

Predictability of real-wold phenomena varies widely. E.g., the time of tomorrows sunset (or the time of sunset on a day 7 years from now) or tomorrows TV program schedules can be forecast quite accurately. In contrast, forecasting tomorrow's lottery numbers is pretty much impossible, though a lot of past lottery-number data may be available. There are many real-world phenomena that are not foreseeable in any meaningful way, or at least not foreseeable with the precision necessary for the forecast to be useful. And it is largely the forecasters responsibility to distinguish between cases where the applied methodology will work, and cases where it won't.

Another problem is that the true uncertainty may not be represented well in the data. The data may seem pretty regular and predictable, until a single, unforeseeable event happens. Then it may be important, if the unforeseeable event can be considered an outlier, i.e., not important for the overall series and the intended applications, or if the outlier is precisely the most important part of the phenomenon and the application. Forecasting as we understand it in this work is not about foreseeing drastic, unexpected changes. It is about procedures that are the every day work-horses in the background of many computer systems used for planning, pricing, control, and administration.

Following Hyndman and Athanasopoulos [34], three factors contribute to predictability: (1) how well we understand the factors that are involved in the creation of the phenomenon we want to forecast, (2) how much data we have, and (3) if the forecast we make will affect the future.

So, in order to forecast, it is important to have data. Then, it is important to understand at least in part the underlying system, in order to assess if the data are representative and sufficient for the forecasting task to perform. And then, after forecasting, the forecast should not change the actual behavior of the target variable.

The aim of using mathematical methods for forecasting, as in this work, is to model adequately repetitive patterns and central characteristics of past data. Noise and unique, unforeseeable events are not useful in such a forecasting framework. So, the central idea is to separate the time series into a part with predictive value, and a part of noise. The part of predictive value then has to be modeled accurately, without distraction caused by the noise.

There are lots of applications where forecasting yields very good results and the forecasts are

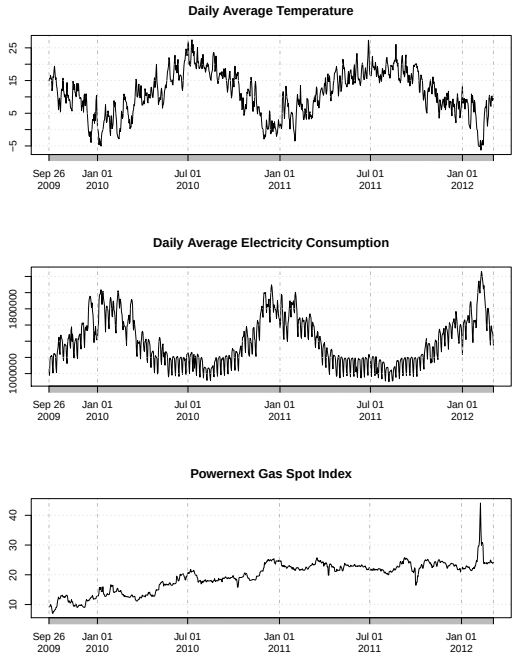

Figure 2.2.: The first series is the daily average temperature for Paris (obtained from: http://academic.udayton.edu/kissock/http/Weather/). The second series is the amounts of daily energy consume of metropolitan France, except Corsica (source: http://www.rte-france.com). The third series is the Powernext gas spot index, which represents the gas price in France (source: http://www.powernext.fr).

used successfully, for example when predicting electricity consumption, ATM cash withdrawal amounts, traffic related data such as car throughput of certain streets and crossings, call center load, warehouse load, hotel bookings, flight bookings, and others. But there are certainly other examples, especially in relation with the 2008 global financial crisis, where forecasting greatly failed. Predictions in Economics of, e.g., prices, exchange rates, or stock quotations, have several difficulties. Data is usually abundant, but the underlying systems are not understood well and the forecasts may greatly affect the future, as such time series are usually not functions of their own past, but of people's expectations about their future.

In the following, we present an example that illustrates some basic aspects of forecasting and predictability. The first series of Figure 2.2 shows the daily average temperature from September 2009 until March 2012 in Paris, France. Weather forecasting is probably the most well-known application of forecasting techniques. The series has a strong seasonal pattern, which could be

used for forecasting (it is pretty easy to foresee that next winter it will be colder than next summer, and that it will be more or less the same temperature as this winter). Also, the series is stable in the sense that the naïve forecast, which is simply to take the last known value as the forecast, would give us a basic but usable short-term forecasting result. However, for weather forecasting there exist specialized research disciplines that try to understand the underlying phenomena, identify the factors that influence, e.g., the temperature (temperature is mostly influenced by the number of daylight hours, thermal advection, wind speed, and cloud cover), and then take adequate, periodic measurements of these variables, in order to perform the forecasting.

The second time series of Figure 2.2 shows the daily consumption of electrical energy in France in the same time period. We see strong weekly patterns and a strong seasonal pattern which has a negative correlation with the temperature: The colder it gets, the more energy is consumed. All this information can be used for forecasting, either by only using the weekly and seasonal patterns from the past of the series (univariate forecasting) or, which is in this case clearly better, the patterns in combination with the domain-specific temperature forecasts. This enables pretty accurate forecasts for the electricity consumption.

One thing we note is that towards the end of the temperature series, there is a period in February 2012 of very cold weather. During that time, there was a strong cold wave in all of central Europe. We see this period of very low temperatures directly reflected in a peak of electricity consumption on those days.

Having this in mind, we now want to take a look at another forecasting task. The third time series is the Powernext gas spot index, which reflects the gas price in France, for the same time period. Note that, during the cold wave of February 2012, the gas prices have a very sharp peak. It is reasonable to assume that this peak is caused by the period of cold weather, probably due to supply shortages or similar incidents. Taking into account only the past of the gas price, the event was clearly not predictable. The series has just not behaved in a similar way before. And, though in this particular case the series has a relationship to the temperature, this relationship is much more indirect than in the electricity consumption case, and in the data before this extreme event, the gas price seems not to have any connection with the temperature. So, it is probably more the case that not the cold weather itself yielded to the high gas price, but more the fact that intense and duration of the cold weather was *unexpected*. In summary, forecasting the gas price in this example is much more difficult than forecasting energy consumption.

2.3. A general forecasting framework

An important concept related to predictability is stationarity. The fundamental idea of stationarity is that the main statistical characteristics of the time series do not change over time. Stationarity can be formally defined as follows (see Cryer and Chan [18]). A time series x_t is strictly stationary, if for any time points $t_1,\ldots,\ t_n$ and any lag parameter k, the joint distribution of $x_{t_1},\ldots,\ x_{t_n}$ and $x_{t_{1-k}},\ldots,\ x_{t_{n-k}}$ is the same.

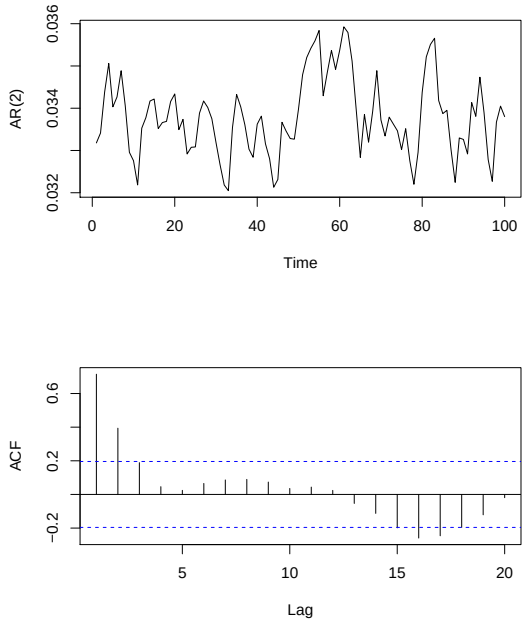

Figure 2.3.: The upper image shows a time series, which is generated from a linear autoregressive process. The lower image shows its autocorrelation function.

A time series x_t is weakly/second order stationary, if the first and second order moments do not change over time. I.e., the mean function is constant over time, and the covariance only depends on the time lag between two time series values (from this follows directly that also the variance is constant over time):

$$E(x_{t_1}) = E(x_{t_{1-k}})$$

and

$$Cov(x_{t_1}, x_{t_2}) = Cov(x_{t_{1-k}}, x_{t_{2-k}})$$

for any time points x_{t_1}, x_{t_2}, and any lag k. If mean and covariance exist, weak stationarity follows from strict stationarity.

It is common practice in forecasting to limit considerations to stationary time series, and to special cases of non-stationarity. The non-stationarity is then typically tackled by pre-processing steps. De-trending and de-seasonalization are used to remove deterministic changes in the mean. Other common methods are, e.g., to differentiate or logarithmize the series to achieve stationarity.

If no appropriate pre-processing can be found to remove non-stationarity, it may be appropriate not to use the whole past of the series for forecasting but only a part from the end [20], or forecasting may not be possible at all [20, 39].

index	lag 4	lag 3	lag 2	lag 1	target
		...	...		
7	14	10	26	11	-13
8	10	26	11	-13	-15
9	26	11	-13	-15	-8
10	11	-13	-15	-8	35
11	-13	-15	-8	35	40
12	-15	-8	35	40	-8
13	-8	35	40	-8	-16
14	35	40	-8	-16	7
15	40	-8	-16	7	17
		...	...		

Table 2.1.: Example of a time series pre-processed for regression, using the last four values (lagged values) to predict the current value.

As in a stationary time series, correlations are constant over time, in the sense that they only depend on the lag, but not on the exact point of time, a meaningful autocorrelation function can be calculated. The autocorrelation function determines the lags of the time series that show (significant) correlation. An example is given in Figure 2.3.

In this way, it is possible to identify the time lags which are relevant for forecasting, construct a matrix whose columns are the lagged versions of the time series (see Table 2.1 for an illustration), and use this matrix as inputs to a regression method. This model architecture is called the autoregressive model of fixed order, and the pre-processing step to construct the matrix is called embedding of the time series. We note that, as stationarity directly justifies the use of the autoregressive model of fixed order, if the series is not stationary, use of this model architecture may not yield good results. Let $x(t)$ be a time series, and let d be the delay parameter of the maximal lag we want to use for forecasting. Finding a forecasting model can then be defined as finding a (linear or non-linear) function F, such that

$$x(t) = F(x(t-1),\ \ldots,\ x(t-d)) + e(t), \tag{2.1}$$

where $e(t)$ is a time series of independent, identically distributed (i.i.d.) error terms. This definition reflects directly the separation of useful information and noise already mentioned in Section 2.2. So, the key concept of this definition is not just to find a separation, but to find the separation in a way to make the errors i.i.d., so that F adequately captures all information

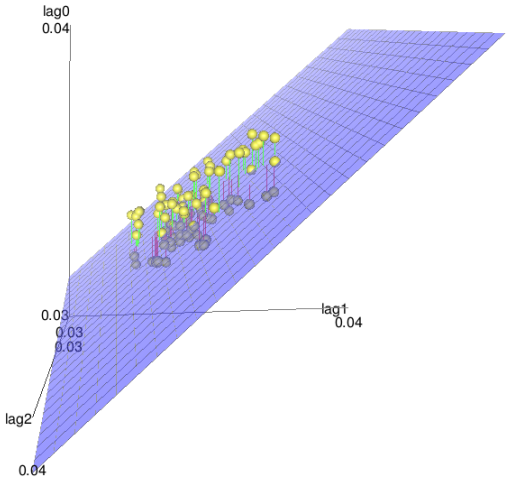

Figure 2.4.: Three dimensional scatter plot of the series from Figure 2.3. the values of the time series at time t are plotted against the values at time $t-1$ and $t-2$. A linear relationship between the three time lags can be identified, and the blue plane shows a linear regression of the data.

valuable for forecasting.

Using the framework of Equation 2.1, a linear autoregressive model (as in Box and Jenkins [13]) is defined by choosing F in a way that it computes a linear combination of past values of the series:

$$x(t) = a_0 + \sum_{i=1}^{d} a_i x(t-i) + e(t). \tag{2.2}$$

Here, $a_0, \ldots, a_d$ are the coefficients of the linear model (that need to be estimated during model building). With $\mathbf{z}(t) = (1,\ x(t-1),\ x(t-2), \ldots,\ x(t-d))^T$ and $\mathbf{a} = (a_0, \ldots, a_d)$, Equation 2.2 becomes in vector notation:

$$x(t) = \mathbf{az}(t) + e(t). \tag{2.3}$$

Figure 2.4 shows a three dimensional scatter plot of the series from Figure 2.3, where the values of the time series at time t are plotted against the values at time $t-1$ and $t-2$. From this plot, we can clearly see the linear relationship between the values at a given time and the values of lag one and lag two.

2.4. Predictor evaluation

After having specified a predictor and estimated its parameters, it is important to assess its quality. A critique of Taleb [73] is that in Economics, scientists perform forecasts without any feedback on how good the forecasts were. Other critiques are that both human experts and automatic forecasting methods are not capable of forecasting the most important variables [50, 51], and that humans tend to greatly underestimate the uncertainty of forecasts. Taking this into account, evaluation is a central aspect in forecasting, as forecasts can be always made, but in order to know whether they will be useful, or whether they may even be dangerous and harmful, we need to assess their quality.

One of the first to systematically assess the quality of forecasting methods was Makridakis in the so-called M-competitions, M [47], M2 [48], and M3 [49]. Within these competitions, the participants were asked to provide forecasts for a lot of different time series. The forecasts were then evaluated on test sets (not disclosed to the participants beforehand) consisting of the following values of the respective series, and errors were calculated using different error measures. However, since the early beginnings of the work on forecast evaluation, the field is still today an active area of research, and an important part of the work on the thesis was carried out in this field.

Besides different types of forecasts, error measures, and ways for data partitioning, another important question is, which consequences wrong forecasts have. The forecasts are usually not performed for the sake of forecasting, but in order to take informed decisions. Depending on the forecast, the decision may vary, and the error should take into account, in which way wrong decisions result from the forecasts. One example here are directional accuracy measures that measure if the forecast is able to correctly predict the direction, i.e., whether a variable rises or falls, as this is typically an important criterion on which decisions are based.

2.5. Objectives of this work

The aim of this work is to explore different aspects of the whole forecasting process, in order to achieve good forecasting performance in practical problems. The idea is that we need to consider model identification, model estimation, and model evaluation, to find and be able to use the best suited models for a concrete task. Then, both regarding models and model evaluation, the aim of this work is to consider theoretical and practical problems, to improve over various aspects of the state of the art of algorithms and methods, best practices, and software. We both investigate on new procedures and present good software implementations, which make research reproducible and help in diffusion and technology transfer. Figure 2.5 illustrates this with a schematic classification of the main parts of the thesis.

Concretely, regarding forecasting models, we propose a new technique for time series prediction, which combines the NCSTAR procedure with a powerful memetic optimization algorithm, and we show that it is competitive to other techniques used in the field, while having the advantages of a statistically well-founded building procedure and the possibility for its interpretation by fuzzy

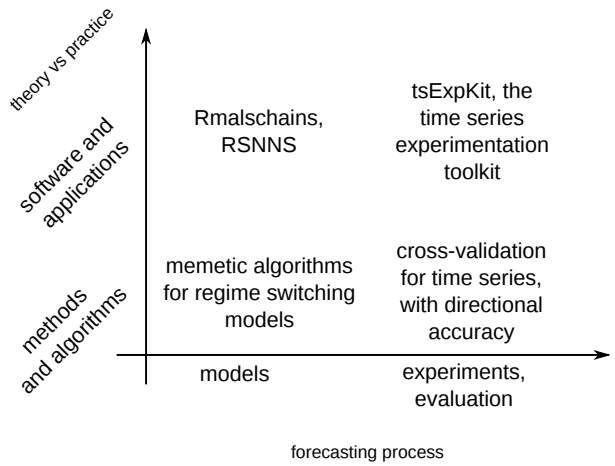

Figure 2.5.: Schematic classification of the main parts of the thesis.

rule-based systems. Related to this work, we provide implementations for the R programming language of the used class of memetic algorithms. Furthermore, we implement a comprehensive neural network package for R, which includes many neural network models that can be used for forecasting, such as multi-layer perceptrons, recurrent Elman- and Jordan-type networks, radial basis function networks, etc.

With respect to predictor evaluation, our aim is to study this field from a Computer Science perspective, and to find best practices for evaluation of predictors that employ methods of Computational Intelligence. We perform a review of existing methods and focus then on cross-validation for time series, where we propose the use of a blocked cross-validation predictor evaluation scheme, thus exploiting the strengths of cross-validation, while circumventing its problems when employed in time series forecasting. Also, we study cross-validation in the special use case of directional forecast accuracy measures. A practical implementation of all the predictor evaluation methodology is presented with the package `tsExpKit` for R. Besides predictor evaluation, the package also standardizes data formats, method descriptions, and experimental setup, in order to facilitate realization of well-documented, reproducible experiments.

2.6. Structure of the document

The thesis consists of two main parts. In the first part, named PhD Dissertation, we resume and connect the content of the whole thesis. This chapter, Chapter 2, provides an introduction to the topic in general. The next chapter, Chapter 3, presents the main concepts and results of the thesis, and Chapter 4 concludes. The second main part of the thesis presents the journal

papers that were written within the work on the thesis. Each chapter in the second part is an independent publication. The first publication presents the approach of regime switching models combined with memetic algorithms for parameter estimation. The second publication presents the software package for memetic algorithms created within this work, `Rmalschains`. The third publication presents another software package for neural networks, `RSNNS`. The forth publication reviews predictor evaluation methodology, and proposes blocked cross-validation as a substitute for usual out-of-sample evaluation. Finally, the fifth publication presents the special case of using blocked cross-validation together with directional accuracy measures.

3. Discussion of the Results

In this chapter we point out and discuss the most important concepts and results of the research.

3.1. Regime-switching models adjusted with evolutionary algorithms

In this section, we describe the work that was performed within the thesis on the time series model family of regime-switching models. We investigated on the use of memetic algorithms for parameter estimation of such models. The associated journal paper is:

> C. Bergmeir, I. Triguero, D. Molina, J.L. Aznarte, and J.M. Benítez. Time Series Modeling and Forecasting Using Memetic Algorithms for Regime-Switching Models. IEEE Transactions on Neural Networks and Learning Systems, 2012, 23(11), 1841-1847.

Furthermore, during this work, the memetic algorithm family that we used was implemented as an R package, which is presented in the publication:

> C. Bergmeir, D. Molina, and J.M. Benítez. Memetic Algorithms with Local Search Chains in R: The Rmalschains Package. Journal of Statistical Software, 2012, (submitted).

The software is discussed in more detail in Section 3.2. In this section, we focus on the theoretical aspects.

With the work of Box and Jenkins [13], linear autoregressive (AR) models became a standard procedure in time series forecasting. However, linearity is a pretty strong assumption, and more flexible models are desirable (in this context, note that the abbreviation "AR" usually stands for linear autoregressive models, and not non-linear ones).

A popular extension of the Box-Jenkins methodology to non-linearity is the approach of Tong [75]. Its central idea is to use several linear models for different parts of the series. The resulting models

are called piece-wise linear models. Which linear model to use concretely is determined by a function, called threshold function, which takes certain characteristics of the time series and/or exogenous signals into account. The threshold function partitions the time series into non-overlapping regions, and different linear models, called "regimes" in this context, are used to model the regions. E.g., in a time series in which general periods of rise and fall alternate, the threshold variable can determine whether at a certain time point the series is in the rising or the falling case. The periods of rise can then be modeled by a linear model, and the periods of fall by another one.

In general, a piece-wise linear model is defined as follows:

$$x(t) = \sum_{j=1}^{k} f_j(th_j(t))\mathbf{a}_j\mathbf{z}(t) + e(t). \tag{3.1}$$

Here, f_j are nonlinear functions that determine, how the combination of the linear models is performed, and th_j are the threshold functions. This general equation enables the definition of various models, depending on the choice of f_j and th_j. In the so-called threshold autoregressive model (TAR), we use the indicator function for combination, which has a logical condition as argument and returns one if the condition is met, and zero otherwise. Using threshold constants $c_0, \ldots, c_k$ with $-\infty = c_0 < c_1 < \ldots < c_k = \infty$, the functions f_j are defined as follows:

$$f_j(th_j(t)) := I\left(th_j(t) \in (c_{j-1}, c_j]\right) \tag{3.2}$$

Then, choosing $th_j(t)$ as a concrete lagged value $x(t-d)$ of the time series, we define the so-called self-exciting TAR (SETAR) model. Use of the indicator function yields very abrupt changes. This may not be natural and not desirable in the model. To get smoother transitions between the regimes, e.g., the logistic function can be used, and we obtain the smooth transition autoregressive model (STAR). The logistic function is defined as follows:

$$f_j(th_j(t)) = \frac{1}{1 + \exp(-\gamma_j(th_j(t) - c_j))}. \tag{3.3}$$

Finally, also within the STAR models, various proposals of threshold functions are present in the literature. In this work, we use the following threshold function to define the neuro-coefficient STAR (NCSTAR) model:

$$th_j(t) = \omega_j\mathbf{w}(t). \tag{3.4}$$

The vector $\mathbf{w}(t)$ contains all relevant lagged and exogenous variables, which are weighted using the weights vector ω_j. The weights vector has the same dimension as $\mathbf{w}(t)$, and is normalized to length one, i.e., $\|\omega_j\| = 1$. In our work, we assume $\mathbf{w}(t) = (x(t-1),\ x(t-2), \ldots,\ x(t-d))^T$, i.e., no exogenous variables are used and all lagged values used in the autoregression also contribute to the threshold. The model is called "neuro-coefficient" STAR, because it can be interpreted as a linear model with coefficients that change over time, and which are determined by a multi-layer perceptron [2].

The model was originally proposed by Medeiros and Veiga [53], who also presented a statistically founded, iterative building procedure, which involves a statistical test of linearity. As long as non-linearity remains in the residuals, the method adds new regimes to the model (i.e., it adds hidden units to the multi-layer perceptron). Another advantage, besides the well-defined building procedure, is that this kind of model can be interpreted in terms of fuzzy rule-based systems (FRBS) [2, 4, 3, 5]. So, the modeling results can be used not only in a black-box manner, but can also be used to interpret the findings and to gain insight into the factors that contribute to the behavior of the time series.

However, a potential weakness in the original NCSTAR building procedure is the fitting algorithm. Medeiros and Veiga use a combination of a grid search and a local search procedure, in order to determine the non-linear parameters of the model. The fitting procedure directly affects the accuracy of the model and the behavior of the iterative building procedure.

Complementarily, evolutionary algorithms [21] have proven to be efficient optimization techniques in a broad range of application fields [76]. In this work, we evaluate a memetic algorithm family, called memetic algorithms with local search chains (MA-LS-Chains) [54], for using it to substitute the original fitting algorithm.

The NCSTAR model has the non-linear parameters γ_j, c_j, and ω_j with $j = 1,\ldots,k$ as in Equations (3.3) and (3.4), which model the transitions between regimes, and linear parameters which represent the regimes. The linear parameters can be computed in a closed-form solution, once the non-linear parameters are fixed, i.e., in every evaluation of the fitness function during optimization. The γ_j scale with the series, so we use normalized data to be able to define a domain for the γ_j.

The original model fitting procedure is a combination of grid search and local search. It draws randomly a huge amount of initial solutions for a new regime to add, and evaluates the fitness of these initial solutions combined with the regimes already present. Then, the best solution from these initial solutions is chosen, and a local search algorithm, concretely the Levenberg-Marquardt algorithm [56], is used to improve on all parameters of this solution.

Within our work, we replaced this fitting procedure with MA-CMA-Chains [54], a memetic algorithm with local search chains that employs CMA-ES [29] as the local search strategy. The MA-LS-Chains paradigm employs a steady-state genetic algorithm [79], together with the BLX–α operator [23] for crossover, the negative assortative mating strategy [27] as its selection method, replace worst as replacement strategy, and the BGA operator [58] for mutation.

In order to compare the performance of the new algorithm, we performed an experimental study on time series from the NNGC1, NN5,[1] and Santa Fe [78] forecasting competitions (30 series in total), where we compared the algorithm to two versions of the original algorithm and to five standard methods in the field.

The results were analyzed both for average performance and using statistical testing procedures. The analysis indicates that the proposed algorithm is able to outperform the compared

[1] See http://www.neural-forecasting-competition.com

methods in terms of accuracy. As interpretability is an important advantage of the NCSTAR model, furthermore we performed an analysis about the amount of regimes (which translates into amount of hidden units or amount of fuzzy rules) that are generated by the original and by the proposed building procedures. The proposed building procedure generates with approx. 6 rules on average slightly more rules than the original procedure with 5.5 rules. However, we do not consider this as an important change w.r.t. interpretability, so that the proposed method yields more accurate results, with mainly preserving interpretability.

3.2. Software for Computational Intelligence in time series research

In this section, we present the software projects that were realized within the thesis. Usually, software development is not appreciated much in Computer Science research. The focus is on inventing new algorithms and methods. Robust, usable implementations, which promote the research results and facilitate reproducing and using the methods are often disregarded and not considered within the scope of research. However, this attitude is changing in the research community. In many fields of Computer Science, there is nowadays a gap between the state of the art and the methods that people outside of the narrow research field are actually using. This has mostly to do with the availability of working, usable implementations.

Also, open source is now a widely accepted licensing and distribution form for software, and initiatives for open data and open access publishing of scientific results gain importance. This openness of data, source code, and research publications, greatly facilitates and accelerates research, verification, reproduction, and exploitation of the results.

Regarding this new way in which research is carried out, a pretty complete environment is the ecosystem around the R programming language [65]. R is an open source programming language for statistical computing. It is an interpreted, functional, rapid prototyping language, and the de-facto standard in statistical computing. R offers the possibility to write plugins (which are called "packages" in R), which can easily be published by uploading them to the Comprehensive R Archive Network (CRAN)[2]. Furthermore, there are tools for in-source documentation [81], report generation [45], and unit testing [80]. With the Journal of Statistical Software, there is furthermore an open access journal where high-quality software of this type can be published.

All our projects were realized as packages for the R programming language. Besides the package `Rmalschains`, which implements the MA-LS-Chains algorithm family, the packages `tsExpKit` and `RSNNS` were implemented. The package `tsExpKit` is an infrastructure package to facilitate time series research, and it implements all of the evaluation methodology presented in Section 3.3. `RSNNS` is a neural network toolbox package that implements many neural network standard architectures and learning algorithms.

[2] http://cran.r-project.org

There are two publications associated with this section. The first publication presents the package `Rmalschains`, explains its theory, its implementation, and performs a comparison of `Rmalschains` and other packages available in R for optimization:

> C. Bergmeir, D. Molina, and J.M. Benítez. Memetic Algorithms with Local Search Chains in R: The Rmalschains Package. Journal of Statistical Software, 2012, (submitted).

The package `RSNNS` is presented in the second publication, where we also give an overview of all the neural network architectures implemented, and compare the package to other neural network implementations in R:

> C. Bergmeir and J.M. Benítez. Neural Networks in R Using the Stuttgart Neural Network Simulator: RSNNS. Journal of Statistical Software, 2012, 46, 1-26.

In the following, we discuss the software packages in more detail:

3.2.1. tsExpKit

The time series experimentation kit `tsExpKit` is a package facilitating experimental setup and evaluation of the experiments in time series research. It implements all methods explained in Section 3.3. Furthermore, it facilitates parallelization of experiments, and implements its own standard format for time series (based on the `zoo` [83] package). It helps with the creation of time series data repositories, where the time series are stored in a unified data format. Experimentation, documentation, and reproducibility are facilitated by, e.g., caching results of preprocessing and storing experiment results in a unified data format.

3.2.2. Rmalschains

The package `Rmalschains` was implemented in the first place for the work presented in Section 3.1. However, the MA-LS-Chains algorithm family is a general optimization framework, with many potential applications. In this section, we focus on the software implementation, its benefit for the R community, and its advantages over available alternatives.

In general optimization, the goal is to find parameter values for a given function that maximize or minimize the output of the function. In this context, a parameter configuration is usually called a solution, the function to optimize is called target- or fitness function, and the output of the function is called fitness. Optimization problems need to be solved in many modern applications and in nearly all fields of modern science and engineering. As nowadays the problems to solve are often complex, a mathematical analysis is not feasible or too expensive and time consuming. Also,

evaluating all solutions in the parameter space in a brute-force manner is usually not possible, due to the size of the parameter space and limitations of computational resources. Problems of this type are traditionally tackled with local search algorithms, which start with one solution and find a way through the parameter space by altering this solution. However, as new solutions are only searched in the neighborhood of the current solution, local characteristics of the search space, such as local optima or flat regions, may lead to unwanted results. In order to tackle this problem, in the literature there are many meta-heuristics proposed, using different methods to find their way through the parameter space to a good solution, evaluating only a small fraction of all possible configurations.

A popular family of meta-heuristics are evolutionary algorithms, which use a set of possible solutions, the population. The solutions in the population, also called individuals in this context, are combined and altered by crossover and mutation operators, and solutions with the best fitness are favored, so that the population moves towards better fitness values. By evaluating a set of solutions in parallel, evolutionary algorithms avoid getting stuck in local optima and perform well exploring the search space. But they may lack exploitative power to find the best solution in a determined region. In contrast, local search techniques can improve quickly on a determined solution, reaching the local optimum.

Memetic algorithms [57, 41] are a hybridization between evolutionary algorithms and local search methods. The idea is to combine the strengths of both algorithm types, thus exploring the search space with the evolutionary algorithm and using the local search to exploit promising regions by finding local optima. In this context, an important question is which regions are considered to be promising, and how the overall amount of function evaluations that is to be dedicated to the local search will be distributed among regions, so that the local search is applied more intensely to more promising regions.

MA-LS-Chains solves this issue by applying the local search various times over the same solution, with a fixed amount of iterations. Then, the final state of the local search is saved, and reused as initial state in a subsequent local search application to the same individual. It uses a steady-state genetic algorithm [79], negative assortative mating [27], the BLX–α crossover operator [23], the replace worst replacement strategy, and the BGA mutation operator [58].

Different local search algorithms are implemented in `Rmalschains`, namely the CMAES algorithm [29], which can be considered the state of the art in local search methods, the Nelder-Mead downhill simplex [60, 59], the Solis-Wets' algorithm [71], and an adaptation of the latter one to high-dimensional problems, where in every iteration a subset of all parameters is chosen for optimization (called "Subgrouping Solis-Wets").

There is already a host of choices of optimization methods present in R. However, MA-LS-Chains performed well in several optimization competitions (BBOB'2009, CEC'2010, and in experiments subsequent to CEC'2005), and it takes high-dimensional optimization into account. Also, following the "No Free Lunch" theorem [82], there cannot be one single algorithm that performs best on all types of problems, so especially in optimization it is valuable to be able to evaluate different algorithms for a concrete problem.

Besides these general justifications and motivations for making available MA-LS-Chains as an R package, we performed a study comparing other packages on CRAN to `Rmalschains`.

Concretely, we use the functions/packages `bobyqa`, which implements the algorithm of Powell [63], `cmaes`, which implements the CMAES algorithm, `DEoptim`, which implements differential evolution [64], `dfoptim`, which implements the Nelder-Mead and the Hooke-Jeeves algorithms, `malschains_cmaes`, `malschains_sw`, which are methods from `Rmalschains` with different local search techniques, `optim_BFGS` and `optim_L-BFGS-B`, which are two different versions of the Broyden-Fletcher-Goldfarb-Shanno algorithm, `PSO`, which implements a particle swarm optimization, `Rsolnp`, which implements a sequential quadratic programming (SQP) solver, `GenSA`, which implements generalized simulated annealing [77], and `rgenoud`, a genetic algorithm which also allows for applying local search.

We use the benchmark of Lozano et al. [46], consisting of 19 benchmark functions, and use it with problem dimensions of 2, 10, 30, 50, 100, 200, 500, and 1000. Here, we consider dimensions 2, 10, 30, 50 low-dimensional, dimensions 100 and 200 medium-dimensional, and dimensions 500 and 1000 high-dimensional. As many algorithms involve randomness in some form, each algorithm is run 25 times on each problem and every dimension. We measure running time and average error of the methods. For the average error, rankings of the algorithms are analyzed.

With application of the methods to problems with very different dimensionality, the focus of the analysis is on scalability of the methods. Our analysis takes into account three different causes for methods not scaling well: execution time, memory usage, and errors/exceptions during program execution.

The results show that the `Rmalschains` methods in terms of average error are very competitive, and perform best for dimensions 30, 50, and 100. However, the CMAES algorithm has a complexity of $O(n^3)$, so it inherently does not scale well. This is why both the available CMAES implementation in R and the MA-CMA-Chains implementation in `Rmalschains` cannot be used for dimensions larger than 100. For higher dimensions, MA-LS-Chains with the Solis-Wets' solver achieves best results in terms of accuracy, so that we can conclude that `Rmalschains` is a competitive implementation of a memetic algorithm family, thus being of use to the R community.

3.2.3. RSNNS

When developing new time series forecasting methods and assessing their performance, besides the evaluation methodology presented in Section 3.3, it is important to compare newly developed methods to the state-of-the-art. Therefore, it is convenient to have a canon of implementations of standard methods. In R there are already many Computational Intelligence (CI) methods and regression methods available, which can be used for autoregression. Also, a lot of statistical forecasting methodology is readily available. However, during the work on the thesis, we observed a lack for a comprehensive neural network standard library, and especially recurrent neural networks that are relevant for time series forecasting were not available in R. So we decided to implement a standard library of neural networks for R. As it is good practice to reuse code and comply to stan-

dards whenever possible, we decided to adapt an existing software, the Stuttgart neural network simulator (SNNS), for this purpose, and the newly implemented package is called RSNNS.

The SNNS is a neural network software implemented in C, which contains many high-quality implementations of neural network standard procedures. In terms of network architectures, SNNS implements multi-layer perceptrons (MLP) [69], recurrent Elman-, and Jordan networks [22, 37], radial basis function networks (RBF) [62], RBF with dynamic decay adjustment [10, 32], Hopfield networks [31], time-delay neural networks [43], self-organizing maps (SOM), associative memories, learning vector quantization networks (LVQ) [40], and different types of adaptive resonance theory networks (ART) [28], concretely ART1 [15], ART2 [14], and ARTMAP [16] nets. Furthermore, it implements a wide variety of initialization, learning, update, activation, and pruning functions, adding up to more than 270 in total. For example, learning functions include standard backpropagation, backpropagation with momentum term, backpropagation through time (BPTT) [70], Quickprop [25], resilient backpropagation [68, 67], backpercolation [38], (recurrent) cascade-correlation [24, 26], counterpropagation [30], scaled conjugate gradient [55], and TACOMA learning [44].

SNNS has not seen active development since 1998. So, although the methods are well implemented in the kernel, the GUI is antiquated and cumbersome, and there are no satisfactory possibilities for automation and parallelization of the learning procedures, as well as for graphical visualization of the results. These issues were resolved in RSNNS. The whole SNNS kernel was ported to C++, encapsulating it in one class to get rid of global variables, and therewith allowing for managing more than one network at a time, which is convenient for automation and essential for parallelization. Furthermore, a high-level interface for the kernel was implemented in R, containing many functions that allow for the use of the original SNNS functions with interfaces that integrate seamlessly into the style of R programming. Though absolute download numbers cannot be assessed due to the distributed nature of CRAN, we note that the package receives a lot of attention in the R community, leading to more than 6000 downloads of the associated journal paper, and a flow of constant bug reports, ask for support, and feature requests by users. So, to conclude, RSNNS fills a gap in the functionality of R, and implements important functionality that many users appreciate.

3.3. Predictor evaluation

The evaluation of predictors is not trivial, and a field of active research. We contribute to this field in the following ways. We perform a systematic review of the whole process of predictor evaluation. Then, we focus on the last step, the model selection procedure, where we perform empirical studies characterizing quantitatively the effects of cross-validation, both with respect to typical application scenarios when using CI methods, and in a study using directional accuracy measures. The review of prediction evaluation procedures together with the study employing CI methods is published as:

C. Bergmeir and J.M. Benítez. On the use of cross-validation for time series predictor evaluation. Information Sciences, 2012, 191, 192-213.

The study about cross-validation with directional accuracy measures is presented in the following publication:

C. Bergmeir, M. Costantini, and J.M. Benítez. On the usefulness of cross-validation for directional forecast evaluation. Journal of Forecasting, 2013, (submitted).

This section summarizes the main points of the review, and the main results of the two empirical studies.

3.3.1. Review of predictor evaluation methodology

Throughout the last decades, a vast amount of different forecasting methods was proposed in the literature. Comparing forecasting methods and finding out which ones perform best for particular applications is an important endeavor. A generic evaluation methodology for regression, classification, and forecasting methods is as follows:

(i) identification and estimation/fitting of the models

(ii) calculation of the errors for every data point (for every data set, and possibly both for training and test set)

(iii) calculation of the (average) error measures per model and training/test set

(iv) use of a model selection procedure to determine the best model (per data set or overall)

In machine learning and other regression and classification tasks, application of this methodology is straightforward, and a commonly accepted consensus exists. After (i) calculating predictions for the target values, (ii) quadratic loss is used, which is optimal under the assumption that the errors are normally distributed, and (iii) the root mean square error is calculated. Finally (iv), k-fold cross-validation is used to randomly partition the data into k sets, obtain k error measures by using each of the k sets once as test set, and averaging over these k measures to get the final error measure.

In time series forecasting, such a consensus does not exist. It usually depends on the application, which forecast horizons are to be used for prediction, and which lagged values are to be taken into account (i). Also the loss that is associated with certain characteristics of the errors in (ii) often depends on the application, so that, e.g., percentage errors, errors relative to benchmark methods,

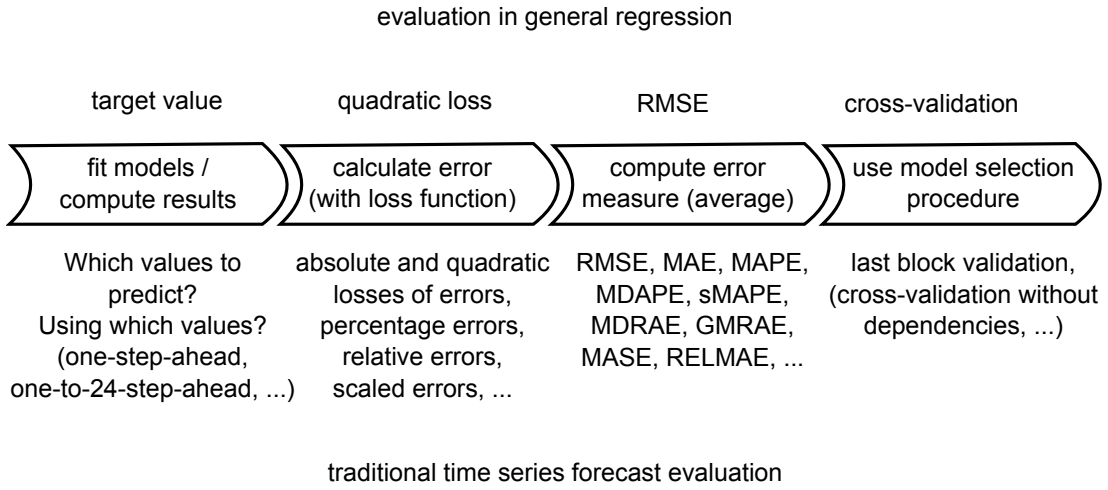

Figure 3.1.: A general evaluation framework for classification, regression, and time series prediction. While in classification and regression, a general consensus for evaluation practice exists, in time series prediction, a host of choices exists for every step of evaluation, many of which are application dependent.

or directional errors are used. Depending on the distribution of the errors, also the methods that are appropriate for averaging (iii) change. E.g., computation of the mean or the median of squared or absolute errors is common [19, 35]. Finally, as a model selection procedure (iv), cross-validation has both theoretical and practical problems. A fundamental assumption of cross-validation is that the data are independent and identically distributed (i.i.d.) [1]. Both assumptions may not hold for a time series, as the data is usually autocorrelated, and time-evolving effects may alter the data distribution over time. Regarding the practical problems, it may be the case that the forecasting method needs sequential input data, and cannot handle missing values (which are reserved for testing) properly. So, in forecasting often the last part of a series is withhold for testing. We call this evaluation mode last block evaluation. This type of evaluation is easy to perform in practice and does not have the theoretical problems of cross-validation. Furthermore, it often coincides with the intended application. However, last block evaluation does not make use of the data in the way cross-validation does, so that there are also various proposals in the literature regarding step (iv) for time series. An illustration of the evaluation procedure gives Fig. 3.1. In the following, we have a closer look at the evaluation process.

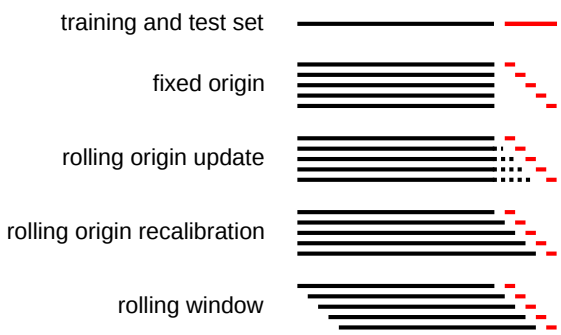

Figure 3.2.: There are many different types of data partitioning and different forecast horizons during forecasting.

Data partitioning and forecasting horizons

Which forecast horizons to use, i.e., what to forecast, and which values to use as inputs, usually depends on the application. This needs to be taken into account when partitioning the data into training and test set.

Following Tashman [74], we classify the different forecasting procedures as follows (see also Fig. 3.2):

- fixed origin: In this type of forecasting, the test set is not available when performing the forecasts. So, every forecast for the test set has a different horizon, depending on its distance in time from the training set.
- rolling origin update: Here, the model is estimated using the training set, but past values from the test set are used as input for the model, as they become available. The model is not altered/retrained during this process. For CI methods, this is the most common use case.
- rolling origin recalibration: In this forecasting type, the model is retrained/readjusted for every prediction in the test set, as the values become available.
- rolling window: As in rolling origin recalibration, the model is retrained for every prediction in the test set. But the size of the training set is fixed, so that the oldest values are discarded, as new data becomes available. This procedure is especially useful when the data is non-stationary (and changes are slow).

All the aforementioned forecasting types can be used with different horizons, and across different time series, which yields to a wide variety of different forecasting procedures that may be examined for a given set of time series. An example for application-dependence is the procedure usually employed in time series forecasting competitions, such as the M3 [49], M4[3], or NN3 [17],

[3] http://m4competition.com

NN5, or NNGC1[4] competitions. As the test set cannot be disclosed to the participants while they perform forecasting, only fixed origin forecasting is applicable here.

Errors and error measures

After forecasting, the predictions need to be compared to the target values in order to assess their quality. Therefore, errors which measure the deviation from the target value are computed, and averaged to an error measure, which ideally represents well the characteristics of the errors a forecaster can expect when using a certain method. Following the classification of Hyndman and Koehler [35], we classify types of errors and error measures into four categories, which are scale-dependent errors, percentage errors, relative errors, and relative error measures. However, as in time series forecasting the expected loss resulting from an error often depends on the application, there are several other evaluation measures. A type of measure which gets increasingly popular in Econometrics are directional accuracy measures [11, 12], which we also consider in our work.

Cross-validation for time series forecasting

A standard evaluation technique in regression and classification is k-fold cross-validation [72]. For this method, the available data is partitioned randomly into k sets. Then, every set is used once as the test set, and the training set consists of all other sets. In this way, k independent realizations of the error measure are obtained, instead of only one. So, a more robust error measure can be computed, and a better estimation of the "true" error can be achieved.

A fundamental assumption of cross-validation is that the data are independent and identically distributed (i.i.d.). In time series prediction, this is usually not the case, as the data have interdependence (which is precisely what is used for the forecasting), and time-evolving effects may occur, so that the distribution of the data may change over time. Because of these problems, there is a lot of work in the literature which proposes adaptations of cross-validation for the time series case.

Correct use of cross-validation for time series has a strong connection with stationarity of the time series. If the series is stationary, it is possible to find a maximal lag order after which the values are approximately independent. As the dependencies are precisely what is used for modeling, correct model order identification, which is crucial for modeling, also becomes important for the evaluation, as the chosen model order can be used as the time period after which two values are approximately independent. For a value in the test set, then all values from the training set within this period have to be omitted. The procedure is presented, e.g., by McQuarrie and Tsay [52]. It is often referred to as modified cross-validation [1]. We call it in the following non-dependent cross-validation. The problem of this procedure is that, depending on the time period chosen to omit, heavy loss of data will occur, which annihilates the advantages of cross-validation of making better use of the data, or use of the procedure may not be feasible at all.

[4] http://www.neural-forecasting-competition.com

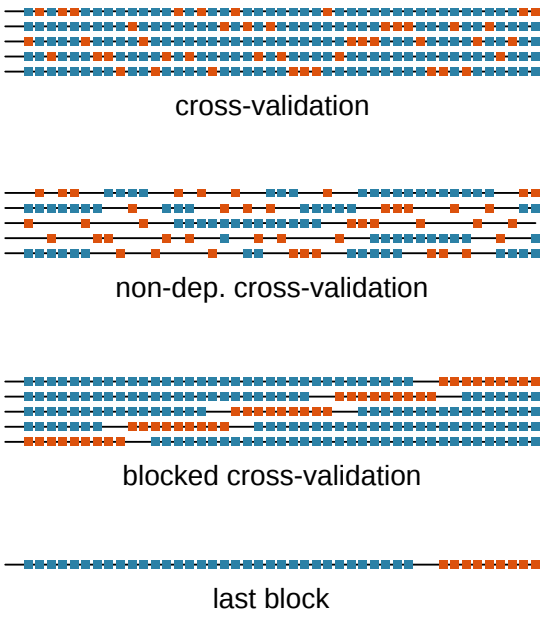

Figure 3.3.: Training (blue) and test (orange) sets for different types of evaluation procedures for time series.

The solution we promote for this problem of wasting data is to choose the training and test sets as blocks of data (following the ideas of Racine [66]), so that values have to be omitted only at the borders of these data blocks. We call this procedure k-fold blocked cross-validation. See Fig. 3.3 for illustrations of traditional cross-validation, non-dependent cross-validation, the proposed block scheme, and last block evaluation. So, using blocked cross-validation, the theoretical issues of unmet preconditions can be resolved in an elegant way. Now we want to have a look at the practical issues. When using CI methods for forecasting, typically only pure non-linear autoregressive models are used. In this context, missing values can be handled without problems, as the autoregression is performed by applying general regression methods to the embedded version of the time series. In this embedded version, which is a data matrix, withholding data is a straightforward task, as only the respective rows need to be removed from the matrix. Compare this to, e.g., exponential smoothing, or models with a moving average part, where for the computation of each new value all past values are necessary.

3.3.2. Experimental studies of cross-validation in predictor evaluation

In this section, we discuss the two experimental studies on the practical consequences of using different model selection procedures. The first study focuses on typical application cases of fore-

casting with CI methods. The second study focuses on the use of cross-validation together with directional accuracy measures.

Cross-validation in forecasting using CI methods

In order to examine and quantify the consequences of the use of different model selection methods in a real-world application scenario, we performed an experimental study. The experiments are designed to answer the following questions, for a typical application setup for CI methods used for forecasting:

- Do the dependencies in the data lead to underestimation of the error, when using traditional cross-validation without modifications?
- Do any time-evolving effects occur, in the sense that data taken from the end of the series for validation yields different results from taking the data from somewhere in between?
- Does the use of cross-validation yield more robust error measures?

In order to answer these questions, the following experimental setup was used. Besides the partitioning into training and test set, which depends on the evaluation procedure used, we withhold an additional validation set, from the end of the series. Then, for each series, predictors are trained, and forecasts are performed for the respective test sets, and error measures E_{in-set} are computed using the model selection procedures. Also, error measures $E_{out-set}$ are computed on the validation set. Then, the quotients

$$Q_E = \frac{E_{out-set}}{E_{in-set}}$$

are computed for every series and every method.

Using Q_E, the questions from above can be answered in the following way: If cross-validation systematically underestimates the error, the Q_Es would be systematically greater than one. The results presented in the associated journal paper indicate that this is not the case. There is a bias in the error measures, but not in the expected way of a continuous underestimation, and the bias is not stable, in the sense that it depends a lot more on the error measure used than on the model selection procedure. But the cross-validation is more robust in the sense that the errors as the results have less spread and fewer outliers. So, within the study no practical consequences of the dependent data could be found, i.e., also when not used correctly, cross-validation yields a good error estimate for the examined cases. Furthermore, no stable time-evolving effects could be found in the study, and it could be verified, that the cross-validation yields more robust results in practice.

Cross-validation with directional accuracy measures

In a second study, we focused on an interesting particular case of the methodology presented, namely for the case when directional accuracy measures are used. As these measures perform a binarization, and therewith loose information, here use of cross-validation can be especially valuable.

In this study, we performed a Monte Carlo simulation with a similar setup as the first study on cross-validation. The difference is that much more series were used, and both univariate and multivariate forecasting was considered, but only linear data generation processes and models were used. The results confirm the results of the first study, that cross-validation yields more robust and precise error estimates. Furthermore, with respect to particularities of the directional accuracy measures, we considered an application of UK interest rate forecasting. Our application shows, that if the amount of data is limited and the models yield similar results, due to the binarization of directional accuracy measures, the models may not be distinguishable any more when using last block evaluation. The cross-validation has considerable advantages here, and allows to discern the models, as it uses more data.

4. Conclusions

In this chapter we present some concluding remarks and lines of future research.

4.1. Summary and concluding remarks

We have developed new methods, new evaluation standards, and readily usable implementations for time series forecasting. So, the thesis aims to resolve problems in all phases of the forecasting procedure, and aims to be of practical use, both by its theory as by its software implementations.

Regime-switching models trained with memetic algorithms are efficient time series modeling and forecasting procedures. They are hybrid methods of statistical models with Computational Intelligence (CI) optimization procedures. They have the advantages of mathematical soundness and interpretability, and can be accurate forecasters, as our study suggests.

The algorithm family of memetic algorithms with local search chains is a state-of-the-art CI optimization procedure. We not only used it for adjusting the parameters of regime-switching models, but also implemented it as an R package, so that it can be used for global optimization by the R community. We showed in a study that it is competitive and often better than many other implementations of optimization algorithms of R. We furthermore implemented two more software packages for the R programming language, tsExpKit facilitates realization of structured, reproducible experiments for time-series forecasting, and RSNNS is a package which provides a neural network toolkit for the R community. It contains many standard implementations of network architectures R was lacking of, and it is pretty successful, with a considerable amount of users.

In our studies of predictor evaluation procedures, we performed an extensive study of the state of the art, and pointed out that in the case of pure autoregressive models for forecasting stationary time series, blocked cross-validation can be used without theoretical and practical problems. When using CI methods for forecasting, this is by far the most common application use case, so that blocked cross-validation could become a standard procedure in the evaluation of CI methods for time series forecasting. Also, cross-validation is of particular interest when directional accuracy measures are used, as this kind of measures performs a binarization and therewith looses information, so that it may occur in practical applications that the traditional out-of-sample evaluation procedure is not capable any more of distinguishing models. Cross-validation can overcome this problem, as it uses more data to calculate the error measures.

4.2. Future work

There are many paths open to follow from the work of the thesis in all three main lines of our research, i.e., regarding new methods, software, and evaluation. Concretely, we plan for the future to continue work on the following:

- Regarding hybrid forecasting methods, we plan to continue our work with the hybridization of exponential smoothing methods, in the line of Bermudez et al. [7, 8, 9]. Therefore, a research stay with Rob Hyndman is scheduled, who is an expert in the field of exponential smoothing [33, 36].
- Work on software is an ongoing process, and all the software packages are implemented in a way that facilitates maintenance. Furthermore, we are currently working on R packages for fuzzy rule-based systems and the exponential smoothing methods.
- With respect to evaluation procedures, next steps can be the work on forecast combination and meta-learning, together with time series characterization. We already implemented some basics in order to work on this topic in the tsExpKit package, and efforts in this field will be extended in the future.

Bibliography

[1] S. Arlot and A. Celisse. A survey of cross-validation procedures for model selection. *Statistics Surveys*, 4:40–79, 2010.

[2] J.L. Aznarte and J.M. Benítez. Equivalences between neural-autoregressive time series models and fuzzy systems. *IEEE Transactions on Neural Networks*, 21(9):1434–1444, 2010.

[3] J.L. Aznarte, J. Manuel Benítez, and J. Luis Castro. Smooth transition autoregressive models and fuzzy rule-based systems: Functional equivalence and consequences. *Fuzzy Sets and Systems*, 158(24):2734–2745, 2007.

[4] J.L. Aznarte, M.C. Medeiros, and J.M. Benítez. Linearity testing for fuzzy rule-based models. *Fuzzy Sets and Systems*, 161(13):1836–1851, 2010.

[5] J.L. Aznarte, M.C. Medeiros, and J.M. Benítez. Testing for remaining autocorrelation of the residuals in the framework of fuzzy rule-based time series modelling. *International Journal of Uncertainty, Fuzziness and Knowlege-Based Systems*, 18(4):371–387, 2010.

[6] C. Bergmeir, M. Costantini, and J.M. Benítez. On the usefulness of cross-validation for directional forecast evaluation. *Journal of Forecasting*, (submitted), 2013.

[7] J.D. Bermúdez, J.V. Segura, and E. Vercher. A decision support system methodology for forecasting of time series based on soft computing. *Computational Statistics and Data Analysis*, 51(1):177–191, 2006.

[8] J.D. Bermúdez, J.V. Segura, and E. Vercher. Bayesian forecasting with the holt-winters model. *Journal of the Operational Research Society*, 61(1):164–171, 2010.

[9] J.D. Bermúdez, J.V. Segura, and E. Vercher. A multi-objective genetic algorithm for cardinality constrained fuzzy portfolio selection. *Fuzzy Sets and Systems*, 188(1):16–26, 2012.

[10] Michael R. Berthold and Jay Diamond. Boosting the performance of rbf networks with dynamic decay adjustment. In *Advances in Neural Information Processing Systems*, pages 521–528. MIT Press, 1995.

[11] O. Blaskowitz and H. Herwartz. Adaptive forecasting of the EURIBOR swap term structure. *Journal of Forecasting*, 28(7):575–594, 2009.

[12] O. Blaskowitz and H. Herwartz. On economic evaluation of directional forecasts. *International Journal of Forecasting*, 27(4):1058–1065, 2011.

[13] G. Box and G. Jenkins. *Time series analysis: Forecasting and control*. Holden-Day, 1970.

[14] Gail A. Carpenter and Stephen Grossberg. Art 2: Self-organization of stable category recognition codes for analog input patterns. *Applied Optics*, 26(23):4919–4930, Dec 1987.

[15] Gail A Carpenter and Stephen Grossberg. A massively parallel architecture for a self-organizing neural pattern recognition machine. *Comput. Vision Graph. Image Process.*, 37:54–115, January 1987.

[16] Gail A. Carpenter, Stephen Grossberg, and John H. Reynolds. Artmap: Supervised real-time learning and classification of nonstationary data by a self-organizing neural network. *Neural Networks*, 4(5):565–588, 1991.

[17] S.F. Crone, M. Hibon, and K. Nikolopoulos. Advances in forecasting with neural networks? empirical evidence from the nn3 competition on time series prediction. *International Journal of Forecasting*, 27(3):635–660, 2011.

[18] Jonathan D. Cryer and Kung-Sik Chan. *Time Series Analysis With Applications in R*. Springer, 2008. ISBN 978-0-387-75958-6.

[19] J.G. De Gooijer and R.J. Hyndman. 25 years of time series forecasting. *International Journal of Forecasting*, 22(3):443–473, 2006.

[20] G. Deco, R. Neuneier, and B. Schürmann. Non-parametric data selection for neural learning in non-stationary time series. *Neural Networks*, 10(3):401–407, 1997.

[21] A.E. Eiben and J.E. Smith. *Introduction to Evolutionary Computing*. Springer, Berlin, 2003.

[22] Jeffrey L. Elman. Finding structure in time. *Cognitive Science*, 14(2):179–211, 1990.

[23] L. J. Eshelman and J. D. Schaffer. Real-coded genetic algorithm and interval schemata. *Foundation of Genetic Algorithms*, pages 187–202, 1993.

[24] Scott Fahlman and Christian Lebiere. The cascade-correlation learning architecture. In *Advances in Neural Information Processing Systems 2*, pages 524–532. Morgan Kaufmann, 1990.

[25] Scott E. Fahlman. An empirical study of learning speed in back-propagation networks. Technical Report CMU-CS-88-162, Carnegie Mellon, Computer Science Dept, 1988.

[26] Scott E. Fahlman. The recurrent cascade-correlation architecture. In *Advances in Neural Information Processing Systems 3*, pages 190–196. Morgan Kaufmann, 1991.

[27] C. Fernandes and A. Rosa. A study on non-random mating and varying population size in genetic algorithms using a royal road function. In *Proceedings of the IEEE Conference on Evolutionary Computation, ICEC*, volume 1, pages 60–66, 2001.

[28] S. Grossberg. *Adaptive Pattern Classification and Universal Recoding. I.: Parallel Development and Coding of Neural Feature Detectors*, chapter I, pages 243–258. MIT Press, 1988.

[29] Nikolaus Hansen and Andreas Ostermeier. Adapting arbitrary normal mutation distributions in evolution strategies: the covariance matrix adaptation. In *Proceedings of the IEEE Conference on Evolutionary Computation*, pages 312–317, 1996.

[30] R. Hecht-Nielsen. Counterpropagation networks. *Applied Optics*, 26(23):4979–4984, 1987.

[31] J.J. Hopfield. Neural networks and physical systems with emergent collective computational abilities. *Proceedings of the National Academy of Sciences of the United States of America*, 79(8):2554–2558, 1982.

[32] Michael Hudak. RCE classifiers: Theory and practice. *Cybernetics and Systems*, 23(5):483–515, 1993.

[33] R. J. Hyndman, A. B. Koehler, J. K. Ord, and R. D. Snyder. *Forecasting with Exponential Smoothing: The State Space Approach*. Springer Series in Statistics. Springer, 2008.

[34] R.J. Hyndman and G. Athanasopoulos. *Forecasting: principles and practice*. (online textbook available at: http://otexts.com/fpp/), 2012.

[35] R.J. Hyndman and A.B. Koehler. Another look at measures of forecast accuracy. *International Journal of Forecasting*, 22(4):679–688, 2006.

[36] R.J. Hyndman, A.B. Koehler, R.D. Snyder, and S. Grose. A state space framework for automatic forecasting using exponential smoothing methods. *International Journal of Forecasting*, 18(3):439–454, 2002.

[37] Michael I Jordan. Serial order: A parallel, distributed processing approach. *Advances in Connectionist Theory Speech*, 121(ICS-8604):471–495, 1986.

[38] Mark Jurik. Backpercolation. Technical report, Jurik Research, 1994.

[39] T.Y. Kim, K.J. Oh, C. Kim, and J.D. Do. Artificial neural networks for non-stationary time series. *Neurocomputing*, 61(1-4):439–447, 2004.

[40] Teuvo Kohonen. *Self-Organization and Associative Memory*, volume 8. Springer-Verlag, 1988.

[41] N. Krasnogor and J. Smith. A tutorial for competent memetic algorithms: Model, taxonomy, and design issues. *IEEE Transactions on Evolutionary Computation*, 9(5):474–488, 2005.

[42] R. Kunst. Cross validation of prediction models for seasonal time series by parametric bootstrapping. *Austrian Journal of Statistics*, 37:271–284, 2008.

[43] K.J. Lang, A.H. Waibel, and G.E. Hinton. A time-delay neural network architecture for isolated word recognition. *Neural Networks*, 3(1):23–43, 1990.

[44] Jan Matti Lange, Hans-Michael Voigt, and Dietrich Wolf. Growing artificial neural networks based on correlation measures, task decomposition and local attention neurons. In *IEEE International Conference on Neural Networks - Conference Proceedings*, volume 3, pages 1355–1358, 1994.

[45] Friedrich Leisch. Sweave: Dynamic generation of statistical reports using literate data analysis. In Wolfgang Härdle and Bernd Rönz, editors, *Compstat 2002 — Proceedings in Computational Statistics*, pages 575–580. Physica Verlag, Heidelberg, 2002. ISBN 3-7908-1517-9.

[46] M. Lozano, Daniel Molina, and Francisco Herrera. Editorial scalability of evolutionary algorithms and other metaheuristics for large-scale continuous optimization problems. *Soft Computing*, 15(11):2085–2087, 2011.

[47] S. Makridakis, A. Andersen, R. Carbone, R. Fildes, M. Hibon, R. Lewandowski, and et al. The accuracy of extrapolative (time series) methods: Results of a forecasting competition. *International Journal of Forecasting*, 1(2):111–153, 1982.

[48] S. Makridakis, C. Chatfield, M. Hibon, M. Lawrence, T. Mills, K. Ord, and L.F. Simmons. The m2-competition: A real-time judgmentally based forecasting study. *International Journal of Forecasting*, 9(1):5–22, 1993.

[49] S. Makridakis and M. Hibon. The m3-competition: Results, conclusions and implications. *International Journal of Forecasting*, 16(4):451–476, 2000.

[50] S. Makridakis and N. Taleb. Decision making and planning under low levels of predictability. *International Journal of Forecasting*, 25(4):716–733, 2009.

[51] S. Makridakis and N. Taleb. Living in a world of low levels of predictability. *International Journal of Forecasting*, 25(4):840–844, 2009.

[52] Allan D. R. McQuarrie and Chih-Ling Tsai. *Regression and time series model selection*. World Scientific Publishing, 1998.

[53] M.C. Medeiros and A. Veiga. A flexible coefficient smooth transition time series model. *IEEE Transactions on Neural Networks*, 16(1):97–113, 2005.

[54] D. Molina, M. Lozano, C. García-Martínez, and F. Herrera. Memetic algorithms for continuous optimisation based on local search chains. *Evolutionary Computation*, 18(1):27–63, 2010.

[55] M.F. Møller. A scaled conjugate gradient algorithm for fast supervised learning. *Neural Networks*, 6(4):525–533, 1993.

[56] J. J. Moré. The Levenberg-Marquardt algorithm: Implementation and theory. In *Numerical Analysis: Proceedings of the Biennial Conference*, pages 104–116. Springer, 1978.

[57] P A Moscato. *Memetic Algorithms: A Short Introduction*, pages 219–234. McGraw-Hill, London, 1999.

[58] Heinz Mühlenbein and Dirk Schlierkamp-Voosen. Predictive models for the breeder genetic algorithm in continuous parameter optimization. *Evolutionary Computation*, 1:25–49, 1993.

[59] J. Nelder and S. Singer. Nelder-mead algorithm. *Scholarpedia*, 4(2):2928, 2009.

[60] J.A. Nelder and R. Mead. A simplex method for function minimization. *Comput. J.*, 7:308–313, 1965.

[61] D.B. Percival and A.T. Walden. *Wavelet Methods for Time Series Analysis*. Cambridge Series on Statistical and Probabilistic Mathematics. Cambridge University Press, 2000.

[62] Tomaso Poggio and Federico Girosi. A theory of networks for approximation and learning. Technical Report A.I. Memo No.1140, C.B.I.P. Paper No. 31, MIT Artifical Intelligence Laboratory, 1989.

[63] M. J. D. Powell. The BOBYQA Algorithm for Bound Constrained Optimization Without Derivatives. Technical report, Centre for Mathematical Sciences, University of Cambridge, UK, 2009.

[64] K.V. Price, R.M. Storn, and J.A. Lampinen. *Differential evolution: a practical approach to global optimization*. Natural computing series. Springer, 2005.

[65] R Development Core Team. *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing, Vienna, Austria, 2009. ISBN 3-900051-07-0.

[66] J. Racine. Consistent cross-validatory model-selection for dependent data: hv-block cross-validation. *Journal of Econometrics*, 99(1):39–61, 2000.

[67] Martin Riedmiller. Rprop - description and implementation details. Technical report, University of Karlsruhe, 1994.

[68] Martin Riedmiller and Heinrich Braun. Direct adaptive method for faster backpropagation learning: The rprop algorithm. In *1993 IEEE International Conference on Neural Networks*, pages 586–591, 1993.

[69] F. Rosenblatt. The perceptron: A probabilistic model for information storage and organization in the brain. *Psychological Review*, 65(6):386–408, 1958.

[70] David E. Rumelhart, James L. Mac Clelland, and PDP Research Group. *Parallel Distributed Processing: Explorations in the Microstructure of Cognition*. MIT Press, 1986.

[71] Francisco J. Solis and Roger J.B. Wets. Minimization by random search techniques. *Mathematics of Operations Research*, 6(1):19–30, 1981.

[72] M. Stone. Cross-validatory choice and assessment of statistical predictions. *Journal of the Royal Statistical Society. Series B (Methodological)*, 36(2):111–147, 1974.

[73] N.N. Taleb. *The Black Swan: The Impact of the Highly Improbable*. Random House Publishing Group, 2010.

[74] L.J. Tashman. Out-of-sample tests of forecasting accuracy: An analysis and review. *International Journal of Forecasting*, 16(4):437–450, 2000.

[75] Howell Tong. *Non-linear time series: a dynamical system approach*. Oxford University Press, Oxford, UK, 1990.

[76] I. Triguero, S. García, and F. Herrera. Ipade: Iterative prototype adjustment for nearest neighbor classification. *IEEE Transactions on Neural Networks*, 21(12):1984–1990, 2010.

[77] Constantino Tsallis and Daniel A. Stariolo. Generalized simulated annealing. *Physica A: Statistical Mechanics and its Applications*, 233(1–2):395 – 406, 1996.

[78] A. S Weigend and N. A. Gershenfeld, editors. *Time series prediction: Forecasting the future and understanding the past*, 1994.

[79] D Whitley. The GENITOR Algorithm and Selection Pressure: Why Rank-Based Allocation of Reproductive Trials is Best. *Proc. of the Third Int. Conf. on Genetic Algorithms*, pages 116–121, 1989.

[80] Hadley Wickham. *testthat: Testthat code. Tools to make testing fun :)*, 2012. R package version 0.7.

[81] Hadley Wickham, Peter Danenberg, and Manuel Eugster. *roxygen2: In-source documentation for R*, 2011. R package version 2.2.2.

[82] D.H. Wolpert and W.G. Macready. No free lunch theorems for optimization. *IEEE Transactions on Evolutionary Computation*, 1(1):67–82, 1997.

[83] Achim Zeileis and Gabor Grothendieck. zoo: S3 infrastructure for regular and irregular time series. *Journal of Statistical Software*, 14(6):1–27, 2005.

Part II.

Publications

Time Series Modeling and Forecasting Using Memetic Algorithms for Regime-Switching Models

C. Bergmeir, I. Triguero, D. Molina, J.L. Aznarte, and J.M. Benítez. Time Series Modeling and Forecasting Using Memetic Algorithms for Regime-Switching Models. IEEE Transactions on Neural Networks and Learning Systems, 2012, 23(11), 1841-1847.

- Status: **Published**
- Impact Factor (JCR 2011): 2.952
- Subject category:
  - COMPUTER SCIENCE, ARTIFICIAL INTELLIGENCE (12/111 Q1)
  - COMPUTER SCIENCE, HARDWARE & ARCHITECTURE (1/50 Q1)
  - COMPUTER SCIENCE, THEORY & METHODS (4/99 Q1)
  - ENGINEERING, ELECTRICAL & ELECTRONIC (19/245 Q1)

Brief Papers

Time Series Modeling and Forecasting Using Memetic Algorithms for Regime-Switching Models

Christoph Bergmeir, Isaac Triguero, Daniel Molina, José Luis Aznarte, and José Manuel Benítez

***Abstract*— In this brief, we present a novel model fitting procedure for the neuro-coefficient smooth transition autoregressive model (NCSTAR), as presented by Medeiros and Veiga. The model is endowed with a statistically founded iterative building procedure and can be interpreted in terms of fuzzy rule-based systems. The interpretability of the generated models and a mathematically sound building procedure are two very important properties of forecasting models. The model fitting procedure employed by the original NCSTAR is a combination of initial parameter estimation by a grid search procedure with a traditional local search algorithm. We propose a different fitting procedure, using a memetic algorithm, in order to obtain more accurate models. An empirical evaluation of the method is performed, applying it to various real-world time series originating from three forecasting competitions. The results indicate that we can significantly enhance the accuracy of the models, making them competitive to models commonly used in the field.**

***Index Terms*— Autoregression, memetic algorithms, neuro-coefficient smooth transition autoregressive model (NCSTAR), regime-switching models, threshold autoregressive model (TAR).**

I. Introduction

Time series prediction and modeling is an important interdisciplinary field of research, involving among others Computer Sciences, Statistics, and Econometrics. Made popular by Box and Jenkins [1] in the 1970s, traditional modeling procedures combine linear autoregression (AR) and moving average. But, since data are nowadays abundantly available, often complex patterns that are not linear can be extracted. So, the need for nonlinear forecasting procedures arises. Commonly used in this context are procedures, such as multilayer perceptrons or support vector machines [2], and recent developments focus on recurrent neural networks [3], [4], generalized regression neural networks [5], and other regression procedures from machine learning.

But following the ideas of Box and Jenkins [1], a special type of nonlinear models, mainly developed by Tong [6], are piecewise linear models, which allow for modeling a series using various linear models assigned to different zones of the series. A threshold variable is then used to switch between the linear models. Many different types of these so-called threshold AR models can be found in the literature [6], [7]. If the threshold variable is chosen to be a lagged value of the time series, the model is called self-exciting threshold AR. Taking into account that the series are usually continuous, it may often be better if the change from one regime to the other is not performed by a sudden change, but merely using a smooth, differentiable function, such as the Gaussian or logistic function, which leads to smooth transition AR models.

Using this theory, Medeiros and Veiga [8] developed the neuro-coefficient smooth transition AR (NCSTAR), which uses a neural network to learn the threshold variable as a weighted sum of the inputs from the training data. Furthermore, those authors presented an iterative building procedure based on statistical testing to define the number of hidden units of the neural network.

All these models (in the following, we call the model family *TAR) have the advantage that there is a theory to translate them to fuzzy rule-based systems (FRBS) [9]–[12], which makes their interpretability more accessible.

Accuracy and interpretability are usually considered contradictory goals. In many modeling approaches, accuracy is strived for and interpretability is hardly considered. In contrast, the focus of fuzzy modeling initially was to obtain interpretable systems with acceptable accuracy, as the seminal purpose of FRBSs is to exploit the descriptive power of linguistic variables in linguistic fuzzy modeling [13]. Only in later developments, the focus was broadened to concentrate solely on accuracy in precise fuzzy modeling, and nowadays often a tradeoff between accuracy and interpretability is aimed at, e.g., by using multiobjective optimization algorithms [13].

So, although the sole use of FRBSs does not guarantee interpretability, their overall design as systems of rules makes them more accessible to humans. Furthermore, the question of measuring interpretability of FRBSs is an important subject of ongoing research in the area [14], and there exists certain consensus that important matters for interpretability are the overall number of rules, and easily understandable rule premises with few input variables [14]. Also, FRBSs are used in the literature to give interpretations to not only *TAR models [9], but also to other model classes like, e.g., neural networks [15], and support vector machines [16].

Manuscript received December 9, 2011; revised August 28, 2012; accepted August 28, 2012. Date of publication October 3, 2012; date of current version October 15, 2012. This work was supported in part by the Spanish Ministry of Science and Innovation under Project TIN-2009-14575. The work of C. Bergmeir was supported by a scholarship from the Spanish Ministry of Education of the Programa de Formación del Profesorado Universitario.

C. Bergmeir, I. Triguero, and J. M. Benítez are with the Department of Computer Science and Artificial Intelligence, CITIC-UGR, University of Granada, Granada 18071, Spain (e-mail: c.bergmeir@decsai.ugr.es; isaaktriguero@gmail.com; j.m.benitez@decsai.ugr.es).

D. Molina is with the Department of Computer Science and Engineering, University of Cadiz, Cadiz 11001, Spain (e-mail: dmolina@decsai.ugr.es).

J. L. Aznarte is with the Department of Artificial Intelligence, Universidad Nacional de Educación a Distancia, Madrid 21110, Spain (e-mail: jlaznarte@decsai.ugr.es).

Color versions of one or more of the figures in this paper are available online at http://ieeexplore.ieee.org.

Digital Object Identifier 10.1109/TNNLS.2012.2216898

As the iterative building procedure of the NCSTAR model controls the overall number of regimes, the resulting models typically have only few rules, and can be considered interpretable in this sense.

For model identification and estimation, the NCSTAR model uses a combination of a grid search (GS) procedure and a local search (LS) to optimize its parameters. This optimization step is crucial during the iterative building procedure, as it both influences the behavior of the test that determines the number of regimes, and the overall accuracy of the method. On the other hand, evolutionary algorithms (EAs) [17] have proven to be very efficient techniques in various fields of optimization problems [18], especially in the optimization of neural network parameters [19], [20]. They have also been applied in various time series prediction problems [21]–[24].

In particular, memetic algorithms [17], [25] are well-suited for continuous optimization, where high precision in the solutions has to be achieved [26], as they combine the evolutionary paradigm with LS strategies. In this brief, we will use memetic algorithms based on local search chains (MA-LS-Chains) [27]. Our aim is to combine the strength of EAs to find good parameters, with the benefits of the NCSTAR model, in order to develop a building procedure which results in equally interpretable, but more accurate models for time series (in comparison to the original NCSTAR method).

The structure of this brief is as follows. Section II details the theoretical background of the threshold autoregressive models. Section III discusses the memetic algorithm scheme employed, namely MA-LS-Chains. Section IV presents the proposed algorithm, which combines the NCSTAR model with the MA-LS-Chains optimization method. Section V presents the performed experimental setup, and Section VI discusses the results. Finally, Section VII concludes this brief.

II. Neuro-Coefficient Smooth Transition Autoregressive Model: NCSTAR

We define an autoregressive model for a time series $x(t)$ with a function F and a series $e(t)$ of independent, identically distributed error terms in the following way:

$$x(t) = F(x(t-1), \ldots, x(t-d)) + e(t). \tag{1}$$

The delay parameter d determines which lagged values are used as input for the method. From this general definition, various time series models can be derived according to the choice of F.

In a linear autoregression, F performs a linear combination of the past values

$$x(t) = a_0 + \sum_{i=1}^{d} a_i x(t-i) + e(t). \tag{2}$$

With $\mathbf{z}(t) = [1, x(t-1), x(t-2), \ldots, x(t-d)]^T$, and $\mathbf{a} = [a_0, \ldots, a_d]$, (2) becomes in vector notation

$$x(t) = \mathbf{a}\mathbf{z}(t) + e(t). \tag{3}$$

When F is to be a nonlinear function, a popular approach is to use mixtures of linear models

$$x(t) = \sum_{j=1}^{k} f_j(th_j(t))\mathbf{a}_j\mathbf{z}(t) + e(t). \tag{4}$$

Here, two important lines of research can be found in the literature [6], [7]: 1) regarding the (nonlinear) functions f_j that are used for mixing and 2) composition of the threshold function $th_j(t)$. This function can, for instance, take into account exogenous variables, lagged values, or combinations of both. In the threshold AR model (TAR), the functions f_j are chosen to be index functions I_j that switch between the different linear models, depending on the current threshold value $th_j(t)$, using threshold constants $c_0, \ldots, c_k$ with $-\infty = c_0 < c_1 < \cdots < c_k = \infty$, in the following way:

$$I_j(th_j(t)) = \begin{cases} 1, & \text{if } th_j(t) \in (c_{j-1}, c_j] \\ 0, & \text{otherwise.} \end{cases} \tag{5}$$

In the self-exciting TAR, for instance, the threshold variable is defined as $th_j(t) := x(t-d)$ [7]. As the index function causes abrupt changes, which might not be desirable, another possibility is to use the logistic function, that is

$$f_j(th_j(t)) = (1 + \exp(-\gamma_j(th_j(t) - c_j)))^{-1}. \tag{6}$$

This defines the logistic smooth transition autoregressive model (LSTAR) [7]. Here, the parameter c_j can still be interpreted as the threshold between two regimes, and the parameter γ_j is the slope parameter of the logistic function, which determines the smoothness of the change in the logistic function [7].

There are some other possibilities to choose the functions f_j and $th_j(t)$, to generate other models. The most relevant for our work, the NCSTAR, is a modification of the LSTAR model, with

$$th_j(t) = \omega_j \mathbf{w}(t). \tag{7}$$

Here, $\mathbf{w}(t)$ is a vector containing all variables that are relevant for the threshold, that is, lagged values and/or exogenous variables. In the following, we will use $\mathbf{w}(t) = [x(t-1), x(t-2), \ldots, x(t-d)]^T$. And ω_j is a vector of weights with $\|\omega_j\| = 1$, which has the same length as $\mathbf{w}(t)$.

The NCSTAR model has some interesting properties. Medeiros and Veiga [8] presented an iterative building procedure based on statistical tests for this type of model, and Aznarte and Benítez [9] showed that "NCSTAR models are functionally equivalent to Additive TSK FRBS with logistic membership function." Thus, with the NCSTAR, we have a model at hand with a powerful iterative building procedure that can be interpreted in terms of a fuzzy rule-based system.

The parameters of NCSTAR can be divided into linear and nonlinear ones. After having fixed the nonlinear parameters, the linear parameters can be computed in a closed-form solution. In the original version of the NCSTAR [8], a combination of GS and LS is used to determine good settings for the nonlinear parameters. The nonlinear parameters are γ_j, c_j, and ω_j for $j = 1, \ldots, k$ as in (5) and (6), with $\omega_{ij} \in [-1, 1]$, and $\|\omega_j\| = 1$. Whenever during the iterative procedure a new regime is added, starting values for this regime j are chosen in the following way [8].

1) ω_j is drawn from a uniform distribution, and normalized afterwards to ensure its norm is 1. If $\omega_{1j} < 0$, $\omega_j := -\omega_j$. This is performed M times, so that we obtain M vectors ω_j^m (with $m = 1, \ldots, M$).

2) For every ω_j^m, c_j is defined to be the median of $\omega_j^m \mathbf{x}$, with $\mathbf{x}$ being the embedded time series.
3) A grid of N values is defined to choose γ_j^n (with $n = 1, \ldots, N$) as $\gamma_j^n := n$.

Hence, a total of $N \times M$ candidate solutions for the new regime are generated. The parameter γ_j is scale-dependent, so the series are normalized before NCSTAR model building. The best parameter set, appended to the regimes yet present, is used then to initialize the LS algorithm, which improves on all values of the current model. In the original publication, the Levenberg–Marquardt algorithm [28] is used for the LS.

III. Memetic Algorithms With Local Search Chains: MA-LS-Chains

Evolutionary algorithms [17] are used nowadays successfully in a wide range of optimization problems. They evaluate a population of candidate solutions and alter and combine them to new solutions, substituting iteratively the candidate solutions by better suited variants. A central idea of EAs states that, with the use of several candidate solutions, better coverage of the search space will be achieved, and getting stuck in a particular local optimum will be avoided.

When using EAs, a tradeoff has to be made between exploring new unknown areas of the search space and exploiting already known good solutions to reach the local optimum in their neighborhood. This problem is important for continuous optimization problems, such as the one addressed in this brief, as results of high precision are required.

Memetic algorithms combine EAs with LS in order to explore more intensely the most promising areas of the search space. Instead of a generational approach for the EA, where the whole population is substituted by a new one, in this context a steady-state approach is better suited, where only single individuals are substituted. When using an elitist algorithm, it is then possible to maintain the results of the LS in the population.

In the MA-LS-Chains paradigm [27], we use a steady-state genetic algorithm [29] designed to maintain population diversity high by combining the BLX–α crossover operator [30] with a high value for its associated parameter (the default is alpha $= 0.5$), the negative assortative mating strategy [31] as its selection method, replace worst as replacement strategy, and the BGA mutation operator [32].

Another central idea of MA-LS-Chains is that, not only are the individuals stored, but also the current state of the LS for each individual. As a result, it becomes possible to interrupt the LS after a fixed number of iterations (the parameter istep of the method), and later resume it from the same state. In this way, MA-LS-Chains adapts the intensity of the LS to a solution in function of the fitness of that solution. The process of interrupting and later continuing LS is called LS chaining. In these LS chainings, the final state of the LS parameters after each LS application becomes the initial point of a subsequent LS application over the same solution, continuing the LS. In this way, MA-LS-Chains applies a higher intensity to the most promising solutions. Finally, the parameter effort controls the ratio of function evaluations used for LS over those used for the genetic algorithm.

Different LS algorithms can be used within the MA-LS-Chains paradigm. MA-CMA-Chains [27] uses the covariance matrix adaptation evolution strategy (CMA-ES) [33]. Though CMA-ES is itself an EA, it performs well in detecting and exploiting local structures. A drawback is that it does not scale well with the amount of parameters, as it employs complex mathematical operations. However, in our application this is of minor importance, as the amount of parameters is relatively low (for instance, a NCSTAR with order 4 and 10 regimes has 60 nonlinear parameters).

See [27] for a more detailed discussion of the MA-CMA-Chains algorithm.

IV. NCSTAR Fitted With MA-LS-Chains

In order to apply the MA-LS-Chains paradigm to replace the combination of GS and LS of the original NCSTAR, some adjustments are necessary.

The individuals of the population of the MA-LS-Chains algorithm consist of vectors $X = [\gamma_1, \ldots, \gamma_k, c_1, \ldots, c_k, \omega_{11}, \ldots, \omega_{d1}, \ldots, \omega_{1k}, \ldots, \omega_{dk}]$, which are realizations of the nonlinear parameters γ_j, c_j, and ω_j with $j = 1, \ldots, k$ of a NCSTAR model with k transitions (and therewith $k + 1$ regimes).

The process for model building is then the following (also shown in Algorithm 1): first, the series is tested for linearity. If the linearity assumption can be rejected, the series is assumed to be nonlinear, and the iterative building procedure is started. Otherwise, a linear model is built (a one-regime NCSTAR is a linear model). Within the iterative procedure, in the kth iteration, a $(k+1)$-regime NCSTAR is built. In every iteration, the following is executed.

1) A randomly initialized regime is added to the last iterations' solution, and this solution is added to the initial population. The rest of the population is initialized randomly. A uniform distribution constrained by the parameter domains given below is used.
2) The nonlinear parameters for all k transitions are fixed with the optimization algorithm (the linear parameters are computed for every evaluation of the fitness function).
3) The residuals of the built model are tested for linearity.
4) If the test indicates that a new regime should be added, the algorithm goes back to step (1). Otherwise, the method terminates.

It is important to properly constrain the values for γ_j and c_j. A regime that is relevant only for few training data leads to numerical problems, and unreasonable linear parameters, which may lead to very unexpected forecasts. Furthermore, if the γ_j are high, the NCSTAR deteriorates to a TAR model.

So, we restrict the nonlinear parameters in the following way.

1) For a time series x, we define the domain of the γ_j to be $\gamma_j \in [0, \gamma_0 \cdot (\max(x) - \min(x))]$, with γ_0 being a parameter of the method.
2) The thresholds c_j are constrained to lie into the $[\min(x), \max(x)]$ interval. In preliminary experiments, we also evaluated the less narrow interval $[-m_{\text{th}}, m_{\text{th}}]$,

Algorithm 1 NCSTAR-MA-LS Algorithm

```
NCSTAR ← a linear model fitted to the series
k ← 0 {the current number of transitions}
Test for linearity (H0 : γ1 = 0).
if H0 is not rejected then
   {The series is assumed to be linear.}
   return NCSTAR
else
   repeat
      {Add a new regime to NCSTAR.}
      k ← k + 1
      Build a random initial population.
      Add a new, randomly initialized regime to NCSTAR.
      Add vector X of non-linear parameters of current
      NCSTAR to the population.
      Run MA-LS-Chains using the initial population.
      Store the result in NCSTAR.
      Test for the addition of another regime (H0 : γk+1 = 0).
   until H0 is not rejected
end if
return NCSTAR
```

with $m_{\text{th}} = \sqrt{d} \cdot \max(|\min(x)|, |\max(x)|)$. But because of the numerical problems mentioned earlier, the former turned out to be a better choice for the threshold domain.

3) In order to handle the constraint $\|\omega_j\| = 1$, ω_j is encoded with n-dimensional polar coordinates [34], so that $\omega_{ij} \in [0, \pi]$ for $i = 1, \ldots, (d-1)$, and $\omega_{dj} \in [0, 2\pi]$.

V. Experimental Setup

In order to analyze the performance of the proposed method, an experimental study was carried out, which is detailed in this section. We comment on the time series data and the algorithms used, as well as on the results that were obtained.

A. Time Series Used for the Experiments

We use data from the NNGC1 and NN5 forecasting competitions,[1] and from the Santa Fe [35] competition. The high-frequency data, that is, weekly, daily, and hourly data from the NNGC1, are used. The weekly data are those related to the oil industry, such as import quantities or prices. The daily series are throughput measures of several car-tunnels, and the hourly data are arrival times of planes at airports and trains at metro stations.

The NN5 competition data are daily cash withdrawal amounts at different ATMs in the UK, measured over a period of 2 years. There is a so-called full dataset consisting of 111 series, and a reduced dataset containing 11 series. We use the reduced dataset. There are missing values present in the series, so we use one of the methods proposed for this dataset [36] to fill the gaps.

Regarding the Santa Fe data, from the six datasets, we only use data where our methods are applicable. Some of the datasets have a special focus and would require special treatment. For instance, there are series with nonuniform measurement intervals, series with missing values, and a problem where the objective is to learn a concept out of many series.

The augmented Dickey–Fuller test [37] was applied to the series, in order to use only stationary series (*TAR models are, as ARMA models, only applicable to stationary series [6]). Furthermore, series are excluded for which the linearity test, performed during the building procedure, suggests linearity. In this case, the one-regime NCSTAR that is built is a linear AR model which has no nonlinear parameters, so that the optimization procedure is not executed at all.

In total, 30 series are used, which are all available in the KEEL-dataset repository [38].[2]

For the experiments, we withhold 20 percent from the end of every series as test set, and the rest of the data is used for model building. Furthermore, as mentioned above, the series are normalized to zero mean and unitary variance. The normalization parameters are computed on the training sets, and then applied to both training set and corresponding test set.

B. Applied Algorithms

The experiments were carried out with the programming language R [39]. Our code is based on the implementations of *TAR models in the package tsDyn [40]. The MA-LS-Chains algorithm is available in the package Rmalschains [41]. Instead of the Levenberg–Marquardt algorithm, the Broyden–Fletcher–Goldfarb–Shanno (BFGS) algorithm was used, which is available in R by the function optim [42].

To analyze the effects of the parameter domains discussed in Section IV, we use the original method furthermore with a box-constrained version of the algorithm, the large-scale bound-constrained BFGS (L-BFGS-B) algorithm, which is available through the same function optim [42]. We call these two algorithms the NCSTAR and the NCSTAR-BOX, respectively. The version of the algorithm using MA-CMA-Chains for optimization is called NCSTAR-MA-CMA.

The original methods were used with $N = 20$, and $M = 1000$, which are the values proposed by the original authors [8]. The $N = 20$ values for each γ_j are chosen equidistant within the domain for the γ_j as defined in Section IV, using $\gamma_0 = 20$ within all of our experiments. The LS algorithm is run with a maximum of 1000 iterations. So, this yields a number of approx. 21 000 function evaluations per newly added regime in total.

To yield comparable results, the MA-CMA-Chains algorithm is used with the same amount of function evaluations. It is used with the parameter settings effort $= 0.5$, alpha $= 0.5$ (which are the default parameters of the algorithm recommended by the authors), istep $= 300$ (the method has a low sensitivity w.r.t. this parameter [27]), and a population size of 70 individuals. Although we evaluated other parameter sets in preliminary experiments, it turned out that these values are good reliable choices.

Besides the comparison within NCSTAR models, we also performed a study comparing the proposed method with other

[1] Available at http://www.neural-forecasting-competition.com.

[2] Available at http://sci2s.ugr.es/keel/timeseries.php.

TABLE I
PARAMETERS USED THROUGHOUT THE EXPERIMENTS

Algorithm	Parameters
SVR	cost = 10, gamma = 0.2, epsilon = 0.1
MLP	size = 15, decay = 0.0147, maxit = 1000
NNET	size = 9, decay = 0.1, maxit = 1000

methods commonly employed for time series forecasting. Namely, we used ϵ-support vector regression (SVR), different versions of the multilayer perceptron (MLP) trained with standard backpropagation and with the BFGS, multivariate adaptive regression splines (MARS) [43], and a linear model.

SVR is available in R through the package e1071, which implements a wrapper to LIBSVM [44]. The BFGS-trained MLP (which in the following we call NNET) is available through the nnet package [42]. The MLP with standard backpropagation is available in the package RSNNS [45] (and will be called MLP in the following). MARS is available from the package mda.

Table I shows the parameters that are used throughout the experiments (for the methods that require parameters to be set). The parameters are determined in the following way. First, a parameter grid is defined empirically for every method. Then, the parameter set performing best on a set of test series (artificial series generated from a NCSTAR process) w.r.t. the root mean squared error (RMSE) on the respective test sets is chosen.

VI. RESULTS AND DISCUSSION

For all of the 30 series, models were trained, predictions were made on the test set, and the RMSE was computed for these predictions. As both the original and the proposed NCSTAR building procedures are nondeterministic, the whole process was executed ten times, and the respective arithmetic means of the RMSE from the ten executions were used. As the series are normalized, comparing the RMSE of the different series is feasible. Fig. 1 shows box and whisker plots of the RMSE, and Table II shows averaged values over all series. The results indicate that NCSTAR-MA-CMA performs best within the compared methods, as it yields the lowest averaged RMSE.

However, this measure may be heavily influenced by outliers, and may not represent the distribution of the errors adequately. Especially in situations like ours, where the distributions are close together (see Fig. 1), this may lead to erroneous conclusions. So, in order to perform a more sophisticated evaluation of the results, we perform an analysis of the frequencies with which the methods outperform each other. Therefore, we use nonparametric statistical tests for multiple comparisons.

Concretely, we use the Friedman rank-sum test for multiple comparisons to detect statistically significant differences, and the *post-hoc* procedure of Hochberg [46] to characterize those differences [47].[3]

[3]More information can be found on the thematic web site of SCI2S about Statistical Inference in computational intelligence and data mining. Available at http://sci2s.ugr.es/sicidm.

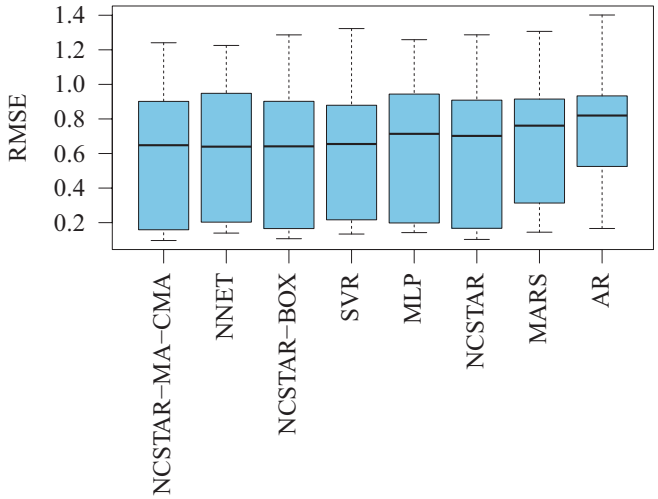

Fig. 1. Box and whisker plots for the RMSE obtained on the test sets of the series for every method. The boxes contain 50% of the data, the middle line is the median. The whiskers extend to the most extreme values.

TABLE II
RMSE AVERAGED OVER ALL SERIES, FOR THE METHODS COMPARED

	RMSE
NCSTAR-MA-CMA	0.579
NNET	0.608
NCSTAR-BOX	0.612
SVR	0.614
MLP	0.617
NCSTAR	0.623
MARS	0.667
AR	0.765

At first, we perform a comparison among the NCSTAR methods, to clearly determine the possible advantages of the method that is proposed in this brief bears. The Friedman test detects highly significant differences on a significance level of $\alpha = 0.01$ (p-value $< 2.81 \cdot 10^{-5}$). Because it obtains the best ranking, NCSTAR-MA-CMA is chosen as the control method, and the Hochberg *post-hoc* procedure is applied. Table III shows the results. As Hochberg's procedure is highly significant (with a significance level of $\alpha = 0.01$) for all compared methods, it is clear that NCSTAR-MA-CMA performs significantly better than the original versions of the algorithm.

In a second step, we compare NCSTAR-MA-CMA to the other benchmarks. The Friedman test shows highly significant differences (p-value $< 2.43 \cdot 10^{-8}$). Table IV shows the results, which indicate that NCSTAR-MA-CMA also performs significantly better than the benchmark methods we compare it to.

In combination, the results indicate that NCSTAR-MA-CMA is the best method, both in terms of the absolute value of the error as well as the frequency with which it outperforms the other methods. It especially outperforms the original algorithm, thus improving the accuracy of the generated NCSTAR models.

TABLE III
COMPARISON WITHIN THE NCSTAR METHODS. AVERAGE RANKS AND ADJUSTED p-VALUES FOR THE FRIEDMAN TEST USING THE *Post-hoc* PROCEDURE OF HOCHBERG

	Average Rank	p_{Hochberg}
NCSTAR-MA-CMA	1.37	–
NCSTAR-BOX	2.27	$4.91 \cdot 10^{-4}$
NCSTAR	2.37	$2.15 \cdot 10^{-4}$

TABLE IV
COMPARISON WITH THE BENCHMARK METHODS. AVERAGE RANKS AND ADJUSTED p-VALUES FOR THE FRIEDMAN TEST USING THE *Post-hoc* PROCEDURE OF HOCHBERG

	Average Rank	p_{Hochberg}
NCSTAR-MA-CMA	1.93	–
NNET	3.23	$7.12 \cdot 10^{-3}$
SVR	3.23	$7.12 \cdot 10^{-3}$
MLP	3.36	$7.12 \cdot 10^{-3}$
MARS	4.53	$2.94 \cdot 10^{-7}$
AR	4.70	$5.09 \cdot 10^{-8}$

TABLE V
AMOUNT OF REGIMES PRESENT IN THE MODELS, AVERAGED OVER ALL SERIES AND OVER TEN RUNS

	Amount of Regimes
NCSTAR-BOX	5.467
NCSTAR	5.513
NCSTAR-MA-CMA	5.940

Another issue is the interpretability of the model. All the NCSTAR building procedures produce the same kind of models, only the amount of regimes varies. The question is whether the NCSTAR-MA-CMA procedure yields a comparable number of rules. Table V shows the results. The original procedure generates on average 5.5 regimes, and NCSTAR-MA-CMA produces with an average of 5.9 regimes approximately 0.4 regimes more than the original methods. We consider this not to be a qualitative change in the interpretability, as an average of up to nine fuzzy rules is considered interpretable by human beings.

VII. CONCLUSION

We investigated the use of memetic algorithms within the iterative NCSTAR building procedure. In every iteration, a statistical test determines if a new regime is to be added, or if the process should terminate. If a new regime is added, the whole model is readjusted using the MA-CMA-Chains algorithm. With the combination of a building procedure that is well-founded on statistics, the possibility to interpret the model as an FRBS, and the advanced optimization procedures, we obtained a model that is flexible and robust in terms of construction, interpretability, and accuracy.

The combination of the NCSTAR model with the MA-CMA-Chains algorithm for optimization produces significantly more accurate results than the original methods.

Moreover, by using a powerful optimization algorithm, NCSTAR is also competitive with other procedures commonly employed in time series forecasting with machine learning procedures. So, NCSTAR-MA-CMA is an algorithm that can be used to build accurate and interpretable models for time series.

REFERENCES

[1] G. Box and G. Jenkins, *Time Series Analysis: Forecasting and Control*. San Francisco, CA: Holden-Day, 1970.

[2] S. F. Crone, M. Hibon, and K. Nikolopoulos, "Advances in forecasting with neural networks? Empirical evidence from the NN3 competition on time series prediction," *Int. J. Forecast.*, vol. 27, no. 3, pp. 635–660, Jul.–Sep. 2011.

[3] L. C. Chang, P. A. Chen, and F. J. Chang, "Reinforced two-step-ahead weight adjustment technique for online training of recurrent neural networks," *IEEE Trans. Neural Netw. Learn. Syst.*, vol. 23, no. 8, pp. 1269–1278, Aug. 2012.

[4] D. Li, M. Han, and J. Wang, "Chaotic time series prediction based on a novel robust echo state network," *IEEE Trans. Neural Netw. Learn. Syst.*, vol. 23, no. 5, pp. 787–799, May 2012.

[5] W. Yan, "Toward automatic time-series forecasting using neural networks," *IEEE Trans. Neural Netw. Learn. Syst.*, vol. 23, no. 7, pp. 1028–1039, Jul. 2012.

[6] H. Tong, *Non-Linear Time Series: A Dynamical System Approach* (Oxford Statistical Science). Oxford, U.K.: Clarendon Press, 1990.

[7] P. H. Franses and D. Van Dijk, *Nonlinear Time Series Models in Empirical Finance*. Cambridge, U.K.: Cambridge Univ. Press, 2000.

[8] M. C. Medeiros and A. Veiga, "A flexible coefficient smooth transition time series model," *IEEE Trans. Neural Netw.*, vol. 16, no. 1, pp. 97–113, Jan. 2005.

[9] J. L. Aznarte and J. M. Benitez, "Equivalences between neural-autoregressive time series models and fuzzy systems," *IEEE Trans. Neural Netw.*, vol. 21, no. 9, pp. 1434–1444, Sep. 2010.

[10] J. L. Aznarte, M. C. Medeiros, and J. M. Benitez, "Linearity testing for fuzzy rule-based models," *Fuzzy Sets Syst.*, vol. 161, no. 13, pp. 1836–1851, Jul. 2010.

[11] J. L. Aznarte, J. M. Benitez, and J. L. Castro, "Smooth transition autoregressive models and fuzzy rule-based systems: Functional equivalence and consequences," *Fuzzy Sets Syst.*, vol. 158, no. 24, pp. 2734–2745, Dec. 2007.

[12] J. L. Aznarte, M. C. Medeiros, and J. M. Benitez, "Testing for remaining autocorrelation of the residuals in the framework of fuzzy rule-based time series modelling," *Int. J. Uncertain., Fuzziness Knowl.-Based Syst.*, vol. 18, no. 4, pp. 371–387, 2010.

[13] J. Casillas, F. Herrera, R. Pérez, M. J. del Jesus, and P. Villar, "Special issue on genetic fuzzy systems and the interpretability-accuracy trade-off," *Int. J. Approx. Reason.*, vol. 44, no. 1, pp. 1–3, 2007.

[14] M J. Gacto, R. Alcalá, and F. Herrera, "Interpretability of linguistic fuzzy rule-based systems: An overview of interpretability measures," *Inf. Sci.*, vol. 181, no. 20, pp. 4340–4360, 2011.

[15] J. M. Benitez, J. L. Castro, and I. Requena, "Are artificial neural networks black boxes?" *IEEE Trans. Neural Netw.*, vol. 8, no. 5, pp. 1156–1164, Sep. 1997.

[16] J. L. Castro, L. D. Flores-Hidalgo, C. J. Mantas, and J. M. Puche, "Extraction of fuzzy rules from support vector machines," *Fuzzy Sets Syst.*, vol. 158, no. 18, pp. 2057–2077, 2007.

[17] A. E. Eiben and J. E. Smith, *Introduction to Evolutionary Computing*. Berlin, Germany: Springer-Verlag, 2003.

[18] I. Triguero, S. García, and F. Herrera, "IPADE: Iterative prototype adjustment for nearest neighbor classification," *IEEE Trans. Neural Netw.*, vol. 21, no. 12, pp. 1984–1990, Dec. 2010.

[19] R. Rojas, *Neural Networks: A Systematic Introduction*. New York: Springer-Verlag, 1996.

[20] C. Harpham, C. W. Dawson, and M. R. Brown, "A review of genetic algorithms applied to training radial basis function networks," *Neural Comput. Appl.*, vol. 13, no. 3, pp. 193–201, 2004.

[21] H. Du and N. Zhang, "Time series prediction using evolving radial basis function networks with new encoding scheme," *Neurocomputing*, vol. 71, nos. 7–9, pp. 1388–1400, 2008.

[22] A. F. Sheta and K. De Jong, "Time-series forecasting using GA-tuned radial basis functions," *Inf. Sci.*, vol. 133, nos. 3–4, pp. 221–228, 2001.

[23] C. G. da Silva, "Time series forecasting with a non-linear model and the scatter search meta-heuristic," *Inf. Sci.*, vol. 178, no. 16, pp. 3288–3299, 2008.

[24] V. M. Rivas, J. J. Merelo, P. A. Castillo, M. G. Arenas, and J. G. Castellano, "Evolving RBF neural networks for time-series forecasting with EvRBF," *Inf. Sci.*, vol. 165, nos. 3–4, pp. 207–220, 2004.

[25] N. Krasnogor and J. Smith, "A tutorial for competent memetic algorithms: Model, taxonomy, and design issues," *IEEE Trans. Evol. Comput.*, vol. 9, no. 5, pp. 474–488, Oct. 2005.

[26] H. Kita, "A comparison study of self-adaptation in evolution strategies and real-coded genetic algorithms," *Evol. Comput.*, vol. 9, no. 2, pp. 223–241, 2001.

[27] D. Molina, M. Lozano, C. García-Martínez, and F. Herrera, "Memetic algorithms for continuous optimisation based on local search chains," *Evol. Comput.*, vol. 18, no. 1, pp. 27–63, 2010.

[28] J. J. Moré, "The Levenberg–Marquardt algorithm: Implementation and theory," in *Proc. Biennial Conf. Numer. Anal.*, 1978, pp. 104-116.

[29] D. Whitley, "The GENITOR algorithm and selection pressure: Why rank-based allocation of reproductive trials is best," in *Proc. 3rd Int. Conf. Genet. Algorithms*, 1988, pp. 116–121.

[30] L. J. Eshelman and J. D. Schaffer, "Real-coded genetic algorithm and interval schemata," in *Proc. Found. Genet. Algorithms*, 1993, pp. 187–202.

[31] C. Fernandes and A. Rosa, "A study of non-random matching and varying population size in genetic algorithm using a royal road function," in *Proc. Congr. Evol. Comput.*, 2001, pp. 60–66.

[32] H. Mühlenbein and D. Schlierkamp-Voosen, "Predictive models for the breeder genetic algorithm in continuous parameter optimization," *Evol. Comput.*, vol. 1, pp. 25–49, Mar. 1993.

[33] N. Hansen and A. Ostermeier, "Adapting arbitrary normal mutation distributions in evolution strategies: The covariance matrix adaptation," in *Proc. IEEE Conf. Evol. Comput.*, May 1996, pp. 312–317.

[34] L. E. Blumenson, "A derivation of n-dimensional spherical coordinates," *Amer. Math. Monthly*, vol. 67, no. 1, pp. 63–66, Jan. 1960.

[35] A. S. Weigend and N. A. Gershenfeld, *Time Series Prediction: Forecasting the Future and Understanding the Past*. Reading, MA: Addison-Wesley, 1994.

[36] J. D. Wichard, "Forecasting the NN5 time series with hybrid models," *Int. J. Forecast.*, vol. 27, no. 3, pp. 700–707, 2011.

[37] S. E. Said and D. A. Dickey, "Testing for unit roots in autoregressive-moving average models of unknown order," *Biometrika*, vol. 71, pp. 599–607, Nov. 1984.

[38] J. Alcalá-Fdez, A. Fernández, J. Luengo, J. Derrac, S. García, L. Sánchez, and F. Herrera, "KEEL data-mining software tool: Data set repository, integration of algorithms and experimental analysis framework," *J. Multiple-Valued Logic Soft Comput.*, vol. 17, nos. 2–3, pp. 255–287, 2011.

[39] R. Development Core Team, *A Language and Environment for Statistical Computing*. Vienna, Austria: R Foundation for Statistical Computing, 2009.

[40] A. F. Di Narzo, J. L. Aznarte, and M. Stigler. (2009). *tsDyn: Time Series Analysis Based on Dynamical Systems Theory* [Online]. Available: http://CRAN.R-Project.org/package=tsDyn

[41] C. Bergmeir, D. Molina, and J. M. Benítez. (2012). *Continuous Optimization Using Memetic Algorithms with Local Search Chains (MA-LS-Chains) in R* [Online]. Available: http://CRAN.RProject.org/package=Rmalschains

[42] W. N. Venables and B. D. Ripley, *Modern Applied Statistics with S*. New York: Springer-Verlag, 2002.

[43] J. H. Friedman, "Multivariate adaptive regression splines," *Ann. Stat.*, vol. 19, pp. 1–67, Mar. 1991.

[44] C.-C. Chang and C.-J. Lin. (2001). *LIBSVM: A Library for Support Vector Machines* [Online]. Available: http://www.csie.ntu.edu.tw/_cjlin/libsvm

[45] C. Bergmeir and J. M. Benítez, "Neural networks in R using the Stuttgart neural network simulator: RSNNS," *J. Stat. Softw.*, vol. 46, no. 7, pp. 1–26, 2012.

[46] Y. Hochberg and D. Rom, "Extensions of multiple testing procedures based on Simes' test," *J. Stat. Plann. Inf.*, vol. 48, no. 2, pp. 141–152, 1995.

[47] S. García, A. Fernández, J. Luengo, and F. Herrera, "Advanced nonparametric tests for multiple comparisons in the design of experiments in computational intelligence and data mining: Experimental analysis of power," *Inf. Sci.*, vol. 180, no. 10, pp. 2044–2064, 2010.

Memetic Algorithms with Local Search Chains in R: The Rmalschains Package

C. Bergmeir, D. Molina, and J.M. Benítez. Memetic Algorithms with Local Search Chains in R: The Rmalschains Package. Journal of Statistical Software, 2012, (submitted).

- Status: **Submitted**
- Impact Factor (JCR 2011): 4.010
- Subject category:
  - COMPUTER SCIENCE, INTERDISCIPLINARY APPLICATIONS (3/99 Q1)
  - STATISTICS & PROBABILITY (1/116 Q1)

Journal of Statistical Software

MMMMMM YYYY, Volume VV, Issue II. *http://www.jstatsoft.org/*

Memetic Algorithms with Local Search Chains in R: The Rmalschains Package

Christoph Bergmeir
University of Granada

Daniel Molina
University of Cádiz

José M. Benítez
University of Granada

Abstract

Global optimization is an important field of research both in Mathematics and Computer Sciences. It has applications in nearly all fields of modern Science and Engineering. Memetic algorithms are powerful problem solvers in the domain of continuous optimization, as they offer a trade-off between exploration of the search space using an evolutionary algorithm scheme, and focused exploitation of promising regions with a local search algorithm. In particular, we describe the memetic algorithms with local search chains (MA-LS-Chains) paradigm, and the R package **Rmalschains**, which implements it. MA-LS-Chains has proven to be effective compared to other algorithms, especially in high-dimensional problem solving. In an experimental study, we explore the advantages of using MA-LS-Chains in comparison to other optimization methods already available in R.

Keywords: continuous optimization, memetic algorithms, MA-LS-Chains, R, **Rmalschains**.

1. Introduction

Global optimization, i.e., finding the inputs to a function that yield minimal/maximal output, is an important mathematical problem with applications in nearly all fields of modern science and engineering. Nowadays, as the functions to optimize are often complex and high-dimensional, mathematical analysis may be difficult, costly or even impossible. In contrast, computational power is abundant, and optimization has evolved to an important line of research in Computer Sciences. Here, meta-heuristics are developed for general optimization that treat the target function in a black-box manner, i.e., no preconditions or further information is required, such as continuity, differentiability, or derivatives of the function.

One such meta-heuristic, which led to a vast amount of successful algorithms and implementations in the past years, is the evolutionary algorithm (EA) framework (Bäck, Fogel, and

Michalewicz 1997). Here, a population of possible solutions is evolved by altering (mutation) the solutions, and by letting them interact with each other (crossover). The candidate solutions are evaluated for their fitness, and newly created solutions replace the solutions with worst fitness, in order to converge around the region with best fitness. Using a population allows EAs to perform good exploration of the search space, but sometimes they are not capable of exploiting a promising region to reach the local optimum of that region. So, the solutions they obtain are sometimes not accurate. Local search (LS) methods, on the other hand, can improve very quickly a solution, but they are not able to explore a complex search domain, as they have the tendency to get stuck in local optima.

Memetic algorithms (MA) (Moscato 1999; Krasnogor and Smith 2005) are a hybridization between EA and LS methods, with the objective to take advantage of both the exploration power of EAs and the exploitative power of the LS, therewith improving the overall results (Goldberg and Voessner 1999). As not all solutions are equally good, MAs can obtain best results if they apply the LS method with a higher *intensity*, i.e., using more evaluations, to the most promising solutions, which are the ones with best fitness.

In this paper, we present the package **Rmalschains** for R (R Development Core Team 2009) that implements various variants of the memetic algorithm with local search chains paradigm (MA-LS-Chains) (Molina, Lozano, García-Martínez, and Herrera 2010). MA-LS-Chains is an MA whose main feature lies in its ability to apply the LS various times on the same solution. The final state of the LS parameters after each LS application becomes the initial point of a subsequent LS application over the same solution, creating an *LS chain*. This way, MA-LS-Chains adapts the intensity of the LS to a solution in function of its quality.

The MA-LS-Chains algorithm family has proven to be very effective in continuous optimization problems in the past. MA-SW-Chains, which employs the Solis-Wets' algorithm (SW) for LS, was the competition winner of CEC'2010 for high-dimensional optimization (Tang, Li, and Suganthan 2010). MA-CMA-Chains, which employs the covariance matrix adaptation evolution strategy (CMAES) as LS (see Section 2), performed very well in the BBOB'2009 competition (Hansen, Auger, Ros, Finck, and Pošík 2010), and also on the data of the CEC'2005 competition (Deb and Suganthan 2005; García, Molina, Lozano, and Herrera 2009); though it did not take part in the official competition, it was evaluated using the same conditions in Molina *et al.* (2010).

There is already a host of choices for continuous optimization methods that are readily available in R, Section 4 gives an overview. However, one important result in Computer Science research on optimization is the "No Free Lunch" theorem (Wolpert and Macready 1997), which states that there cannot be a general, non-problem specific optimization algorithm that performs always better than all other methods. This is due to the fact that a method which takes into account problem specific knowledge has the potential to perform better than general methods. And though most optimization algorithms do not take into account problem-specific knowledge explicitly, they are usually implicitly better/worse suited for certain types of problems. So, taking this into account together with the good performance of MA-LS-Chains, especially for high-dimensional problems, we find it justified to present another package for optimization to the R community. The **Rmalschains** package also performed well in a recent study by Burns (2012), and in Section 5, we perform a comparison of our package to other methods, with a focus on high-dimensional problems.

The algorithm is implemented in C++, and encapsulated in a library called **librealea** (Molina

2012), so that it can also be used outside of R. **Rmalschains** uses **Rcpp** (Eddelbuettel and François 2011) to make the functionality of **librealea** accessible from within R. The package **Rmalschains** is available from the Comprehensive R Archive Network (CRAN) at `http://CRAN.R-project.org/package=Rmalschains`. Also, the interested reader can find further information on the state of the art of EAs on the thematic web site of our research group on "Evolutionary Algorithms and other Metaheuristics for Continuous Optimization Problems" (`http://sci2s.ugr.es/EAMHCO/`).

The remainder of this paper is structured as follows. Section 2 presents the theory of the MA-LS-Chains algorithm, and Section 3 shows a brief example of the usage of the package. Section 4 gives an overview on the other packages available in R for continuous optimization, and Section 5 shows experiments comparing the methods already present in R with **Rmalschains**. Section 6 concludes the paper.

2. The theory of the algorithm

In the following, we describe briefly the general scheme of the MA-LS-Chains algorithm and its main components, i.e., the EA and the LS methods employed. For more details, the reader may refer to Molina *et al.* (2010).

2.1. General scheme

The algorithm was designed with the idea that the LS should be applied with higher intensity on the most promising regions. As promising regions, we consider the areas/regions where solutions with good fitness are located.

MA-LS-Chains is a steady-state MA that is combined with different methods for the LS. It uses a steady-state genetic algorithm (SSGA) as EA (Whitley 1989). Different from a generational algorithm, where the genetic operators are applied to large parts of the population simultaneously, in a steady-state EA only single individuals are used at a time to generate offspring, which replaces other single individuals of the population.

MA-LS-Chains allows for improving the same solution several times, thus creating an *LS chain*. Also, it uses a mechanism to store the final state of the LS parameters along with the solution, after each LS application. In this way, the final state of an LS application on a solution can be used for the initialization of a subsequent LS application on the same solution, *continuing* the LS.

The general algorithm is shown in Algorithm 1. After generating the initial population, in a loop the following is executed: The SSGA is run with a certain amount of evaluations n_{frec}. Then, the set S_{LS} is built with the individuals of the population that have never been improved by the LS, or that have been improved by the LS but with an improvement (in fitness) superior to δ_{LS}^{min}, where δ_{LS}^{min} is a parameter of the algorithm (by default $\delta_{LS}^{min} = 10^{-8}$). If $|S_{LS}| \neq 0$, the LS is applied with an intensity of I_{str} to the best individual in S_{LS}. If S_{LS} is empty, the whole population is reinitialized except for the best individual which is maintained in the population.

With this mechanism, if the SSGA obtains a new best solution, it should be improved by the LS in the following application of the LS method.

Algorithm 1 Pseudocode of MA-LS-Chains

```
Generate the initial population
while not termination-condition do
    Perform the SSGA with n_frec evaluations
    Build the set S_LS of individuals which can be refined by LS
    Pick the best individual c_LS in S_LS
    if c_LS belongs to an existing LS chain then
        Initialize the LS operator with the LS state stored with c_LS
    else
        Initialize the LS operator with the default LS parameters
    end if
    Apply the LS algorithm to c_LS with I_str, giving c_LS^r
    Replace c_LS by c_LS^r
    Store the final LS state with c_LS^r
end while
```

2.2. The evolutionary algorithm

The SSGA applied is specifically designed to promote high population diversity levels by means of the combination of the $BLX-\alpha$ crossover operator (Eshelman and Schaffer 1993) with a high value for its associated parameter (we use a default of $\alpha = 0.5$) and the negative assortative mating (NAM) strategy (Fernandes and Rosa 2001). Diversity is favored as well by means of the BGA mutation operator. The replacement strategy used is Replacement Worst (RW) (Goldberg and Deb 1991). The combination NAM-RW produces a high selective pressure.

Crossover: The $BLX-\alpha$ operator (Eshelman and Schaffer 1993) performs crossover in the following way. Let $a, b \in \mathbb{R}$ be the respective numbers at the ith position of two individuals. Without loss of generality, we assume $a < b$. Using the distance $d = b - a$, the outcome z of the crossover operation is a random number chosen uniformly from the interval $[a - d \cdot \alpha,\ b + d \cdot \alpha]$. It can be shown (Nomura and Shimohara 2001) that values of $\alpha > \frac{\sqrt{3}-1}{2} \approx 0.366$ yield a spread of the individuals in the distribution, whereas smaller values of α lead to a concentration of the individuals.

Negative assortative mating Assortative mating means that the individuals which are crossed are not chosen fully at random, but depending on their similarity. According to whether crossing of similar or dissimilar individuals is favored, the strategy is called positive or negative assortative mating. We use a mating strategy proposed by Fernandes and Rosa (2001), which favors diversity in the population. The algorithm chooses 4 individuals, and computes the similarity (in form of the Euclidean distance) between the first one and all others. Then, the first individual and the individual with maximal distance from it are chosen for mating.

Mutation: The BGA operator This is the operator of the breeder genetic algorithm (BGA) (Mühlenbein and Schlierkamp-Voosen 1993). Its main purpose is to assure diversity

in the population. Let $c \in [a,\ b]$ be the value at the ith position of the individual subject to mutation, with $a, b \in \mathbb{R}$ being the corresponding upper and lower domain bounds. Let r be the mutation range, normally defined as $0.1 \cdot (b-a)$. Then, a new value c' for the ith position of the chromosome, lying in the interval $[c-r,\ c+r]$, is computed in the following way:

$$c' = c \pm r \cdot \sum_{k=0}^{15} \alpha_k 2^{-k},$$

where addition or subtraction are chosen with a probability of 0.5, and the α_k are chosen as either zero or one, with a probability for one of $p(\alpha_k = 1) = \frac{1}{16}$. So, the probability of generating a c' in the neighborhood of c is very high.

The replacement worst strategy This is a standard replacement strategy, where the worst individuals are replaced by better ones. It generates high selective pressure, so that in combination with the negative assortative mating, many different solutions are generated throughout the search, but only the best ones are kept in the population.

2.3. The local search method

Within the MA-LS-Chains paradigm, different methods for the LS can be used, depending on the application. Usually, the CMAES strategy works best. But as the CMAES algorithm does not scale well with the amount of parameters, for high-dimensional problems other LS strategies, such as the Solis-Wets' or the Subgrouping Solis-Wets' solver are to be preferred (Molina, Lozano, Sánchez, and Herrera 2011).

CMAES The CMAES algorithm (Hansen, Müller, and Koumoutsakos 2003) can be considered the state of the art in continuous optimization. Thanks to the adaptability of its parameters, its convergence is very fast and obtains very good results. An implementation is present in R in the package **cmaes** (Trautmann, Mersmann, and Arnu 2011). CMAES is an algorithm that uses a distribution function to obtain new solutions, and adapt the distribution around the best created solutions. The global scheme can be observed in Figure 1.

Its only parameters are the initial average of the distribution $\vec{m}$ and the initial σ. MA-CMA-Chains sets the individual to optimize c_{LS} as $\vec{m}$, and as the initial σ value the half of the distance of c_{LS} to its nearest neighbor in the EA's population.

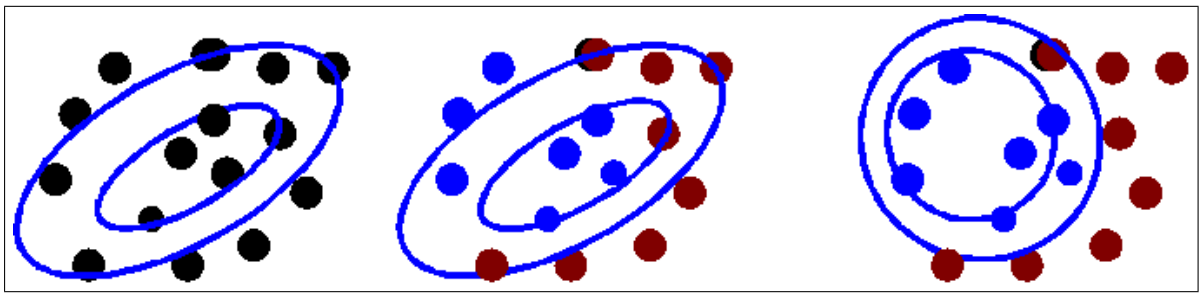

Figure 1: Example of convergence of the CMAES algorithm.

Solis-Wets' algorithm The algorithm presented by Solis and Wets (1981) is a randomized hill climber with adaptive step size. Starting from the current position c_{LS} in the search space, two candidate solutions are generated in the following way. Using a multivariate normal distribution that has the same dimension as c_{LS} and a standard deviation of ρ, a sample is drawn and used as distance d to compute the candidate solutions $c_{LS} + d$ and $c_{LS} - d$. If the better one of the candidate solutions is better than c_{LS}, c_{LS} is updated to this new solution and a success is recorded. If c_{LS} is better than both of the candidate solutions, c_{LS} is not updated and a failure is recorded. After several successes/failures in a row, ρ is increased/decreased. Furthermore, there is a bias term added, to put the search momentum to regions that are promising. This term is continuously updated using its previous value and d. For details, see Molina *et al.* (2010).

Subgrouping Solis-Wets' In this adaptation to high-dimensional data of the Solis-Wets' algorithm, a subgroup of the overall amount of parameters is chosen randomly, and then optimized for a certain amount of evaluations (defined by the parameter `maxEvalSubset`). Then, a new subset is chosen. In the current implementation, the subsets contain 20% of the overall amount of variables.

Nelder-Mead downhill simplex This method, presented by Nelder and Mead (1965) (see also Nelder and Singer (2009)), is a popular standard algorithm for optimization without using derivatives. In R, it is the standard method of the `optim` function (Venables and Ripley 2002). Also, it is implemented in the packages **neldermead** (Bihorel and Baudin 2012), **dfoptim** (Varadhan, University, Borchers, and Research. 2011), **gsl** (Hankin 2006), and **nloptr** (Johnson 2012). A simplex, which is the generalization of a triangle (in 2 dimensions), or a tetrahedron (in 3 dimensions) in n dimensions, is entirely defined by $n + 1$ points. In this algorithm, in an n-dimensional parameter space, the simplex is initialized using $n + 1$ candidate solutions. Then, the simplex is evolved according to the fitness of its vertices using the operations: reflection (about the opposite face of the simplex), reflection and expansion, contraction, and shrinkage. In this way, the simplex is moved "downhill" to find a local minimum of the function. Motivated by this form of movement, the method is sometimes also called the amoeba method (Press, Teukolsky, Vetterling, and Flannery 2007). The method needs for initialization $n + 1$ candidate solutions. However, in MA-LS-Chains, there is only one candidate solution c_{LS} from which the LS is started. A common way to solve this problem is to generate the n other candidate solutions from c_{LS} as $c_{LS} + \lambda_i e_i$, $i = 1, \ldots, n$, with the e_i being unit vectors and λ_i a scaling factor for the respective parameter dimension, which is usually set to one.

3. A simple example

We use minimization of the n-dimensional Rastrigin function as an example, which is a common benchmark in global optimization (Mühlenbein, Schomisch, and Born 1991). The function is defined as follows:

$$f(\vec{x}) = 10n + \sum_{i=1}^{n} \left(x_i^2 - 10\cos(2\pi x_i)\right)$$

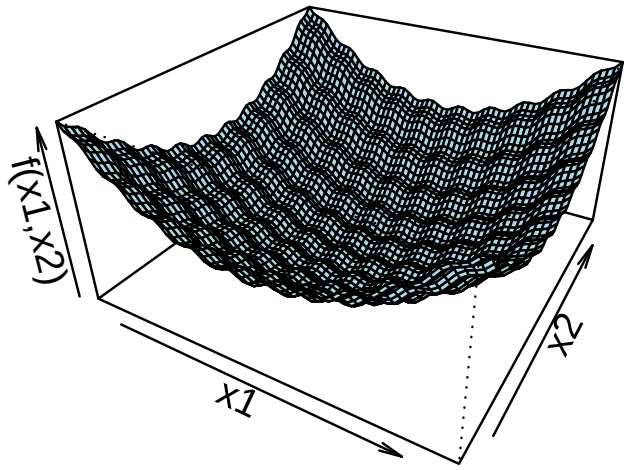

Figure 2: The 2-dimensional Rastrigin function, in the $[-5.12, 5.12]$-interval of the two input parameters.

It has a global minimum at $f(\vec{0}) = 0$. The cosine causes a lot of local optima, so that this function is considered a difficult optimization problem. In R, it can be implemented as follows:

```
R> rastrigin <- function(x) 10 * length(x) + sum(x^2 - 10 * cos(2 * pi * x))
```

Figure 2 shows the 2-dimensional case for the input parameters in the $[-5.12, 5.12]$-interval. The `malschains` function can then be used to minimize the function (for maximization, the objective function would have to be inverted). For the 30-dimensional Rastrigin function, the optimization is performed, e.g., with the following command:

```
R> res <- malschains(rastrigin, lower = seq(-5.12, -5.12, length = 30),
+                    upper = seq(5.12, 5.12, length = 30),
+                    maxEvals = 200000, trace = FALSE,
+                    control = malschains.control(popsize = 50,
+                    istep = 300, ls = "cmaes"))

LS: CMAESHansen: cmaes
CMAES::Neighborhood: 0.5
RatioLS: 0.500000
Istep: 300
LS::Effort: 0.500000
EA::MaxEval: 200000
Popsize: 50
```

```
Time[ALG]: 670.00
Time[LS]: 2070.00
Time[MA]: 2750.00
RatioTime[ALG/MA]: 24.36
RatioTime[LS/MA]: 75.27
NumImprovement[EA]:41%
```

Here, the first parameter is the objective function, the parameters `lower` and `upper` define the lower and upper bounds of the search space, and also define the amount of parameters that the optimizer assumes the objective function to have. The parameter `maxEvals` defines the maximal number of function evaluations that are to be used. The parameter `trace` is used to control whether the output will be verbose or not. Finally, the `control` parameter can be used to define parameters of the optimization method itself. In the example, we use the parameter `popsize` to set the population size of the EA, the parameter `istep` to set the amount of function evaluations within each run of the LS, and the parameter `ls`, to set the type of LS to use.

In this configuration, i.e., with `trace = FALSE`, the output indicates the type of LS, the parameters applied, the time that was spent for both EA and LS, and the improvement that was achieved with the EA as a percentage of the total improvement.

The solution is a list containing the individual with the lowest objective function (called `sol`), and its fitness:

```
R> res

$fitness
[1] 9.689643e-09

$sol
 [1]  1.636650e-06 -1.747430e-07 -9.084372e-07  1.413892e-06
 [5] -1.416422e-06 -1.604717e-06  3.015395e-07 -4.882431e-07
 [9] -9.831059e-07 -1.982626e-06  1.193813e-07 -3.979552e-07
[13] -1.356330e-06 -4.948536e-07 -1.410579e-07  7.184787e-07
[17] -2.792303e-06 -2.515960e-06  1.019090e-06  1.250221e-06
[21] -3.079762e-07 -2.549811e-06  3.145459e-07 -2.071679e-06
[25] -9.839809e-07 -1.692443e-07 -1.455749e-07  7.567179e-07
[29] -4.598565e-07  1.090013e-06
```

It can be seen, that the solution is near to the global optimum of zero both for all values of the solution and the fitness.

4. Other packages in R for continuous optimization

Throughout the last years, a rich variety of packages in R for optimization was developed. A constantly updated overview is provided at the "CRAN Task View: Optimization and Mathematical Programming" (http://cran.r-project.org/web/views/Optimization.html). In the following, we present methods from the section "General Purpose Continuous Solvers" of

that task view, in which our package is also present. Some packages are omitted in the following overview because they are not applicable in our context or because they are very similar to other packages (e.g., there are various implementations of the Nelder-Mead method). The following non-population-based methods are present:

- The `optim` function (Venables and Ripley 2002) from the **stats** package is the standard optimization function in R. It implements a number of LS methods, like the Broyden-Fletcher-Goldfarb-Shanno (BFGS) method, a box-constrained version of it, called L-BFGS-B, simulated annealing, the Nelder-Mead method, and a conjugate-gradient algorithm.
- The package **dfoptim** (Varadhan *et al.* 2011) implements derivative-free optimization algorithms. Concretely, the Nelder-Mead and the Hooke-Jeeves algorithms are implemented. The Nelder-Mead algorithm can be used within our package as the LS.
- The package **minqa** (Bates, Mullen, Nash, and Varadhan 2012) implements derivative-free optimization methods using quadratic approximation. Within the package, the function `bobyqa` implements the algorithm of Powell (2009).
- **Rsolnp** (Ghalanos and Theussl 2011) uses a nonlinear augmented Lagrange multiplier method solver, based on sequential quadratic programming (SQP), for optimization.
- The package **GenSA** (Yang Xiang, Gubian, Suomela, and Hoeng 2011) implements generalized simulated annealing (Tsallis and Stariolo 1996).
- The package **cmaes** (Trautmann *et al.* 2011) implements the CMAES algorithm, which we use as an LS algorithm, and presented in Section 2.3.

And there are also some implementations of population-based methods:

- The packages **DEoptim**, and **RcppDE** (Mullen, Ardia, Gil, Windover, and Cline 2011; Ardia, Boudt, Carl, Mullen, and Peterson 2011) implement differential evolution (DE), an EA that uses difference vectors of the individuals for crossover and mutation (Price, Storn, and Lampinen 2005). **RcppDE** is a reimplementation of **DEoptim** in C++ using the package **Rcpp**, in order to achieve shorter computation times. The packages yield the same results in terms of accuracy.
- The package **rgenoud** (Mebane Jr. and Sekhon 2011) implements a genetic algorithm that is also able to use an LS algorithm for improvement of single individuals. The LS employed is the BFGS algorithm. It is applied after each iteration to the individual with the best fitness or used as a genetic operator.
- The package **PSO** (Bendtsen 2006) implements a particle swarm optimization (PSO), known as Standard PSO 2007 (SPSO-07)[1].

5. Experimental study: Comparison with other algorithms

[1] http://www.particleswarm.info/Programs.html

In this section, we compare **Rmalschains** with the packages discussed in Section 4. The comparison is performed using a benchmark which contains 19 scalable objective functions with different characteristics. Our analysis considers accuracy and execution time of the methods with a fixed amount of function evaluations. Accuracy measures directly the quality of the solution, and may be considered the primary criterion to assess performance of a method. But execution time may be critical as well, in the sense that application of many methods is not feasible if problem dimension and complexity grow. Experiments are performed in different use cases, with medium-, and high-dimensional data, to analyze the behavior of the methods in detail, especially regarding the high-dimensional use case. Results are presented as diagrams, showing execution time and ranking of average accuracy, and as statistical tests assessing the significance of the results obtained for accuracy.

5.1. Test suite and experimental conditions

We use the well-known benchmark of Lozano, Molina, and Herrera (2011), which is especially good to test the scalability of the algorithms. This test set is composed of 19 scalable function optimization problems (also see `http://sci2s.ugr.es/eamhco/testfunctions-SOCO.pdf`):

- 6 Functions: $F_1 - F_6$ of the CEC'2008 test suite. A detailed description may be found in Tang (2008).
- 5 Shifted Functions: Schwefel's Problem 2.22 (F_7), Schwefel's Problem 1.2 (F_8), Extended f10 (F_9), Bohachevsky (F_{10}), and Schaffer (F_{11}).
- 8 Hybrid Composition Functions ($F_{12} - F_{19}$): These functions are non-separable functions built by combining two functions belonging to the set of functions $F_1 - F_{11}$.

In the comparison, we follow the guideline proposed by the authors of the benchmark. We apply the test suite for dimensions 2, 10, 30, 50, 100, 200, 500, 1000. We consider the cases with dimensions 2, 10, 30, and 50 as low-dimensional problems, the cases with dimensions 100 and 200 as medium-dimensional problems, and the cases with dimensions 500 and 1000 as high-dimensional problems. Each algorithm is run 25 times for each test function, and the average error, with respect to the known global optimum, is obtained. Each run stops when a maximum of fitness evaluations is reached, called *MaxFES*, depending on the dimension D in the following way: $MaxFES = 5000 \cdot D$.

To use a *MaxFES* instead of a maximum iteration number allows us to make a fair comparison between optimization methods that have a very different structure. Unfortunately, for several of the considered packages (**rgenoud**, **cmaes**, **DEoptim**, **RcppDE**), only the maximum of iterations can be defined, but not a *MaxFES*. In these cases, we count the number of fitness evaluations, and we return the best solution of the first *MaxFES* evaluations. Then, we use the `callCC` function from the **base** package to stop the algorithm.

The experiments are performed on a Sun Grid Engine (SGE) cluster. Parallelization is performed in the way that the different algorithms are run sequentially on the benchmark functions, and execution of different algorithms is parallelized. The nodes of the cluster have each an Intel(R) Core(TM) i7 CPU with a frequency of 2.80 GHz, and 24 GB of RAM. We establish the following limits: One R session (one algorithm executed with a particular dimension on all benchmarks 25 times) is limited to 6 GB of RAM, and a maximal global

execution time of 10 days. This is approx. 30 min for every execution of one algorithm, and seems reasonable taking into account that computation of the fitness within the benchmark functions is considerably less expensive than in usual real-world problems.

5.2. Parameters and used methods

The aim of this work is to make a comparison between different packages on CRAN that can be used for optimization. In order to simulate the most general use of the packages, we have decided to compare the results using the packages in the most simple and straightforward way, which is with their default parameters.

The only parameter that we define for the benchmark is the population size, as usually this parameter should be adapted in function of the dimensionality, and maintaining constant this parameter for all dimensions could result in degeneration of the method's errors. According to the authors of the test suite (Lozano *et al.* 2011), this parameter is set to $\min(10 \cdot D, 100)$ for algorithms involving a population like DE and PSO.

The functions `optim` and `malschains` apply an improvement method indicated as a parameter from a predefined list. We apply `optim` using the BFGS and L-BFGS-B algorithms, and our function `malschains` using CMAES (the default method), and SW (the Solis-Wets' solver), because it is a more adequate method for higher dimensional problems, due to its lower computation time. These methods will be called in the following `optim_BFGS`, `optim_L-BFGS-B`, `malschains_cmaes`, and `malschains_sw`. All other applied methods have the same name as their respective packages/functions, namely `bobyqa` (from the package **minqa**), **cmaes**, **DEoptim**, **RcppDE**, **dfoptim**, **PSO**, and **Rsolnp**. The packages **rgenoud** and **GenSA** are not included, because their memory requirements prevented obtaining results for dimensions greater than 30.

Some packages require an initial solution, for these algorithms we generate a solution randomly within the parameter domains (using function `runif`), and pass this initial solution to these functions.

5.3. Results in average error

In this section, we study the average error for medium dimension ($D \leq 200$). Results with higher dimensions (up to 1000) are analyzed in Section 5.5. For each function we rank the algorithms according to the average error. The algorithms with best results have the lowest ranks.

Figure 3 shows the average ranking for all the algorithms considered for dimension ≤ 100. The lower the ranking, the better the algorithm. An interesting conclusion we can draw from the figure is that **DEoptim** (and with the same results **RcppDE**) is initially the algorithm with best results, but with increasing dimensionality, the algorithm performs worse. With respect to our package, we can observe that the **Rmalschains** methods perform best for dimension 50 and greater. The packages **PSO** and **cmaes** also perform well.

Figure 4 shows the results for dimensions 10, 50, 100, and 200. Methods `malschains_cmaes` and **cmaes** are omitted here, as their execution time reached the established limit. As rankings are computed, it is not possible to omit the algorithms only in the higher dimensions, which is why the results are presented here again also for the lower dimensions. In Figure 4 we can observe that `malschains_sw` obtains the best results, while the performance of **DEoptim**

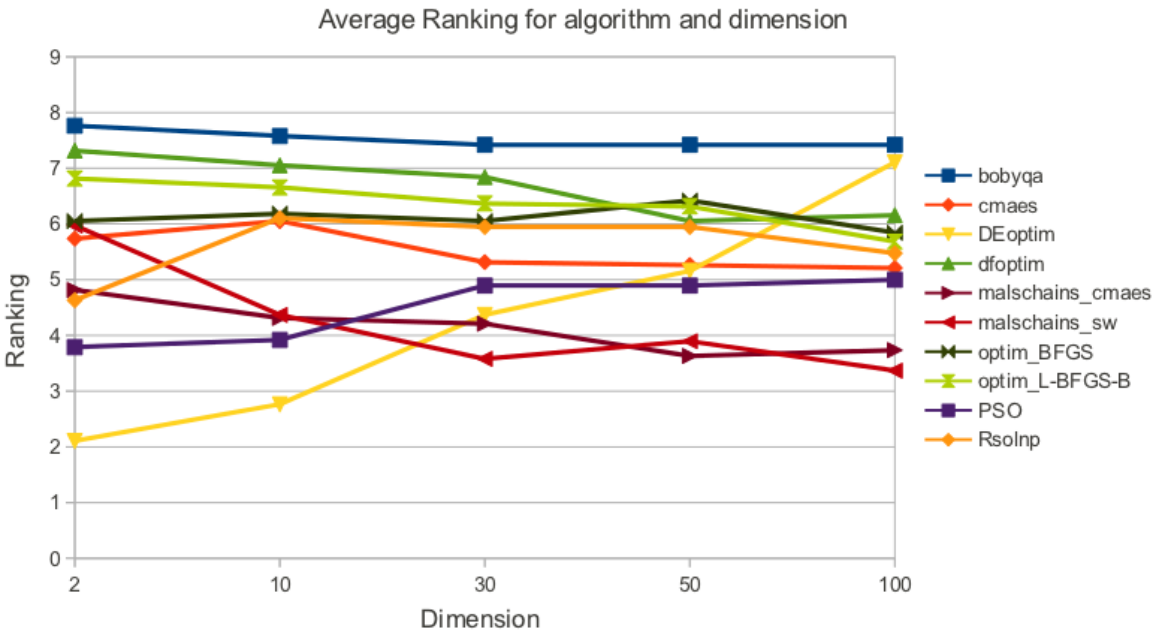

Figure 3: Average ranking for algorithms and dimension ≤ 100.

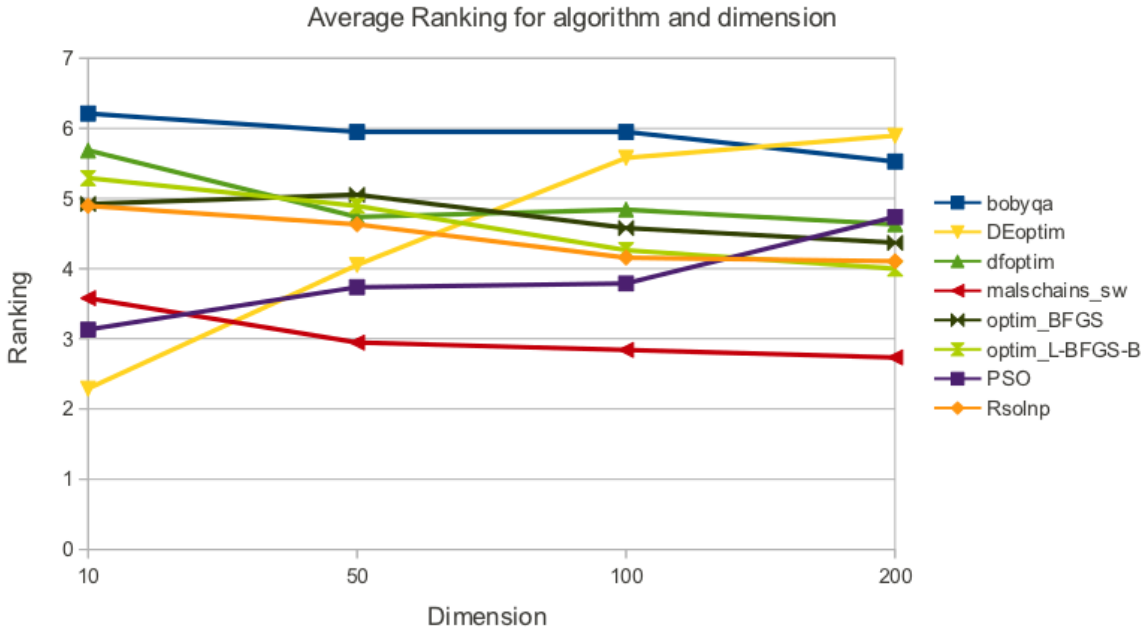

Figure 4: Average ranking for algorithms and dimension ≤ 200.

decreases very quickly as the dimension grows.

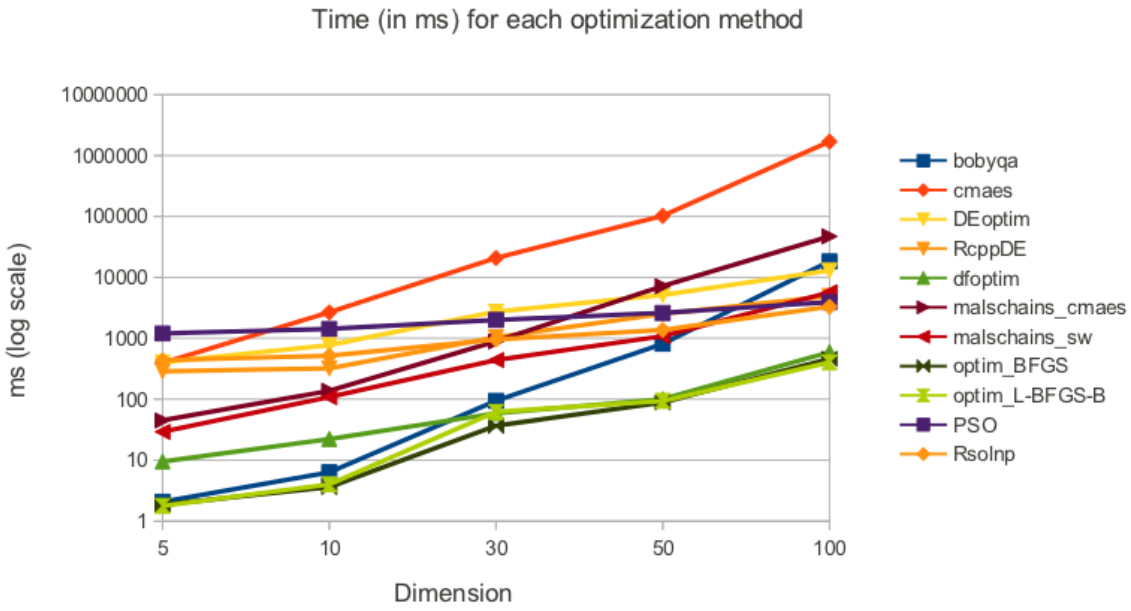

Figure 5: Average computation time for each package (in ms, log scale).

5.4. Analysis of computation time

Though computation time depends greatly on implementation details (e.g., if everything is implemented in pure R, or if C or C++ code is used), from a user perspective, when a package has to be chosen for a concrete application, such an analysis can be very valuable.

For each package and function we run once the optimization function and we measure its computation time (using the **microbenchmark** package (Mersmann 2011)) removing every I/O operation (by function `capture.output` from the **utils** package). Table 1 shows the average computation time in milliseconds. This is graphically illustrated in Figure 5 (note that in the analysis of computation time **DEoptim** and **RcppDE** are shown separately).

Figure 5 shows that `optim` algorithms are the fastest, and **cmaes** is the slowest method for higher dimensions. **PSO** has an average computation time, and it is the algorithm whose running time increases the least with the dimension. The `malschains_cmaes` algorithm has the same increasing ratio as **cmaes** but it maintains for dimension ≤ 100 a moderate computation time (in dimension 100, 47 seconds on average against the 28 minutes of **cmaes**).

5.5. Scalability

Many real problems require optimization of large numbers of variables. Thus, an important aspect of an algorithm is its scalability. Unfortunately, many algorithms have serious limitations in the number of dimensions they can handle. The main reason is usually the increasing computation time (as it can be seen in Section 5.4), but there are others, such as memory requirements, program errors, or accumulated error with the dimension.

In this section, we study the scalability of the different packages. First, we identify the

Algorithm\Dim	5	10	30	50	100	200	500	1000
bobyqa	2.08	6.34	94.03	811.94	18270.63	254440.10	-T-	–
DEoptim	402.30	770.45	2727.22	5138.34	12972.36	37580.78	177020.90	656181.60
RcppDE	287.83	322.06	1044.56	2515.88	4917.35	14383.89	85628.93	361631.10
cmaes	393.15	2671.98	20786.80	102113.30	1695430.00	-T-	–	–
dfoptim	9.51	22.26	59.07	100.07	592.74	1809.29	4615.43	58617.32
malschains_cmaes	44.85	137.69	888.95	7188.50	47237.20	352899.50	-T-	–
malschains_sw	29.14	108.08	440.32	1085.85	5693.48	17961.84	121082.20	570921.00
optim_BFGS	1.85	3.61	36.66	88.57	462.28	3144.69	11872.38	-E-
optim_L-BFGS-B	1.77	4.01	61.94	93.80	404.21	1887.32	-E-	–
PSO	1200.48	1427.22	2002.28	2611.18	3934.63	6655.85	15833.53	35383.74
Rsolnp	435.31	517.61	957.73	1373.02	3305.19	9321.46	-T-	–
rgenoud	39695.38	-M-	–	–	–	–	–	–
GenSA	216.62	537.95	-M-	–	–	–	–	–

Table 1: Time (in ms) for each optimization package. The different errors are: T: time limit was reached. M: memory limit was reached. E: program exited with error.

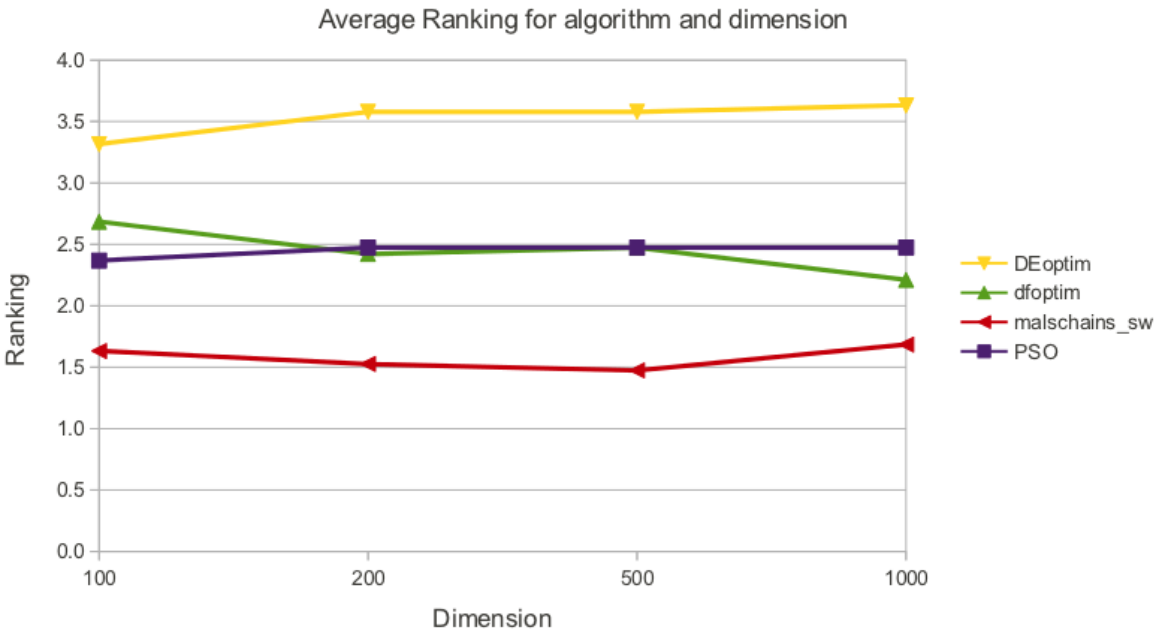

Figure 6: Average ranking for scalable algorithms and dimension ≤ 1000.

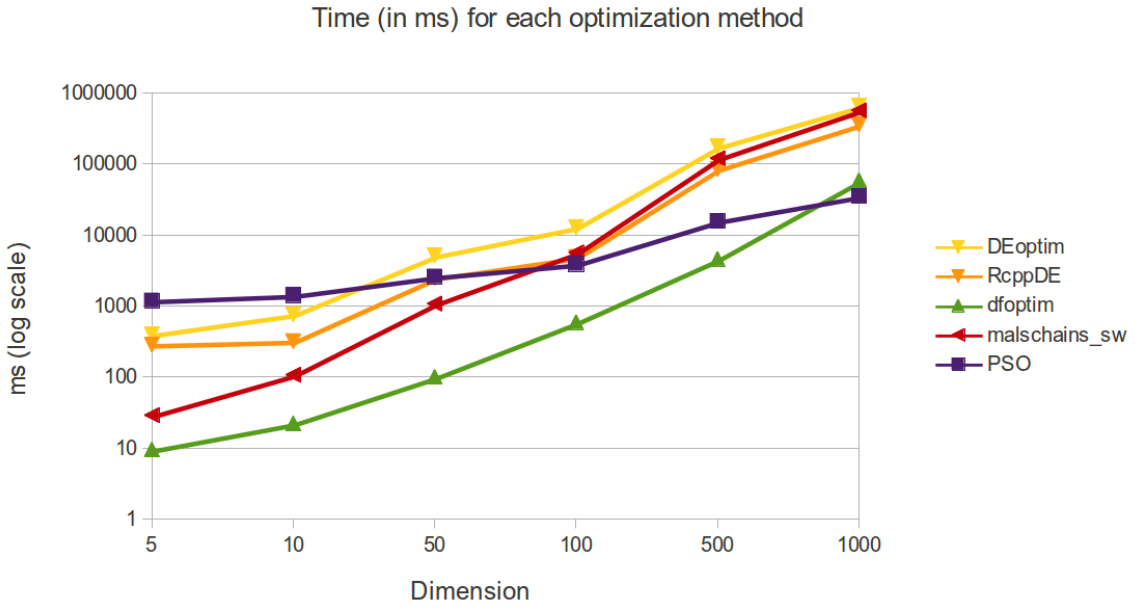

Figure 7: Average computation time for large dimension(in ms, log scale).

packages that have scalability problems, considering Table 1:

- The package **rgenoud** is not scalable, as it reaches the established memory limit for dimension 10. Furthermore, the average computation time for dimension 5 is 39 seconds, when the average of the other algorithms is only around 200 milliseconds.
- The package **bobyqa** is not scalable for higher dimensions, as it reaches the established limit of computation time.
- The package **GenSA** is not scalable by memory requirement from dimension 30 onward.
- Package **cmaes** with its default parameters is not very scalable, since the computational complexity of the CMAES algorithm is $O(n^3)$. From dimension 10 on it is the slowest algorithm, and reaches the time limit at dimension 200.
- Package **Rmalschains** with the cmaes method requires a lower computation time than **cmaes**, but with a similar increasing velocity.
- Although `optim` is the function with the lowest computation time for the majority of dimension values, for higher dimensions, namely 500 with method BFGS and 1000 with method L-BFGS-B, it terminates with an error message ("initial value of 'vmmin' not finite").
- Though in Table 1 it seems that package **Rsolnp** is not too expensive, it takes a variable computing time, and the overall computations for *dimension* ≥ 200 was not finished within the established time limits.

In terms of accuracy, Figure 6 shows the ranking for the algorithms that could be executed up to dimension 1000. We can observe that `malschains_sw` obtains the best results for high dimensional problems.

Results for the execution time are shown in Figure 7. We can observe that the differences in time between the majority of algorithms, except **PSO** and **dfoptim**, are very similar. **dfoptim** is the algorithm with lowest computation time, but taking into account its fast increase in computation time, it probably performs similar as the other methods for dimensions higher than 1000. **PSO** is the algorithm that maintains the lowest increase in computation time with the dimension.

5.6. Statistical analysis of the results on accuracy

So far we performed graphical analyses of the results. In this section, we complement the analysis of the accuracy using statistical tests, to confirm that the differences found so far are "real", i.e., unlikely to be generated by pure chance.

For the comparisons we use the non-parametric test framework presented by Derrac, García, Molina, and Herrera (2011). In particular, we use Friedman's test for detecting if the differences are statistically relevant. Then, we apply the *post-hoc* Holm's test to compare which algorithms are statistically worse than the best one.

First, we apply Friedman's test. Results are in Table 2, and we can observe that in all cases there are statistically significant differences, on a significance level of $\alpha = 0.01$.

Because there are statistically significant differences, we apply the Holm's test against the best algorithm. Table 3 shows the results for dimensions 2 and 10 against **DEoptim**, the

Dimension	*Test value*	*#Algorithms*	*p value*	*Hypothesis*
2	8.475	10	8.967e-9	**Rejected**
10	6.497	10	7.470e-7	**Rejected**
30	3.611	10	0.00076	**Rejected**
50	3.159	10	0.00243	**Rejected**
100	3.947	9	0.00326	**Rejected**
200	4.195	9	0.00353	**Rejected**
500	3.947	5	0.00326	**Rejected**
1000	12.307	4	1.015e-5	**Rejected**

Table 2: Results of Friedman's test for different dimensions, significance level $\alpha = 0.01$

Algorithm	$z = (R_0 - R_i)/SE$	*p value*	α/i	*Hypothesis*
Dimension 2				
bobyqa	5.7598	8.419E-09	0.00556	**Rejected**
dfoptim	5.3044	1.130E-07	0.00625	**Rejected**
optim_L-BFGS-B	4.7954	1.623E-06	0.00714	**Rejected**
optim_BFGS	4.0185	5.857E-05	0.00833	**Rejected**
malschains_sw	3.9381	8.212E-05	0.01000	**Rejected**
cmaes	3.6970	0.0002	0.01250	**Rejected**
malschains_cmaes	2.7594	0.0058	0.01667	**Rejected**
Rsolnp	2.5718	0.0101	0.02500	**Rejected**
PSO	1.7146	0.0864	0.05000	**Not Rejected**
Dimension 10				
bobyqa	4.9026	9.459E-07	0.00556	**Rejected**
dfoptim	4.3668	1.261E-05	0.00625	**Rejected**
optim_L-BFGS-B	3.9649	7.342E-05	0.00714	**Rejected**
optim_BFGS	3.4827	0.0005	0.00833	**Rejected**
Rsolnp	3.4023	0.0007	0.01000	**Rejected**
cmaes	3.3487	0.0008	0.01250	**Rejected**
malschains_sw	1.6342	0.1022	0.01667	**Not Rejected**
malschains_cmaes	1.5806	0.1140	0.02500	**Not Rejected**
PSO	1.1788	0.2385	0.05000	**Not Rejected**

Table 3: Results of the Holm's test for dimension using **DEoptim** as reference algorithm, significance level $\alpha = 0.05$

algorithm with best ranking for these dimensions. **DEoptim** is significantly better than the other algorithms except **PSO** and, for dimension 10, the **Rmalschains** methods. For dimensions 30, 50, and 100, `malschains_sw` is the best algorithm. Table 4 shows the results of Holm's test. We can see that `malschains_sw` is significantly better than `bobyqa`, **dfoptim**, and the `optim` methods.

Table 5 shows the results for higher dimensions, namely dimensions 200, 500, and 1000, for the algorithms that were applicable in these dimensions. We can see that `malschains_sw` is the best algorithm, significantly better than **DEoptim**, `bobyqa`, **dfoptim**, and **PSO** for dimensions

Algorithm	$z = (R_0 - R_i)/SE$	*p value*	α/i	*Hypothesis*
Dimension 30, malschains_sw as reference algorithm				
bobyqa	3.9113	9.179E-05	0.00556	**Rejected**
dfoptim	3.3220	0.0009	0.00625	**Rejected**
optim_L-BFGS-B	2.8397	0.0045	0.00714	**Rejected**
optim_BFGS	2.5183	0.0118	0.00833	**Not Rejected**
Rsolnp	2.4111	0.0159	0.01000	**Not Rejected**
cmaes	1.7681	0.0770	0.01250	**Not Rejected**
PSO	1.3395	0.1804	0.01667	**Not Rejected**
DEoptim	0.8037	0.4216	0.02500	**Not Rejected**
malschains_cmaes	0.6430	0.5203	0.05000	**Not Rejected**
Dimension 50, malschains_cmaes as reference algorithm				
bobyqa	3.8578	0.0001	0.00556	**Rejected**
optim_BFGS	2.8397	0.0045	0.00625	**Rejected**
optim_L-BFGS-B	2.7326	0.0063	0.00714	**Rejected**
dfoptim	2.4647	0.0137	0.00833	**Not Rejected**
Rsolnp	2.3575	0.0184	0.01000	**Not Rejected**
cmaes	1.6610	0.0967	0.01250	**Not Rejected**
DEoptim	1.5538	0.1202	0.01667	**Not Rejected**
PSO	1.2859	0.1985	0.02500	**Not Rejected**
malschains_sw	0.2679	0.7888	0.05000	**Not Rejected**
Dimension 100, malschains_sw as reference algorithm				
bobyqa	4.1257	3.697E-05	0.00556	**Rejected**
DEoptim	3.8042	0.0001	0.00625	**Rejected**
dfoptim	2.8397	0.0045	0.00714	**Rejected**
optim_BFGS	2.5183	0.0118	0.00833	**Not Rejected**
optim_L-BFGS-B	2.3575	0.0184	0.01000	**Not Rejected**
Rsolnp	2.1432	0.0321	0.01250	**Not Rejected**
cmaes	1.8753	0.0608	0.01667	**Not Rejected**
PSO	1.6610	0.0967	0.02500	**Not Rejected**
malschains_cmaes	0.3751	0.7076	0.05000	**Not Rejected**

Table 4: Results of the Holm's test for dimension, significance level $\alpha = 0.05$

200 or 500, and significantly better than **DEoptim** for dimension 1000.

In summary, for dimension ≥ 30, **Rmalschains** offers the best results. Especially for higher dimensions, i.e., dimension ≥ 100, `malschains_sw` does not only scale well, but also achieves the best results in terms of accuracy.

6. Conclusions

MA-LS-Chains is an algorithm framework of memetic algorithms with local search chains. It uses a steady-state genetic algorithm in combination with an LS method. Different LS methods are implemented. The algorithm chooses the individual on which to run the LS on according to its fitness and its possibility to be enhanced with the LS. The LS is run for a fixed

Algorithm	$z = (R_0 - R_i)/SE$	*p value*	α/i	*Hypothesis*
Dimension 200				
DEoptim	4.0872	4.366E-05	0.00625	**Rejected**
bobyqa	3.4356	0.0006	0.00714	**Rejected**
PSO	2.7248	0.0064	0.00833	**Rejected**
dfoptim	2.4879	0.0129	0.01000	**Not Rejected**
optim_BFGS	2.1325	0.0330	0.01250	**Not Rejected**
Rsolnp	1.8363	0.0663	0.01667	**Not Rejected**
optim_L-BFGS-B	1.7178	0.0858	0.02500	**Not Rejected**
malschains_cmaes	0.2369	0.8127	0.05000	**Not Rejected**
Dimension 500				
DEoptim	5.0273	4.974E-07	0.01250	**Rejected**
dfoptim	2.5649	0.0103	0.01667	**Rejected**
PSO	2.4623	0.0138	0.02500	**Rejected**
optim_BFGS	1.7442	0.0811	0.05000	**Not Rejected**
Dimension 1000				
DEoptim	4.6493	3.331E-06	0.01667	**Rejected**
PSO	1.8848	0.0595	0.02500	**Not Rejected**
dfoptim	1.2566	0.2089	0.05000	**Not Rejected**

Table 5: Results of the Holm's test for high dimension using `malschains_sw` as reference algorithm, significance level $\alpha = 0.05$

number of iterations, with the possibility to be continued on in a later stage of the algorithm. The algorithm is very effective, especially in high-dimensional problems. This was proven in various optimization competitions, and in our experiments. We presented an implementation in R, thus making the algorithm available to the R community and facilitating its use in general. We performed a rigorous experimental study comparing it to other general purpose optimizers already available in R, both with respect to quality of the results, as with respect to execution time. The study proved the good performance of the algorithm, especially in higher-dimensional problems.

Acknowledgements

This work was supported in part by the Spanish Ministry of Science and Innovation (MICINN) under Project TIN-2009-14575. C. Bergmeir holds a scholarship from the Spanish Ministry of Education (MEC) of the "Programa de Formación del Profesorado Universitario (FPU)".

References

Ardia D, Boudt K, Carl P, Mullen KM, Peterson BG (2011). "Differential Evolution with **DEoptim**." *The R Journal*, **3**(1), 27–34.

Bäck T, Fogel DB, Michalewicz Z (eds.) (1997). *Handbook of Evolutionary Computation*. IOP Publishing Ltd., Bristol, UK. ISBN 0750303921.

Bates D, Mullen KM, Nash JC, Varadhan R (2012). ***minqa**: Derivative-Free Optimization Algorithms by Quadratic Approximation*. R package version 1.2-1, URL http://CRAN.R-project.org/package=minqa.

Bendtsen C (2006). *Standard PSO 2006*. R package version 1.0-1, URL http://CRAN.R-project.org/package=pso.

Bihorel S, Baudin M (2012). ***neldermead**: R Port of the Scilab* neldermead *Module*. R package version 1.0-7, URL http://CRAN.R-project.org/package=neldermead.

Burns P (2012). "A Comparison of Some Heuristic Optimization Methods." *Technical report*, Burns Statistics. URL http://www.portfolioprobe.com/2012/07/23/a-comparison-of-some-heuristic-optimization-methods.

Deb K, Suganthan P (2005). "Special Session on Real-Parameter Optimization. 2005 IEEE CEC, Edinburgh, UK, Sept 2-5. 2005."

Derrac J, García S, Molina D, Herrera F (2011). "A Practical Tutorial on the Use of Non-parametric Statistical Tests as a Methodology for Comparing Evolutionary and Swarm Intelligence Algorithms." *Swarm and Evolutionary Computation*, **1**(1), 3–18.

Eddelbuettel D, François R (2011). "**Rcpp**: Seamless R and C++ Integration." *Journal of Statistical Software*, **40**(8), 1–18.

Eshelman L, Schaffer JD (1993). "Real-coded Genetic Algorithms in Genetic Algorithms by Preventing Incest." *Foundation of Genetic Algorithms 2*, pp. 187–202.

Fernandes C, Rosa A (2001). "A Study on Non-Random Mating and Varying Population Size in Genetic Algorithms Using a Royal Road Function." In *Proceedings of the IEEE Conference on Evolutionary Computation, ICEC*, volume 1, pp. 60–66.

García S, Molina D, Lozano M, Herrera F (2009). "A Study on the Use of Non-Parametric Tests for Analyzing the Evolutionary Algorithms' Behaviour: A Case Study on the CEC'2005 Special Session on Real Parameter Optimization." *Journal of Heuristics*, **15**(6), 617–644.

Ghalanos A, Theussl S (2011). ***Rsolnp**: General Non-Linear Optimization Using Augmented Lagrange Multiplier Method*. R package version 1.11.

Goldberg DE, Deb K (1991). "A Comparative Analysis of Selection Schemes Used in Genetic Algorithms." *Foundations of Genetic Algorithms*, pp. 69–93.

Goldberg DE, Voessner S (1999). "Optimizing Global-Local Search Hybrids." In *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO) 1999.*, pp. 220–228.

Hankin RKS (2006). "Special Functions in R: Introducing the **gsl** Package." *R News*, **6**.

Hansen N, Auger A, Ros R, Finck S, Pošík P (2010). "Comparing Results of 31 Algorithms from the Black-Box Optimization Benchmarking BBOB-2009." In *Proceedings of the 12th Annual Genetic and Evolutionary Computation Conference, GECCO '10 - Companion Publication*, pp. 1689–1696.

Hansen N, Müller SD, Koumoutsakos P (2003). "Reducing the Time Complexity of the Derandomized Evolution Strategy with Covariance Matrix Adaptation (CMA-ES)." *Evolutionary Computation*, **1**(11), 1–18.

Johnson SG (2012). *The **NLopt** Nonlinear-Optimization Package*. URL http://ab-initio.mit.edu/nlopt.

Krasnogor N, Smith J (2005). "A Tutorial for Competent Memetic Algorithms: Model, Taxonomy, and Design Issues." *IEEE Transactions on Evolutionary Computation*, **9**(5), 474–488.

Lozano M, Molina D, Herrera F (2011). "Editorial Scalability of Evolutionary Algorithms and Other Metaheuristics for Large-Scale Continuous Optimization Problems." *Soft Computing*, **15**(11), 2085–2087.

Mebane Jr W, Sekhon J (2011). "Genetic Optimization Using Derivatives: The **rgenoud** package for R." *Journal of Statistical Software*, **42**(11), 1–26.

Mersmann O (2011). ***microbenchmark**: Sub Microsecond Accurate Timing Functions*. R package version 1.1-3, URL http://CRAN.R-project.org/package=microbenchmark.

Molina D (2012). ***librealea**: An Implementation of Evolutionary Algorithms for Real Optimisation*. URL https://bitbucket.org/dmolina/librealea.

Molina D, Lozano M, García-Martínez C, Herrera F (2010). "Memetic Algorithms for Continuous Optimisation Based on Local Search Chains." *Evolutionary Computation*, **18**(1), 27–63.

Molina D, Lozano M, Sánchez A, Herrera F (2011). "Memetic Algorithms Based on Local Search Chains for Large Scale Continuous Optimisation Problems: MA-SSW-Chains." *Soft Computing*, **15**(11), 2201–2220.

Moscato PA (1999). *Memetic Algorithms: A Short Introduction*, pp. 219–234. McGraw-Hill, London.

Mühlenbein H, Schlierkamp-Voosen D (1993). "Predictive Models for the Breeder Genetic Algorithm – I. Continuous Parameter Optimization." *Evolutionary Computation*, **1**, 25–49.

Mühlenbein H, Schomisch M, Born J (1991). "The Parallel Genetic Algorithm as Function Optimizer." *Parallel Computing*, **17**(6–7), 619 – 632.

Mullen K, Ardia D, Gil D, Windover D, Cline J (2011). "**DEoptim**: An R Package for Global Optimization by Differential Evolution." *Journal of Statistical Software*, **40**(6), 1–26.

Nelder J, Mead R (1965). "A Simplex Method for Function Minimization." *Comput. J.*, **7**, 308–313.

Nelder J, Singer S (2009). "Nelder-Mead Algorithm." *Scholarpedia*, **4**(2), 2928. doi:doi:10.4249/scholarpedia.2928.

Nomura T, Shimohara K (2001). "An Analysis of Two-Parent Recombinations for Real-Valued Chromosomes in an Infinite Population." *Evolutionary Computation*, **9**(3), 283–308.

Powell MJD (2009). "The BOBYQA Algorithm for Bound Constrained Optimization Without Derivatives." *Technical report*, Centre for Mathematical Sciences, University of Cambridge, UK. URL http://www.damtp.cam.ac.uk/user/na/NA_papers/NA2009_06.pdf.

Press WH, Teukolsky SA, Vetterling WT, Flannery BP (2007). *Numerical Recipes 3rd Edition: The Art of Scientific Computing*. Cambridge University Press. ISBN 9780521880688.

Price K, Storn R, Lampinen J (2005). *Differential Evolution: A Practical Approach to Global Optimization*. Natural computing series. Springer-Verlag. ISBN 9783540209508.

R Development Core Team (2009). *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing, Vienna, Austria. URL http://www.R-project.org.

Solis FJ, Wets RJ (1981). "Minimization by Random Search Techniques." *Mathematics of Operations Research*, **6**(1), 19–30.

Tang K (2008). "Summary of Results on CEC'08 Competition on Large Scale Global Optimization." *Technical report*, Nature Inspired Computation and Application Lab (NICAL). URL http://nical.ustc.edu.cn/papers/CEC2008_SUMMARY.pdf.

Tang K, Li X, Suganthan P (2010). "Session on Large Scale Global Optimization. 2010 IEEE World Congress on Computational Intelligence (CEC@WCCI-2010), July 18-23, 2010, Barcelona, Spain."

Trautmann H, Mersmann O, Arnu D (2011). ***cmaes**: Covariance Matrix Adapting Evolutionary Strategy*. R package version 1.0-11, URL http://CRAN.R-project.org/package=cmaes.

Tsallis C, Stariolo DA (1996). "Generalized Simulated Annealing." *Physica A: Statistical Mechanics and its Applications*, **233**(1–2), 395 – 406.

Varadhan R, University JH, Borchers HW, Research AC (2011). ***dfoptim**: Derivative-Free Optimization*. R package version 2011.8-1, URL http://CRAN.R-project.org/package=dfoptim.

Venables WN, Ripley BD (2002). *Modern Applied Statistics with S*. Springer-Verlag. ISBN 0387954570. URL http://www.stats.ox.ac.uk/pub/MASS4.

Whitley D (1989). "The GENITOR Algorithm and Selection Pressure: Why Rank-Based Allocation of Reproductive Trials is Best." *Proc. of the Third Int. Conf. on Genetic Algorithms*, pp. 116–121.

Wolpert D, Macready W (1997). "No Free Lunch Theorems for Optimization." *IEEE Transactions on Evolutionary Computation*, **1**(1), 67–82.

Yang Xiang, Gubian S, Suomela B, Hoeng J (2011). ***GenSA**: Generalized Simulated Annealing*. R package version 1.0-2, URL http://CRAN.R-project.org/package=GenSA.

Affiliation:

Christoph Bergmeir and José M. Benítez
Department of Computer Science and Artificial Intelligence,
E.T.S. de Ingenierías Informática y de Telecomunicación,
CITIC-UGR, University of Granada,
18071 Granada, Spain
E-mail: c.bergmeir@decsai.ugr.es, j.m.benitez@decsai.ugr.es

Daniel Molina
Computer Science and Engineering,
University of Cádiz,
Cádiz, Spain
Email: daniel.molina@uca.es

URL: http://dicits.ugr.es, http://sci2s.ugr.es

Journal of Statistical Software http://www.jstatsoft.org/
published by the American Statistical Association http://www.amstat.org/
Volume VV, Issue II *Submitted:* yyyy-mm-dd
MMMMMM YYYY *Accepted:* yyyy-mm-dd

Neural Networks in R Using the Stuttgart Neural Network Simulator: RSNNS

C. Bergmeir, and J.M. Benítez. Neural Networks in R Using the Stuttgart Neural Network Simulator: RSNNS. Journal of Statistical Software, 2012, 46, 1-26.

- Status: **Published**
- Impact Factor (JCR 2011): 4.010
- Subject category:
  - COMPUTER SCIENCE, INTERDISCIPLINARY APPLICATIONS (3/99 Q1)
  - STATISTICS & PROBABILITY (1/116 Q1)

Journal of Statistical Software

January 2012, Volume 46, Issue 7. *http://www.jstatsoft.org/*

Neural Networks in R Using the Stuttgart Neural Network Simulator: RSNNS

Christoph Bergmeir
University of Granada

José M. Benítez
University of Granada

Abstract

Neural networks are important standard machine learning procedures for classification and regression. We describe the R package **RSNNS** that provides a convenient interface to the popular Stuttgart Neural Network Simulator **SNNS**. The main features are (a) encapsulation of the relevant **SNNS** parts in a C++ class, for sequential and parallel usage of different networks, (b) accessibility of all of the **SNNS** algorithmic functionality from R using a low-level interface, and (c) a high-level interface for convenient, R-style usage of many standard neural network procedures. The package also includes functions for visualization and analysis of the models and the training procedures, as well as functions for data input/output from/to the original **SNNS** file formats.

Keywords: neural networks, **SNNS**, R, **RSNNS**.

1. Introduction

This paper presents the package **RSNNS** (Bergmeir and Benítez 2012) that implements an R (R Development Core Team 2011) interface to the Stuttgart Neural Network Simulator (**SNNS**, Zell *et al.* 1998). The **SNNS** is a comprehensive application for neural network model building, training, and testing. Today it is still one of the most complete, most reliable, and fastest implementations of neural network standard procedures. The main advantages of **RSNNS**, rendering it a general purpose comprehensive neural network package for R, are threefold. (a) The functionality and flexibility of the **SNNS** kernel is provided within R. (b) Convenient interfaces to the most common tasks are provided, so that the methods of **SNNS** integrate seamlessly into R, especially with respect to the scripting, automation, and parallelization capabilities of R. Finally, (c) enhanced tools for visualization and analysis of training and testing of the networks are provided.

The **RSNNS** is available from the Comprehensive R Archive Network (CRAN) at `http:`

//CRAN.R-project.org/package=RSNNS. Moreover, a web page including fully detailed examples of usage at different abstraction levels and further information is available at http://sci2s.ugr.es/dicits/software/RSNNS.

The remainder of this paper is structured as follows. Section 2 describes the main features of the **SNNS** software, together with references to general introductions to neural networks and to original publications. Section 3 presents the general software architecture and implementation details of the package. Section 4 presents the high-level interface of the package. Section 5 gives an overview of the included example datasets, and Section 6 shows some examples for the usage of the package. Section 7 compares the presented package with packages and solutions yet available in R, and discusses strengths and limitations. Finally, Section 8 concludes the paper.

2. Features of the original SNNS software

SNNS consists of three main components: a simulation kernel, a graphical user interface (GUI), and a set of command line tools. It was written for Unix in C and the GUI uses X11. Windows ports exist. **SNNS** was developed at University of Stuttgart and is now maintained at University of Tübingen. The last version where the authors added new functionality is version 4.2 that was released in 1998. In 2008, version 4.3 was released, which includes some patches contributed by the community (http://developer.berlios.de/projects/snns-dev/) that mainly add a Python wrapping. Furthermore, in this version a license change was performed from a more restrictive, academic license to the Library General Public License (LGPL).

To the best of our knowledge **SNNS** is the neural network software that supports the highest number of models. The neural network types implemented differ greatly in their manner of operation, their architecture, and their type of learning. As giving a comprehensive introduction to neural networks in general or detailed descriptions of the methods' theories is beyond the scope of this paper, we give a brief overview of the methods present in **SNNS** in the following, together with references to the original literature. Various comprehensive introductions to neural networks exist, to which the interested reader may refer, e.g., Rojas (1996) covers many of the neural networks implemented in **SNNS**. Haykin (1999) and Bishop (2003) also give comprehensive introductions, and Ripley (2007) gives a good introduction from a statistical point of view.

Network architectures implemented in **SNNS** contain multi-layer perceptrons (MLP, Rosenblatt 1958), recurrent Elman-, and Jordan networks (Elman 1990; Jordan 1986), radial basis function networks (RBF, Poggio and Girosi 1989), RBF with dynamic decay adjustment (Berthold and Diamond 1995; Hudak 1993), Hopfield networks (Hopfield 1982), time-delay neural networks (Lang *et al.* 1990), self-organizing maps (SOM), associative memories, learning vector quantization networks (LVQ, Kohonen 1988), and different types of adaptive resonance theory networks (ART, Grossberg 1988), namely ART1 (Carpenter and Grossberg 1987b), ART2 (Carpenter and Grossberg 1987a), and ARTMAP (Carpenter *et al.* 1991) nets. It implements a wide variety of initialization, learning, update, activation, and pruning functions, adding up to more than 270 in total. For example, update functions allow serial, randomized, topologically ordered, or synchronous update of the units. The adaptive resonance theory networks can be trained step-wise or directly until they reach a stable state. Learning functions include standard backpropagation, backpropagation with momentum term, back-

propagation through time (BPTT, Rumelhart *et al.* 1986), Quickprop (Fahlman 1988), resilient backpropagation (Riedmiller and Braun 1993; Riedmiller 1994), backpercolation (Jurik 1994), (recurrent) cascade-correlation (Fahlman and Lebiere 1990; Fahlman 1991), counterpropagation (Hecht-Nielsen 1987), scaled conjugate gradient (Møller 1993), and TACOMA learning (Lange *et al.* 1994). Activation functions include many activation functions common in neural networks, such as different step functions, the logistic and tanh functions, the linear function, the softmax function, etc. An example of a pruning function implemented is optimal brain damage (OBD, Cun *et al.* 1990). For a comprehensive overview of all functions along with explanations of their parameters, as well as some theoretical background, we refer to Zell *et al.* (1998), and Zell (1994).

The GUI offers tools for easy development and precise control of topologies, learning and analyzing processes. Finally, some additional functionality is scattered along a set of command line tools which for example allow to create networks, convert networks to standalone C code, or to perform some analysis tasks.

3. Package architecture and implementation details

RSNNS provides convenient access to the major part of the **SNNS** application. To achieve this, the **SNNS** kernel and some parts of the GUI code for network generation (see Section 3.1) were ported to C++ and encapsulated in one main class. All code is included in a library, which ships with **RSNNS**. We call this fork of **SNNS** (as well as its main class) `SnnsCLib` throughout this paper and the software. To make the `SnnsCLib` functionality accessible from within R, the package **Rcpp** (Eddelbuettel and François 2011) is used. The `SnnsCLib` class has an equivalent S4 class in R, called `SnnsR`: the low-level interface of the package. As `SnnsR` can be used to access directly the kernel API (called "`krui`" within **SNNS**), all of the functionality and flexibility of the **SNNS** kernel can be taken advantage of. But, as it maps C++ functions directly, its usage might seem unintuitive and laborious to R programmers. Therefore, a high-level R interface is present by the S3 class `rsnns` and its subclasses. This interface provides an intuitive, easy to use, and still fairly flexible interface so that the most common network topologies and learning algorithms integrate seamlessly into R. All of these components are detailed in the remainder of this section.

3.1. The SNNS fork `SnnsCLib`

SNNS is written as pure C code. It can only manage one neural network at a time that is represented as a global state of the program. Porting it to C++ and encapsulating it in a class bears the advantage that various instances can be used sequentially or in parallel. The code basis of `SnnsCLib` is the **SNNS** version 4.3 with a reverse-applied Python patch[1]. All code from the **SNNS** kernel and some code taken from the GUI to create networks of different architectures (within **SNNS** known as the "bignet" tool)[2] was copied to the `src` folder of the

[1]The patch prepares **SNNS** for Python wrapping, which is not needed in our context. Furthermore, it caused some problems during the porting to C++. So we decided to reverse-apply it, so that everything included in the patch was removed from the source.

[2]There is a tool in **SNNS** that assists in network generation, the so-called bignet tool. Though this tool is located within the source code of the GUI, there are a lot of important functions for network generation that depend strongly on kernel functionality and whose only connection to the GUI is, that they receive some input from there. These functions were also included in `SnnsCLib`.

package. **SNNS** has header files with a file name extension of `.h`, containing public function declarations, and others with a file name extension of `.ph`, the private headers. These were merged into one `.h` file, each. The body file extensions were renamed from `.c` to `.cpp`. As all code is to be encapsulated in one C++ class, the merged `.h` files are included (with `#include`) in the private section of the `SnnsCLib` class definition. This procedure has the advantage that in the original code relatively few things had to be changed: In the header files, the changes mainly include removal of `static` keywords and the moving of variable initializations to the constructor of `SnnsCLib`. As these variables are not anymore initialized automatically to zero, variable initializations for all variables of this type were added. In the body files, (a) **SNNS**-internal includes were substituted with an include of `SnnsCLib.h`, (b) `static` variables within functions were turned into member variables of `SnnsCLib`, (c) all function declarations were changed to declarations of member functions of `SnnsCLib`, and (d) calls to the function table are now done using the `this` pointer and C++ -style function pointers.

3.2. The R low-level interface class and C++ wrapping

To use the `SnnsCLib` functionality from R we employ **Rcpp**. Every function from `SnnsCLib` has a corresponding wrapper function, e.g., the wrapper for the function to set the current learning function `setLearnFunc` is implemented as follows:

```
RcppExport SEXP SnnsCLib__setLearnFunc( SEXP xp, SEXP learning_func ) {
 Rcpp::XPtr<SnnsCLib> snnsCLib( xp );

  std::string p1 = Rcpp::as<std::string>( learning_func );

  int err = snnsCLib->krui_setLearnFunc( const_cast<char*>( p1.c_str() ) );
  return Rcpp::List::create( Rcpp::Named( "err" ) = err );
}
```

A pointer to the current `SnnsCLib` object is in all such functions present as the first parameter. The other parameters are converted and passed to the corresponding **SNNS** function. In this example, the only parameter `learning_func` gives the name of the function to be set as the current learning function. The result of the wrapper function is typically a named list. If a return value for `err` is present, then an error handling is implemented (see below).

The corresponding part of `SnnsCLib` on the R side is the S4 class `SnnsR`. Each `SnnsR` object has an external pointer to its corresponding `SnnsCLib` object. If the object factory is used, a `SnnsCLib` object is generated automatically and the pointer is initialized with this object:

```
snnsObject <- SnnsRObjectFactory()
```

`SnnsR` contains a convenient calling mechanism for member functions of its instances based on the calling mechanism suggested by **Rcpp**. The `$` operator is overloaded and can be used for function calling as illustrated by the following example. The function used earlier as an example, `SnnsCLib__setLearnFunc`, can be called like this:

```
snnsObject$setLearnFunc("Quickprop")
```

The `$` operator first searches an R function with the name `SnnsR__setLearnFunc`. If no such function is present, R's calling mechanism is used to call the corresponding C++ function from

R class	Description
`mlp`	Multi-layer perceptron
`dlvq`	Dynamic learning vector quantization network
`rbf`	Radial basis function network
`rbfDDA`	RBF network using the dynamic decay adjustment algorithm
`elman`	Recurrent Elman network
`jordan`	Recurrent Jordan network
`som`	Self-organizing map
`art1`	ART network for binary patterns
`art2`	ART network for real-valued patterns
`artmap`	ART network for supervised learning of binary patterns
`assoz`	Autoassociative memory

Table 1: Models directly accessible from the R high-level interface.

`SnnsCLib`. This has the advantage that a function from `SnnsCLib` can be replaced completely transparently by a function implemented in R, which typically might call the original `SnnsCLib` function after some error checking and preprocessing, or perform some postprocessing. Also, functions not present in the original **SNNS** can be added in this way, either implemented in R or in C++. A preprocessing currently implemented is automatic loading of an object serialization of the `SnnsCLib` object, if it is not present. This mechanism is used by the high-level interface objects to allow them to be saved and loaded by the usual R mechanisms for saving and loading. A postprocessing currently implemented in the `$` operator is an error handling. If in the result list the member `err` is present and not equal to zero, then the **SNNS** function `error` is called, which translates the error code to a text message. This text message then is displayed as an R warning.

3.3. R high-level interface classes

The most convenient way to use the **RSNNS** is through its high-level R interface composed of the S3 class `rsnns` and its subclasses. The classes currently implemented are shown in Table 1.

The subclasses typically implement a process of five steps:

1. Check and preprocess the input.
2. Generate an `rsnns` object using the object factory `rsnnsObjectFactory`, setting training, update, and initialization function and their parameters.
3. Create/load a network architecture by directly accessing functions of the `SnnsR` object created within the `rsnns` object: `rsnnsObject$snnsObject$`...
4. Train the network, using either the function `train.rsnns` or direct access again.
5. Postprocess the output.

The `train.rsnns` function directly saves all training results in the `rsnns` object, e.g., in `rsnnsObject$fitted.values`.

4. Using the R high-level interface

The high-level interface of the **RSNNS** package provides a comfortable way to develop and deploy the most common models of neural nets. The classes present in this interface enable convenient use of the models in R, as a model can be built with a single command, and many standard methods known from other models in R are present, such as `predict` to apply the models to new data, or `print` and `summary` to show characteristics of the models.

The interface is very similar for all models. As an example of use, we show the interface for the multi-layer perceptron `mlp`:

```
R> mlp(x, y, size = 5, maxit = 100, initFunc = "Randomize_Weights",
+    initFuncParams = c(-0.3, 0.3), learnFunc = "Std_Backpropagation",
+    learnFuncParams = c(0.2, 0.0), updateFunc = "Topological_Order",
+    updateFuncParams = 0.0, hiddenActFunc = "Act_Logistic",
+    shufflePatterns = TRUE, linOut = FALSE, inputsTest = NULL,
+    targetsTest = NULL)
```

The parameters reflect the five processing steps discussed in Section 3.3:

- *Training data:* `x`, `y`. `x` is a matrix or vector containing the training inputs. If the method is a supervised learning method, also training targets `y` have to be supplied.
- *The network architecture parameters:* `size`. The `mlp` has one architecture parameter, `size`, which defines the amount of neurons in the hidden layers.
- *Number of iterations:* `maxit`. The parameter `maxit` defines the number of iterations, i.e., the number of training epochs to perform.
- *Initialization function:* `initFunc`, `initFuncParams`. The initialization function initializes the components of the network, i.e., mainly the weights of the connections between the units. Depending on the network architecture and the learning method to use, an appropriate initialization function has to be chosen, e.g., there are learning procedures where weights have to be initialized with a determined value such as 0.0 or 1.0, in other procedures, random initialization may be best, or in methods where the weights represent prototypes they are usually initialized in a way to cover the whole input space. As the initialization function has a close connection to the architecture of the network, it is usually not necessary to alter the defaults that are set in the high-level functions.
- *Learning function:* `learnFunc`, `learnFuncParams`. The learning function defines how learning takes place in the network. It is a central characteristic of the network. For some network architectures, e.g., the ART networks, there is essentially only one type of learning function, which has to be used then. For other networks, such as the MLP, a host of choices exists. Depending on the function, the parameters have different meanings, e.g., for `Std_Backpropagation`, there are two parameters, the step width of the gradient descent η, and the maximal difference d_{max} between target value and output that is tolerated, i.e., that is propagated as error of value zero.
- *Update function:* `updateFunc`, `updateFuncParams`. The update function defines how activation is propagated through the network. As it is the case for the initialization

function, an update function that suits the network architecture has to be chosen, so that it usually is not necessary to alter the defaults. E.g., in feed-forward networks normally the function `Topological_Order` is used, which calculates unit activations in the topological order of the units, which means especially that activation of the units in the input layer is calculated before activations in the first hidden layer etc., so that the activation is propagated through the network from the input to the output layer. In many architectures, the same parameters that are used for the learning function can be used for the update function.

- *Special parameters for the method:* `hiddenActFunc`, `shufflePatterns`, `linOut`. Often, parameters particular to the method are present. In the `mlp`, the `hiddenActFunc` parameter specifies the activation function of the hidden units, `shufflePatterns` defines, whether patterns will be internally shuffled by **SNNS** before training or not, and `linOut` defines, whether the activation function of the output units will be the linear or the logistic function, making it suitable for classification or regression.

- *Optional test data:* `inputsTest`, `targetsTest`. Providing test data directly at the model building stage has the advantage that after each epoch the error on the test set is computed, and can later be used to analyze the training process. As the data at this stage is exclusively used to compute this iterative error, providing test data here is only interesting for supervised learning methods. In any case, the `predict` function can be used later to obtain results on test data, both in supervised and unsupervised learning scenarios.

Initialization, learning, and update function names together with their parameters are directly passed to the **SNNS** kernel. For their documentation, we refer to Zell *et al.* (1998). In the **SNNS** kernel, each of the parameter sets has a maximal length of five. A list of all available functions in the **SNNS** kernel can be obtained by `getSnnsRFunctionTable`. Their name, type, number of input- and output parameters are shown:

```
R> getSnnsRFunctionTable()[196:202, ]
```

```
                  name type #inParams #outParams
196 Counterpropagation    4         3          3
197        Dynamic_LVQ    4         5          5
198            Hebbian    4         3          3
199              JE_BP    4         3          3
200     JE_BP_Momentum    4         5          5
201       JE_Quickprop    4         5          5
202           JE_Rprop    4         4          4
```

As a short example, the easiest way to train a model is using the default parameters, only setting the training inputs, and – if required by the model – the training outputs. The generic `print` function implemented for `rsnns` objects can then be used to get information about the architecture, the functions, and the parameters used by the model:

```
R> encoder <- mlp(inputs, targets)
R> encoder
```

```
Class: mlp->rsnns
Number of inputs: 8
Number of outputs: 8
Maximal iterations: 100
Initialization function: Randomize_Weights
Initialization function parameters: -0.3 0.3
Learning function: Std_Backpropagation
Learning function parameters: 0.2 0
Update function:Topological_Order
Update function parameters: 0
Patterns are shuffled internally: TRUE
Compute error in every iteration: TRUE
Architecture Parameters:
$size
[1] 5

All members of model:
 [1] "nInputs"               "maxit"
 [3] "initFunc"              "initFuncParams"
 [5] "learnFunc"             "learnFuncParams"
 [7] "updateFunc"            "updateFuncParams"
 [9] "shufflePatterns"       "computeIterativeError"
[11] "snnsObject"            "archParams"
[13] "IterativeFitError"     "fitted.values"
[15] "nOutputs"
```

The `summary` function that is implemented in the `rsnns` class generates a textual representation of the net in the original **SNNS** file format, and displays it (output of the function is omitted here):

```
R> summary(encoder)
```

After training, the model can be used for prediction on new data, or the visualization capabilities of R can be used to analyze the training process and the models performance on training and test data. Examples are given in Section 6.

5. Included datasets

A fairly convenient feature in R is the inclusion of datasets along with software packages. This is important for standardized tests as well as for examples of the usage of the package (as presented in Section 6). Various datasets are present in the original **SNNS** examples, covering a wide variety of network types and possible applications. The functions presented in Section 6.5 were used to convert all pattern files with fixed-size patterns included as example data in **SNNS** to data accessible within R. All the data is available in **RSNNS** through the list `snnsData`:

```
R> data("snnsData")
R> names(snnsData)
```

```
 [1] "art1_letters.pat"          "artmap_test.pat"
 [3] "eight_016.pat"             "laser_999.pat"
 [5] "letters_with_classes.pat" "spirals.pat"
 [7] "trainValidMAP.pat"         "xor_rec1.pat"
 [9] "art2_tetra_high.pat"       "artmap_train.pat"
[11] "eight_160.pat"             "letseq_test.pat"
[13] "nettalk.pat"               "sprach_test.pat"
[15] "validMAP.pat"              "xor_rec2.pat"
[17] "art2_tetra_low.pat"        "bdg_TDNN.pat"
[19] "encoder.pat"               "letseq_train.pat"
[21] "patmat.pat"                "sprach_train.pat"
[23] "art2_tetra_med.pat"        "font.pat"
[25] "letters_auto.pat"          "som_cube_norm.pat"
[27] "testMAP.pat"               "art2_tetra.pat"
[29] "dlvq_ziff_100.pat"         "laser_1000.pat"
[31] "letters.pat"               "som_cube.pat"
[33] "trainMAP.pat"              "xor.pat"
```

The columns of the datasets are named according to whether they are input or output to the net. The convenience functions `inputColumns` and `outputColumns` can be used to pick the right columns according to their names. Furthermore, the function `splitForTrainingAndTest` can be used to split the data in a training and a test set. So, for example the `laser` dataset can be loaded and preprocessed with:

```
R> laser <- snnsData$laser_1000.pat
R> inputs <- laser[, inputColumns(laser)]
R> targets <- laser[, outputColumns(laser)]
R> patterns <- splitForTrainingAndTest(inputs, targets, ratio = 0.15)
```

Also, use of datasets from the KEEL dataset repository (http://sci2s.ugr.es/keel/datasets.php) is possible. Reference results for various models are provided at the package homepage at http://sci2s.ugr.es/dicits/software/RSNNS/.

6. Examples

In this section we illustrate the use of the package with examples for regression, classification, and clustering. Furthermore, we comment on some other useful functionality and on the benefits of using both **RSNNS** and a separate installation of the original **SNNS** software. All examples shown here and various other ones that illustrate both the use of the high-level and the low-level interface are included in the package as demos. In R the command `demo()` gives a list of available demos, e.g., the demo `"laser"` can be started with `demo("laser")`. The examples can also be found at the package web page referred to above.

6.1. Recurrent neural networks for regression

In this example, we show the use of an Elman network (Elman 1990) for time series regression. After loading the data as seen in Section 5, the model is trained in a way similar to the one

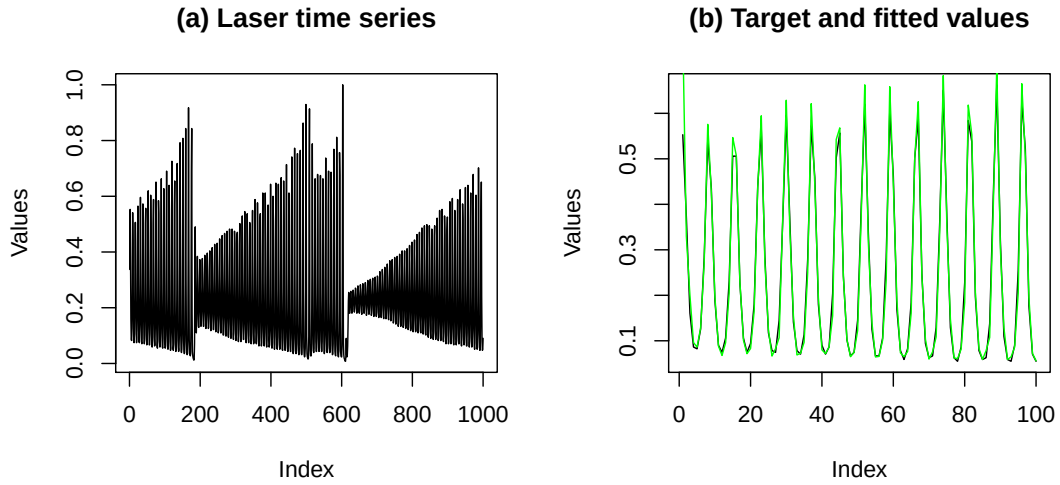

Figure 1: (a) The laser example time series. (b) The first 100 values of the series (black), and the corresponding fits (green).

shown in Section 4. Meaningful default values (especially for the function to use) are already given, so it is often sufficient just to adjust the learning parameters:

```
R> model <- elman(patterns$inputsTrain, patterns$targetsTrain,
+     size = c(8, 8), learnFuncParams = c(0.1), maxit = 500,
+     inputsTest = patterns$inputsTest, targetsTest = patterns$targetsTest,
+     linOut = FALSE)
```

The powerful tools for data visualization in R come handy for neural net modeling. For example, input data and fitted values can be visualized in the following way (the plots are shown in Figure 1):

```
R> plot(inputs, type = "l")
R> plot(targets[1:100], type = "l")
R> lines(model$fitted.values[1:100], col = "green")
```

In addition to the visualization tools already available in R, various other methods for visualization and analysis are offered by the package. The function `plotIterativeError` generates an iterative error plot that shows the summed squared error (SSE), i.e., the sum of the squared errors of all patterns for every epoch. If a test set is provided, its SSE is also shown in the plot, normalized by dividing the SSE through the test set ratio (which is the amount of patterns in the test set divided by the amount of patterns in the training set). The function `plotRegressionError` can be used to generate a regression plot which illustrates the quality of the regression. It has target values on the x-axis and fitted/predicted values on the y-axis. The optimal fit would yield a line through zero with gradient one. This optimal line is shown, as well as a linear fit to the actual data. Using standard methods of R, also other evaluation techniques can be implemented straightforwardly, e.g., an error histogram (the plots are shown in Figure 2):

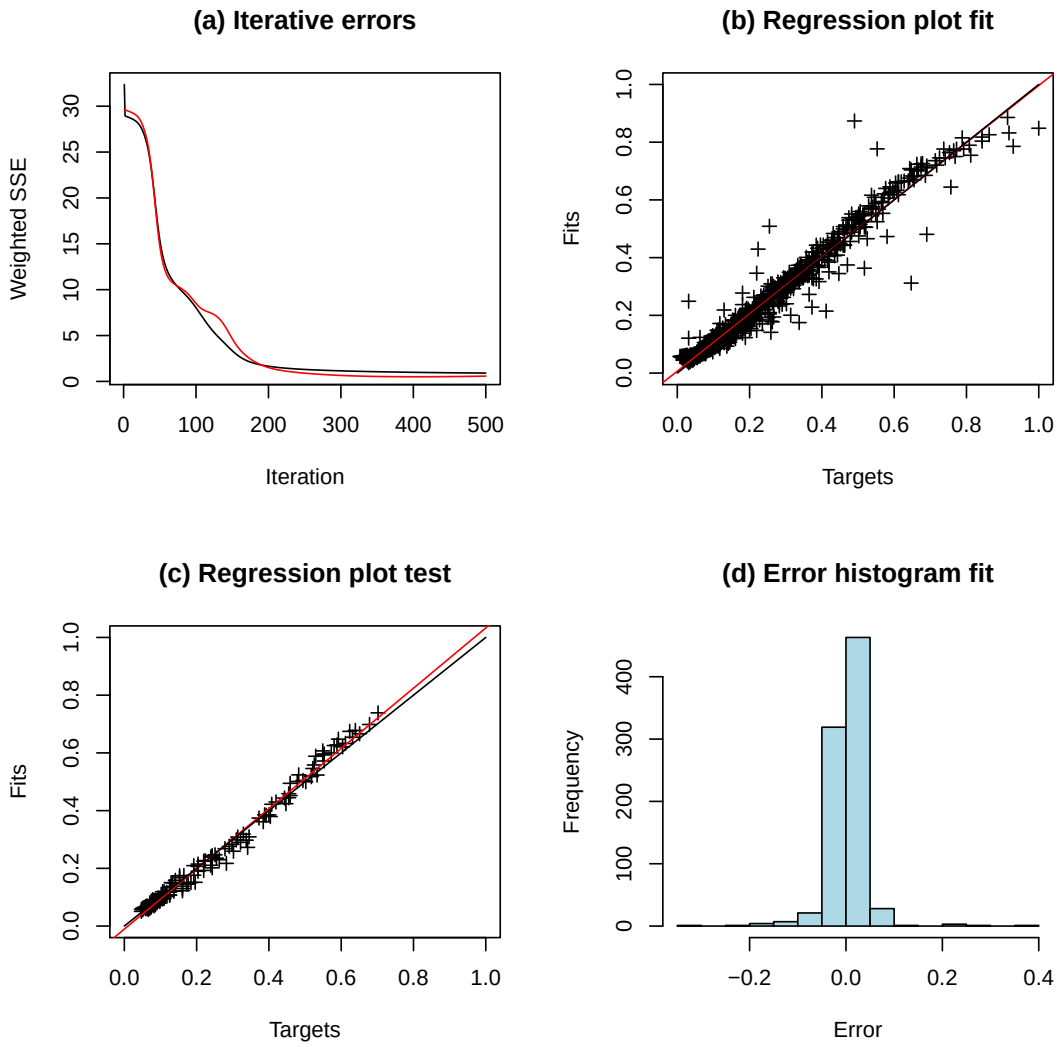

Figure 2: An Elman net trained with the laser example dataset. (a) The iterative error plot of both training (black) and test (red) error. (b) Regression plot for the training data, showing a linear fit in the optimal case (black), and to the current data (red). (c) Regression plot for the test data. (d) An error histogram of the training error.

```
R> plotIterativeError(model)
R> plotRegressionError(patterns$targetsTrain, model$fitted.values)
R> plotRegressionError(patterns$targetsTest, model$fittedTestValues)
R> hist(model$fitted.values - patterns$targetsTrain)
```

6.2. Multi-layer perceptron for classification

Performing classification or regression is very similar with the package. The neural output is typically set to the logistic function instead of the linear function, and an output neuron is used for each possible class. Training targets force the activation of the neuron representing

the correct class, i.e., using the logistic function its output should be close to one. The other neurons should output values close to zero. Methods of pre- and postprocessing to facilitate such a procedure are present in **RSNNS**. In the following, we present an example of how to train a multi-layer perceptron using the standard backpropagation algorithm (Rumelhart *et al.* 1986). We use the well-known `iris` dataset included in R for this example.

The data is loaded, shuffled, and preprocessed. The function `decodeClassLabels` generates a binary matrix from an integer-valued input vector representing class labels. With `splitForTrainingAndTest`, the data is split into training and test set, that can then be normalized using the function `normTrainingAndTestSet`, which has different normalization types implemented. We use a normalization to zero mean and variance one, which is the default setting:

```
R> data("iris")
R> iris <- iris[sample(1:nrow(iris), length(1:nrow(iris))), 1:ncol(iris)]
R> irisValues <- iris[, 1:4]
R> irisTargets <- iris[, 5]
R> irisDecTargets <- decodeClassLabels(irisTargets)
R> iris <- splitForTrainingAndTest(irisValues, irisDecTargets, ratio = 0.15)
R> iris <- normTrainingAndTestSet(iris)
```

The training data of this structure can then be used for training the multi-layer perceptron (or any other supervised learning method):

```
R> model <- mlp(iris$inputsTrain, iris$targetsTrain, size = 5,
+    learnFuncParams = 0.1, maxit = 60, inputsTest = iris$inputsTest,
+    targetsTest = iris$targetsTest)
R> predictions <- predict(model, iris$inputsTest)
```

Again, iterative and regression error plots can be used for analysis, but the regression error plot is less informative than for a regression problem (see Figure 3). Also, a function for displaying receiver operating characteristics (ROC) is included in the package. ROC plots are usually used for the analysis of classification problems with two classes. With more classes, for every class a ROC plot can be generated by combining all other classes to one class, and using the output of only the output of the corresponding neuron for drawing the ROC plot (see Figure 3 for examples):

```
R> plotIterativeError(model)
R> plotRegressionError(predictions[, 2], iris$targetsTest[, 2], pch = 3)
R> plotROC(fitted.values(model)[, 2], iris$targetsTrain[, 2])
R> plotROC(predictions[, 2], iris$targetsTest[, 2])
```

However, using ROC plots in this way might be confusing, especially if many classes are present in the data. Yet in the given example with three classes it is probably more informative to analyze confusion matrices, using the function `confusionMatrix`. A confusion matrix shows the amount of times the network erroneously classified a pattern of class X to be a member of class Y. If the class labels are given as a matrix to the function `confusionMatrix`, it encodes them using the standard setting. Currently, this standard setting is a strict winner-takes-all (WTA) algorithm that classifies each pattern to the class that is represented by the

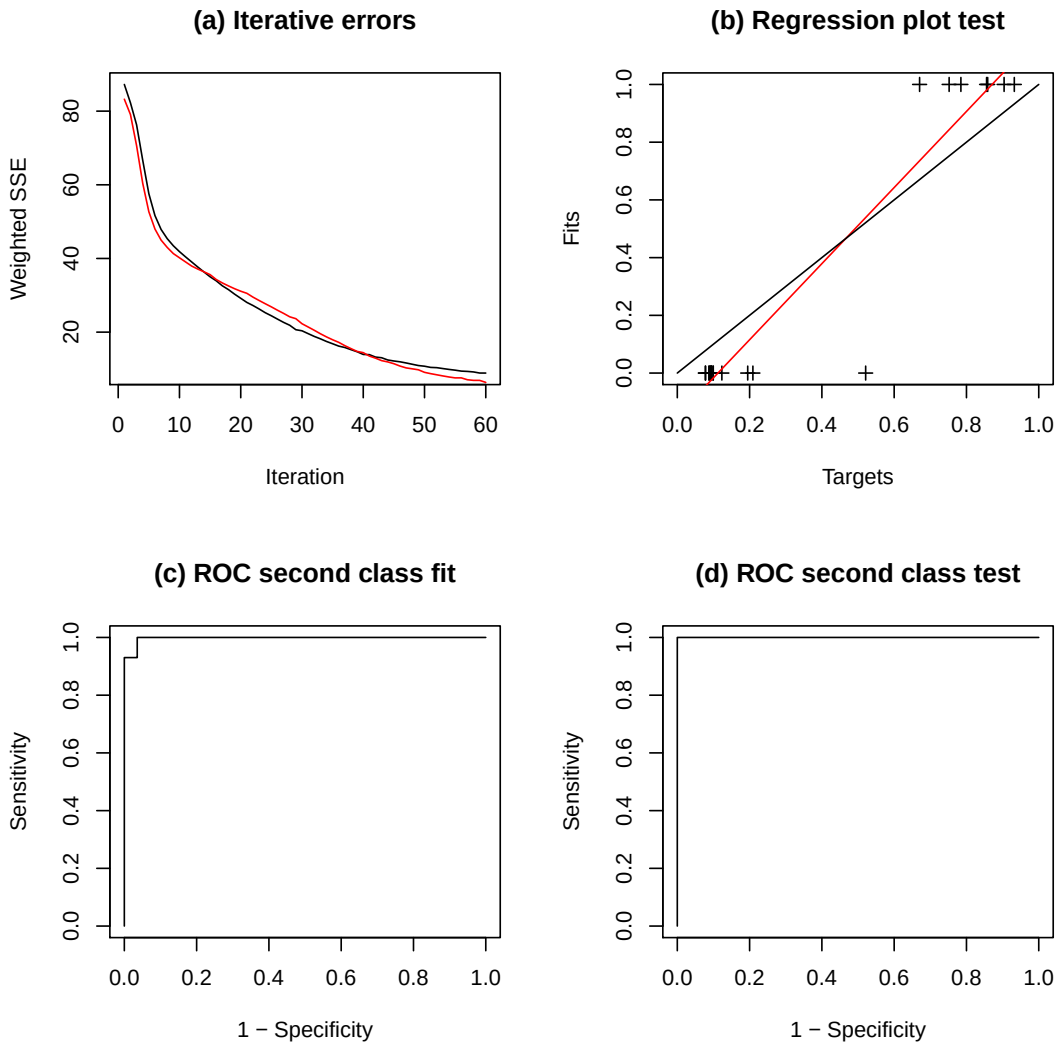

Figure 3: A multi-layer perceptron trained with the `iris` dataset. (a) The iterative error plot of both training (black) and test (red) error. (b) The regression plot for the test data. As a classification is performed, ideally only the points $(0, 0)$ and $(1, 1)$ would be populated. (c) ROC plot for the second class against all other classes, on the training set. (d) Same as (c), but for the test data.

neuron having maximal output activation, regardless of the activation of other units. For other encoding algorithms, the class labels can be encoded manually. In the following example, besides the default, the `402040` method is used, which is named after its default parameters, the two thresholds $l = 0.4$, and $h = 0.6$. In this configuration, these two thresholds divide the $[0, 1]$-interval into a lower part with 40% of the values, a middle part of 20% of the values and an upper part containing 40% of the values. The method classifies a pattern to the corresponding class of an output neuron, if this output neuron has an activation in the upper part, and all other neurons have an activation in the lower part. Otherwise, the pattern is

treated as unclassified. In the current implementation, unclassified patterns are represented by a zero as the class label. If `WTA` is used with standard settings, no unclassified patterns occur. Both `402040` and `WTA` are implemented as described in Zell *et al.* (1998). In the following, we show the calculation of confusion matrices for the training and the test dataset:

```
R> confusionMatrix(iris$targetsTrain, fitted.values(model))

       predictions
targets  1  2  3
      1 42  0  0
      2  0 40  3
      3  0  1 41

R> confusionMatrix(iris$targetsTest, predictions)

       predictions
targets 1 2 3
      1 8 0 0
      2 0 7 0
      3 0 0 8

R> confusionMatrix(iris$targetsTrain, encodeClassLabels(fitted.values(model),
+    method = "402040", l = 0.4, h = 0.6))

       predictions
targets  0  1  2  3
      1  0 42  0  0
      2  5  0 38  0
      3  3  0  0 39
```

Finally, we can have a look at the weights of the newly trained network, using the function `weightMatrix` (output is omitted):

```
R> weightMatrix(model)
```

6.3. Self-organizing map for clustering

A self-organizing map is an unsupervised learning method for clustering (Kohonen 1988). Similar input patterns result in spatially near outputs on the map. For example, a SOM can be trained with the `iris` data by:

```
R> model <- som(irisValues, mapX = 16, mapY = 16, maxit = 500,
+    targets = irisTargets)
```

The `targets` parameter is optional. If given, a labeled version of the SOM is calculated, to see if patterns from the same class are present as groups in the map. As for large pattern sets calculation of the outputs can take a long time, the parameters `calculateMap`, `calculateActMaps`, `calculateSpanningTree`, and `saveWinnersPerPattern` can be used to control which results are computed. Component maps are always computed. The results in more detail are:

- `model$actMaps`: An activation map is simply the network activation for one pattern. The activation maps list `actMaps` contains a list of activation maps for each pattern. If there are many patterns, this list can be very large. All the other results can be computed from this list. So, being an intermediary result, with limited use especially if many patterns are present, it is not saved if not explicitly requested by the parameter `calculateActMaps`.

- `model$map`: The most common representation of the self-organizing map. For each pattern, the winner neuron is computed from its activation map. As unit activations represent Euclidean distances, the winner is the unit with minimal activation. The map shows then, for how many patterns each neuron was the winner.

- `model$componentMaps`: For each input dimension there is one component map, showing where in the map this input component leads to high activation.

- `model$labeledMap`: A version of the map where for each unit the target class number is determined, to which the majority of patterns where the neuron won belong to. So, performance of the (unsupervised) SOM learning can be controlled using a problem that could also be trained with supervised methods.

- `model$spanningTree`: It is the same as `model$map`, except that the numbers do not represent the amount of patterns where the neuron won, but the number identifies the last pattern that led to minimal activation in the neuron. In contrast to the other results of SOM training, the spanning tree is available directly from the **SNNS** kernel. As the other results are probably more informative, the spanning tree is only interesting if the other functions require high computation times, or if the original **SNNS** implementation is needed.

The resulting SOM can be visualized using the `plotActMap` function which displays a heat map, or by any other R standard method, e.g., `persp`. If some units win much more often than most of the others, a logarithmic scale may be appropriate (plots are shown in Figure 4):

```
R> plotActMap(model$map, col = rev(heat.colors(12)))
R> plotActMap(log(model$map + 1), col = rev(heat.colors(12)))
R> persp(1:model$archParams$mapX, 1:model$archParams$mapY, log(model$map + 1),
+    theta = 30, phi = 30, expand = 0.5, col = "lightblue")
R> plotActMap(model$labeledMap)
```

The component maps can be visualized in the same way as the other maps (Figure 5 shows the plots):

```
R> for(i in 1:ncol(irisValues)) plotActMap(model$componentMaps[[i]],
+    col = rev(topo.colors(12)))
```

6.4. An ART2 network

ART networks (Grossberg 1988) are unsupervised learning methods for clustering. They offer a solution to a central problem in neural networks, the stability/plasticity dilemma, which

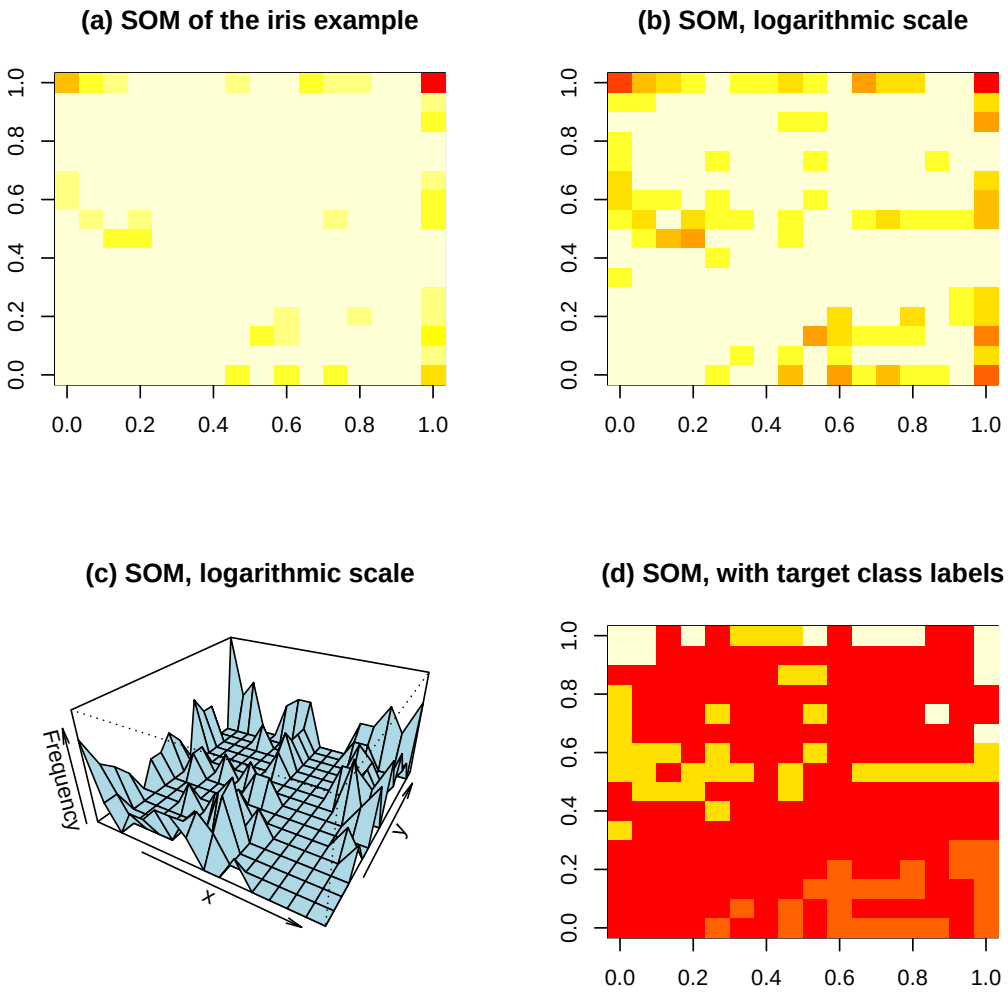

Figure 4: A SOM trained with the `iris` dataset. (a) A heat map that shows for each unit the amount of patterns where the unit won, from no patterns (white) to many patterns (red). (b) Same as (a), but on a logarithmic scale. (c) Same as (b), but as a perspective plot instead of a heat map. (d) Labeled map, showing for each unit the class to which the majority of patterns belong, for which the unit won.

means, that in general it is difficult in neural networks to learn new things without altering/deleting things already present in the net. In ART networks, plasticity is implemented in the way that new input patterns may generate new prototypes/cluster centers, if they are not represented yet by another prototype of the net. Stability is present, as a prototype is not altered by all new patterns, but only by new patterns that are similar to the prototype. The ART1 networks (Carpenter and Grossberg 1987b) only allow binary input patterns, ART2 (Carpenter and Grossberg 1987a) was developed for real-valued inputs. In the **SNNS** example `art2_tetra`, which we reimplement here, the inputs are noisy coordinates of the corners of a tetrahedron.

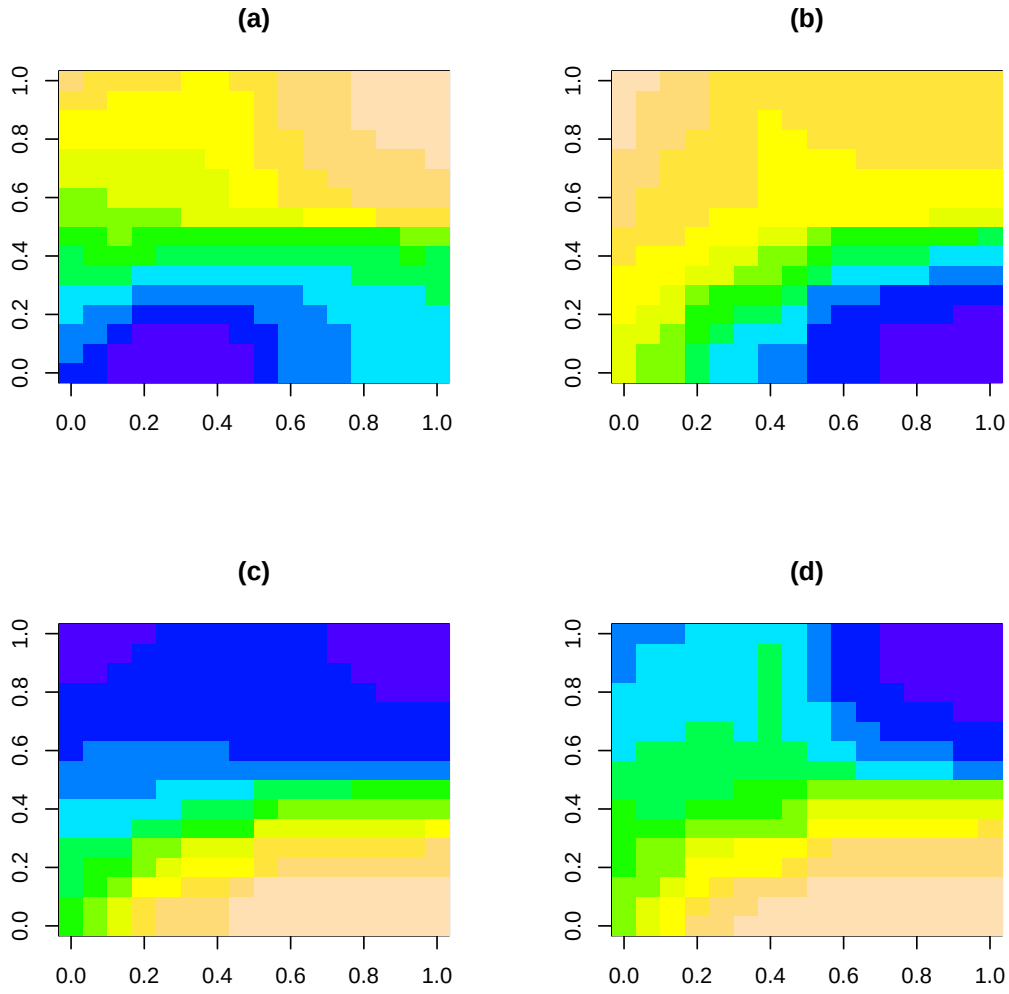

Figure 5: (a)–(d) Component maps for the SOM trained with the `iris` data. As the `iris` dataset has four inputs, four component maps are present that show for each input, where in the map it leads to high activation.

The architecture parameter `f2Units` defines the amount of hidden units present in the f2-layer of the network, and so gives the maximal amount of clusters that can be saved in the net (for details, see Zell *et al.* (1998) and Herrmann (1992)).

The model can be built in the following way:

```
R> patterns <- snnsData$art2_tetra_med.pat
R> model <- art2(patterns, f2Units = 5,
+    learnFuncParams = c(0.99, 20, 20, 0.1, 0),
+    updateFuncParams = c(0.99, 20, 20, 0.1, 0))
```

For visualization of this example, it is convenient to use the R package **scatterplot3d** (Ligges

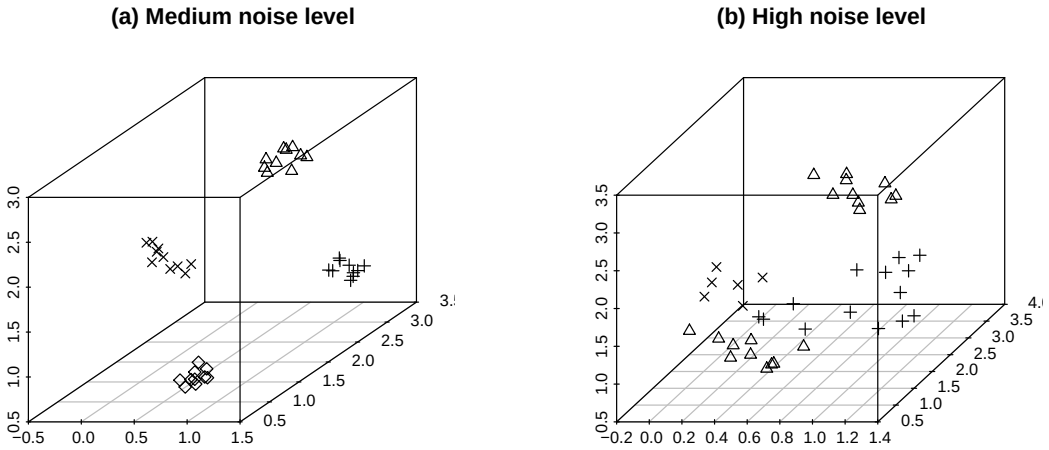

Figure 6: An ART2 network trained with noisy input data which represent the corners of a three-dimensional tetrahedron. (a) Data with medium noise level, the method correctly assumes four different clusters and clusters the points correctly. (b) Data with high noise level. The method generates three clusters. Note that the points of two of the corners are in one cluster though they are not spatially near. As ART2 uses normalized vectors, it only is able to take into account the direction of the vectors, which yields this unintuitive result (Herrmann 1992).

and Maechler 2003), for three-dimensional scatter plots:

```
R> library("scatterplot3d")
R> scatterplot3d(patterns, pch = encodeClassLabels(model$fitted.values))
```

Figure 6 shows this scatter plot, and the result of the same processing, but using as input data `snnsData$art2_tetra_high.pat`, which has more noise added.

6.5. Combined use of RSNNS and independent SNNS software

While **RSNNS** includes the major part of **SNNS** functionality, there are some bits left out, which can be interesting for some specific purposes: for example, the **SNNS** GUI is a very convenient tool for interactive visual development of a neural network topology or for manual analysis of a network weight's role. **RSNNS** offers functionality to interact with **SNNS**. This includes mainly reading/writing native dataset files (`.pat`, where files containing patterns with variable length are currently not supported) or neural network files (`.net`), as well as a rudimentary parser for `.res` files. This way, data and networks can be interchanged with an installation of the original **SNNS**, e.g., to visualize the network architecture in **SNNS**, or to train a net in **SNNS** and use R to analyze the capabilities of the net.

```
R> exportToSnnsNetFile(model, filename)
R> readPatFile(filename)
R> savePatFile(inputs, targets, filename)
R> readResFile(filename)
```

The `.pat` file methods make use of the original **SNNS** methods. Furthermore, `.net` files can be loaded and saved with the normal **SNNS** kernel methods `loadNet` and `saveNet`.

7. Neural network packages in R

In this section, we review the packages available directly in R or from CRAN implementing neural networks, and compare their functionality with algorithms available through **RSNNS**. As all neural network packages we are aware of on CRAN implement either feed-forward networks or Kohonen networks (LVQ, SOM), we discuss the packages grouped accordingly in this section. Furthermore, we give a general discussion about the functionality that has become available through **RSNNS** to R, and on general limitations of both **SNNS** and the wrapper package.

The only package present on CRAN yet that tackles connection between **SNNS** and R is **write.snns** (Castejón Limas *et al.* 2007), which implements a function to export data from R to a **SNNS** pattern file (`.pat`). This functionality is included in **RSNNS**, using directly the original **SNNS** functions for input and output of pattern files.

7.1. Feed-forward neural networks

There are several packages available for R that implement multi-layer perceptrons. Furthermore, implementations of quantile regression neural networks and flexible radial basis function networks exist.

nnet The package **nnet** (Venables and Ripley 2002) is part of the recommended R packages that usually ship directly with R. So, **nnet** can be considered the R standard neural network package. It implements a multi-layer perceptron with one hidden layer. For weight adjustment, it does not use backpropagation nor one of its variants, but a general quasi-Newton optimization procedure, the BFGS algorithm. Ripley (2007) argues that to "use general algorithms from unconstrained optimization [...] seems the most fruitful approach". A similar method, the scaled conjugate gradient (SCG, Møller 1993), is implemented in **SNNS**. SCG combines a conjugate gradient approach with ideas from the Levenberg-Marquardt algorithm. Møller (1993) compares SCG with standard backpropagation, another conjugate gradient algorithm, and the BFGS algorithm. In his comparison, the SCG performs best.

AMORE The package **AMORE** (Castejón Limas *et al.* 2010) implements the "TAO-robust backpropagation learning algorithm" (Pernía Espinoza *et al.* 2005), which is a backpropagation learning algorithm designed to be robust against outliers in the data. Furthermore, adaptive backpropagation and adaptive backpropagation with momentum term, both in online and batch versions, are implemented. The algorithms use an individual learning rate for each unit. Adaptive backpropagation procedures are not implemented in this way in **SNNS**, but using different learning rates for the units or the weights is a common idea for enhancing neural network learning procedures, and e.g., resilient backpropagation, which is implemented in **SNNS**, adapts the learning rate for each weight.

Furthermore, the package aims at giving a general framework for the implementation of neural networks (`http://rwiki.sciviews.org/doku.php?id=packages:cran:amore`), i.e.,

for defining units, their activation functions, connections, etc. The **SNNS** kernel interface implements such structures that are used internally by the algorithms implemented in **SNNS**. This interface can be accessed by the low-level interface functions of **RSNNS**.

neuralnet The package **neuralnet** (Fritsch *et al.* 2010) implements standard backpropagation and two types of resilient backpropagation (Riedmiller and Braun 1993; Riedmiller 1994). These algorithms are also available in **SNNS**, in implementations of the original authors of the algorithms. Furthermore, the package implements a "modified globally convergent version of the algorithm" presented by Anastasiadis *et al.* (2005), and a method for "the calculation of generalized weights", which are not present in **SNNS**.

monmlp The package **monmlp** (Cannon 2011a) implements a multi-layer perceptron with partial monotonicity constraints (Zhang and Zhang 1999). The algorithm allows for the definition of monotonic relations between inputs and outputs, which are then respected during training. If no constraints are defined, the algorithms behave as the usual, unconstrained versions. Implemented are standard backpropagation, and learning using a nonlinear least-squares optimizer. Furthermore, a stopping criterion using a bootstrap procedure is implemented. In **SNNS**, monotonic constraints methods are not implemented. Regarding the standard procedures, backpropagation and SCG, which uses a general optimizer, are present in **SNNS**. Stopping criteria are not implemented in **SNNS**, but as this part is controlled by **RSNNS** in the R code, could be considered for future versions of the package.

qrnn The package **qrnn** (Cannon 2011b) implements a quantile regression neural network, which can be used to produce probability density forecasts, especially in environments with both continuous and discrete input variables. This is not implemented in **SNNS**.

frbf The package **frbf** (Martins 2009) implements an algorithm for flexible radial basis functions (Falcao *et al.* 2006). The algorithm can be used for classification only. The algorithm constructs in a first phase of unsupervised learning the network topology from the training data, and later uses different kernels for each class. The algorithm is not included in **SNNS**, but standard radial basis functions are. Furthermore, an algorithm that also is only suitable for classification and constructs the network topology on its own is present with the RBF dynamic decay adjustment algorithm (Berthold and Diamond 1995).

7.2. Kohonen networks

Several packages in R implement SOMs. The version that is implemented in **SNNS** uses Euclidean distance and a quadratic neighborhood.

class The package **class** (Venables and Ripley 2002) is one of the recommended packages in R . It implements a SOM with rectangular or hexagonal grid, and the learning vector quantization algorithms LVQ1, LVQ2.1, LVQ3, and OLVQ. LVQ1 only adapts the winning prototype, whereas LVQ2.1 and LVQ3 also adapt the second best prototype. OLVQ uses a different learning rate for every unit. An implementation of LVQ is present in **SNNS**, where it is called dynamic LVQ, because it starts with an empty network/codebook, and adds successively new units.

som The package **som** (Yan 2010) implements a self-organizing map, focused on its application in gene clustering. It has hexagonal and rectangular topologies implemented, as well as basic visualization functionality.

kohonen The package **kohonen** (Wehrens and Buydens 2007) implements a standard SOM, as well as a supervised SOM with two parallel maps, and a SOM with multiple parallel maps. It is based on the package **class**. It has hexagonal and quadratic neighborhood relationships implemented, as well as toroidal maps. Besides the usual Euclidean distance, for class labels the Tanimoto distance can be used.

When class labels are available, there are various potential possibilities of how to use them during SOM training. An easy possibility is to use them (as in Section 6.3) merely during visualization. Another possibility that can be used with the standard SOM, is to add the class labels as additional feature, possibly with a weighting. However, the method implemented in this package, presented by Melssen *et al.* (2006), uses two different maps. The algorithm is an enhancement of counterpropagation (Hecht-Nielsen 1987), which is implemented in **SNNS**.

Besides the supervised versions of the SOM, the package also offers various visualization possibilities for the maps.

wccsom The package **wccsom** (Wehrens 2011) is another package from the authors of the **kohonen** package. It implements another distance measure, the weighted cross-correlation (WCC). Furthermore, it implements maps that start with few units, and add additional units during training by interpolation. This way, training can be performed faster.

7.3. Possibilities and limitations of RSNNS

So far, neural network methods are scattered along several packages. **RSNNS** addresses this lack of a standard neural network package in R by making the **SNNS** functionality usable from R, and therewith offering a uniform interface to many different standard learning procedures and architectures. For those models or algorithms for which there are already implementations available in R, such as for resilient propagation, standard backpropagation, or DLVQ, **RSNNS** offers alternatives that can be used complementary and for comparison. Regarding SOMs, there exist yet sophisticated packages in R, which offer powerful visualization procedures and flexible network construction, so use of the SOM standard implementation in **RSNNS** will usually not have benefits. But **RSNNS** also makes available a lot of architectures and learning procedures that were not present for R before, to the best of our knowledge, such as partial recurrent neural networks (Jordan and Elman nets), the ART theory, associative memories, Hopfield networks, the cascade correlation networks, or network pruning algorithms.

SNNS is a robust and fast software with standard implementations of a large amount of neural network techniques (see Section 2), validated by many years of employment by a huge group of users. Though **RSNNS** successfully overcomes some of the main problems that the use of **SNNS** in a modern experiment design raises, some others persist. As active development terminated in 1998, newer network types are not present. However, building a comprehensive up-to-date neural network standard package is a difficult task. The packages currently available from CRAN mainly focus on implementation of special types of architectures and/or learning functions. An important question in this context seems, if **RSNNS** could provide a suitable architecture to add new network types. Though **SNNS** in general is well-written

software with a suitable architecture and kernel API, to extend the kernel, knowledge of its internals is necessary, and the networks would need to be added directly in `SnnsCLib`, as the current wrapping mechanism of **RSNNS** is one-way, i.e., **RSNNS** provides no mechanisms that allow to implement learning, initialization, or update functions for use from within the **SNNS** kernel in R.

Regarding other limitations of the wrapping mechanism itself, the low-level interface offers full access to not only the kernel user interface, but also some functions that were not part of the kernel (bignet), and some functions that were not part of the original **SNNS**, but implemented for `SnnsCLib`. Some functions of the kernel interface are currently excluded from wrapping (see the file KnownIssues, that installs with the package), but this does not limit the functionality of **RSNNS**, as those functions usually implement redundant features, or features that can be easily reimplemented from R. So, with the low-level interface, all of the functionality of **SNNS** can be used.

In contrast to the direct mapping of the **SNNS** kernel to R functions of the low-level interface, the high-level interface implements task-oriented functionality that is mainly inspired by the original **SNNS** examples. The high-level functions define the architecture of the net, and the way the learning function is called (e.g., iteratively or not). So, the high-level functions are still pretty flexible, and e.g., the function `mlp` is suitable for a wide range of learning functions. Naturally, there are cases where such flexibility is not necessary, so for example the ART networks are only suitable for their specific type of learning.

8. Conclusions

In this paper, we presented the package **RSNNS** and described its main features. It is essentially a wrapper in R for the well-known **SNNS** software. It includes an API at different levels of trade-off between flexibility/complexity of the neural nets and convenience of use. In addition it provides several tools to visualize and analyze different features of the models. **RSNNS** includes a fork of **SNNS**, called `SnnsCLib`, which is the base for the integration of the **SNNS** functionality into R, an environment where automatization/scriptability and parallelization play an important role. Through the use of **Rcpp**, the wrapping code is kept straightforward and well encapsulated, so that `SnnsCLib` could also be used on its own, in projects that do not use R. Flexibility and full control of the networks is given through the **RSNNS** low-level interface `SnnsR`. The implemented calling mechanism enables object serialization and error handling. The high-level interface `rsnns` allows for seamless integration of many common **SNNS** methods in R programs.

In addition, various scenarios exist where the combined usage of the original **SNNS** and **RSNNS** can be beneficial. **SNNS** can be used as an editor to build a network which afterwards can be trained and analyzed using **RSNNS**. Or, a network trained with **RSNNS** can be saved along with its patterns, and **SNNS** can be used for a detailed analysis of the behavior of the net (or parts of the net or even single units or connections) on certain patterns.

As discussed in Section 7, packages currently available in R are focused on distinct network types or applications, so that **RSNNS** is the first general purpose neural network package for R, and could become the new standard for neural networks in R.

Acknowledgments

This work was supported in part by the Spanish Ministry of Science and Innovation (MICINN) under Project TIN-2009-14575. C. Bergmeir holds a scholarship from the Spanish Ministry of Education (MEC) of the "Programa de Formación del Profesorado Universitario (FPU)".

References

Anastasiadis AD, Magoulas GD, Vrahatis MN (2005). "New Globally Convergent Training Scheme Based on the Resilient Propagation Algorithm." *Neurocomputing*, **64**(1–4), 253–270.

Bergmeir C, Benítez JM (2012). *Neural Networks in R Using the Stuttgart Neural Network Simulator: **RSNNS***. R package version 0.4-3, URL http://CRAN.R-Project.org/package=RSNNS.

Berthold MR, Diamond J (1995). "Boosting the Performance of RBF Networks with Dynamic Decay Adjustment." In *Advances in Neural Information Processing Systems*, pp. 521–528. MIT Press.

Bishop CM (2003). *Neural Networks for Pattern Recognition*. Oxford University Press.

Cannon AJ (2011a). ***monmlp**: Monotone Multi-Layer Perceptron Neural Network*. R package version 1.1, URL http://CRAN.R-project.org/package=monmlp.

Cannon AJ (2011b). "Quantile Regression Neural Networks: Implementation in R and Application to Precipitation Downscaling." *Computers & Geosciences*, **37**(9), 1277–1284.

Carpenter GA, Grossberg S (1987a). "ART 2: Self-Organization of Stable Category Recognition Codes for Analog Input Patterns." *Applied Optics*, **26**(23), 4919–4930.

Carpenter GA, Grossberg S (1987b). "A Massively Parallel Architecture for a Self-Organizing Neural Pattern Recognition Machine." *Computer Vision, Graphics and Image Processing*, **37**, 54–115.

Carpenter GA, Grossberg S, Reynolds JH (1991). "ARTMAP: Supervised Real-Time Learning and Classification of Nonstationary Data by a Self-Organizing Neural Network." *Neural Networks*, **4**(5), 565–588.

Castejón Limas M, Ordieres Meré JB, de Cos Juez FJ, Martínez de Pisón Ascacibar FJ (2007). ***write.snns**: Function for Exporting Data to SNNS Pattern Files*. R package version 0.0-4.2, URL http://CRAN.R-project.org/package=write.snns.

Castejón Limas M, Ordieres Meré JB, González Marcos A, Martínez de Pisón Ascacibar FJ, Pernía Espinoza AV, Alba Elías F (2010). ***AMORE**: A MORE Flexible Neural Network Package*. R package version 0.2-12, URL http://CRAN.R-project.org/package=AMORE.

Cun YL, Denker JS, Solla SA (1990). "Optimal Brain Damage." In *Advances in Neural Information Processing Systems*, pp. 598–605. Morgan Kaufmann.

Eddelbuettel D, François R (2011). "**Rcpp**: Seamless R and C++ Integration." *Journal of Statistical Software*, **40**(8), 1–18.

Elman JL (1990). "Finding Structure in Time." *Cognitive Science*, **14**(2), 179–211.

Fahlman S, Lebiere C (1990). "The Cascade-Correlation Learning Architecture." In *Advances in Neural Information Processing Systems 2*, pp. 524–532. Morgan Kaufmann.

Fahlman SE (1988). "An Empirical Study of Learning Speed in Back-Propagation Networks." *Technical Report CMU-CS-88-162*, Carnegie Mellon, Computer Science Department.

Fahlman SE (1991). "The Recurrent Cascade-Correlation Architecture." In *Advances in Neural Information Processing Systems 3*, pp. 190–196. Morgan Kaufmann.

Falcao AO, Langlois T, Wichert A (2006). "Flexible Kernels for RBF Networks." *Neurocomputing*, **69**(16-18), 2356–2359.

Fritsch S, Guenther F, Suling M (2010). ***neuralnet**: Training of Neural Networks*. R package version 1.31, URL http://CRAN.R-project.org/package=neuralnet.

Grossberg S (1988). *Adaptive Pattern Classification and Universal Recoding. I.: Parallel Development and Coding of Neural Feature Detectors*, chapter I, pp. 243–258. MIT Press.

Haykin SS (1999). *Neural Networks: A Comprehensive Foundation*. Prentice Hall.

Hecht-Nielsen R (1987). "Counterpropagation Networks." *Applied Optics*, **26**(23), 4979–4984.

Herrmann KU (1992). *ART – Adaptive Resonance Theory – Architekturen, Implementierung und Anwendung*. Diplomarbeit 929, IPVR, University of Stuttgart.

Hopfield JJ (1982). "Neural Networks and Physical Systems with Emergent Collective Computational Abilities." *Proceedings of the National Academy of Sciences of the United States of America*, **79**(8), 2554–2558.

Hudak M (1993). "RCE Classifiers: Theory and Practice." *Cybernetics and Systems*, **23**(5), 483–515.

Jordan MI (1986). "Serial Order: A Parallel, Distributed Processing Approach." *Advances in Connectionist Theory Speech*, **121**(ICS-8604), 471–495.

Jurik M (1994). "BackPercolation." *Technical report*, Jurik Research. URL http://www.jurikres.com/.

Kohonen T (1988). *Self-Organization and Associative Memory*, volume 8. Springer-Verlag.

Lang KJ, Waibel AH, Hinton GE (1990). "A Time-Delay Neural Network Architecture for Isolated Word Recognition." *Neural Networks*, **3**(1), 23–43.

Lange JM, Voigt HM, Wolf D (1994). "Growing Artificial Neural Networks Based on Correlation Measures, Task Decomposition and Local Attention Neurons." In *IEEE International Conference on Neural Networks – Conference Proceedings*, volume 3, pp. 1355–1358.

Ligges U, Maechler M (2003). "**scatterplot3d** – An R Package for Visualizing Multivariate Data." *Journal of Statistical Software*, **8**(11), 1–20. URL http://www.jstatsoft.org/v08/i11/.

Martins F (2009). ***frbf**: Implementation of the 'Flexible Kernels for RBF Network' Algorithm.* R package version 1.0.1, URL http://CRAN.R-project.org/package=frbf.

Melssen W, Wehrens R, Buydens L (2006). "Supervised Kohonen Networks for Classification Problems." *Chemometrics and Intelligent Laboratory Systems*, **83**(2), 99–113.

Møller MF (1993). "A Scaled Conjugate Gradient Algorithm for Fast Supervised Learning." *Neural Networks*, **6**(4), 525–533.

Pernía Espinoza AV, Ordieres Meré JB, Martínez de Pisón Ascacibar FJ, González Marcos A (2005). "TAO-Robust Backpropagation Learning Algorithm." *Neural Networks*, **18**(2), 191–204.

Poggio T, Girosi F (1989). "A Theory of Networks for Approximation and Learning." *Technical Report A.I. Memo No.1140, C.B.I.P. Paper No. 31*, MIT Artifical Intelligence Laboratory.

R Development Core Team (2011). *R: A Language and Environment for Statistical Computing.* R Foundation for Statistical Computing, Vienna, Austria. ISBN 3-900051-07-0, URL http://www.R-project.org/.

Riedmiller M (1994). "Rprop – Description and Implementation Details." *Technical report*, University of Karlsruhe.

Riedmiller M, Braun H (1993). "Direct Adaptive Method for Faster Backpropagation Learning: The RPROP Algorithm." In *1993 IEEE International Conference on Neural Networks*, pp. 586–591.

Ripley BD (2007). *Pattern Recognition and Neural Networks.* Cambridge University Press.

Rojas R (1996). *Neural Networks: A Systematic Introduction.* Springer-Verlag.

Rosenblatt F (1958). "The Perceptron: A Probabilistic Model for Information Storage and Organization in the Brain." *Psychological Review*, **65**(6), 386–408.

Rumelhart DE, Clelland JLM, Group PR (1986). *Parallel Distributed Processing: Explorations in the Microstructure of Cognition.* MIT Press.

Venables WN, Ripley BD (2002). *Modern Applied Statistics with S.* Springer-Verlag, New York. URL http://www.stats.ox.ac.uk/pub/MASS4.

Wehrens R (2011). ***wccsom**: SOM Networks for Comparing Patterns with Peak Shifts.* R package version 1.2.4, URL http://CRAN.R-project.org/package=wccsom.

Wehrens R, Buydens LMC (2007). "Self- and Super-Organizing Maps in R: The **kohonen** Package." *Journal of Statistical Software*, **21**(5), 1–19. URL http://www.jstatsoft.org/v21/i05/.

Yan J (2010). ***som**: Self-Organizing Map.* R package version 0.3-5, URL http://CRAN.R-project.org/package=som.

Zell A (1994). *Simulation Neuronaler Netze*. Addison-Wesley.

Zell A, *et al.* (1998). *SNNS Stuttgart Neural Network Simulator User Manual, Version 4.2*. IPVR, University of Stuttgart and WSI, University of Tübingen. URL http://www.ra.cs.uni-tuebingen.de/SNNS/.

Zhang H, Zhang Z (1999). "Feedforward Networks with Monotone Constraints." In *Proceedings of the International Joint Conference on Neural Networks*, volume 3, pp. 1820–1823.

Affiliation:

Christoph Bergmeir, José M. Benítez
Department of Computer Science and Artificial Intelligence
E.T.S. de Ingenierías Informática y de Telecomunicación
CITIC-UGR, University of Granada
18071 Granada, Spain
E-mail: c.bergmeir@decsai.ugr.es, j.m.benitez@decsai.ugr.es
URL: http://dicits.ugr.es/, http://sci2s.ugr.es/

Journal of Statistical Software http://www.jstatsoft.org/
published by the American Statistical Association http://www.amstat.org/
Volume 46, Issue 7 *Submitted:* 2010-12-20
January 2012 *Accepted:* 2011-10-03

On the use of cross-validation for time series predictor evaluation

C. Bergmeir, and J.M. Benítez. On the use of cross-validation for time series predictor evaluation. Information Sciences, 2012, 191, 192-213.

- Status: **Published**
- Impact Factor (JCR 2011): 2.833
- Subject category:
  - COMPUTER SCIENCE, INFORMATION SYSTEMS (9/135 Q1)

Information Sciences 191 (2012) 192–213

Contents lists available at SciVerse ScienceDirect

Information Sciences

journal homepage: www.elsevier.com/locate/ins

On the use of cross-validation for time series predictor evaluation

Christoph Bergmeir *, José M. Benítez

Department of Computer Science and Artificial Intelligence, E.T.S. de Ingenierías Informática y de Telecomunicación, CITIC-UGR, University of Granada, 18071 Granada, Spain

ARTICLE INFO

Article history:
Received 30 November 2010
Received in revised form 24 October 2011
Accepted 28 December 2011
Available online 4 January 2012

Keywords:
Cross-validation
Time series
Predictor evaluation
Error measures
Machine learning
Regression

ABSTRACT

In time series predictor evaluation, we observe that with respect to the model selection procedure there is a gap between evaluation of traditional forecasting procedures, on the one hand, and evaluation of machine learning techniques on the other hand. In traditional forecasting, it is common practice to reserve a part from the end of each time series for testing, and to use the rest of the series for training. Thus it is not made full use of the data, but theoretical problems with respect to temporal evolutionary effects and dependencies within the data as well as practical problems regarding missing values are eliminated. On the other hand, when evaluating machine learning and other regression methods used for time series forecasting, often cross-validation is used for evaluation, paying little attention to the fact that those theoretical problems invalidate the fundamental assumptions of cross-validation. To close this gap and examine the consequences of different model selection procedures in practice, we have developed a rigorous and extensive empirical study. Six different model selection procedures, based on (i) cross-validation and (ii) evaluation using the series' last part, are used to assess the performance of four machine learning and other regression techniques on synthetic and real-world time series. No practical consequences of the theoretical flaws were found during our study, but the use of cross-validation techniques led to a more robust model selection. To make use of the "best of both worlds", we suggest that the use of a blocked form of cross-validation for time series evaluation became the standard procedure, thus using all available information and circumventing the theoretical problems.

1. Introduction

Time series forecasting methods have become indispensable tools in a broad field of applications to understand and forecast, e.g., technical, physical, and economic data. They are nowadays used to make important decisions with far-reaching consequences, and evaluating the procedures' performance is crucial. Assessing an estimate of the error a predictor produces on average can be done for different purposes. Besides the common case, where a general measure of the reliability and accuracy of a system is needed, evaluation often becomes necessary to choose the best one out of different methods and/or parameter sets. Also, researchers proposing a new method are interested in the question, whether the new method performs better than the state-of-the-art methods. This is usually determined by the application and comparison of all the methods on a set of benchmarking data or within competitions. The steps usually involved in such an assessment are (i) computing the methods' results on the test data (ii) calculating the errors of these results with respect to reference data under the use of a loss function, (iii) computing an error measure from the errors, and (iv) using a model selection procedure to examine the distribution of the error measure and/or to find the best model that would be used in the final application.

* Corresponding author.
E-mail addresses: c.bergmeir@decsai.ugr.es (C. Bergmeir), j.m.benitez@decsai.ugr.es (J.M. Benítez).

0020-0255/$ - see front matter © 2012 Elsevier Inc. All rights reserved.
doi:10.1016/j.ins.2011.12.028

In classification and regression, the standard procedures are to use a quadratic loss function (as a normal distribution of the error is assumed), the root mean squared error as error measure, and cross-validation as the model selection procedure. However, in time series prediction such a consensus does not exist yet. Characteristics of the series such as the amount of observed values, the periodicity, or the complexity of the generating processes can be very different, as well as the types of forecasts needed (one-step-ahead, one-to-24-step-ahead, etc.). With the different nature of series, types of forecasts, and evaluation purposes, not only the methods that perform well and yield accurate forecasts vary, but also the evaluation techniques that guarantee robust, meaningful error estimates. The errors computed should be the ones that are relevant for the intended application, and the measure should be a good summary of the error distribution. With respect to this, important questions are whether the procedures achieve adequacy and diversity. Adequacy means that for each relevant horizon enough forecasts are available, whereas diversity means that the measured error should not depend on special events within the time series [47]. Finally, the model selection procedure should represent and use the distribution of the error measure reasonably.

In traditional forecasting of economic data, the analyzed series mainly represent yearly, quarterly, or monthly acquired data, so that series hardly reach a length longer than a couple of hundreds of values. Furthermore, data generation processes are often complex and poorly represented in the time series itself, so that the amount of information that can potentially be extracted and used for forecasting is limited. Methods widely used are linear methods (made popular by Box and Jenkins [8]) such as autoregression (AR), moving average (MA), or combinations of these (ARMA) Based on these methods, various approaches exist to tackle non-stationarity in the series. A possibility is to use the derivative of the series as input (ARIMA). Or seasonal decomposition procedures, e.g., the STL procedure [16], are used to decompose the series into (deterministic) trend and seasonality, and a stochastic process of the residuals that is stationary.

Regarding non-linear methods, threshold AR models (TAR) are used to partition the (stationary) time series into linear pieces, so that they can be modeled by linear methods. A regime-switching mechanism then decides, which linear model to use [48]. These models have evolved to a host of choices, and combining them with machine learning techniques is getting increasingly popular [6,20,24]. Neural networks are also popular [7,21,52], but as for this type of time series problems complex methods do not necessarily yield better results (this was one of the conclusions of the M3 forecasting competition [36]), their use is not always appropriate.

For evaluation, usually a part at the end of each series is reserved and not used during model generation. This is often called out-of-sample evaluation [47]. To avoid confusion with other evaluation techniques, we will call validation using a set taken from the end of the series *last block* validation.

Another application scenario is the problem a forecaster faces when implementing a concrete application in fields where time series are longer and the underlying processes are better represented, such as in electrical load or price forecasting, traffic-related data or other technical or physical data. Within these applications, the forecaster is typically interested in the forecast of only one concrete time series, and methods that perform well on that particular series. Machine learning techniques and traditional methods seem to outperform each other on time series with different characteristics. Though these characteristics are to the best of our knowledge not well specified so far [21], machine learning techniques usually work well on long series with high-frequency data [7]. So, in this scenario the use of neural networks, machine learning techniques in general and other regression techniques is popular [1,2,13,14,27,39], also as it is often possible to find non-linear patterns in the data, due to the large amount of observed values. The machine learning techniques and other general regression methods used in this context take lagged values of the time series as input to perform an autoregression. With the use of regression methods, also evaluation techniques known from this field are applied to the time series forecasts. Besides the use of statistical measures such as the Akaike and Bayesian information criteria (AIC, BIC), it has become common practice to use cross-validation techniques within regression, and so such methods are also used in the literature to evaluate autoregressions on time series [1,13,39].

We observe that, especially with respect to the model selection procedure, researchers might often be unaware of the advantages and risks of the methods they use. Cross-validation makes full use of the data, i.e., all available data is used both for testing and training. Hence, diversity and adequacy of the evaluation is achieved. But, as during autoregression the same values are used both as part of the input and as reference data, the training set and the test set are not independent if randomly chosen. And the time series might be generated by a process that evolves over time, thus hurting the fundamental assumptions of cross-validation that the data are independent and identically distributed (i.i.d.) [3]. On the other side, besides simulating the real-world application, last block evaluation solves these theoretical problems straightforwardly, and practical problems of missing values during training (as they are reserved for testing) do not arise. But it does not make full use of the data, so that especially in a competition situation, where the validation set is not available to the participants, some problems with respect to adequacy and diversity arise. Only one forecast per series and horizon can be calculated, and the error measure might, rather than being representative, reflect characteristics of the validation set not present neither in the rest of the series nor in future data.

The most important questions regarding this issue seem to be whether the theoretical problems are relevant in practice and therefore standard cross-validation might mislead the user and is applied erroneously in such situations, whether advantages of cross-validation prevail, yielding more robust results, and whether the theoretical shortcomings of cross-validation can be solved in a practical manner. With the aim of gaining further insight into the issue, we present a comprehensive and rigorous empirical study on this, using various cross-validation and last block techniques to evaluate machine learning and general regression methods. Performing an empirical study on this topic is also motivated by the fact, that asymptotic behavior known from theory of the evaluation methods might be quite different from their performance on small test sets [12].

Furthermore, we analyze the different problems of cross-validation in more detail. The problem of dependencies within training and test set can be solved by using blocks of data rather than choosing data randomly. The problem of time evolving effects is closely related to stationarity of the series, so that it can be tackled with known tools for detecting and removing stationarity from the series.

Complete experimental results as well as high-resolution color graphics and complementary information can be found at http://sci2s.ugr.es/dicits/papers/CV-TS.

The remainder of the paper is structured as follows. Section 2 presents traditional methods of evaluation in detail, regarding data splitting in training and test set, and error measures proposed for forecast evaluation. Section 3 details how regression techniques are used for forecasting and how they are evaluated, using cross-validation. Section 4 discusses the design of the experiments that are carried out within our work, and Section 5 shows the results. Section 6 summarizes the conclusions drawn from the paper.

2. Traditional predictor evaluation

The most common approach for data partitioning within traditional forecast evaluation is last block evaluation, as choosing the validation set in this way typically corresponds to the later use case of the system (the continuous forecasting of upcoming values), and the model can be trained and used as in a normal application situation. Furthermore, as we assume that the future depends on the past, the natural dependencies in the data are respected.

However, also within last block evaluation there exist different possibilities for choosing how to use the available data for model building and for evaluation, which are discussed in Section 2.1.

After choice and computation of pairs of forecasts and known reference values, important issues are the choice of a loss function, i.e., how the errors are computed and scaled, and the choice of an error measure that defines which errors are averaged in what way. Literature on this topic is discussed in Section 2.2.

2.1. Data partitioning and forecast horizons

Depending on the length of the last block and the process applied to the individual series, it may be the case that only few forecasts per time series and/or horizon are available (see Section 2.1.1). Adequacy and diversity of the error measure may then be obtained by averaging over different series, or over different horizons, see Section 2.1.2.

2.1.1. Individual series

When evaluating forecasts on individual series, there are mainly four possibilities for training and evaluation, which we name similar to Tashman [47] *fixed-origin*, *rolling-origin-recalibration*, *rolling-origin-update*, and *rolling-window evaluation*. In the following, let the *forecast origin* be the time point of the last known value, from which the forecast is performed. For example, if a daily time series over a certain period, that ends on day t with value x_t, is used to forecast the value x_{t+k} of day $t + k$, the forecast origin is t.

Fixed-origin evaluation is typically applied during forecasting competitions. A forecast for each value present in the test set is computed using only the training set. The forecast origin is fixed to the last point in the training set. So, for each horizon only one forecast can be computed. Obvious drawbacks of this type of evaluation are, that characteristics of the forecast origin might heavily influence evaluation results, and, as only one forecast per horizon is present, averaging is not possible within one series and one horizon.

Within rolling-origin-recalibration evaluation, forecasts for a fixed horizon are performed by sequentially moving values from the test set to the training set, and changing the forecast origin accordingly. For each forecast, the model is recalibrated using all available data in the training set, which often means a complete retraining of the model.

Rolling-origin-update evaluation is probably the normal use case of most applications. Forecasts are computed in analogy to rolling-origin-recalibration evaluation, but values from the test set are not moved to the training set, and no model recalibration is performed. Instead, past values from the test set are used merely to update the input information of the model. Both types of rolling-origin evaluation are often referred to as n-step-ahead evaluation, with n being the forecast horizon used during the evaluation. Tashman [47] argues that model recalibration probably yields better results than updating. But recalibration may be computationally expensive, and within a real-world application, the model typically will be built once by experts, and later it will be used with updated information as new values are available, but it will certainly not be rebuilt.

Rolling-window evaluation is similar to rolling-origin evaluation, but the amount of data used for training is kept constant, so that as new data is available, old data from the beginning of the series is discarded. Rolling-window evaluation is only applicable if the model is rebuilt in every window, and has merely theoretical statistical advantages, that might be noted in practice only if old values tend to disturb model generation.

2.1.2. Different series and horizons

Using forecasts of different horizons to compute an average error raises some problems. Values of different horizons have different statistical properties. With increasing horizon the uncertainty and the variance increases. This issue might be

addressed with a loss function that takes into account the horizon (as defined, e.g., by Kunst and Jumah [35]). But as the relative performance of methods depends on the forecast horizon used [18,36], an average calculated from forecasts of different horizons is potentially misleading.

Time series of different lengths are normally combined by keeping the length of the validation set constant, though depending on the overall procedure this may not be necessary. When averaging over different time series, it should be taken into account that there are very different types of time series. A method might be suited well for a certain type of series, but show weak performance on other types. So, without notions neither of the intended application nor of the time series types present in the evaluation database, averaging over different time series might be misleading. An obvious case of this problem is that time series with different time intervals such as yearly, monthly, and daily data should not be used for the computation of an averaged error measure.

A good test database should contain (according to Tashman [47]) heterogeneous groups of homogeneous time series. Within a homogeneous group adequacy may be achieved, and within the groups, diversity may be achieved. Thus, a good test database could compensate for the shortcomings of fixed-origin evaluation, what is especially important during forecasting competitions.

2.2. Accuracy measures

The purpose of error measures is to obtain a clear and robust summary of the error distribution [15]. It is common practice to calculate error measures by first calculating a loss function (usually eliminating the sign of the single errors) and then computing an average. Let in the following y_t be the observed value at time t, also called the reference value, and let $\widehat{y}_t$ be the forecast for y_t. The error E_t is then computed by $y_t - \widehat{y}_t$. Hyndman and Koehler [31] give a detailed review of different accuracy measures used in forecasting and classify the measures into these groups:

2.2.1. Scale-dependent measures

Standard error measures, where absolute errors $\mathrm{AE}_t = |y_t - \widehat{y}_t|$ or squared errors $\mathrm{SE}_t = (y_t - \widehat{y}_t)^2$ are averaged by arithmetic mean M or median MD, leading to the mean absolute error MAE, the median absolute error MDAE, the mean squared error MSE or the root mean squared error:

$$\mathrm{RMSE} = \sqrt{\frac{1}{n}\sum_{t=1}^{n}(y_t - \widehat{y}_t)^2}. \tag{1}$$

As these widely used standard error measures are scale-dependent, they cannot be used to compare and average errors across heterogeneous time series.

2.2.2. Percentage errors

To overcome scale-dependency, the error can be divided by the reference value, thus defining the percentage error:

$$\mathrm{PE}_t = 100\ \frac{y_t - \widehat{y}_t}{y_t}. \tag{2}$$

In analogy to scale-dependent measures, the mean absolute percentage error:

$$\mathrm{MAPE} = \frac{1}{n}\sum_{t=1}^{n}\left|100\ \frac{y_t - \widehat{y}_t}{y_t}\right|, \tag{3}$$

the median absolute percentage error MDAPE, the root mean squared percentage error RMSPE, or the root median squared percentage error RMDSPE can be computed.

The main problem regarding these measures is that they have to deal with infinite values if one y_t equals zero, or with undefined values, if $\widehat{y}_t = y_t = 0$ for one t. Time series containing zero values are common in many applications, e.g., in return on investment curves, or in traffic delay time series. Also, if the time series consists of real numbers and it is unlikely that exact zeros occur, percentage errors at small values are high (as the denominator gets small), and so the measures show a skewed distribution. By using the median for averaging (MDAPE, RMDSPE), these problems are easier to deal with, as single infinite or undefined values do not necessarily result in an infinite or undefined measure.

The so-called symmetric measures such as sMAPE or sMDAPE try to overcome these shortcomings. However, they are not as symmetric as their names suggest: the symmetry with respect to the interchange of forecast and reference value is obtained at the cost of loosing the symmetry that forecasts only differing in sign should result in the same error [26]. Furthermore, in their original definitions these measures are even able to take negative values; the definition of the sMAPE as it is used in the NNGC1[1] competition circumvents at least the latter shortcoming by using absolute values in the denominator:

$$\mathrm{sMAPE} = \frac{1}{n}\sum_{t=1}^{n} 100\ \frac{|y_t - \widehat{y}_t|}{m_t},\ \text{with}\ \ m_t = \frac{|y_t| + |\widehat{y}_t|}{2}. \tag{4}$$

[1] http://www.neural-forecasting-competition.com.

But the main problems when dealing with reference values and forecasts close and equal to zero persist.

2.2.3. Relative errors

Another possibility is not to scale using the reference value, but with the help of the error of a benchmark method B, where normally the naïve method is used, which uses the last known reference value as forecast. The relative error is defined as:

$$\mathrm{RE}_t = \frac{y_t - \widehat{y_t}}{y_t - \hat{y}_{tB}}, \tag{5}$$

where $\hat{y}_{tB}$ is the forecast for y_t obtained by the benchmark method. Using the RE, e.g., the mean relative absolute error MRAE, or the median relative absolute error MDRAE can be defined. However, general problems of the percentage error measures persist. If the naïve method is used as benchmark, and two subsequent values are zeros, the relative error measures might evaluate to infinity. If in addition the forecast is correct and takes a value of zero, the result is undefined.

2.2.4. Relative measures

Instead of calculating a relative error for each forecast value, averaged error measures can be computed for the forecasting method and a benchmark method. Using, e.g., the MAE, the relative MAE can be defined as:

$$\mathrm{RELMAE} = \frac{\mathrm{MAE}}{\mathrm{MAE}_B}. \tag{6}$$

The RELRMSE with the naïve method as benchmark is also known under the name of Theil's U [31]. Relative measures are able to circumvent many of the problems the other error measures have, as they do not have problems with zeros in the forecasts or the reference values (only if all values would be zero). A shortcoming of these measures is, that they cannot be computed as straightforward over various time series as measures based on percentage errors or relative errors. A second averaging step has to be performed to calculate the average of the relative measures computed on various time series. This is especially a problem, if there is only one forecast per time series and/or horizon available (as it is the case in forecasting competitions). Then, calculating, e.g., the RELMAE for each series with only one forecast and later calculating the mean over this measure across different series results in computing the overall MRAE, with all problems discussed.

Another problem, present in both relative errors and relative measures, is that the use of a benchmark may introduce unexpected or undesired behavior of the measures. The desired behavior of comparing the methods' performance to a benchmark may lead to the behavior that the measure represents rather characteristics of the benchmark. E.g., if there is a negative relation of lag one present in the series, the series tends to short-term oscillations and is less smooth, so that the benchmark will perform badly. Then, low values of the RELMAE in this situation, if compared or averaged with values of a similar series with positive feedback of lag one may be misleading. This shortcoming can be overcome by the usage of a more sophisticated benchmark like an ARIMA model. However, this might induce new problems and complicate error calculation and interpretability in general. Additionally, the performance of the naïve forecast may be sensitive to the horizon used, which might also lead to misinterpretation.

2.2.5. Others

As illustrated, all commonly used error measures have shortcomings, and no commonly accepted measure exists so far that is robust, scale-independent, easy to compute, use, and interpret. There is ongoing research on this topic. Hyndman and Koehler [31] propose to scale errors using the in-sample error of a benchmark method, namely the naïve method. They define the scaled error as

$$\mathrm{SE}_t = \frac{y_t - \hat{y}_t}{\frac{1}{n-1}\sum_{i=2}^{n}|y_i - y_{i-1}|}. \tag{7}$$

With the help of this error, e.g., the mean absolute scaled error MASE or the median absolute scaled error MDASE can be computed. The MASE has the advantage, that it has no problems with zeros in the input data in practice and that it can be used in a straightforward way to average across time series. However, its interpretation might be difficult and it uses a benchmark method, which has the shortcomings already discussed.

Chen and Yang [15] present several further normalization approaches, as well as some measures theoretically founded on the Kullback–Leibler divergence. Some of these are further discussed by Kunst and Jumah [35].

3. Cross-validation and regression for time series

Cross-validation is one of the most important tools in the evaluation of regression and classification methods. Its use is outlined in Section 3.1. If general regression techniques are used in time series forecasting (see Section 3.2), cross-validation is often used for evaluation in these applications as well, in spite of the theoretical problems that may arise. On the contrary, in traditional forecasting standard cross-validation receives little attention due to both theoretical and practical problems. Instead, a variety of validation techniques customized for time series can be found in the literature, which are discussed in Section 3.4.

Table 1
Example of a time series preprocessed for regression, using the last four values (lagged values) to predict the current value. Lines in boldface are values that are used for testing within the current iteration of cross-validation, the other ones are used for training.

Index	Lag4	Lag3	Lag2	Lag1	Target
		...	...		
7	14	10	26	11	−13
8	10	26	11	−13	−15
9	**26**	**11**	**−13**	**−15**	**−8**
10	11	−13	−15	−8	35
11	−13	−15	−8	35	40
12	**−15**	**−8**	**35**	**40**	**−8**
13	**−8**	**35**	**40**	**−8**	**−16**
14	**35**	**40**	**−8**	**−16**	**7**
15	40	−8	−16	7	17
		...	...		

3.1. Cross-validation and related methods in regression

In regression and classification, the main concern is usually the generalization ability of the system, i.e., its performance on unseen data. To get a valid estimation of this performance, data used for testing is typically not used for model building.

This raises two problems. Firstly, the data used for testing has to be omitted during training, although the system would probably yield better results if it had been trained with all available data, especially if the amount of data is small. Secondly, in a statistical sense the data available to the researcher is only one possible realization of a stochastic process sample, and so also the acquired error measure is one sample of a stochastic variable that has its possible realizations and probability distribution. This relates directly to adequacy and diversity of the accuracy measure. Depending on the amount of data available, the test set often cannot be chosen large, as the data in the test set cannot be used during training, which decreases adequacy. Diversity is decreased, as the measure computed might represent characteristics only observable in the test set, not in the rest of the data.

To tackle these problems, in classification and regression it is common practice to use *k-fold cross-validation* [3,46], where all available data is randomly partitioned into k sets. Then, the whole training or model fitting procedure, as well as the calculation of the error is performed k times, with every set being once used as the test set, and the other sets being used for model building. So, the method finally acquires k independent realizations of the error measure, and all data is used as well for training as for testing. Averaging the k obtained error measures yields an overall error measure that typically will be more robust than single measures.

In traditional regression and model selection theory, another popular way is to consider complexity of the models, as more complex models are more likely to overfit the data, which yields bad generalization abilities. In this context, a model with not more than the necessary amount of complexity is called a parsimonious model [19]. Model complexity is usually defined by the amount of parameters the model requires. Measures computing this kind of error, i.e., the error of fit with penalizing the amount of parameters are, e.g., the Akaike and Bayesian information criteria (AIC, BIC). Some authors discuss the relation of these measures and cross-validation. Shao [43] demonstrates that AIC and leave-one-out cross-validation (LOOCV) converge and asymptotically show the same behavior. However, Arlot and Celisse [3] argue that in a practical situation cross-validation is applicable to a wide range of problems, so that without knowledge of the data it is likely to yield better results than penalized information criteria.

3.2. Regression for time series prediction

As the name autoregressive (AR) model suggests, AR calculates an estimate for a future value using determined lagged values from the past. So, a regression of the time series on itself is performed. To use standard regression techniques for autoregression, time series must be preprocessed through an embedding step. The lags that are to be used as inputs are identified and a data matrix is built as seen in Table 1.

If autoregression is performed in this way, technically cross-validation can be performed as in normal regression. However, as the embedding procedure may use heavily overlapping parts of the time series (as illustrated in Table 1), the data used for regression is not statistically independent, so that one of the key assumptions of cross-validation is not met [3].

3.3. Stationarity and cross-validation

An important concept in time series research is stationarity, which means that the basic statistics of the time series do not change over time. A series is defined to be stationary (see, e.g., Cryer and Chan [19]), if for any time points $t_1, \ldots, t_n$, and any lag parameter k, the joint distribution of $x_{t_1}, \ldots, x_{t_n}$ and $x_{t_1-k}, \ldots, x_{t_n-k}$ is the same. From this follows especially, with $n = 1$, that the x_t are identically distributed, which is important for our work as this is one of the assumptions of cross-validation.

A related but weaker definition, sometimes referred to as second-order stationarity, is to define a series as (second-order) stationary if the mean remains constant throughout the series, and the autocorrelation of two values only depends on the

relative position of the values (the lag) within the series. From this definition follows that all values of the series have the same mean and variance. If all the joint distributions are Gaussian, the two concepts of stationarity are identical [19].

Non-stationarity has to be taken into account throughout the whole modeling process, not only during model selection. Depending on the type of stationarity, it can be easily removed by a preprocessing step such as differentiation (as done in ARIMA models) or a procedure that removes trend and seasonality. Also, with the Dickey–Fuller unit root test [42], the series can be checked for stationarity. If non-stationarity cannot be removed by such a preprocessing step, the model building procedure may require a processing step that determines, which parts of the series to include in the modeling, as proposed by Deco et al. [22], or prediction of the series might even be an impossible task [22,33].

Furthermore, for non-stationary series last block evaluation might be misleading as well, as the block chosen for testing might be very different from the training data, and the unknown future may also be different from the training data, the test data, or from both of these. Following Inoue and Kilian [32], and also Kunst [34], we could argue that in time series forecasting, the last block of a series might be the most relevant one, being probably most related to the data to predict. However, cases are easily imaginable where this is not the case, and if the last block in fact is the most important part, we suggest that it would be more appropriate to take this information into account while building the model (e.g., by weighting), and not just for its evaluation. Because of these difficulties it is common practice in time series forecasting to assume stationary time series.

Also, w.r.t. the application of cross-validation, the problem of dependent data can be dealt with more easily in the framework of stationarity. As the autocorrelation function only depends on the lags, it can be analyzed as a function of these. In a stationary series, often the number of lags with significant autocorrelation is small [19]. So we can assume that there is a constant h such that x_i and x_j are approximately independent, if $|i-j| > h$ [3,37,41].

It is worth noting, that actually the method outlined in Section 3.2 also is motivated by a stationarity assumption, as choosing particular lagged values as inputs for the forecasting procedure would not make sense, if their dependence on the value that is to be forecast would continuously change. So, if we assume that the model was built taking into account all lagged values with relevant correlations, the order of the model gives a good estimation on the number of values that are dependent and should therefore be omitted during cross-validation.

3.4. Cross-validation and related methods for time series

In addition to the theoretical problems of unmet basic assumptions, using cross-validation for the evaluation of traditional forecasting procedures leads to practical problems. As the partitions are chosen randomly, missing values have to be taken into account during model construction. Depending on the forecasting method, this can be a straightforward or a very difficult task. E.g., if the autocorrelation function is used during the model construction process, missing values might skew that function and eventually yield bad results.

Also, practical consequences of the theoretical problems have been observed in some cases in the literature, which will be discussed in the following. And various evaluation methods especially for time series have been proposed.

3.4.1. Methods based on the last block

As evaluation techniques based on the last block circumvent the discussed problems, various authors describe evaluation techniques based on the last block. Hjorth [29,30] proposes a procedure called "forward validation", which basically computes a weighted sum of one-step-ahead forecasts by the rolling-origin-recalibration procedure. The weights are normalized and depend on the number of parameters and the forecast origin: As the model is recalibrated, depending on the forecast origin more or less data is available for model construction. Finally, the method yielding the minimal error in forward validation is chosen. Additionally, the author presents an estimation for the bias that might be introduced by the model selection procedure in the error measure. Therefore, the forward validation procedure is performed using the chosen model subsequently on subseries of the last block, so that a set of error samples is obtained. The difference between the error measure of the last block and the mean of these subsequently calculated error measures can then be used as an estimate for the bias introduced by the model selection procedure. Wagenmakers et al. [50] use the so-called accumulative prediction error (APE) for model selection, which is the sum of one-step-ahead forecasts computed by rolling-origin-recalibration evaluation. The authors argue that this process has a strong theoretical justification by being related to the AIC and BIC. The advantage using the APE instead of AIC or BIC is according to them, that the APE is not only sensitive to the number of parameters, but also to their functional form. Both the forward validation method and the APE use rolling-origin-recalibration evaluation.

Inoue and Kilian [32] show, that information criteria such as AIC or BIC asymptotically perform better than evaluation on the last block of the series. They also admit the shortcoming that evaluation with the last block only uses a part of the information available, and so looses potentially important information.

3.4.2. Cross-validation with omission of dependent data

A brief review for methods of this type of cross-validation is given by Arlot and Celisse [3]. To solve the theoretical problems of cross-validation with correlated data, stationarity is assumed, and not only data that is used for testing is removed from the training set, but also data that is not independent from the data that is used for testing. The procedure is described, e.g., by McQuarrie and Tsai [37], and is often called modified cross-validation (MCV) [3]. We call it in the following *non-dependent cross-validation*. Burman et al. [9] present a related approach which they call *h*-block cross-validation. They

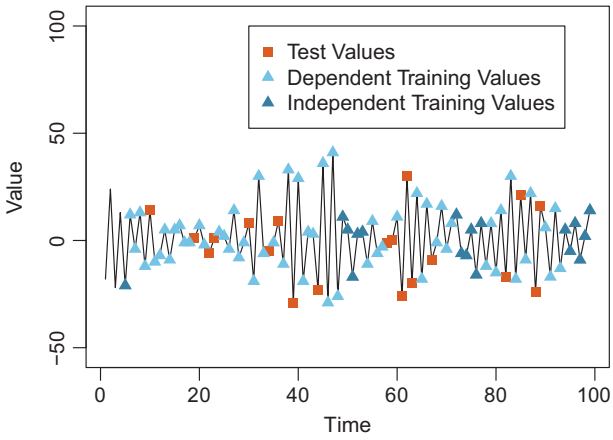

Fig. 1. Example of values that are used for training and test within one iteration of a 5-fold cross-validation procedure. Assuming a forecast procedure that uses the last four values to predict the next one, for every point in the test set values in a radius of four cannot be used for training, if independence has to be achieved. This leads to areas, in which nearly all points have to be excluded from training.

perform LOOCV, and remove the values correlated with the test values in both axis directions from the training set. Hart [28] presents a procedure he names time series cross-validation (TSCV), which simultaneously estimates an optimal bandwidth and the autocorrelation function of the data. Kunst [34] proposes the use of cross-validation with removed dependent values with a subsequent bootstrapping procedure. A related field is non-parametric regression with correlated errors. As in this field, no autoregression is performed and the series are not necessarily stationary, it can be seen as a preprocessing step for trend estimation within time series problems. Opsomer et al. [38] show that standard cross-validation chooses smaller bandwidths in a kernel estimator regression framework if autocorrelation of the error is high, so that the method overfits the data. They state, that depending on the regression method and the parameters to choose, short-range dependence has weak influence, whereas long-range dependence often has a high influence on the model selection process. Carmack et al. [10] present a method they call far casting cross-validation (FCCV), which is similar to h-block cross-validation, and tackles especially the problem of multivariate data by defining a neighborhood radius of dependent data to remove. They also present results with their method in a bandwidth selection framework for non-parametric regression, and show that LOOCV underestimates the error in certain cases.

Depending on the amount of lags used and the number of folds during cross-validation, omission of dependent values can lead to a significant loss of data or even to the removal of all data available for training (see Fig. 1). So, non-dependent cross-validation methods are only applicable in certain cases, where folds contain a low percentage of the overall data, or the amount of relevant lags is small. It has to be noted, that LOOCV is not only computationally costly, but in contrast to k-fold cross-validation it is also asymptotically not consistent (which would mean that with the amount of available data going to infinity, the probability of selecting the best model goes to one) [44].

3.4.3. Cross-validation with blocked subsets

Snijders [45] uses cross-validation with "non-interrupted time series as validating sub-samples". We call this type of cross-validation in the following *blocked cross-validation*. In particular, the last block can be one of these validating sub-samples. In that early study, the author compares blocked cross-validation with last block evaluation, using basic linear forecasting methods. As there is no clear tendency in the results, the use of last block evaluation is suggested as it is less costly to compute: last block evaluation only involves one training and evaluation step, whereas cross-validation involves these steps for every sub-sample. Racine [41] presents a method named hv-block cross-validation. It extents the h-block method in the way that not a single value is used for testing, but a block of data of size v. The author points out, that the method is asymptotically consistent for general stationary processes. In his experimental study that focuses on model selection capabilities and does not explicitly state error values, he shows that hv-blocked cross-validation yields better model selection performance than v-blocked cross-validation (where $h = 0$, so that no dependent values are omitted), if the series are large, i.e., longer than 500 values; on series with 5000 values his method achieves 10% higher probability for choosing the correct model. However, in practice such long series are often not available, and the task is not the choice of a true model (often there is no true underlying model at all), but the determination of a model that yields good forecasting performance. In particular, Kunst [34] showed that the model with the best forecasting performance is not always the true model, i.e., the model that generated the data.

4. Design of the experiments

As seen in Section 3.4, many authors state the theoretical problems when cross-validation is to be used for time series prediction, and many methods have been proposed to circumvent these problems. Furthermore, some results on synthetic

Table 2
The parameter grid that is generated for the neural network method, which has two parameters, size and decay. The size is chosen by the model selection procedures from {3, 5, 9}, and the decay from {0.00316, 0.0147, 0.1}.

	Size	Decay
1	3	0.00316
2	5	0.00316
3	9	0.00316
4	3	0.01470
5	5	0.01470
6	9	0.01470
7	3	0.10000
8	5	0.10000
9	9	0.10000

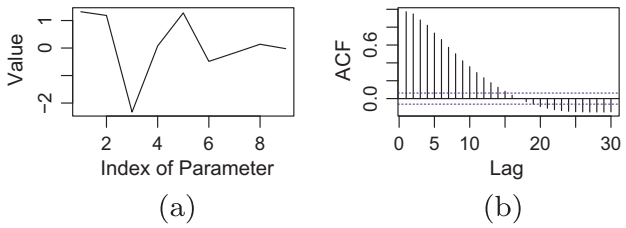

Fig. 2. (a) Generated parameters to simulate an AR or MA model, respectively. In order to obtain a stationary (AR) or invertible (MA) process, the coefficients have to fulfill certain constraints that are respected within our generation process, e.g., the sum of all coefficients has to be smaller than one and the absolute value of the last coefficient has to be smaller than one [19]. Higher coefficients tend to be small. (b) Autocorrelation function of a time series generated using the AR model with the parameters from (a).

and real-world data within bandwidth selection in regression suggest, that standard cross-validation might favor overfitting methods.

But it remains unclear, if these problems have significant consequences in practice when using machine learning techniques in real-world applications. This question is important, as in such a scenario often standard cross-validation is used for evaluation. Another question is, if standard cross-validation could be replaced by another, theoretically better-founded method (last block, blocked cross-validation, etc.), that is equally robust and easy in its use, yielding the same quality of results.

To address those issues we have developed a thorough experimental study with the following objectives:

- To determine if dependency within the data has effects on the cross-validation, e.g., in the way, that the cross-validation procedure systematically underestimates the error. This can be done by comparing randomly chosen evaluation sets to blocked sets.
- To determine if effects of temporal evolution can be found, by comparing evaluations that use data from the end of the series to evaluations that use data from somewhere in between. It has to be noted, that we will consider in this study only (second-order) stationary series, which is common practice in time series forecasting, as stated in Section 3.3.
- To determine if cross-validation yields a more robust error measure, by making full use of the data in the way that it uses all data for training and testing.

In order to cover a broad amount of application situations, the experiments on different model selection procedures will be carried out using machine learning and general regression methods, synthetic and real-world datasets, and various error measures.

4.1. Applied models and algorithms

We have considered a selection of representative machine learning and general regression methods: a support vector regression, a multi-layer perceptron, a linear fitting method, and lasso regression [23]. All methods used are available in packages within the statistical computing language R [40]. We use the implementation of an epsilon support vector regression algorithm with a radial kernel from the LIBSVM [11] (that is wrapped in R by the package `e1071`), and employ a multi-layer perceptron of the `nnet` package [49]. Furthermore, we use lasso regression and the linear fit model present in the R base package. The latter one is a traditional (linear) forecasting method, i.e., an AR model with fixed order that can be applied in the same way as the other regression methods (without the potential need for treatment of missing values during order determination). In the following, the methods will be called `svmRadial`, `nnet`, `lasso`, and `lm`.

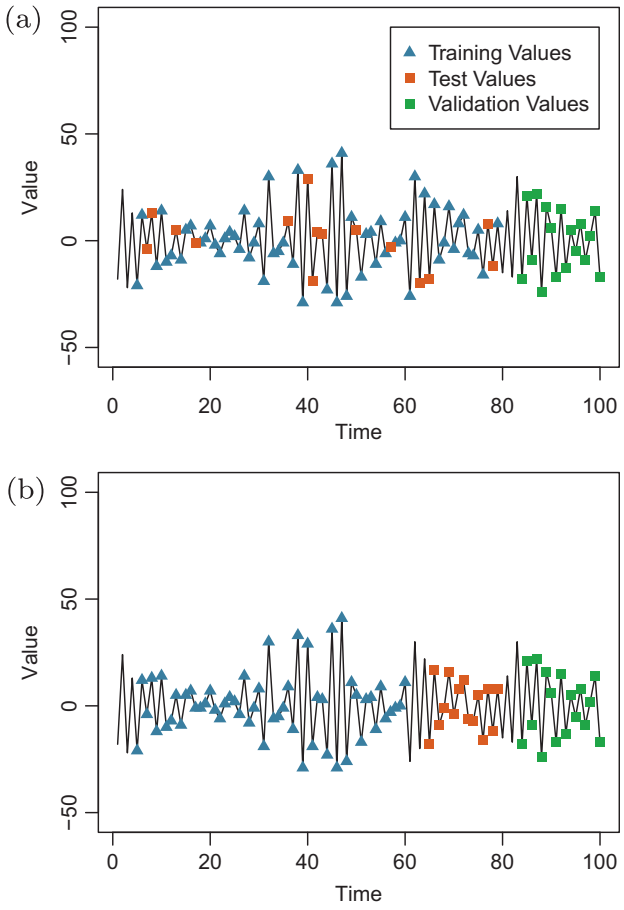

Fig. 3. Within the study, a validation set is withhold from model building and model selection: the out-set. From the other values, the in-set, values are chosen for training and testing according to the model selection procedure, for example (a) one set of standard cross-validation, and (b) last block evaluation. Due to the embedding, values from the very beginning of the series cannot be used as targets, and to make the validation independent of the in-set, the first values of the out-set are omitted as well.

All methods are applied with different parameter configurations. Therefore, for each method a parameter grid is determined empirically (on time series that are available in R, but not used throughout our study, e.g., the "canadian lynx" dataset), which is fixed throughout all of the experiments. The model selection procedures choose for each model and time series the best parameter combination from the grid. The `nnet` has two parameters, size and decay. The model selection procedures choose the size from {3, 5, 9}, and the decay from {0.00316, 0.0147, 0.1}. As an example, the grid is shown in Table 2. The `svmRadial` method has two parameters, cost and gamma, which we defined to be chosen from {0.1, 1, 10, 100, 1000}, and {0.0001, 0.001, 0.01, 0.2}, respectively. The `lasso` has one parameter, fraction, that was chosen from {0.10, 0.36, 0.63, 0.90}. The linear model `lm` has no free parameters to be determined during model selection.

4.2. Benchmarking data

Both synthetic and real-world data were used throughout our study, in order to analyze the evaluation methods' behavior under controlled conditions and in real-world application scenarios. Using the data, three use cases were defined for the experiments, see Section 4.2.3. All data is made available in the KEEL-dataset repository.[2]

4.2.1. Synthetic data

Linear and non-linear time series are simulated for the study. Linear series are generated by an ARMA process:

$$y_t = \alpha_1 y_{t-1} + \alpha_2 y_{t-2} + \cdots + \alpha_k y_{t-k} + w_t - \beta_1 w_{t-1} - \beta_2 w_{t-2} - \cdots - \beta_l w_{t-l}, \tag{8}$$

where l and k are the numbers of lags that influence the current value, and $w_t, \ldots, w_{t-l}$ are i.i.d. Gaussian distributed random variables. In order to obtain a process that has a stationary AR part and an invertible MA part, the coefficients $\alpha_1, \ldots, \alpha_k$, and $\beta_1, \ldots, \beta_l$, have to be chosen in a way that the roots of their characteristic polynomials have an absolute value greater than one, respectively [19]. In analogy to model fitting, where the final model usually is checked by unit root tests for stationarity

[2] http://sci2s.ugr.es/keel/timeseries.php.

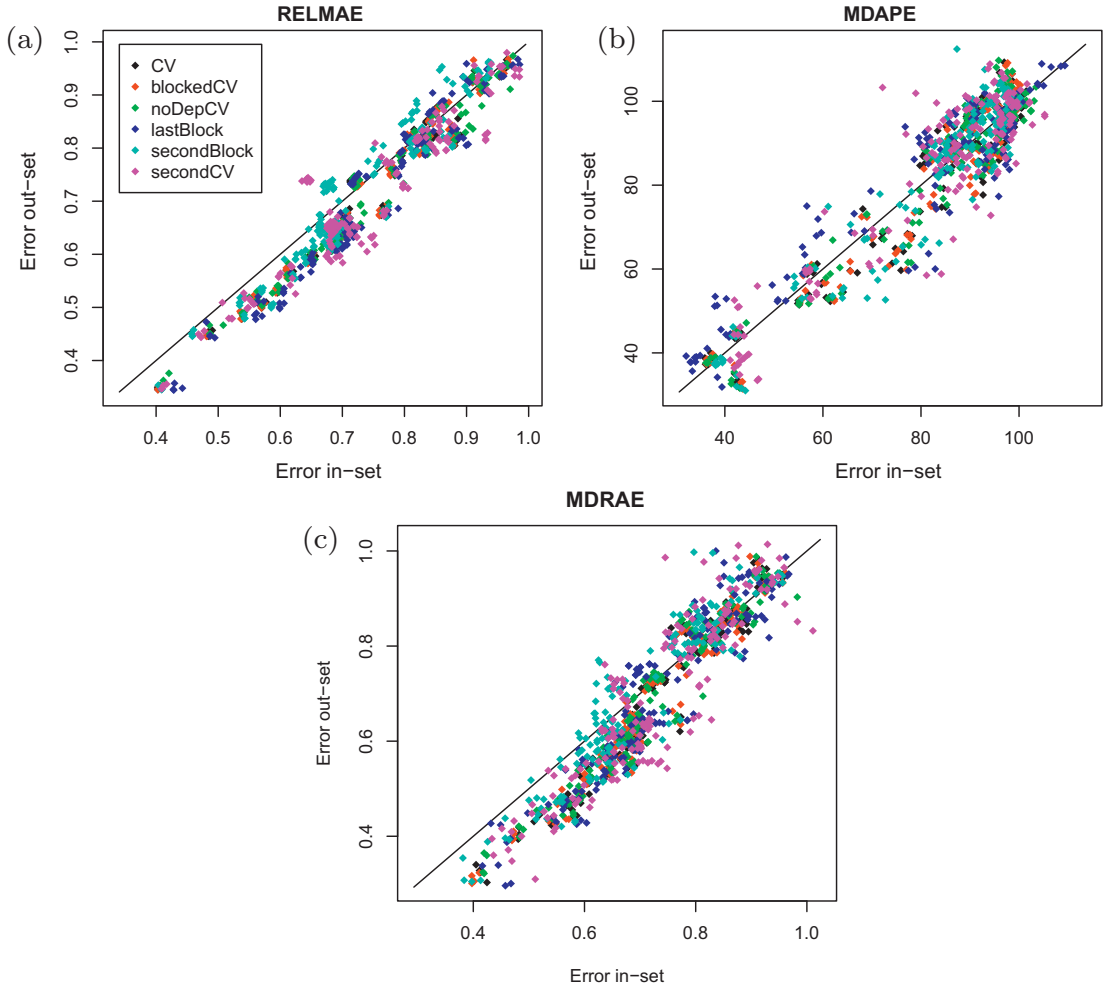

Fig. 4. Point plots for scenario (AS1) (synthetic data with few significant lags), using different error measures. Every point represents the in-set and out-set errors of one method applied to one dataset, with its parameters selected by one of the model selection procedures. As we use four methods and, within this scenario 40 datasets, 160 points are present for every one of the six model selection procedures.

and invertibility (e.g., by using the Dickey–Fuller unit root test [42]), parameters could be generated randomly and then checked for validity. However, if l (or k, respectively) is large finding valid parameters is not trivial (see Fig. 2) and random generation may take a long time, as lots of potential parameters have to be generated and tested. Instead of this generate-and-test approach, we sample the roots of the characteristic polynomials randomly from a uniform distribution in the interval $[-root_{max}, -1.1] \wedge [1.1, root_{max}]$, with $root_{max}$ being a parameter to be chosen. From this roots, the coefficients can then be computed by algebraic standard methods. It has to be noted, that the characteristic polynomials are constrained to have real-valued roots. Initial values are chosen randomly, and the first $2 \cdot max(l, k) + 1$ values of every series are discarded to remove effects of these values. The procedure is used to simulate AR processes by setting the coefficients of the MA part to zero, and vice versa.

Non-linear time series are simulated by a similar procedure, introducing non-linearities in the following way. Parameters for an AR model are generated as described above. Then, for every lag, a non-linear function is chosen randomly from $\cos(x)$, $\sin(x)$, $\arctan(x)$, $\tanh(x)$, and $\exp\left(-\frac{x}{c}\right)$ (where c is a constant value; throughout our experiments we used $c = 10{,}000$). Series are simulated as within the AR model, but with application of the corresponding non-linear function to every $y_{t-1}, \ldots, y_{t-l}$.

4.2.2. Real-world data

Data from the Santa Fe forecasting competition [51] and the NNGC1[3] competition are used.

The Santa Fe competition data set consists of six time series sets, all of them with several thousands of values. Five of these sets are taken from real-world applications, i.e., laser generated data, physiological data, currency exchange rate data, astrophysical data, and audio data. The sixth series is computer generated. Out of this data, we use five time series: the laser generated data, two series of the physiological data, the computer generated series, and a continuous part of the astrophysical

[3] http://www.neural-forecasting-competition.com/datasets.htm.

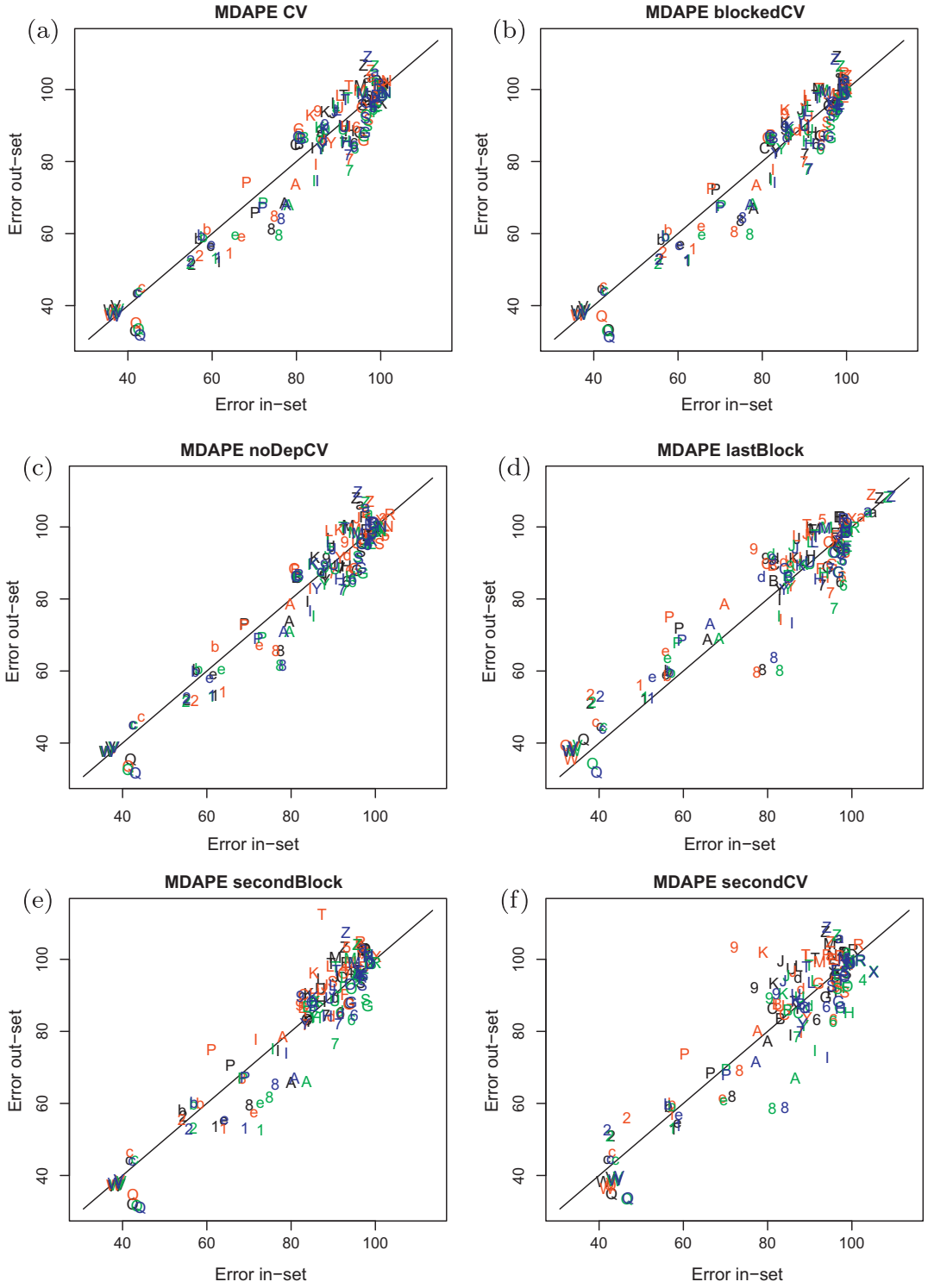

Fig. 5. Point plots for scenario (AS1), using MDAPE as error measure, with single plots for every model selection procedure. Each symbol indicates a different dataset and each color a method. The methods are: (black) `svmRadial`, (red) `nnet`, (blue) `lasso`, (green) `lm`. (For interpretation of the references to colour in this figure legend, the reader is referred to the web version of this article.)

data. The other data is not used as it requires special treatment (time stamps instead of uniform intervals, missing values, learning one concept from many time series), which is not the focus of our work.

From the NNGC1 data, the high-frequency data is used, i.e., weekly, daily, and hourly data. The weekly data are economic data related to the oil industry (gasoline prices, amount of imports to the U.S., etc.). The daily time series are measures of traffic volume passing several tunnels, and the hourly data are average late arrival times and arrival times of airports and metro systems.

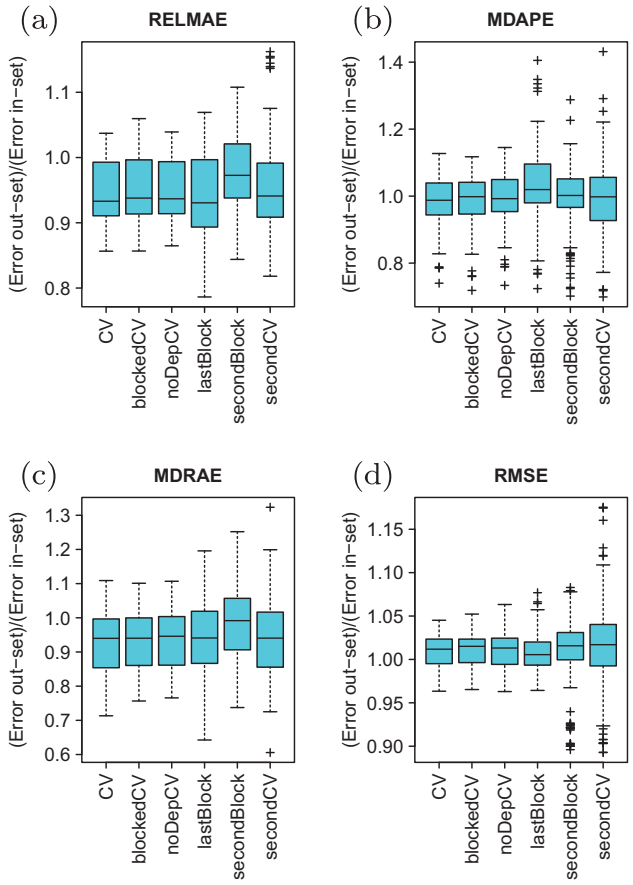

Fig. 6. Box plots of scenario (AS1).

4.2.3. Application scenarios

Application of cross-validation without dependent values is not feasible if relevant lags are too large. So, synthetic and real-world data was used with only few and small relevant lags. Additionally, a simulation study was performed with data containing larger and more lags. Then, cross-validation without dependent values cannot be applied. Thus, three different scenarios were analyzed within the study:

- For application scenario one (AS1), synthetic data with significant autocorrelations in only few and small lags (between one and five) were considered, so that cross-validation without dependent values is applicable. Time series with 1000 values were generated with the simulation methods for AR, MA, ARMA, and non-linear time series presented in Section 4.2.1. The ARMA series were simulated with the same order of the AR and the MA part, and autocorrelations in the first one to five lags were used, i.e., $l = k$, $l, k \in \{1, 2, 3, 4, 5\}$. For every lag, values of 5.0 and 10.0 were used for the $root_{max}$ parameter. So, in total 40 time series were simulated.
- To analyze behavior of the methods on time series that have autocorrelations in more and larger lags, application scenario two (AS2) uses synthetic data with autocorrelations in the last 10–30 lags. Cross-validation without dependent values has to be omitted then. Series were simulated in analogy to scenario (AS1), but with k, $l \in \{10, 15, 20, 25, 30\}$.
- Application scenario three (AS3) considers real-world data (with using four lags, to enable the use of cross-validation without dependent values). All real-world series are tested with the Augmented Dickey–Fuller test [42] for stationarity. The series that do not pass the test are not used. Though the non-stationary series are most likely to show poor performance under cross-validation, as stated in Section 3.3, experiments are likely to be only valid for these particular series. Also, it is likely that the forecasting models would show poor performance as well, possibly indicating that these series would require special preprocessing for non-stationarity, which is not the focus of this study. So, finally a total of 29 real-world time series is used, five from the Santa Fe competition, and 24 from the NNGC1 competition data.

4.3. Data preparation and partitioning

During this study we will use rolling-origin-update evaluation with one-step-ahead forecasting, since this is the most common way such models are used.

As discussed earlier, last block evaluation simulates the typical real-world application scenario of forecasting systems. So, we withhold from every series a percentage p_{ds} of values as "unknown future" for validation. In the following, we will call

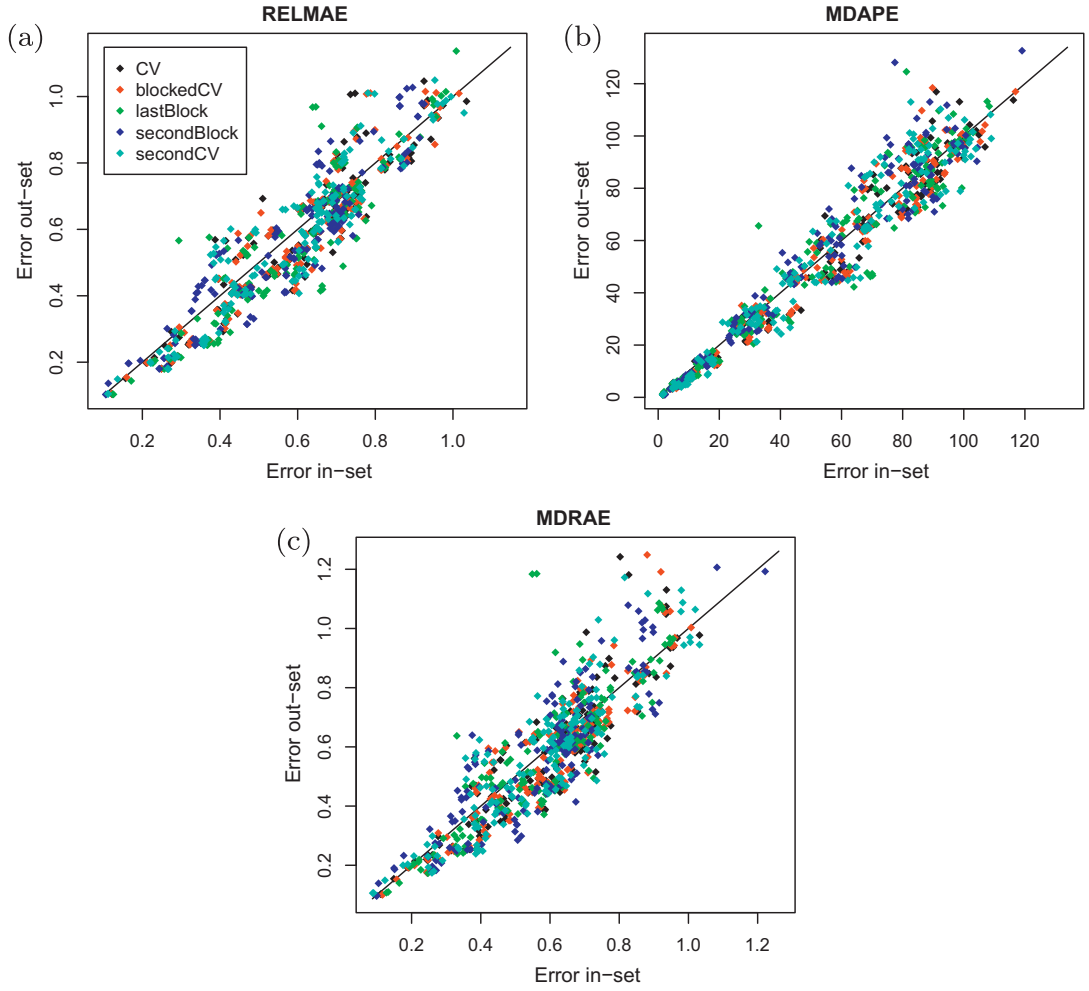

Fig. 7. Point plots for scenario (AS2) (synthetic data with more relevant lags), using different error measures. Every point represents the in-set and out-set errors of one method applied to one dataset, with its parameters selected by one of the model selection procedures. As we use four methods and, within this scenario 40 datasets, 160 points are present for every one of the five model selection procedures.

this dataset the validation set, the out-of-sample set, or shortly the out-set, as it is completely withhold from all other model building and model selection processes. The remaining data accordingly will in the following be called the in-set. The out-set is chosen in such a way that the later in-set evaluation can be performed without problems. That is, values from its beginning are removed according to the number of lags that later will be used for training, so that the out-set is independent of the other data, and it is chosen in a way that the amount of data remaining in the in-set can be partitioned equally, e.g., if 5-fold cross-validation is to be used, the remaining amount of data will be divisible by five. The process is illustrated in Fig. 3. Throughout our experiments, we use $p_{ds} = 0.8$, i.e., 20% of the data is used as out-set.

The in-set data is used for model building and model selection in the following way: the lags l_{ds} to be used for forecasting are chosen. For synthetic series, the lags are known, as they were specified during data generation, for the real-world data they have to be estimated. To have the possibility to use all model selection procedures (especially the procedure that removes dependent values, see Section 4.4), four lags are used for real-world series. This seems feasible, as the focus of this study does not lie in the actual performance of the methods, but in the performance of the model selection procedures. Then, the data is embedded as already discussed in Section 3.2.

4.4. Compared model selection procedures

From the embedded versions of the data, training and test sets are generated according to different model selection strategies. The procedures are then used for choosing the parameters of each method, as the model selection procedures are applied with each method and each parameter configuration, and for each method, the parameter set that produced the minimal error within the model selection procedure is chosen. This error is furthermore used as an estimate of the overall error, and is later compared to the errors that the methods (with these parameter configurations) produce on the out-set. We consider six different model selection strategies, named `CV`, `blockedCV`, `noDepCV`, `lastBlock`, `secondBlock`, and `secondCV`.

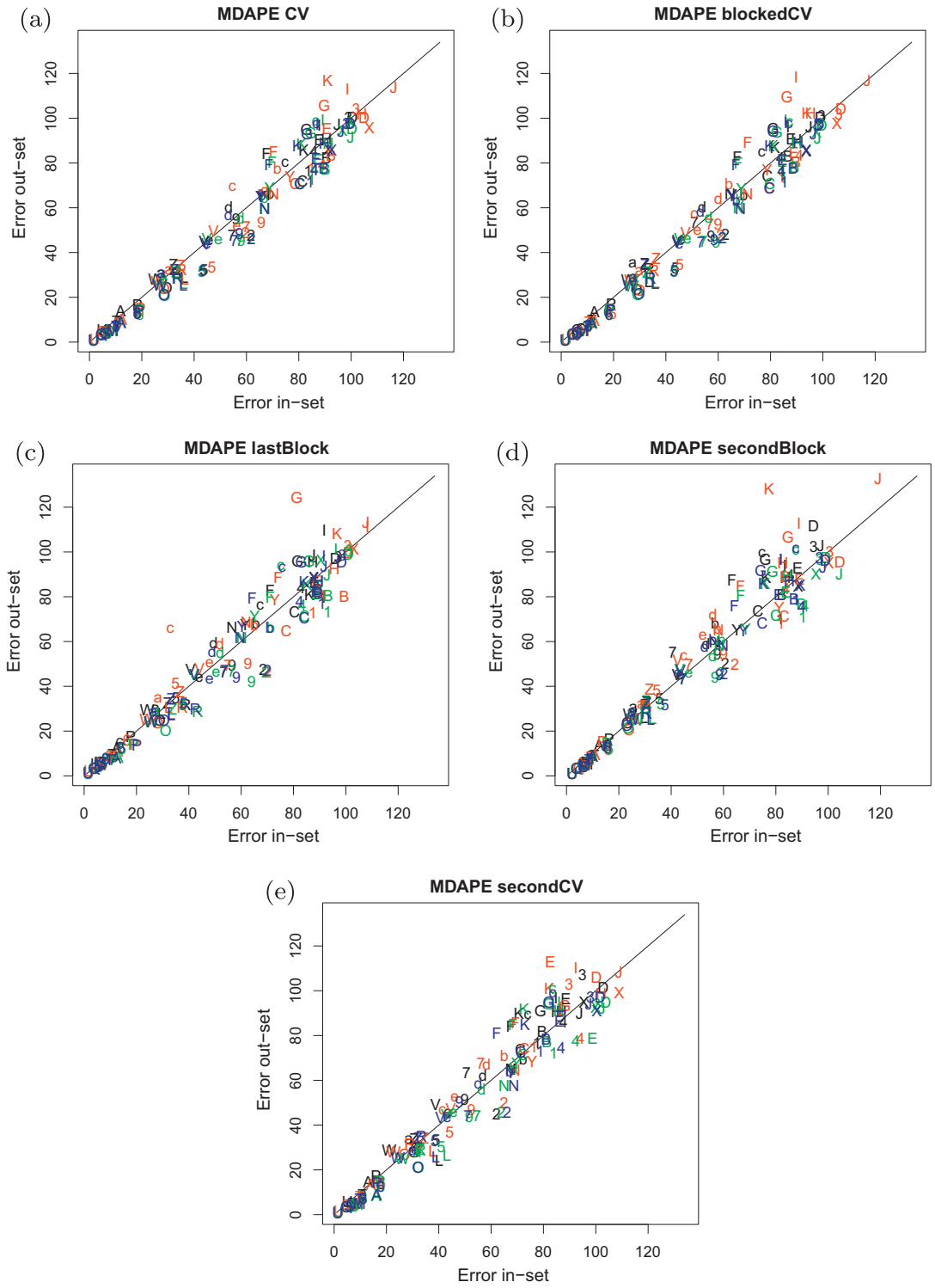

Fig. 8. Point plots for scenario (AS2), using MDAPE as error measure, with single plots for every model selection procedure. Each symbol indicates a different dataset and each color a method. The methods are: (black) `svmRadial`, (red) `nnet`, (blue) `lasso`, (green) `lm`. (For interpretation of the references to colour in this figure legend, the reader is referred to the web version of this article.)

For `CV`, standard 5-fold cross-validation is applied, i.e., the embedded data is partitioned randomly into five sets, and within five turns every set is used as test set, while the other sets are used for training. For `blockedCV`, 5-fold cross-validation is applied on data that is not partitioned randomly, but sequentially into five sets. So, the problem of dependent values is resolved (except for some values at the borders of the blocks, which can be removed). The problem that a system evolving over time might have generated the time series and render results of cross-validation incorrect remains. During `noDepCV`, 5-fold cross-validation without the dependent values is applied: the sets generated for `CV` are used, but according to the lags used for embedding, dependent values are removed from the training set. As stated earlier, depending on the lags used a lot

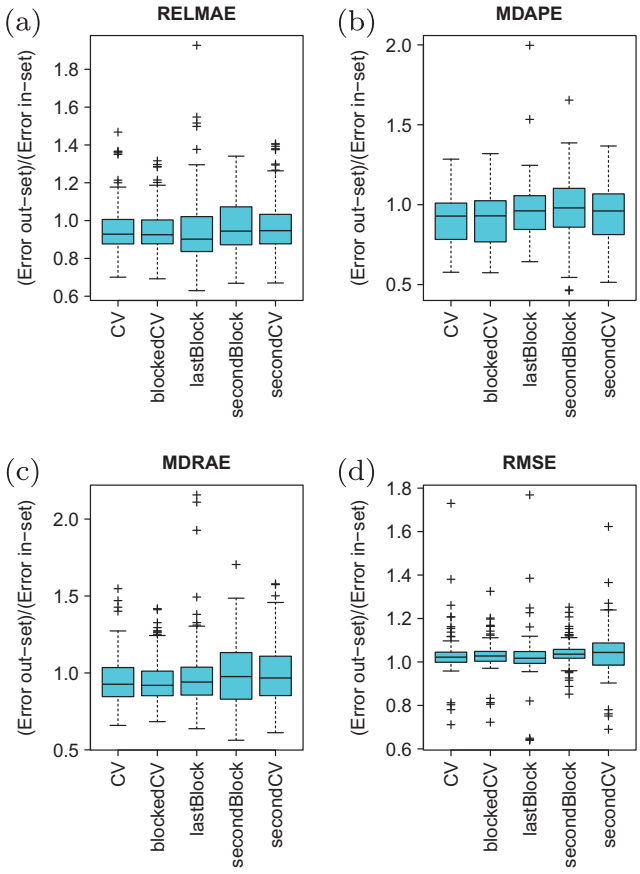

Fig. 9. Box plots of scenario (AS2).

of values have to be removed, so that this model selection procedure only can be applied if the amount of lags is not large compared to the number of cross-validation subsets, i.e., in AS1 and AS3. The evaluation type `lastBlock` uses only the last set of `blockedCV` for evaluation. To study in more detail the effects of using a connected evaluation set or random chosen values, and the effect of using data from the end or from somewhere in between, `secondBlock` evaluation uses not the last but the second block of `blockedCV` for evaluation, and `secondCV` uses only the second subset of `CV`. The different types of data sampling within the model selection procedures illustrates Fig. 3.

The other methods discussed in Section 3.4, namely forward validation and APE, cannot be used, as no recalibration of the models is performed within our experiments.

4.5. Computed error measures

Though we do not compute average measures over multiple time series, as the study employs various different time series, only the use of scale-independent error measures is feasible in order to achieve results that can be compared among themselves. As some of the series contain zero values, measures based on percentage errors or on scaled errors are only applicable, if a robust averaging is used, e.g., the median. So, we calculate the MDAPE and MDRAE. Furthermore, as for every time series and the horizon used many values are available, the use of relative measures (e.g., RELMAE) is possible. Within these, we use the naïve forecast (i.e., the last known value) as a benchmark. It has to be noted, that by using this benchmark, a difference between blocked and unblocked validation modes exists, as during unblocked modes the naïve forecasts might also be present in the training set as lagged and target values, whereas this is not the case for blocked validation modes (if the borders of the sets are removed).

Furthermore, the additional validation procedure we apply enables the scaling of the error directly by the error on the out-set. For this purpose, also scale-dependent measures such as the RMSE can be used.

4.6. Plots and statistical tests

The in-set error, estimated by the model selection procedure, is compared to the error on the out-set. If the model selection procedure produces a good estimate for the error the two errors should be very similar. Therefore, we analyze plots of points $(E_{in-set}, E_{out-set})$. If the errors are equal, these points all lie on a line with origin zero and gradient one. In the following, we call this type of evaluation *point plots*.

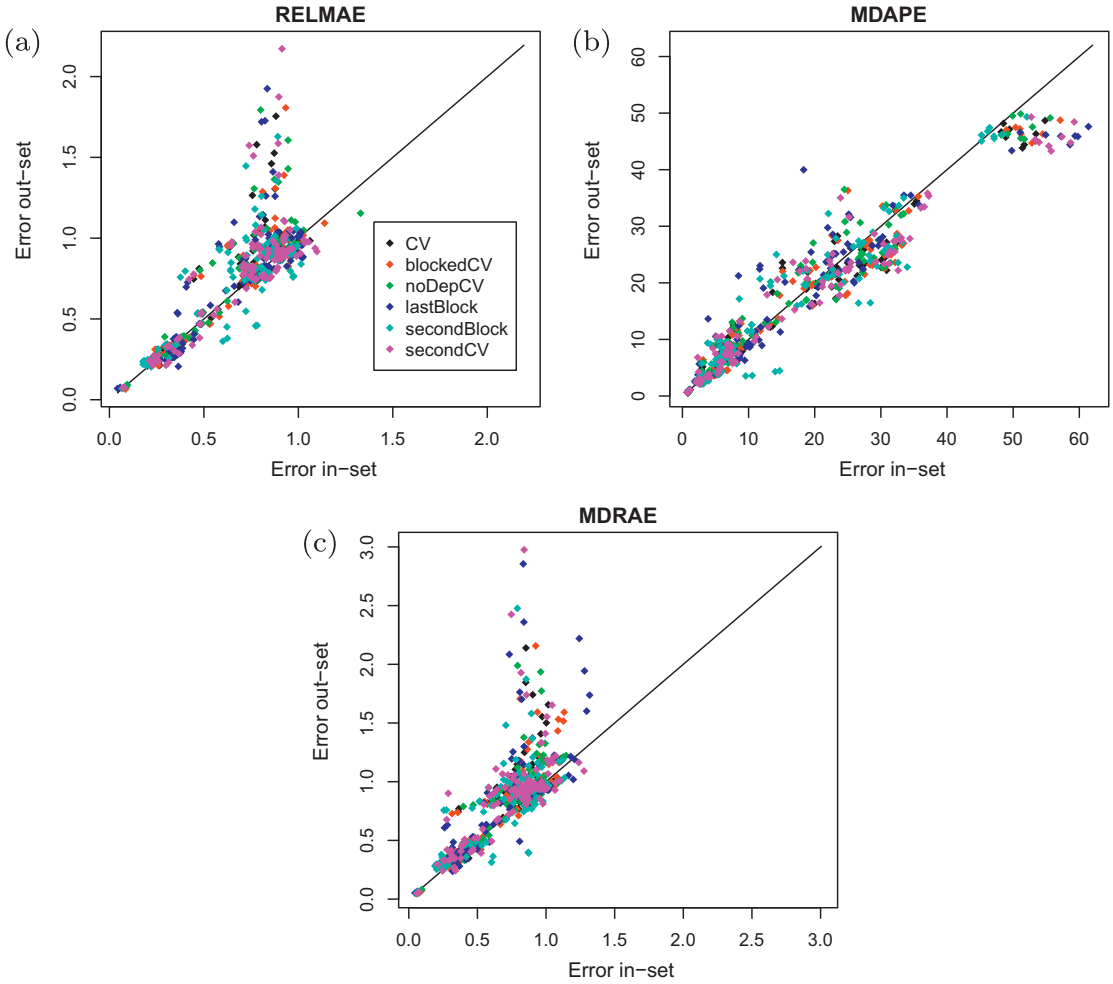

Fig. 10. Point plots for scenario (AS3) (real-world data), using different error measures. Every point represents the in-set and out-set errors of one method applied to one dataset, with its parameters selected by one of the model selection procedures. As we use four methods and, within this scenario 29 datasets, 116 points are present for every one of the six model selection procedures.

Additionally to the point plots, we analyze box-and-whisker plots containing directly the value of the quotient ($E_{out-set}/E_{in-set}$), which is especially interesting with the use of scale-dependent measures like the RMSE, as with using the quotient a normalization takes place, so that the results are comparable.

Statistical significance of the results is explored with the following non-parametric tests: the Friedman test in its implementations of García et al. [25] is used to determine if the distributions of the quotient ($E_{out-set}/E_{in-set}$) for the model selection procedures differ in their location parameter (the median). And the Fligner–Killeen test [17] (that is available in R) is used to determine if these distributions differ in their dispersion.

5. Experimental results and analysis

The complete results can be found at http://sci2s.ugr.es/dicits/papers/CV-TS. In the following, a selection of the results is presented.

5.1. Plots of the results

Fig. 4 shows point plots for scenario AS1, using different error measures. It can be observed that the RELMAE yields a less scattered distribution than MDAPE and MDRAE. The in-set error tends to overestimate the out-set error, especially when using relative measures, i.e., RELMAE or MDRAE. No systematical difference between different model selection procedures can be determined in this plot. To further examine the different model selection procedures, Fig. 5 shows the results of Fig. 4 in more detail, only for the MDAPE measure, but with the results of every model selection procedure in a different plot. Fig. 6 shows the results of Fig. 5 as a box plot, again including all error measures of Fig. 4, and furthermore including the RMSE. Figs. 5 and 6 show graphically that the methods using only a single set for evaluation, i.e., `lastBlock`, `secondBlock`, and `secondCV` lead in general to more disperse, less robust results. Few differences between the model selection procedures

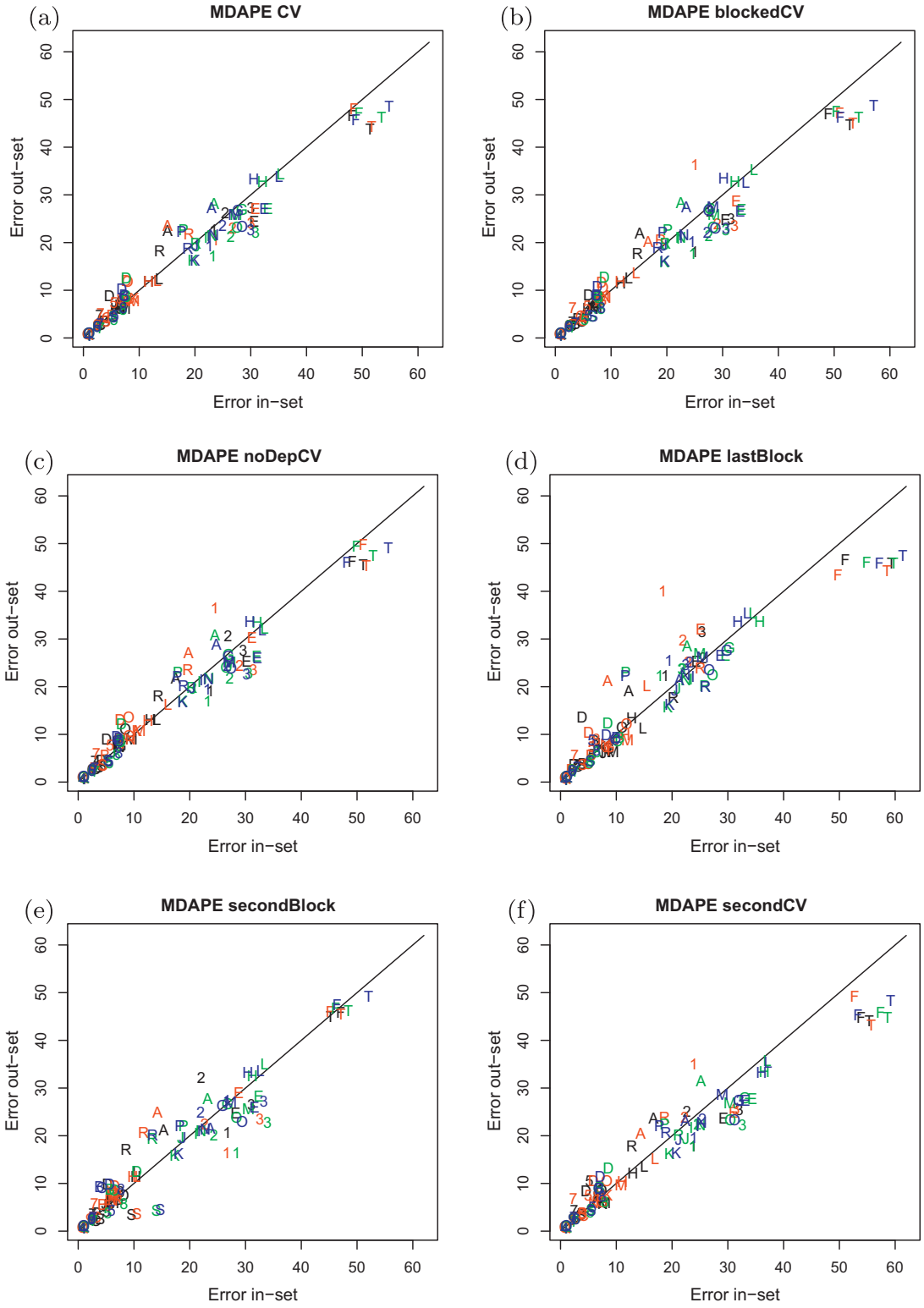

Fig. 11. Point plots for scenario (AS3), using MDAPE as error measure, with single plots for every model selection procedure. Each symbol indicates a different dataset and each color a method. The methods are: (black) `svmRadial`, (red) `nnet`, (blue) `lasso`, (green) `lm`. (For interpretation of the references to colour in this figure legend, the reader is referred to the web version of this article.)

of the form of a bias are present. Furthermore, the difference is not in the way expected w.r.t. the theoretical problems of cross-validation (due to the dependencies within the values, an underestimation when applying cross-validation could occur).

Within the scenario AS2, `noDepCV` is not applicable any more. Point plots and box plots analogous to the plots of scenario AS1 are shown in Figs. 7–9. The results confirm the results of scenario AS1.

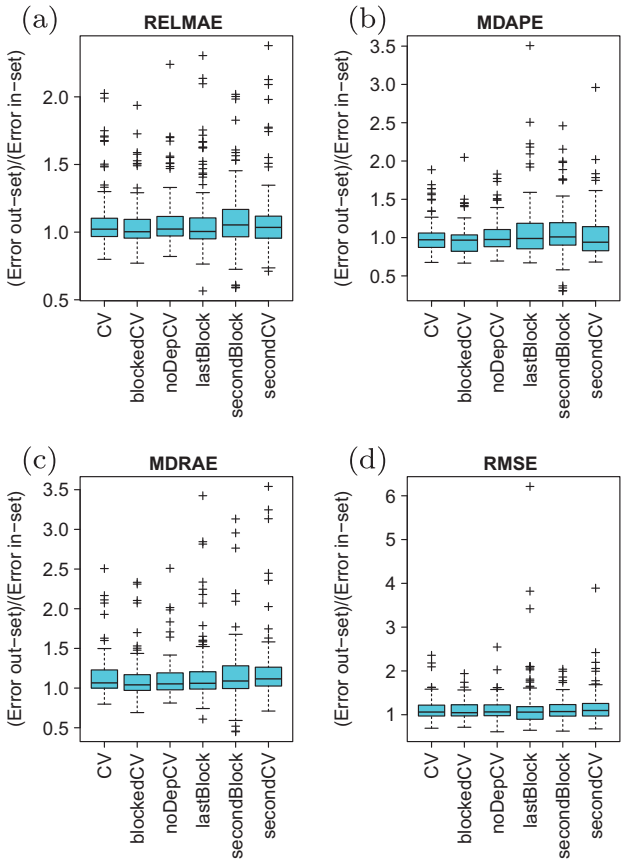

Fig. 12. Box plots of scenario (AS3).

Figs. 10–12 show the results of scenario AS3, where real-world data is used. The results on real-world data basically confirm the simulation results, but contain more noise. For real-world data *a priori* it is unknown, which delayed values are relevant, and no sophisticated methods to solve such problems were used within our study, so on some datasets the methods perform worse than the naïve forecast. This leads to the "traces" in Fig. 10, when measures that employ a benchmark are used.

5.2. Some results of parameter selection

Tables 3 and 4 show examples in scenario AS3 for the differences in choosing the parameters by distinct model selection procedures, and error measures, respectively. The examples illustrate that both the model selection procedure and the error measure used influences in the choice of the parameter set.

5.3. Statistical evaluation

The Friedman test shows high significance ($p < 0.001$) in the difference of medians for the groups of all model selection procedures, for all combinations of scenarios and error measures, except for AS3 with RMSE, where a p-value of 0.126 was obtained. As the data results from different methods (e.g., not from a test and a control sample), we cannot expect the resulting points to have exactly the same distributions. So, the statistical significance in the difference shows merely that the amount of data points is high enough to obtain a stable result. As a statistical test does not express the size or relevance of the difference found, their consequences for the practice are often unclear [53]. Also, in the forecasting community the use of statistical tests in general is discussed controversially [4,5]. Therefore, we perform an analysis of the medians and their differences in Table 5. The table shows that with respect to under or overestimation of the error, no difference between the model selection procedures can be found. The differences in the accuracy with which the in-set error predicts the out-set error are small, and vary with the characteristics of the data and the error measures. So, e.g., choice of the error measure seems more relevant than choice of the model selection procedure.

Table 6 shows results of the Fligner–Killeen test. Though the difference between `lastBlock` and the cross-validation procedures is not always statistically significant, the table clearly shows the trend that the difference between the last block evaluation and the cross-validation methods is bigger than difference among the cross-validation methods.

Table 3
Parameter sets that are chosen by the different model selection procedures, using MDAPE, for the laser data of the Santa Fe competition. The numbers represent the index of the parameter set within the parameter grid, as shown in Table 2 for `nnet`. For `svmRadial`, the relevant sets 18, 19, and 20 all define the parameter gamma to be 0.2, and the parameter cost to be 10, 100, and 1000, respectively. In the method `lasso`, 4 means the only present parameter fraction is set to 0.9.

	svmRadial	nnet	lasso
CV	19	4	4
blockedCV	19	8	4
noDepCV	20	4	4
lastBlock	19	3	4
secondBlock	18	7	4
secondCV	18	4	4

Table 4
Parameter sets that are chosen by the different error measures, using standard cross-validation, for the laser data of the Santa Fe competition. The numbers represent the index in the parameter grid, as in Table 3.

	svmRadial	nnet	lasso
RELMAE	19	3	4
MDAPE	19	4	4
MDRAE	19	4	3
RMSE	19	3	4

Table 5
Medians and differences in the median. The columns are: CV, bCV, and lB: Median of $(E_{out-set}/E_{in-set})$ values for the procedures `CV`, `blockedCV`, and `lastBlock`, diminished by one. The optimal ratio of the errors is one (which would result in a zero in the table), as then the in-set error equals the out-set error, and hence is a good estimate. Negative values in the table indicate a greater in-set error, i.e., the out-set error is overestimated. A positive value, on the contrary, indicates underestimation. CV-lB, CV-bCV, and bCV-lB: differences of the absolute values of CV, bCV, and lB. A negative value indicates that the minuend in the difference leads to a value nearer to one, that is, to a better estimate of the error.

		CV	bCV	lB	CV-lB	CV-bCV	bCV-lB
RELMAE	AS1	−0.067	−0.062	−0.069	−0.002	0.005	−0.007
	AS2	−0.072	−0.075	−0.098	−0.026	−0.003	−0.023
	AS3	0.022	0.003	0.005	0.017	0.019	−0.002
MDAPE	AS1	−0.012	−0.002	0.020	−0.007	0.011	−0.018
	AS2	−0.071	−0.070	−0.039	0.033	0.001	0.031
	AS3	−0.028	−0.033	−0.012	0.017	−0.005	0.022
MDRAE	AS1	−0.060	−0.060	−0.059	0.001	0.000	0.001
	AS2	−0.074	−0.081	−0.059	0.014	−0.007	0.021
	AS3	0.065	0.041	0.060	0.005	0.025	−0.020
RMSE	AS1	0.012	0.015	0.006	0.006	−0.003	0.010
	AS2	0.022	0.027	0.017	0.005	−0.005	0.010
	AS3	0.061	0.046	0.060	0.002	0.015	−0.013

Table 6
p-Values of the Fligner test. First column: Fligner test for differences in variance, applied to the group of all model selection procedures (6 procedures for AS1 and AS3, and 5 procedures for AS2). Columns 2–4: Tests of interesting pairs of methods (without application of a post hoc procedure).

		All	CV, lB	bCV, lB	CV, bCV
RELMAE	AS1	0.000	0.002	0.000	0.307
	AS2	0.010	0.119	0.087	0.849
	AS3	0.005	0.115	0.050	0.618
MDAPE	AS1	0.000	0.018	0.004	0.444
	AS2	0.108	0.943	0.424	0.529
	AS3	0.005	0.007	0.013	0.784
MDRAE	AS1	0.000	0.020	0.007	0.701
	AS2	0.001	0.346	0.369	0.983
	AS3	0.054	0.256	0.163	0.737
RMSE	AS1	0.000	0.448	0.508	0.707
	AS2	0.000	0.230	0.047	0.433
	AS3	0.078	0.028	0.008	0.644

6. Conclusions

In this paper, we reviewed the methodology of evaluation in traditional forecasting and in regression and machine learning methods used for time series. We observed that with the use of general regression methods and machine learning techniques, also techniques usually used for their evaluation (especially cross-validation) are used within the time series problems. This raises theoretical concerns, as the data contains dependencies and time-evolving effects may occur. On the other hand, in traditional forecasting only the last part of every series is typically used for evaluation, thus not exhausting the available information.

Because of the shortcomings that the commonly used methods have, various other methods have been proposed in the literature. We grouped these in Section 3.4 into methods based on the last block, methods that use non-dependent cross-validation, and blocked cross-validation methods.

In order to analyze the shortcomings of the popular methods and to evaluate the potential benefit from the use of other methods in common application situations, we performed a thorough empirical study. It includes the comparison of six model selection procedures (among others cross-validation and last block evaluation) on forecasts of four different methods for synthetic and real-world time series.

Using standard 5-fold cross-validation, no practical effect of the dependencies within the data could be found, regarding whether the final error is under- or overestimated. On the contrary, last block evaluation tends to yield less robust error measures than cross-validation and blocked cross-validation. The non-dependent cross-validation procedure also yields robust results, but might lead to a waste of data and therewith is not applicable in many cases.

Regarding time-evolving effects, no differences could be found, as using the last block and using a block taken from somewhere within the data (we used the second block of the blocked cross-validation) showed a similar behavior. This is not surprising, as we limited the study to stationary time series. However, assuming stationary time series is a common and reasonable assumption in many applications.

Though no practical problems with standard cross-validation could be found, we suggest the use of blocked cross-validation, together with an adequate control for stationarity, since it makes full use of all available information both for training and testing, thus yielding a robust error estimate. And theoretical problems are solved in a practical manner, as assuming stationarity and approximate independence after a certain amount of lags coincides with common assumptions of many application scenarios.

Acknowledgements

This work was supported in part by the Spanish Ministry of Science and Innovation (MICINN) under Project TIN-2009-14575. C. Bergmeir holds a scholarship from the Spanish Ministry of Education (MEC) of the "Programa de Formación del Profesorado Universitario (FPU)".

References

[1] N. Amjady, F. Keynia, Application of a new hybrid neuro-evolutionary system for day-ahead price forecasting of electricity markets, Applied Soft Computing Journal 10 (3) (2010) 784–792.
[2] R.D.A. Araujo, Swarm-based translation-invariant morphological prediction method for financial time series forecasting, Information Sciences 180 (24) (2010) 4784–4805.
[3] S. Arlot, A. Celisse, A survey of cross-validation procedures for model selection, Statistics Surveys 4 (2010) 40–79.
[4] J.S. Armstrong, Significance tests harm progress in forecasting, International Journal of Forecasting 23 (2) (2007) 321–327.
[5] J.S. Armstrong, Statistical significance tests are unnecessary even when properly done and properly interpreted: Reply to commentaries, International Journal of Forecasting 23 (2) (2007) 335–336.
[6] J.L. Aznarte, J.M. Benítez, Equivalences between neural-autoregressive time series models and fuzzy systems, IEEE Transactions on Neural Networks 21 (9) (2010) 1434–1444.
[7] S.D. Balkin, J.K. Ord, Automatic neural network modeling for univariate time series, International Journal of Forecasting 16 (4) (2000) 509–515.
[8] G. Box, G. Jenkins, Time Series Analysis: Forecasting and Control, Holden-Day, 1970.
[9] Prabir Burman, Edmond Chow, Deborah Nolan, A cross-validatory method for dependent data, Biometrika 81 (2) (1994) 351–358.
[10] P.S. Carmack, W.R. Schucany, J.S. Spence, R.F. Gunst, Q. Lin, R.W. Haley, Far casting cross-validation, Journal of Computational and Graphical Statistics 18 (4) (2009) 879–893.
[11] Chih-Chung Chang, Chih-Jen Lin, LIBSVM: a library for support vector machines, 2001 <http://www.csie.ntu.edu.tw/cjlin/libsvm>.
[12] C. Chatfield, Model uncertainty and forecast accuracy, Journal of Forecasting 15 (7) (1996) 495–508.
[13] B.-J. Chen, M.-W. Chang, C.-J. Lin, Load forecasting using support vector machines: a study on eunite competition 2001, IEEE Transactions on Power Systems 19 (4) (2004) 1821–1830.
[14] Y. Chen, B. Yang, Q. Meng, Y. Zhao, A. Abraham, Time-series forecasting using a system of ordinary differential equations, Information Sciences 181 (1) (2011) 106–114.
[15] Zhuo Chen, Yuhong Yang, Assessing forecast accuracy measures. Technical Report 2004–10, Iowa State University, Department of Statistics & Statistical Laboratory, 2004.
[16] R.B. Cleveland, W.S. Cleveland, J.E. McRae, I. Terpenning, Stl: a seasonal-trend decomposition procedure based on loess (with discussion), Journal of Official Statistics 6 (1990) 3–73.
[17] W.J. Conover, Mark E. Johnson, Myrle M. Johnson, Comparative study of tests for homogeneity of variances, with applications to the outer continental shelf bidding data, Technometrics 23 (4) (1981) 351–361.
[18] S.F. Crone, M. Hibon, K. Nikolopoulos, Advances in forecasting with neural networks? Empirical evidence from the nn3 competition on time series prediction, International Journal of Forecasting 27 (3) (2011) 635–660.
[19] Jonathan D. Cryer, Kung-Sik Chan, Time Series Analysis With Applications in R, Springer, 2008. ISBN 978-0-387-75958-6.
[20] C.G. da Silva, Time series forecasting with a non-linear model and the scatter search meta-heuristic, Information Sciences 178 (16) (2008) 3288–3299.

[21] J.G. De Gooijer, R.J. Hyndman, 25 Years of time series forecasting, International Journal of Forecasting 22 (3) (2006) 443–473.
[22] G. Deco, R. Neuneier, B. Schürmann, Non-parametric data selection for neural learning in non-stationary time series, Neural Networks 10 (3) (1997) 401–407.
[23] B. Efron, T. Hastie, I. Johnstone, R. Tibshirani, H. Ishwaran, K. Knight, J.-M. Loubes, P. Massart, D. Madigan, G. Ridgeway, S. Rosset, J.I. Zhu, R.A. Stine, B.A. Turlach, S. Weisberg, T. Hastie, I. Johnstone, R. Tibshirani, Least angle regression, Annals of Statistics 32 (2) (2004) 407–499.
[24] M. Gan, H. Peng, X. Peng, X. Chen, G. Inoussa, A locally linear rbf network-based state-dependent ar model for nonlinear time series modeling, Information Sciences 180 (22) (2010) 4370–4383.
[25] S. García, A. Fernández, J. Luengo, F. Herrera, Advanced nonparametric tests for multiple comparisons in the design of experiments in computational intelligence and data mining: experimental analysis of power, Information Sciences 180 (10) (2010) 2044–2064.
[26] P. Goodwin, R. Lawton, On the asymmetry of the symmetric mape, International Journal of Forecasting 15 (4) (1999) 405–408.
[27] C. Hamzaçebi, Improving artificial neural networks' performance in seasonal time series forecasting, Information Sciences 178 (23) (2008) 4550–4559.
[28] Jeffrey D. Hart, Automated kernel smoothing of dependent data by using time series cross- validation, Journal of the Royal Statistical Society: Series B (Methodological) 56 (3) (1994) 529–542.
[29] J.S.U. Hjorth, Computer Intensive Statistical Methods, Validation, Model Selection and Bootstrap, Chapman and Hall, 1994.
[30] Urban Hjorth, Model selection and forward validation, Scandinavian Journal of Statistics 9 (2) (1982) 95–105.
[31] R.J. Hyndman, A.B. Koehler, Another look at measures of forecast accuracy, International Journal of Forecasting 22 (4) (2006) 679–688.
[32] A. Inoue, L. Kilian, On the selection of forecasting models, Journal of Econometrics 130 (2) (2006) 273–306.
[33] T.Y. Kim, K.J. Oh, C. Kim, J.D. Do, Artificial neural networks for non-stationary time series, Neurocomputing 61 (1–4) (2004) 439–447.
[34] R. Kunst, Cross validation of prediction models for seasonal time series by parametric bootstrapping, Austrian Journal of Statistics 37 (2008) 271–284.
[35] Robert M. Kunst, Adusei Jumah, Toward a theory of evaluating predictive accuracy, Economics Series, vol. 162, Institute for Advanced Studies, 2004.
[36] S. Makridakis, M. Hibon, The m3-competition: results, conclusions and implications, International Journal of Forecasting 16 (4) (2000) 451–476.
[37] Allan D.R. McQuarrie, Chih-Ling Tsai, Regression and Time Series Model Selection, World Scientific Publishing, 1998.
[38] J. Opsomer, Y. Wang, Y. Yang, Nonparametric regression with correlated errors, Statistical Science 16 (2) (2001) 134–153.
[39] N.M. Pindoriya, S.N. Singh, S.K. Singh, An adaptive wavelet neural network-based energy price forecasting in electricity markets, IEEE Transactions on Power Systems 23 (3) (2008) 1423–1432.
[40] R Development Core Team, R: A Language and Environment for Statistical Computing. R Foundation for Statistical Computing, Vienna, Austria, 2009, ISBN 3-900051-07-0.
[41] J. Racine, Consistent cross-validatory model-selection for dependent data: hv-block cross-validation, Journal of Econometrics 99 (1) (2000) 39–61.
[42] S.E. Said, D.A. Dickey, Testing for unit roots in autoregressive-moving average models of unknown order, Biometrika 71 (1984) 599–607.
[43] J. Shao, An asymptotic theory for linear model selection, Statistica Sinica 7 (1997) 221–264.
[44] Jun Shao, Linear model selection by cross-validation, Journal of the American Statistical Association 88 (422) (1993) 486–494.
[45] T.A.B. Snijders, On cross-validation for predictor evaluation in time series, in: T.K. Dijkstra (Ed.), On Model Uncertainty and its Statistical Implications, 1988, pp. 56–69.
[46] M. Stone, Cross-validatory choice and assessment of statistical predictions, Journal of the Royal Statistical Society: Series B (Methodological) 36 (2) (1974) 111–147.
[47] L.J. Tashman, Out-of-sample tests of forecasting accuracy: an analysis and review, International Journal of Forecasting 16 (4) (2000) 437–450.
[48] H. Tong, Non-Linear Time Series: A Dynamical System Approach, Clarendon Press, Oxford, 1990.
[49] W.N. Venables, B.D. Ripley, Modern Applied Statistics with S, fourth ed., Springer, New York, 2002. ISBN 0-387-95457-0.
[50] E.-J. Wagenmakers, P. Grünwald, M. Steyvers, Accumulative prediction error and the selection of time series models, Journal of Mathematical Psychology 50 (2) (2006) 149–166.
[51] A.S Weigend, N.A. Gershenfeld, (Eds.), Time Series Prediction: Forecasting the Future and Understanding the Past, 1994.
[52] G.P. Zhang, A neural network ensemble method with jittered training data for time series forecasting, Information Sciences 177 (23) (2007) 5329–5346.
[53] S.T. Ziliak, D.N. McCloskey, Size matters: the standard error of regressions in the american economic review, Journal of Socio-Economics 33 (5) (2004) 527–546.

On the usefulness of cross-validation for directional forecast evaluation

C. Bergmeir, M. Costantini, and J.M. Benítez. On the usefulness of cross-validation for directional forecast evaluation. Journal of Forecasting, 2013, (submitted).

- Status: **Submitted**
- Impact Factor (JCR 2011): 0.930
- Subject category:
  - MANAGEMENT (101/168 Q3)
  - PLANNING & DEVELOPMENT (27/54 Q3)

On the usefulness of cross-validation for directional forecast evaluation

Journal:	*Journal of Forecasting*
Manuscript ID:	FOR-13-0011
Wiley - Manuscript type:	Research Article
Date Submitted by the Author:	22-Jan-2013

Complete List of Authors:	Bergmeir, Christoph; University of Granada, Department of Computer Sciences and Artificial Intelligence Costantini, Mauro; Brunel University, Department of Economics and Finance Benítez, José; University of Granada, Department of Computer Science and Artificial Intelligence

Keywords:	blocked cross-validation, out-of-sample evaluation, forecast directional accuracy

Abstract:	The purpose of this work is to investigate the usefulness of a predictor evaluation framework which combines a blocked cross-validation scheme with directional accuracy measures. The advantage of using a blocked cross-validation scheme with respect to the standard out-of-sample procedure is that cross-validation yields more precise error estimates since it makes full use of the data. In order to quantify the gain in precision when directional accuracy measures are considered, we provide a Monte Carlo analysis using univariate and multivariate models. The experiments indicate that more precise estimates are obtained with the blocked cross-validation procedure. An application is carried out on forecasting UK interest rate for illustration purposes. The results show that the cross-validation scheme has considerable advantages over the standard out-of-sample evaluation procedure as it makes possible to compensate for the loss of information due to the binary nature of the directional accuracy measures.

On the usefulness of cross-validation for directional forecast evaluation

Christoph Bergmeir[1], Mauro Costantini[2], and José M. Benítez[1*]

[1]Department of Computer Science and Artificial Intelligence, E.T.S. de Ingenierías Informática y de Telecomunicación, University of Granada, Spain.
[2]Department of Economics and Finance, Brunel University, United Kingdom

Abstract

The purpose of this work is to investigate the usefulness of a predictor evaluation framework which combines a blocked cross-validation scheme with directional accuracy measures. The advantage of using a blocked cross-validation scheme with respect to the standard out-of-sample procedure is that cross-validation yields more precise error estimates since it makes full use of the data. In order to quantify the gain in precision when directional accuracy measures are considered, we provide a Monte Carlo analysis using univariate and multivariate models. The experiments indicate that more precise estimates are obtained with the blocked cross-validation procedure.
An application is carried out on forecasting UK interest rate for illustration purposes. The results show that the cross-validation scheme has considerable advantages over the standard out-of-sample evaluation procedure as it makes possible to compensate for the loss of information due to the binary nature of the directional accuracy measures.

KEY WORDS Blocked cross-validation; out-of-sample evaluation; forecast directional accuracy; Monte Carlo analysis; linear models.

*Corresponding author. DECSAI, ETSIIT, UGR, C/ Periodista Daniel Saucedo Aranda s/n, 18071 - Granada, Spain, E-mail address: j.m.benitez@decsai.ugr.es

1 Introduction

Assessing and evaluating the accuracy of forecasting models and forecasts is an important and long-standing problem which a forecaster always faces when choosing among various available forecasting methods. This paper aims at investigating the usefulness of a blocked cross-validation (BCV) scheme along with directional accuracy measures for forecast evaluation. Several forecast error measures such as scale-dependent, percentage and relative measures have been largely used for forecast evaluation (see Armstrong and Fildes (1995); Taylor and Bunn (1999); Parigi et al. (2000); Meade (2002); Baffigi et al. (2004); Hyndman and Koehler (2006); Schumacher (2007); Marcellino (2008); Jumah and Kunst (2008); Costantini and Pappalardo (2010); Frale et al. (2010); Costantini and Kunst (2011); Ahumada (2012), Pavlidis et al. (2012) among others).

However, Blaskowitz and Herwartz (2009) point out that directional forecasts can provide a useful framework for assessing the economic forecast value when loss functions (or success measures) are properly formulated to account for the realized signs and realized magnitudes of directional movements. In this regard, Blaskowitz and Herwartz (2009, 2011) propose several directional accuracy measures which assign a different loss to the forecast, depending on whether it correctly forecasts the direction (rise/fall) of the time series or not. The idea behind this kind of measures is that there are many situations where the correct prediction of the direction of the time series can be very useful, even if the forecast is biased (an investor buys stock, if its price is expected to rise, Blaskowitz and Herwartz (2009); a central bank tends to increase the interest rate, if the inflation is expected to rise, Kim et al. (2008)). For purposes of out-of-sample (OOS) forecast evaluation, the sample is divided into two parts. A fraction of the sample is reserved for initial parameter estimation while the remaining fraction is used for evaluation. However, this procedure may fail to work well when the overall amount of data is limited and/or a lot of parameters are to be estimated. As the directional accuracy measures use the predictions in a binary way (correct/incorrect prediction of direction), the problems are even more prominent when using such measures. Here, the cross-validation scheme can considerably improve the forecast directional accuracy provided that the data used for forecasting are stationary (see Arlot and Celisse (2010)). In this context, the use of the directional forecasting accuracy is also recommended since changes in the sign are frequent with stationary data (no increasing/decreasing trend).

This paper makes a contribution to the existing literature by investigating whether, and to what extent, the k-fold BCV procedure proposed by Bergmeir and Benítez (2012) may provide better results in terms of forecast

directional accuracy than the standard OOS procedure. The use of the k-fold blocked scheme is suggested since it yields more precise error measures, and this paper aims at evaluating if this benefit is also retained when the forecasts are tested for directional accuracy. To this end, we provide a Monte Carlo analysis using simple univariate and multivariate linear autoregressive models. These models are likely to show a rather conservative behavior compared to more complex models regarding the differences in the outcome of the forecast evaluation, as more complex models require more data for parameter estimation, so that the observed effects may be even stronger with complex models. The Monte Carlo results show that the advantage of using a BCV scheme is quite remarkable.

Furthermore, we offer an empirical application to the UK interest rate data. The forecast results show that the BCV scheme has considerable advantages over the standard OOS evaluation procedure as it makes possible to compensate for the loss of information due to the binary nature of the directional accuracy measures.

The rest of the paper is organized as follows. Section 2 reviews the BCV procedure. Section 3 describes the directional accuracy measures. Section 4 provides the Monte Carlo results. Section 5 discusses our empirical findings, and Section 6 concludes.

2 Blocked cross-validation

In k-fold cross-validation (see Stone (1974)), the overall available data is randomly partitioned into k sets of equal size: each of the k sets is used once to measure the OOS forecast accuracy, and the other $k-1$ sets are used to build the model. The k resulting error measures are averaged using the mean to calculate the final error measure. The advantage of cross-validation is that all the data is used both for training (initial estimation) and testing, and the error measure can be computed k times instead of only one. Therefore, by averaging over the k measures, a more reliable evaluation of the true model performance can be obtained. Since the cross-validation scheme requires the data to be i.i.d. (see Arlot and Celisse (2010)), modified versions of cross-validation for time series analysis have been proposed (for a large survey see Bergmeir and Benítez (2012)). Identical distribution translates to stationarity of the series (Racine, 2000), and independence can be assured by omitting dependent values during testing (see, e.g., McQuarrie and Tsai (1998) or Kunst (2008) for the procedure). Depending on the amount of folds and the specification of the model, removal of dependent values may yield to a considerable loss of data, and even may result in an insufficient amount

of data for model estimation. A solution to this problem is to choose the k folds as blocks of sequential data (see also Racine (2000)). This scheme along with the procedure of omitting only the data from the borders of these data blocks during evaluation yields k-fold BCV (Bergmeir and Benítez, 2012).

In Figure 1, we present a simple example to show how the BCV procedure works in practice (see also Bergmeir and Benítez (2011)). For the OOS evaluation, the test set is chosen from the end of the series, and the training set is not interrupted by the test set. For the BCV scheme, there are cases in which the test set interrupts the training set, so that it consists of two non-continuous parts. Because of this, for some forecasting models (e.g., exponential smoothing methods or models with a moving average part) the use of cross-validation may not be straightforward, as it may be difficult for the model estimation procedure to handle missing values in the training set. However, this is not an issue in the broad class of (linear or non-linear) pure autoregressive models of fixed order, as in the embedded form of the series only the respective rows have to be removed before estimating the model.

3 Directional accuracy measures

Conventional measures of forecasting accuracy are based on the idea of a quadratic loss function in that larger errors carry proportionally greater weight than smaller ones. Such measures respect the view that forecast evaluation should concentrate on all large disturbances whether or not they are associated with directional errors which are of no special interest in and of themselves. However, several studies argue that incorrectly predicted directions are among the most serious errors a forecast can make (see Chung and Hong (2007); Kim et al. (2008); Solferino and Waldmann (2010); Blaskowitz and Herwartz (2009, 2011), Clatworthy et al. (2012) among others). In this respect, this study applies some directional accuracy measures (Blaskowitz and Herwartz (2009, 2011)) for forecast evaluation.

Using the indicator function I, the directional error (DE) for h-step-ahead forecasting is defined as:

$$\mathrm{DE}_t = I(I((\hat{y}_{t+h} - y_t) > 0) = I((y_{t+h} - y_t) > 0))$$

where y_t is the current value of the series, $\hat{y}_{t+h}$ is the value of the forecast, and y_{t+h} is the true value of the series at time $t + h$. Using DE, a general framework for the directional accuracy (DA) can be obtained:

$$\mathrm{DA}_t = \begin{cases} a & \text{for} \quad \mathrm{DE}_t = 1 \\ b & \text{for} \quad \mathrm{DE}_t = 0 \end{cases}$$

In this framework, a correct prediction of the direction gets assigned a value a, which can be interpreted as a reward, and an incorrect prediction gets assigned a value b, a penalty. In our study, we use $a = 1$ and $b = -1$. Based on the DA, several directional accuracy measures can be defined.
The mean directional accuracy (MDA) is defined straightforwardly as the mean of the DA:

$$\text{MDA} = \text{mean}(\text{DA}_t)$$

This measure acquires well the degree up to which the predictor is able to correctly predict the direction of the forecast, and it is robust to outliers. However, it does not take into account the actual size of the change, so it does not measure the economic value of the forecast (the predictor can be able to forecast the direction in cases of low volatility quite well, but it can fail when the volatility is high). Therefore, we use the directional forecast value (DV), which multiplies DA by the absolute value of the real changes, thus assessing better the actual benefit/loss of a correct/incorrect direction of the prediction.
The mean DV (MDV) is defined as:

$$\text{MDV} = \text{mean}(|y_{t+h} - y_t| \cdot \text{DA}_t)$$

In order to have a scale-free measure, the absolute value of the change can be divided by the current value of the series (Blaskowitz and Herwartz, 2011). Then, the mean directional forecast percentage value (MDPV) can be defined as follows:

$$\text{MDPV} = \text{mean}\left(\left|\frac{y_{t+h} - y_t}{y_t}\right| \cdot \text{DA}_t\right)$$

4 Monte Carlo experiment

In this section, we provide a Monte Carlo analysis. Specifically we consider two different experiments. In the first experiment, we generate series from a stable AR(3) process, while in the second one the data is generated from a bivariate VAR(2) model. In both experiments, one-step-ahead predictions are considered. The experiments are performed with the R programming language (R Development Core Team, 2009) in the following way.

Series are first generated and partitioned into a data set which is available to the forecaster, the *in-set*, and a set of data from the end of the series as unknown future, the *out-set*. We use 70 percent of the data as in-set, and the rest of the data as out-set. Then, the in-set is partitioned into training and

test sets using the OOS and 5-fold BCV procedures (20 percent of the in-set are used as test set, so that the OOS evaluation coincides with the last fold of the BCV). Models are then built and values of the test sets are predicted and the directional accuracy measures presented in Section 3 are calculated using the OOS and 5-fold BCV procedures. In this way, we obtain error estimates using only the data of the in-set. Then, we build models using all data of the in-set, and predict the values of the out-set and calculate the directional accuracy measures on the out-set (see also Figure 2). So, for each model type we obtain an error estimate using only the in-set data, and a reference error measure on the future of the series, the out-set data. This enables a comparison of the in-set error estimates (using OOS and BCV) and the out-set errors. See also Bergmeir and Benítez (2012) for the procedure. We calculate the root mean squared error of the in-set estimates with respect to the out-set errors, and call this measure in the following the root mean squared predictive accuracy error (RMSPAE). It is defined as follows:

$$\text{RMSPAE} = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(M_i^{out-set} - M_i^{in-set})^2}$$

Here, n is the number of series, i.e., trials in the Monte Carlo simulation, and M is the directional accuracy measure in consideration (MDA, MDV, or MDPV), calculated for one model and one series. In the case of M^{in-set}, the OOS or BCV procedure is used on the in-set, and in the case of $M^{out-set}$, the data of the in-set are used for training, and the data of the out-set are used for testing. Series with lengths of 50, 70, 100, 150, 200, 250, 300, 350, 400, 450, 500, 550, and 600 values are used. For each length, 1000 experiments are conducted.

4.1 Univariate case

Series are generated from a stable AR(3) process. Real-valued roots of the characteristic polynomial are chosen randomly from a uniform distribution in the interval $[-r_{max}, -1.1] \cup [1.1, r_{max}]$, with r_{max}=5.0. From these roots, the coefficients of the AR model are computed (for a more detailed description of the procedure, see Bergmeir and Benítez (2012)). The first 100 time series observations are discarded to avoid possible initial value effects. For each iteration, new coefficients and a new series are generated in this way. For forecasting purposes, we consider AR models that use 1 to 5 lagged values, AR(1) - AR(5). Evaluation is performed using OOS and 5-fold BCV. As percentage measures such as the MAPE and the MDPV are heavily skewed when the series have values close to zero (see, e.g., Hyndman and Koehler

(2006))[1], for each series we subtract the minimum of the series from all values (to obtain a series of non-negative values), and then we increment all values by 1, to achieve a series which only contains values greater 1.

Table 1 reports the RMSPAE results for the directional accuracy measures discussed in Section 3 for BCV and OOS schemes. A series length of 100 is considered in the table.[2]

We clearly see that the values for BCV are consistently smaller than the respective values of OOS evaluation. This means that the measures calculated on the in-set using BCV estimates more precisely the out-set measures. So, using BCV we can estimate more precisely the directional accuracy that is to be expected for a given method when using it for forecasting unknown future values of the series.

Figure 3 shows the RMSPAEs for the series of all lengths when an AR(3) model is considered. The results indicate that the RMSPAE in general decreases with increasing length of the series, so that the directional accuracy is estimated more precisely if more data are available. Also, advantages of cross-validation are greatest if the series are short which can be the case in empirical applications.

4.2 Multivariate case

The purpose of the multivariate Monte Carlo simulation study is to verify the robustness of the results in Section 4.1. The data generating process is a bivariate VAR(2) model. Series are generated as in Section 4.1. Eigenvalues for the companion matrix of the VAR model are generated, with an absolute value smaller than 1, in order to obtain a stable model (Lütkepohl, 2006). The companion matrix is generated from these eigenvalues by the procedure described by Boshnakov and Iqelan (2009). The covariance matrix is randomly chosen by generating an upper triangular matrix from a uniform distribution in the $[-1, 1]$ interval where the elements on the diagonal are set equal to 1. Therefore, a random symmetric matrix is built up. The random values necessary for the VAR process are then drawn from a Gaussian distribution, and multiplied by the Cholesky form of the covariance matrix. As in the univariate experiment, the first 100 observations are discarded and the resulting series are shifted to prevent problems with percentage measures (by incrementing each value by 1 and subtracting the minimum of the series).

Along with the VAR(2) model, two other models are used for forecasting purposes, namely the bivariate VAR(1) and VAR(3) model. In Table 2, the

[1] This is also confirmed in unreported preliminary experiments.

[2] To save space, results for other series lengths are not reported here. They are available upon request.

Monte Carlo results of RMSPAE for the directional accuracy measures using BCV and OOS evaluation procedures for series of length 100 are reported.[3] The results confirm those in the univariate case. From Table 2, it can be seen that the RMSPAE is consistently smaller using the BCV procedure, so that BCV provides a more precise estimate of the directional accuracy. Figure 4 shows the Monte Carlo results in terms of RMSPAE for series of all lengths when a VAR(2) model is considered. The findings observed in the univariate are then confirmed and the advantage of using the cross-validation scheme is preserved and it is also bigger with shorter series.

5 Empirical application

In this section, we offer an application to UK interest rate as an example of the use of BCV in practice. While the findings of the Monte Carlo simulation show superiority of BCV over OOS in general, we now focus on the particularity of directional accuracy measures of binary output, which potentially leads to a loss of information. The main purpose of our study is neither to support or establish an economic theory nor to show the suitability of a particular method, but to investigate the usefulness of the cross-validation scheme along with directional accuracy measures. Therefore we consider simple linear models for forecasting the UK quarterly interest rate.[4] In order to have a realistic setup with regard to the evaluation procedures, we do not perform a partition of the data into in-set and out-set, but just use OOS and BCV evaluation in the way it would be used in an empirical application. Therefore, we do not compute the RMSPAE, but we provide the results of the directional accuracy measures. The data set consists of quarterly annualized real GDP growth, quarterly annualized inflation rate and the three-month Treasury bill rate. The data is taken from the OECD Main Economic Indicators database and it covers the period 1965:1-2011:1.[5] GDP growth is defined as 400 times the log difference of GDP and inflation is similarly defined using CPI. The interest rate is used without any change. The series are shown in Figure 5. All the series have been tested for stationarity using the DF-GLS unit root test of Elliott et al. (1996). The results show that the inflation and GDP growth rates are stationary at 5% level (the statistics are -3.102 and -5.129, respectively) while interest rate is stationary at 10% level (the statistics is -1.920).

[3] As in the univariate case, results for series of other lengths are not reported here to save space. They are available upon request.

[4] For a forecasting exercise on UK interest data see also Barnett et al. (2012).

[5] The CPI data has been seasonally adjusted using Tramo seats.

In the application, we consider a trivariate VAR model with interest, CPI inflation and GDP growth rates (VAR_3), a bivariate VAR model with interest and CPI inflation rates (VAR_{cpi}) and a bivariate VAR model with interest and GDP growth rates (VAR_{gdp}). All the VAR models are of order two (a common or a recommended model in macroeconomic systems, see Lütkepohl (2006)).

Table 3 reports the results. It should be noted that the MDV measure yields negative values for the OOS procedure. This result may be due to the fact that the last part of the sample period of the series is fairly stable. Furthermore, we note that the OOS procedure is not capable of distinguishing the VAR_3 model from the VAR_{gdp} model as we obtain the exactly same values for all the directional measures, while the cross-validation procedure determines the model performances more precisely since it uses all the sample period for the forecast evaluation. This result is examined in more detail in Figure 6. It should be noticed that in the last fold of BCV, which is also used for OOS evaluation, all models yield identical results in terms of the directional accuracy, with the exception of the VAR_{cpi} which yields an incorrect directional forecast in one case (the other two models are able to correctly predict the direction). Using OOS, it is harder to distinguish the forecasting performance among the models (the VAR_3 and VAR_{gdp} models yields the same results). In this regard, findings show that BCV may help distinguish models' forecasting performances in terms of directional forecast accuracy.

6 Conclusions

This paper investigates the usefulness of a predictor evaluation framework which combines a k-fold blocked cross-validation scheme with directional accuracy measures. The advantage of using a blocked cross-validation scheme with respect to other procedures such as the standard out-of-sample procedure is that the cross-validation allows to obtain more precise error estimates since it makes full use of the data. In this paper we evaluate whether, and to what extent, the k-fold blocked cross-validation procedure may provide more precise results than the standard out-of-sample procedure even when dealing with directional forecast accuracy. To this end, a Monte Carlo analysis is performed using simple univariate and multivariate linear models.

An empirical application is carried out on forecasting UK interest rate data. The results show that the standard out-of-sample procedure may fail to detect the best performance among different forecasting models while the cross-validation helps obtain this. This may suggest the use of the block

cross-validation scheme when dealing with directional forecast evaluation.

References

H.A. Ahumada. Break detectability and mean square forecast error ratios for selecting estimation windows. *Journal of Forecasting*, 31(8):688–705, 2012.

S. Arlot and A. Celisse. A survey of cross-validation procedures for model selection. *Statistics Surveys*, 4:40–79, 2010.

S. Armstrong and R. Fildes. On the selection of error measures for comparisons among forecasting methods. *Journal of Forecasting*, 14:67–71, 1995.

A. Baffigi, R. Golinellli, and G. Parigi. Bridge models to forecast the euro area GDP. *International Journal of Forecasting*, 20:447–460, 2004.

A. Barnett, H. Mumtaz, and K. Theodoridis. Forecasting UK GDP growth, inflation and interest rates under structural change: a comparison of models with time-varying parameters. Working Paper 450, Bank of England, 2012.

C. Bergmeir and J.M. Benítez. Forecaster performance evaluation with cross-validation and variants. In *International Conference on Intelligent Systems Design and Applications, ISDA*, pages 849–854, 2011.

C. Bergmeir and J.M. Benítez. On the use of cross-validation for time series predictor evaluation. *Information Sciences*, 191:192–213, 2012.

O. Blaskowitz and H. Herwartz. Adaptive forecasting of the euribor swap term structure. *Journal of Forecasting*, 28(7):575–594, 2009.

O. Blaskowitz and H. Herwartz. On economic evaluation of directional forecasts. *International Journal of Forecasting*, 27(4):1058–1065, 2011.

G.N. Boshnakov and B.M. Iqelan. Generation of time series models with given spectral properties. *Journal of Time Series Analysis*, 30(3):349–368, 2009.

J. Chung and Y. Hong. Model-free evaluation of directional predictability in foreign exchange markets. *Journal of Applied Econometrics*, 22:855–889, 2007.

M.A. Clatworthy, D.A. Peel, and P.F. Pope. Are analysts' loss functions asymmetric? *Journal of Forecasting*, 31(8):736–756, 2012.

M. Costantini and R. Kunst. Combining forecasts based on multiple encompassing tests in a macroeconomic core system. *Journal of Forecasting*, 30: 579–596, 2011.

M. Costantini and C. Pappalardo. A hierarchical procedure for the combination of forecasts. *International Journal of Forecasting*, 26(4):725–743, 2010.

G. Elliott, T.J. Rothenberg, and J.H. Stock. Efficient tests for an autoregressive unit root. *Econometrica*, 64:813–836, 1996.

C. Frale, M. Marcellino, G. Mazzi, and T. Proietti. Survey data as coincident or leading indicators. *Journal of Forecasting*, 29:109–131, 2010.

R.J. Hyndman and A.B. Koehler. Another look at measures of forecast accuracy. *International Journal of Forecasting*, 22(4):679–688, 2006.

A. Jumah and R. M. . Kunst. Seasonal prediction of european cereal prices: Good forecasts using bad models. *Journal of Forecasting*, 27:391–406, 2008.

T.-H. Kim, P. Mizen, and T. Chevapatrakul. Forecasting changes in UK interest rates. *Journal of Forecasting*, 27(1):53–74, 2008.

R. Kunst. Cross validation of prediction models for seasonal time series by parametric bootstrapping. *Austrian Journal of Statistics*, 37:271–284, 2008.

H. Lütkepohl. *New Introduction to Multiple Time Series Analysis*. Springer, 2006. ISBN 9783540262398.

M. Marcellino. A linear benchmark for forecasting GDP growth and inflation? *Journal of Forecasting*, 27(4):305–340, 2008.

Allan D. R. McQuarrie and Chih-Ling Tsai. *Regression and time series model selection*. World Scientific Publishing, 1998.

N. Meade. A comparison of the accuracy of short term foreign exchange forecasting methods. *International Journal of Forecasting*, 18:67–83, 2002.

G. Parigi, R. Golinelli, and G. Bodo. Forecasting industrial production in the euro area. *Empirical Economics*, 25:541–561, 2000.

E.G. Pavlidis, I. Paya, and D.A. Peel. Forecast evaluation of nonlinear models: The case of long-span real exchange rates. *Journal of Forecasting*, 31 (7):580–595, 2012.

R Development Core Team. *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing, Vienna, Austria, 2009. URL `http://www.R-project.org`. ISBN 3-900051-07-0.

J. Racine. Consistent cross-validatory model-selection for dependent data: hv-block cross-validation. *Journal of Econometrics*, 99(1):39–61, 2000.

C. Schumacher. Forecasting german GDP using alternative factor models based on large datasets. *Journal of Forecasting*, 26:271–302, 2007.

N. Solferino and R. Waldmann. Predicting the signs of forecast errors. *Journal of Forecasting*, 29(5):476–485, 2010.

M. Stone. Cross-validatory choice and assessment of statistical predictions. *Journal of the Royal Statistical Society. Series B (Methodological)*, 36(2): 111–147, 1974.

J.W. Taylor and D.W. Bunn. Investigating improvements in the accuracy of prediction intervals for combinations of forecasts: a simulation study. *International Journal of Forecasting*, 15:325–339, 1999.

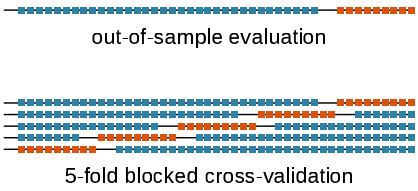

Figure 1: Training and test sets chosen for traditional OOS evaluation, and 5-fold BCV. Blue dots represent points from the time series in the training set and orange dots represent points in the test set. In the example, we assume that the model uses two lagged values for forecasting, which is why at the borders always two values are omitted.

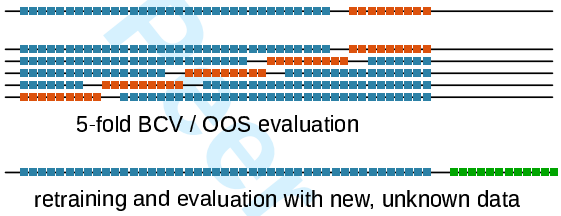

Figure 2: Illustration of the Monte Carlo experiments. The data are partitioned into an in-set, which is used for BCV and OOS evaluation, and an out-set (green), which is completely withheld. After model estimation and estimation of the directional accuracy measures, the models are estimated again, using all available data in the in-set, to forecast the unknown future (the out-set). This is a typical application scenario of forecasting. In our experiments, then the directional accuracy measures are calculated on the out-set, and the error estimates given by BCV and OOS evaluation can be compared to the reference errors calculated on the out-set.

Table 1: Univariate results. OOS and 5-fold BCV procedures.

	MDA	MDV	MDPV
RMSPAE 5-fold BCV			
AR(1)	0.1803	0.9035	0.2922
AR(2)	0.1830	0.9025	0.2926
AR(3)	0.1820	0.9045	0.2927
AR(4)	0.1838	0.9059	0.2928
AR(5)	0.1864	0.9069	0.2933
RMSPAE OOS			
AR(1)	0.2697	1.1790	0.3908
AR(2)	0.2775	1.1798	0.3908
AR(3)	0.2756	1.1817	0.3911
AR(4)	0.2733	1.1799	0.3905
AR(5)	0.2743	1.1812	0.3909

Notes: Series of length 100. The RMSPAE is calculated over 1000 trials.

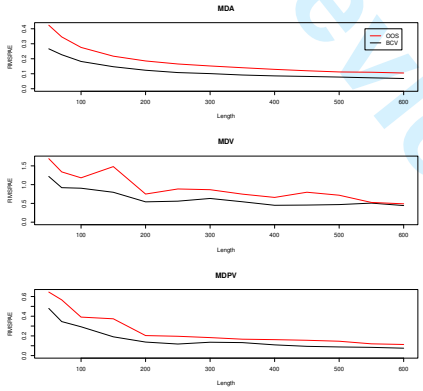

Figure 3: Monte Carlo standard deviations of the forecast errors for an AR(3) model. Series of different lengths.

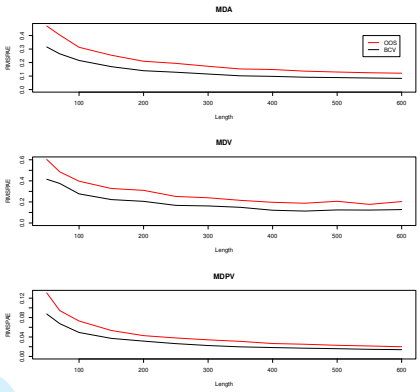

Figure 4: Monte Carlo standard deviations of the forecast errors for a VAR(2) model. Series of different lengths.

Table 2: Multivariate results. OOS and 5-fold BCV procedures.

	MDA	MDV	MDPV
RMSPAE 5-fold BCV			
VAR(1)	0.2203	0.3050	0.0452
VAR(2)	0.2157	0.2765	0.0493
VAR(3)	0.2212	0.2772	0.0522
RMSPAE OOS			
VAR(1)	0.3220	0.4106	0.0664
VAR(2)	0.3132	0.3968	0.0729
VAR(3)	0.3299	0.4182	0.0781

Notes: Series of length 100. The RMSPAE is calculated over 1000 trials.

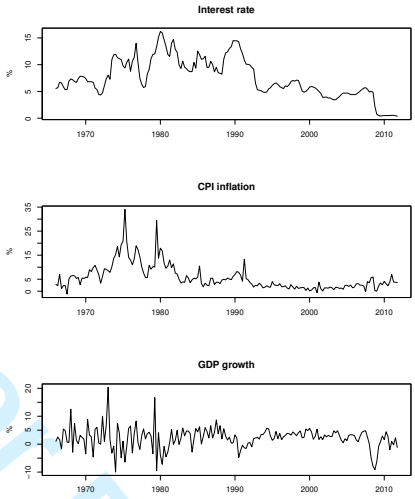

Figure 5: UK quarterly interest rate, CPI inflation rate and GDP growth rate.

Table 3: UK interest rate forecasting results. OOS and 5-fold BCV procedures.

	MDA	MDV	MDPV
5-fold BCV			
VAR_3	0.1778	0.0819	0.0165
VAR_{cpi}	0.2444	0.1122	0.0152
VAR_{gdp}	0.1556	0.1013	0.0181
OOS			
VAR_3	0.1667	-0.0048	0.0191
VAR_{cpi}	0.1111	-0.0161	0.0036
VAR_{gdp}	0.1667	-0.0048	0.0191

Notes: The values for 5-fold BCV are averaged error measures over the 5 folds. VAR_3 includes interest, CPI inflation and GDP growth rates; VAR_{cpi} interest and CPI inflation rates; VAR_{gdp} interest and GDP growth rates. Note that the table does not report the RMSPAE as Tables 1 and 2, but it shows the values of the directional accuracy measures.

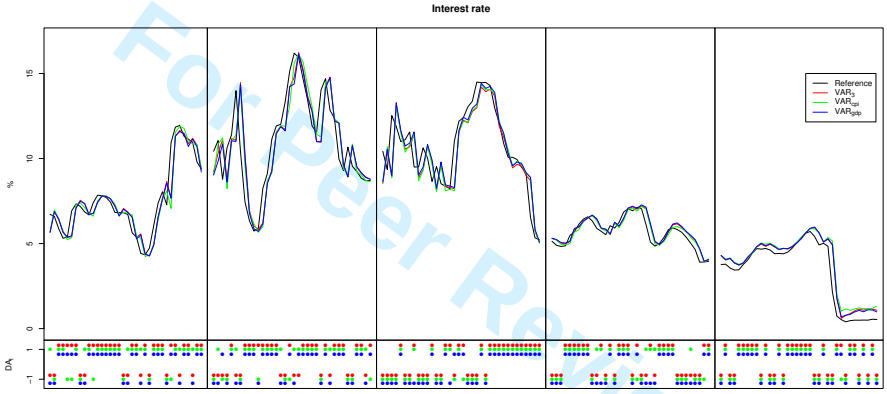

Figure 6: UK quarterly interest rate forecasts. Forecasts and directional accuracy of the three different VAR models are shown. The directional accuracy as defined in Section 3 only takes the values -1 and 1. As the models obtain very similar results in terms of forecasts, it may be difficult to distinguish them w.r.t. their directional accuracy performance.