Time Series Methods in the R package **mlr**

Steve Bronder
sab2287@columbia.edu

October 16, 2016

Abstract

The **mlr** package is a unified interface for machine learning tasks such as classification, regression, cluster analysis, and survival analysis. **mlr** handles the data pipeline of pre-processing, resampling, model selection, model tuning, ensembling, and prediction. This paper details new methods for developing time series models in **mlr**. It includes standard and novel tools such as auto-regressive and LambertW transform data generating processes, fixed and growing window cross validation, and forecasting models in the context of univariate and multivariate time series. Examples from forecasting competitions will be given in order to demonstrate the benefits of a unified framework for machine learning and time series.

1 Introduction

There has been a rapid developement in time series methods over the last 25 years [10]. Time series models have not only become more common, but more complex. The R language [18] has a large task view with many packages available for forecasting and time series methods. However, without a standard framework, many packages have their own sub-culture of style, syntax, and output. The **mlr** [4] package, short for Machine Learning in R, works to give a strong syntatic framework for the modeling pipeline. By automating many of the standard tools in machine learning such as preprocessing and cross validation, **mlr** reduces error in the modeling process that is derived from the user.

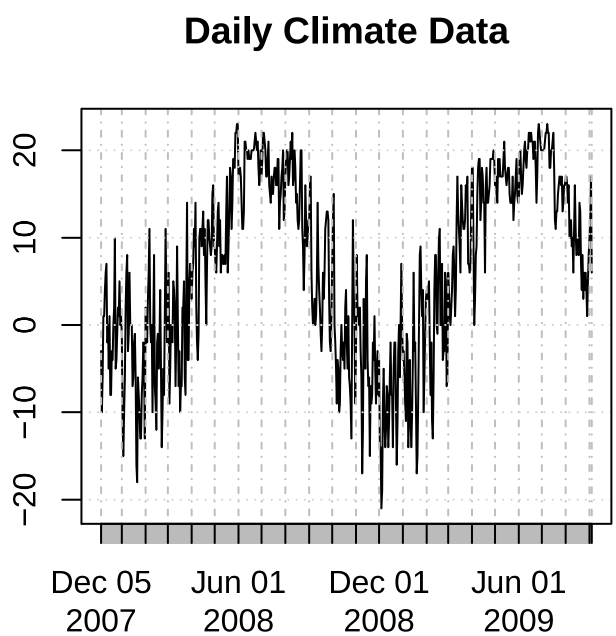
While there are some time series methods available in **caret** [8], development of forecasting models in **caret** is difficult due to computational constraints and design choices. The highly modular structure of **mlr** makes it the best choice for implementing time series methods and models. This paper will show how using **mlr**'s strong syntatic structure allows for time series packages such as **forecast** [14] and **rugarch** [9] to use machine learning methedologies such as automated parameter tuning, data preprocessing, model blending, cross validation, performance evaluation, and parallel processing techniques for decreasing model build time.

2 Forecasting Example with the M4 Competition

Professional forecasters attempt to predict the future of a series based on its past values. Forecasting can be used in a wide range of tasks including forecasting stock prices, [11], weather patterns [1], international conficts [5], and earthquakes [23]. In order to evaluate

mlr's forecasting framework we need a large set of possible time series to make sure our methods generalize well.[1]

The Makridakis competitions [17] are forecasting challenges organized by the International Institute of Forecasters and led by Spyros Makridakis to evaluate and compare the accuracy of forecasting methods. The most recent of the competitions, the M4 competition, contains 10,000 time series on a yearly, quarterly, monthly, and daily frequency in areas such as finance, macroeconomics, climate, microeconomics, and industry. To show examples of how **mlr**'s forecasting features work we will look at a particular climate series. The data is daily with the training subset starting on September 6th, 2007 and ending on September 5th, 2009 while the testing subset is from September 6th, 2009 to October 10th, 2009 for a total of 640 training periods and 35 test periods to forecast.



3 Forecasting Tasks

mlr provides uses the S3 object system to clearly define a predictive modeling task. Tasks contain the data and other relevant information such as the task id and which variable you are targeting for supervised learning problems. Forecasting tasks are handled in **mlr** by the function `makeForecastRegrTask()`. The forecasting task inherets from `makeRegrTask`, but has two noticable differences in parameters.

data: Instead of a data frame, an xts object from **xts** [22] containing the time series.

[1] Very goofy sentence need to fix

frequency: An integer with the number of periods your time series contains. For example, daily data with a weekly periodicity has a frequency of 7 while daily data with a yearly periodicity has a frequency of 365.

```r
library(mlr)

## Loading required package:  ParamHelpers

climate.task = makeForecastRegrTask(id = "M4 Climate Data",
                                    data = m4Train1,
                                    target = "target_var",
                                    frequency = 365L)
climate.task

## Task: M4 Climate Data
## Type: fcregr
## Observations: 640
## Dates:
##  Start: 2007-12-05
##  End: 2009-09-04
## Frequency: 365
## Features:
## numerics  factors  ordered
##        1        0        0
## Missings: FALSE
## Has weights: FALSE
## Has blocking: FALSE
```

4 Building a forecast learner

The `makeLearner()` function provides a structured model building framework to the 7 forecasting models currently implimented in **mlr**. As an example, we will build a simple AutoRegressive Integrated Moving Average (ARIMA) model. The ARIMA model is of the form

$$y_t \sim \alpha + \beta_1 \Delta_d y_{t-1} ... \beta_p \Delta_d y_{t-p} + \phi_1 \epsilon_{t-1} + ... + \phi_q \epsilon_{t-q} + \epsilon_t \tag{1}$$

$$y_t \sim \alpha + \sum_{i=1}^{p} \beta_i \Delta_d y_{t-i} + \sum_{i=1}^{q} \phi_i \epsilon_{t-i} + \epsilon_t \tag{2}$$

In equation three, α is a constant, β_p is the coefficient associated with the lagged observations of y with Δ_d being the dth difference operator. The coefficient for the one step forecast error ϵ_{t-q} is ϕ_q. ARIMA is one of the most well known forecasting models and is avaible in mlr along with models such as BATS, TBATS, THIEF [13], ETS, several GARCH variants, and autoregressive neural networks. In addition, preprocessing features have been added to allow arbitrary supervised machine learning models to be used in the context of forecasting.

To impliment this model we use `makeLearner()`, supplying the class of learner, order, the number of steps to forecast, and any additional arguments to be passed to `Arima` for **forecast**.

```
tbatsMod =makeLearner("fcregr.tbats", use.box.cox = FALSE,
                      use.trend = TRUE,
                      seasonal.periods = TRUE, max.p = 3, max.q = 2,
                      stationary = FALSE, use.arma.errors = TRUE,
                      h = 35)
```

To train the model we simply call train, supplying the model and task.

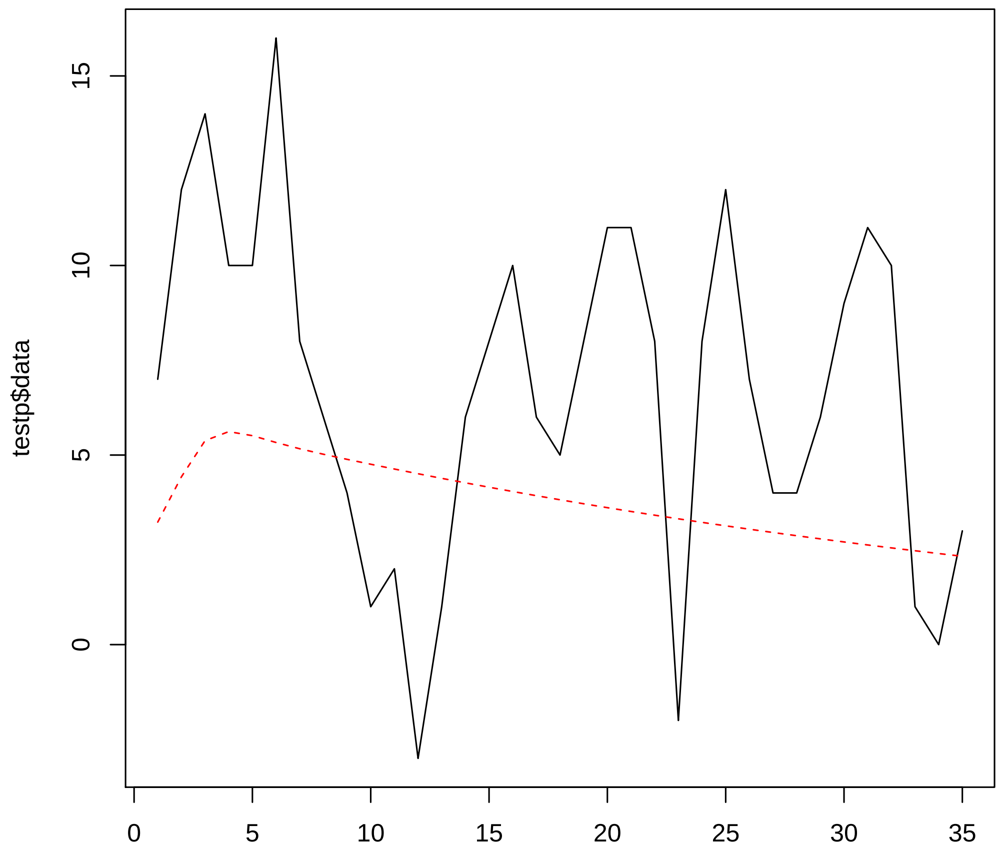
```
trainTbats= train(tbatsMod, climate.task )
trainTbats

## Model for learner.id=fcregr.tbats; learner.class=fcregr.tbats
## Trained on: task.id = M4 Climate Data; obs = 640; features = 1
## Hyperparameters: use.box.cox=FALSE,use.trend=TRUE,seasonal.periods=TRUE,max.p=3,max.q=2,statio

testp = predict(trainTbats, newdata = m4Test1)
performance(testp, mase, task = climate.task)

##      mase
## 0.0676601

matplot(testp$data, type = "l")
```



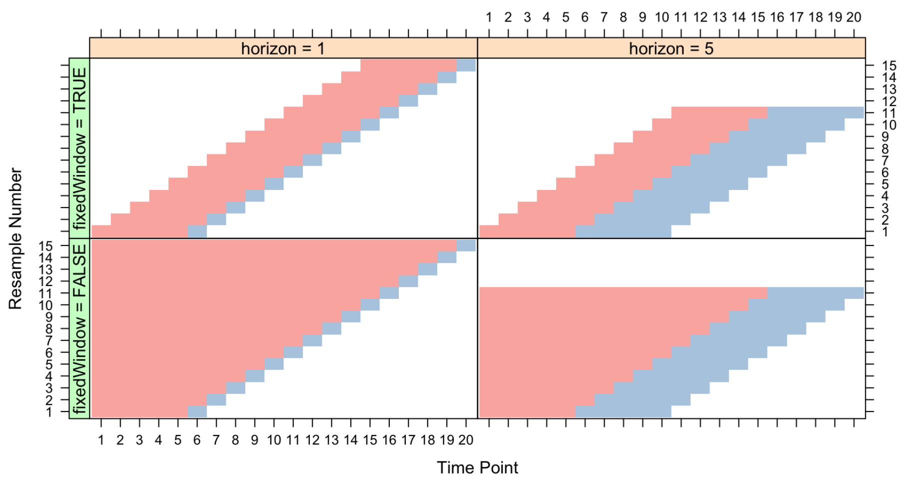
While ARIMA is one of the most well known time series models, the order selection process can be subjective and difficult for users. One of the first proposals for automatic order selection comes from [6] where innovations are obtained by fitting high order autoregressive model to the data and then computing the likelihood of potential models through a series of standard regresssions. Proprietary algorithms from software such as Forecast Pro [20] and Autobox [19] are well known and have performed to high standards in competitions such as the M3 forecasting competition [17]. One of the most well known R packages for automated forecast is **forecast** [14] which contains several methods for automated forecasting including exponential smoothing based methods and step-wise algorithms for forecasting with ARIMA models.

Forecasting in **mlr** takes a machine learning approach, creating a parameter set for a given model and using an optimization method to search over the parameter space. To do this, we will use a windowing resampling scheme to train over the possible models.

5 Resampling with Time

Resampling schemes such as cross-validation, bootstrapping, etc. are common in machine learning for dealing with the bias-variance tradeoff [7] [21]. When their is a time component to the data, windowing schemes are useful in allowing a valid resampling scheme while still maintaining the time properties of the series.[2]. Figure one gives an example of what fixed and growing windows look like. Given a horizon and initial starting point the window slides forward one step each time while either shifting in the fixed case or enlarging by one in the growing case. Growing and fixed window resampling such as from [12] are now available in the `resampling()` function of **mlr**.

Figure 1: Resampling with a window scheme as exampled by caret [16]



[2]crap

A windowing resampling process is created in the function `makeResampleDesc()` by supplying the resampling type, horizon, initial window, the length of the series, and an optional parameter to skip over some windows for the sake of time.

```
resampDesc = makeResampleDesc("GrowingCV", horizon = 35L,
                              initial.window = 400L,
                              size = nrow(getTaskData(climate.task)), skip = 6L)
resampDesc

## Window description:
##  growing with 30 iterations:
##  400 observations in initial window and 35 horizon.
## Predict: test
## Stratification: FALSE
```

To make a parameter set to tune over **mlr** uses **ParamHelpers** [3]. For this example we use the random search procedure from [2] to search our parameter space for the most optimal model.

```
parSet = makeParamSet(
  makeLogicalParam(id = "use.box.cox",
                   default = FALSE,
                   tunable = TRUE),
  makeLogicalParam(id = "use.trend",
                   default = FALSE,
                   tunable = TRUE),
  makeLogicalParam(id = "use.damped.trend",
                   default = FALSE,
                   tunable = TRUE),
  makeLogicalParam(id = "seasonal.periods",
                   default = FALSE,
                   tunable = TRUE),
  makeIntegerParam(id = "max.p",
                   upper = 20,
                   lower = 0),
  makeIntegerParam(id = "start.p",
                   upper = 10,
                   lower = 1,
                   trafo = function(x) x*2),
  makeIntegerParam(id = "max.q",
                   upper = 20,
                   lower = 0),
  makeIntegerParam(id = "start.q",
                   upper = 10,
                   lower = 1,
                   trafo = function(x) x*2),
  makeIntegerParam("max.P",
                   lower = 0,
```

```r
                    upper = 5),
  makeIntegerParam("max.Q",
                    lower = 0,
                    upper = 5),
  makeDiscreteParam("ic",
                     values = c("aicc","aic","bic")),
  makeDiscreteParam("test",
                     values = c("kpss","adf","pp")),
  makeDiscreteParam("seasonal.test",
                     values = c("ocsb", "ch")),
  makeLogicalParam("biasadj", default = FALSE),
  makeIntegerParam(id = "h",
                    default = 35,
                    tunable = FALSE,
                    lower = 35,
                    upper = 36)
)

#Specify tune by grid estimation
ctrl = makeTuneControlIrace(maxExperiments = 500L)
```

Using `tuneParams()` the model is tuned for the task using the specified resampling scheme, parameter set, tune control, and measure. For this tuning task we use MASE [15] as a measure of performance [3].

```r
#
library("parallelMap")
parallelStartSocket(6)
configureMlr(on.learner.error = "warn")
set.seed(1234)
tbatsTune = tuneParams("fcregr.tbats", task = climate.task, resampling = resampDesc,
                  par.set = parSet, control = ctrl, measures = mase)
parallelStop()
tbatsTune
```

```
## Tune result:
## Op. pars: use.box.cox=TRUE; use.trend=TRUE; use.damped.trend=TRUE; seasonal.periods=FALSE; max
## mase.test.mean=0.0688
```

The best model's parameters are extracted using `setHyperPars()` and the best model is passed to `train()` to go over the full data set.

```r
lrn = setHyperPars(makeLearner("fcregr.tbats"), par.vals = tbatsTune$x)
m = train(lrn, climate.task)
```

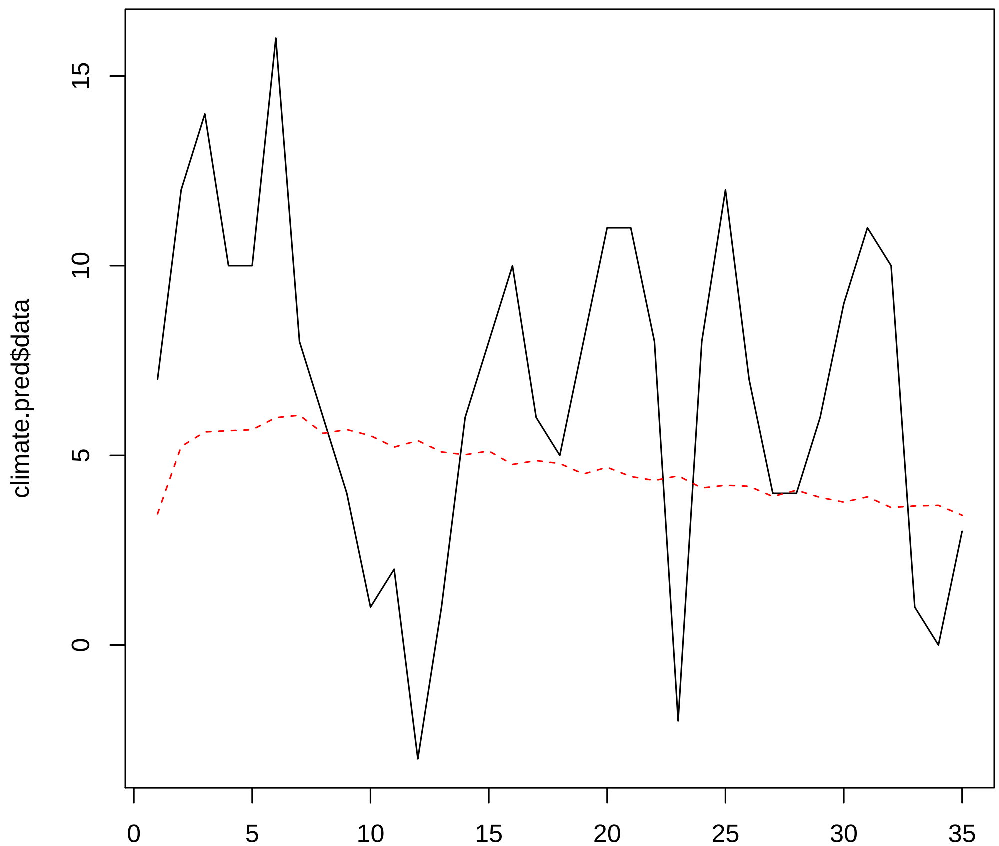
To make predictions for our test set we simply pass our model, task, and test data to `predict()`

[3] Models with a seasonal difference > 0 may be favorably biased as we use the non-seasonal MASE score

```r
climate.pred = predict(m, newdata = m4Test1)
performance(climate.pred, measures = mase, task = climate.task)

##       mase
## 0.06145902

matplot(climate.pred$data, type = "l")
```




```r
par_set = makeParamSet(
  makeDiscreteParam(id = "model", values = c("sGARCH", "csGARCH")),
  makeIntegerVectorParam(id = "garchOrder", len = 2L, lower = c(1,1), upper = c(4,4)),
  makeIntegerVectorParam(id = "armaOrder", len = 2L, lower = c(5,1), upper = c(8,3)),
  makeLogicalParam(id = "include.mean"),
  makeLogicalParam(id = "archm"),
```

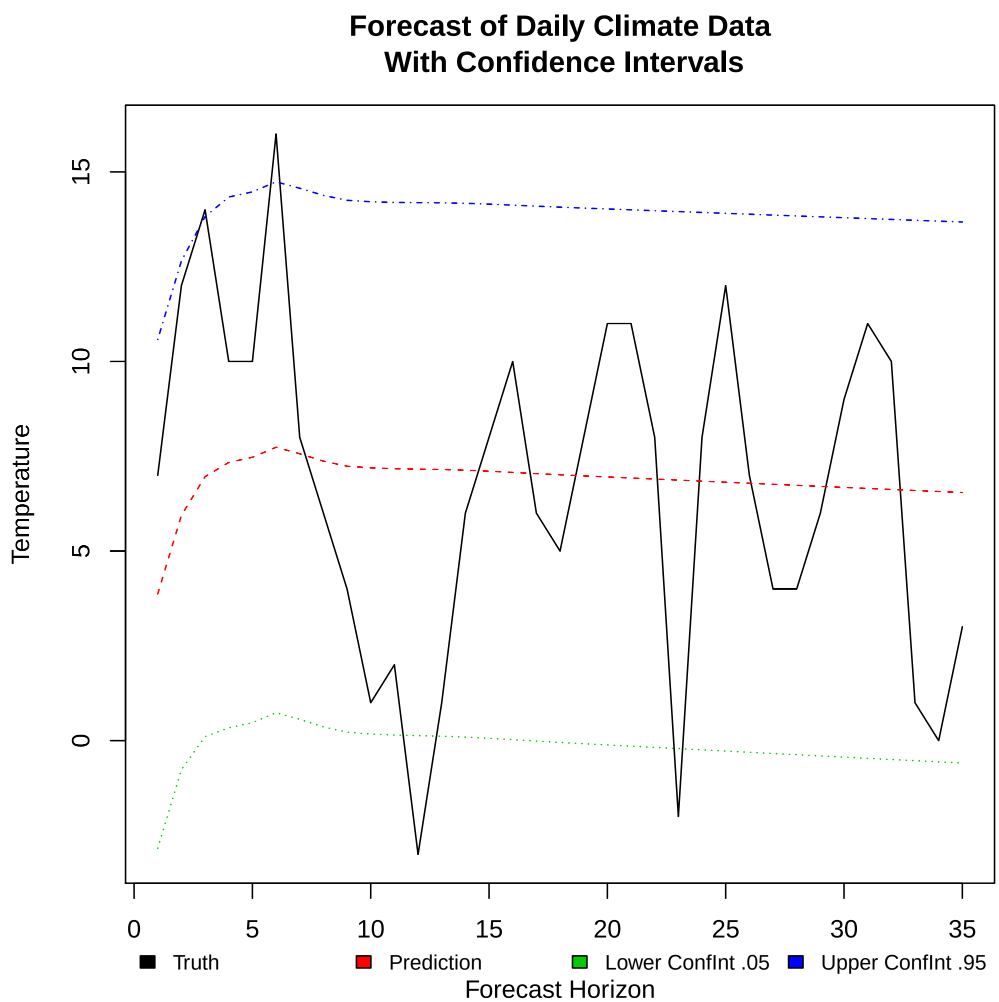
```r
  makeDiscreteParam(id = "distribution.model", values = c("norm","std","jsu")),
  makeDiscreteParam(id = "stationarity", c(0,1)),
  makeDiscreteParam(id = "fixed.se", c(0,1)),
  makeDiscreteParam(id = "solver", values = "nloptr"),
  makeIntegerParam(id = "n.ahead", default = 35L, lower = 35L,
                   upper = 36L, tunable = FALSE)
)

#Specify tune by grid estimation
ctrl = makeTuneControlIrace(maxExperiments = 400L)

parallelStartSocket(6)
configureMlr(on.learner.error = "warn")
set.seed(1234)
garchTune = tuneParams("fcregr.garch", task = climate.task, resampling = resampDesc,
                 par.set = par_set, control = ctrl, measures = mase)
parallelStop()
garchTune
```

```
## Tune result:
## Op. pars: model=sGARCH; garchOrder=4,1; armaOrder=7,2; include.mean=TRUE; archm=FALSE; distrib
## mase.test.mean=0.0789
```

```
##       mase
## 0.05589035
```



Forecast of Daily Climate Data
With Confidence Intervals
Temperature
0
5
10
15
0
5
10
15
20
25
30
35
Truth
Prediction
Lower ConfInt .05
Upper ConfInt .95
Forecast Horizon

References

[1] Robert L. Winkler Allan H. Murphy. Probability forecasting in meterology. *Journal of the American Statistical Association*, 79(387):489–500, 1984.

[2] James Bergstra and Yoshua Bengio. Random search for hyper-parameter optimization. *J. Mach. Learn. Res.*, 13(1):281–305, February 2012.

[3] Bernd Bischl, Michel Lang, Jakob Bossek, Daniel Horn, Jakob Richter, and Pascal Kerschke. *ParamHelpers: Helpers for Parameters in Black-Box Optimization, Tuning and Machine Learning*, 2016. R package version 1.9.

[4] Bernd Bischl, Michel Lang, Jakob Richter, Jakob Bossek, Leonard Judt, Tobias Kuehn, Erich Studerus, and Lars Kotthoff. *mlr: Machine Learning in R*, 2015. R package version 2.7.

[5] Thomas Chadefaux. Early warning signals for war in the news. *Journal of Peace Research*, 51(1):5–18, 2014.

[6] J. Rissanen E. J. Hannan. Recursive estimation of mixed autoregressive-moving average order. *Biometrika*, 69(1):81–94, 1982.

[7] Jerome H. Friedman. On bias, variance, 0/1—loss, and the curse-of-dimensionality. *Data Mining and Knowledge Discovery*, 1(1):55–77, 1997.

[8] Max Kuhn. Contributions from Jed Wing, Steve Weston, Andre Williams, Chris Keefer, Allan Engelhardt, Tony Cooper, Zachary Mayer, Brenton Kenkel, the R Core Team, Michael Benesty, Reynald Lescarbeau, Andrew Ziem, Luca Scrucca, Yuan Tang, and Can Candan. *caret: Classification and Regression Training*, 2015. R package version 6.0-62.

[9] Alexios Ghalanos. *rugarch: Univariate GARCH models.*, 2015. R package version 1.3-6.

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

[11] Clive W.J. Granger. Forecasting stock market prices: Lessons for forecasters. *International Journal of Forecasting*, 8(1):3 – 13, 1992.

[12] R.J. Hyndman and G. Athanasopoulos. *Forecasting: principles and practice:*. OTexts, 2014.

[13] Rob Hyndman and Nikolaos Kourentzes. *thief: Temporal Hierarchical Forecasting*, 2016. R package version 0.2.

[14] Rob J Hyndman and Yeasmin Khandakar. Automatic time series forecasting: the forecast package for R. *Journal of Statistical Software*, 26(3):1–22, 2008.

[15] Rob J. Hyndman and Anne B. Koehler. Another Look at Measures of Forecast Accuracy. Monash Econometrics and Business Statistics Working Papers 13/05, Monash University, Department of Econometrics and Business Statistics, May 2005.

[16] Max Kuhn. caret data splitting methods, 1999.

[17] Spyros Makridakis and Michèle Hibon. The m3-competition: results, conclusions and implications. *International Journal of Forecasting*, 16(4):451 – 476, 2000. The M3-Competition.

[18] R Core Team. *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing, Vienna, Austria, 2015.

[19] David Reilly. The autobox system. *International Journal of Forecasting*, 16(4):531–533, 2000.

[20] Goodrich RL. The forecast pro methodology. *International Journal of Forecasting*, 16(4):533–535, 2000.

[21] J. D. Rodriguez, A. Perez, and J. A. Lozano. Sensitivity analysis of k-fold cross validation in prediction error estimation. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 32(3):569–575, March 2010.

[22] Jeffrey A. Ryan and Joshua M. Ulrich. *xts: eXtensible Time Series*, 2016. R package version 0.10-0.

[23] Tuncel M. Yegulalp. Forecasting for largest earthquakes. *Management Science*, 21(4):418–421, 1974.