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

Steve Bronder
sab2287@columbia.edu

December 4, 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 [14] whereby time series models have not only become more common, but more complex. The R language [25] has a large task view with many packages available for forecasting and time series methods, but the open source nature of R has left users without a standard framework. Many packages have their own sub-culture of style, syntax, and output. The **mlr** [5] 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 from the user during the modeling process.

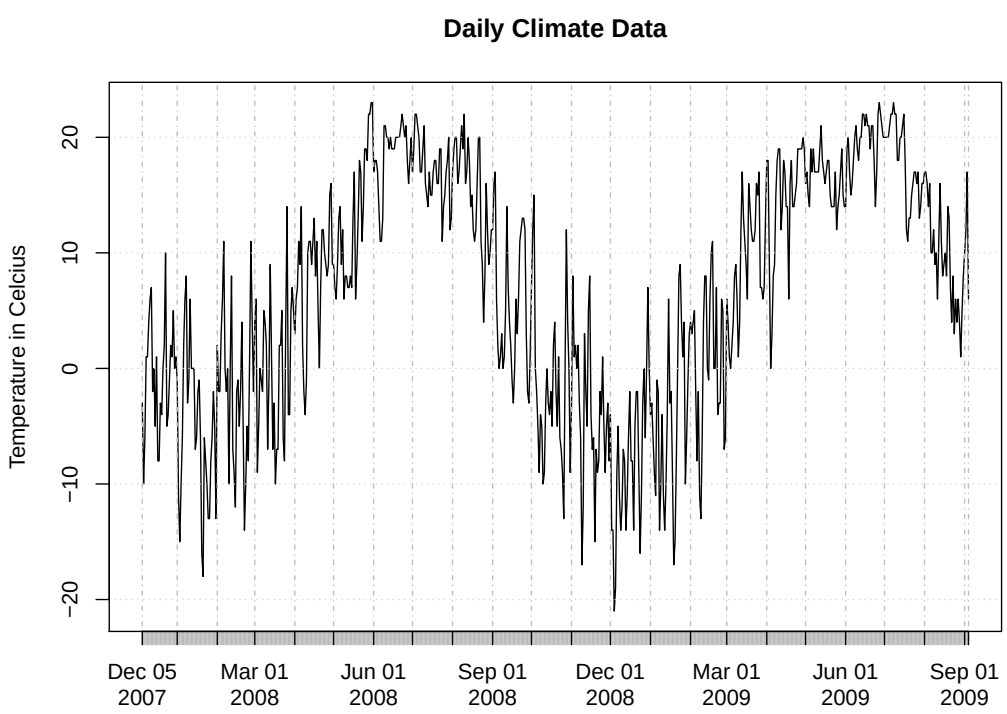
While there are some time series methods available in **caret** [10], development of forecasting models in **caret** is difficult due to computational constraints and design choices within the package. The highly modular structure of **mlr** makes it the best choice for implementing time series methods and models. This paper will show how using **mlr**'s strong syntatic structure allows for time series packages such as **forecast** [18], **rugarch** [11], and [24] 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, [15],

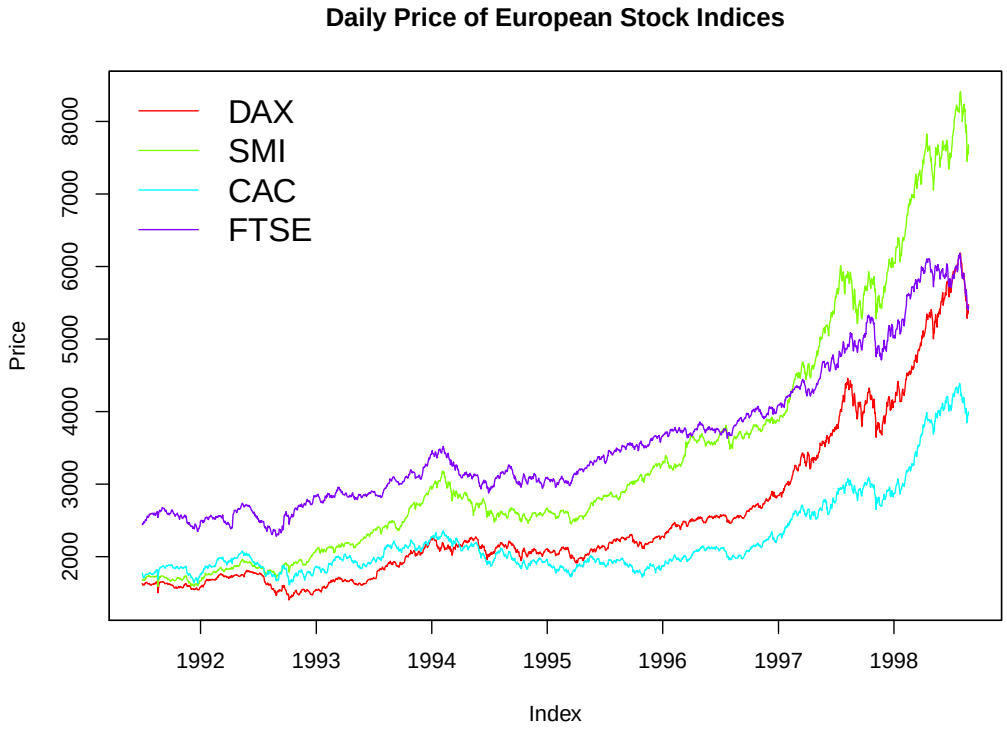
weather patterns [1], international conficts [6], and earthquakes [32]. In order to evaluate **mlr**'s forecasting framework we need a large set of possible time series to make sure our methods generalize well.

The Makridakis competitions [22] is a set of 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 works 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.



This series was chosen for its obvious seasonality and time features. Our data set should be large enough that the tuning method can take multiple windows of the data. Some series in M4 only contain 12 observations, which is not enough data to accurately train a model. We can see figure one is what most people imagine when they think of a time series. There is a clear seasonal time trend with individual points moving about the seasonal periods. The data can be found in the package **M4comp** [2] under sets `M4[28]` and `M4[29]`.

For multivariate forecasting, we will use the EUStockMarkets data set from the **datasetes** [26]. It contains a set of DAX, SMI, CAC, and FTSE European stock indices from July 1st, 1991 to August 24th, 1998 totaling 1828 training observations and 32 test observations.



Note that each stock index tends to follow a similar, but diverging, trend. This will be important to note when we perform windowing cross validation as it will let us see how well the models adapt to what appears to be nonstationary data.

3 Univariate and Multivariate Forecasting Tasks

mlr 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 most of it's arguments from `makeRegrTask`, but has two noticable differences in arguments.

data: Instead of a data frame, an xts object from **xts** [30] containing the time series.

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, daily data with a yearly periodicity has a frequency of 365, and weekly data with a yearly frequency has a periodicity of 52.

```
library(mlr)
climate.task = makeForecastRegrTask(id = "M4 Climate Data",
                                    data = m4.train,
                                    target = "target_var",
```

```
                                  frequency = 183L)
climate.task

## Task: M4 Climate Data
## Type: fcregr
## Target: target_var
## Observations: 640
## Dates:
##  Start: 2007-12-05
##  End:   2009-09-04
## Frequency: 183
## Features:
## numerics  factors  ordered
##        0        0        0
## Missings: FALSE
## Has weights: FALSE
## Has blocking: FALSE
```

Like a regression task, this records the type of the learning problem and basic information about the data set such as the start and end dates, frequency, and whether we have missing values. Note that there are zero features in our task because we only have a target variable, which the model itself will use to build features.

One common problem with forecasting is that it is difficult to use additional explanatory variables or forecast multiple targets that are dependent on one another. If we are at time t and want to forecast 10 periods in the future, we need to know the values of the explanatory variables at time $t + 10$, which is often not possible. A new set of models [24] which treats explanatory variables endogenously instead of exogenously allows us to forecast not only our target, but addititional explanatory variables. This is done by treating all the variables as targets, making them endogeneous to the model. To use these models, we create a multivariate forecasting task. The function `makeMultiForecastRegrTask()` has the same arguments as `makeForecastRegrTask()` with one exception. The `target` argument can contain either a single target variable, multiple target variables, or `All` which treats all variables endogeneously.

```
mfcregr.univar.task = makeMultiForecastRegrTask(id = "bigvar",
                                                data = EuStockMarkets,
                                                target = "FTSE",
                                                frequency = 365L)

## Error in eval(expr, envir, enclos):  could not find function "makeMultiForecastRegrTask"

mfcregr.univar.task

## Error in eval(expr, envir, enclos):  object 'mfcregr.univar.task' not found
```

Like `makeForecastRegrTask()`, `mfcregr.univar.task` we get the standard output, but notice now that there are three three features. Alternatively, `mfcregr.all.task` contains multiple target values with no features. The difference between each of these multivariate tasks is that `mfcregr.univar.task` will act similar to `makeForecastRegrTask()`, giving

the output, predictions, and even using the measures for univariate forecasting tasks. Both of these tasks will still forecast all of the underlying series, which allows us take exogeneous models and treat them endogeneously for multi-step forecasts.

```
mfcregr.all.task = makeMultiForecastRegrTask(id = "bigvar",
                                             data = eu.train,
                                             target = "all",
                                             frequency = 365L)

## Error in eval(expr, envir, enclos):  could not find function "makeMultiForecastRegrTask"

mfcregr.all.task

## Error in eval(expr, envir, enclos):  object 'mfcregr.all.task' not found
```

4 Building and Tuning a forecast learner

4.1 Univariate Forecasting

The `makeLearner()` function provides a structured model building framework to the several forecasting models currently implimented in **mlr**. As an example, we will build the Trigonometric exponential smoothing state space model with Box-Cox transformation, ARMA errors, Trend and Seasonal Components (TBATS) [21]. To creae the original BATS model we let y_t^ω be a box cox transformed observation with parameter ω. Then let m_T be the seasonal periods, l_t be the local level in the period, b the long term trend with b_t being the short term trend, s_t^i being the ith seasonal component, d_t being an ARIMA(p, q) model with gaussian white noise process ϵ_t. Smoothing parameters are given by α, β, and γ and ϕ is the damping constant of the trend.

$$y_t^\omega = \begin{cases} \frac{y_t^\omega - 1}{\omega}, \omega \neq 0 \\ log(y_t), \omega = 0 \end{cases} \tag{1}$$

$$y_t^\omega = l_{t-1} + \phi b_{t-1} + \sum_{i=1}^{T} s_{t-m_i}^i + d_t \tag{2}$$

$$l_t = l_{t-1} + \phi b_{t-1} + \alpha d_t \tag{3}$$

$$b_t = (1 - \phi)b + \phi b_{t-1} + \beta d_t \tag{4}$$

$$s_t^i = s_{t-m_i}^i + \gamma_i d_t \tag{5}$$

$$d_t = \sum_{i=1}^{p} \psi_i d_{t-i} + \sum_{i=1}^{q} \theta_i \epsilon_{t-i} + \epsilon \tag{6}$$

The trigonometric part of the model comes from the representation of the seasonal components based on fourier series [31]. Let the stochastic level of the ith seasonal component be $s_{j,t}^{(i)}$ and the stochastic growth in the level of the ith seasonal component that is needed to describe the change in the seasonal component over time by $s_{j,t}^{*(i)}$.

$$s_t^{(i)} = \sum_{j=1}^{k_i} s_{j,t}^{(i)} \tag{7}$$

$$s_{j,t}^{(i)} = s_{j,t-1}^{(i)} \cos \lambda_j^{(i)} + s_{j,t-1}^{(i)} \sin \lambda_j^{(i)} + \gamma_1^{(i)} d_t \tag{8}$$

$$s_{j,t}^{*(i)} = -s_{j,t-1} \sin \lambda_j^{(i)} + s_{j,t-1}^{*(i)} \cos \lambda_j^{(i)} + \gamma_2^{(i)} d_t \tag{9}$$

The smoothing parameters are defined by $\gamma_1^{(i)}$ and $\gamma_2^{(i)}$ with $\lambda_j^{(i)} = 2\pi j/m_i$ being a trigonometric smoothing parameter associated with the seasonal periods. The parameter k_i is number of harmonics necessary in the ith seasonal component. It's possible to show a simpler deterministic representation of the seasonal components by setting the smoothing parameters to zero. The TBATS model is created by replacing the seasonal component $s_t^{(i)}$ in equation 5 with the trigonometric seasonal equations as well as replacing the measure equation in 2 with

$$y_t^{\omega} = l_{t-1} + \phi b_{t-1} + \sum_{i=1}^{T} s_{t-1}^{(i)} + d_t \tag{10}$$

TBATS is one of the most well known forecasting models and is available in mlr along with models such as BATS, ARIMA, 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 the TBATS model we use `makeLearner()`, supplying the class of learner, order, the number of steps to forecast, and any additional arguments to be passed to `tbats` for **forecast**.

```
tbats.mod =makeLearner("fcregr.tbats", use.box.cox = TRUE,
                       use.trend = TRUE,
                       seasonal.periods = TRUE, max.p = 60, max.q = 60,
                       stationary = FALSE, use.arma.errors = TRUE,
                       h = 35, predict.type = "response")
```

We can also supply a predict type for forecasting models to either receive point estimates (`response`) or point estimates with quantiles of confidence intervals (`quantile`). To train the model we simply call train, supplying the forecasting model and task. After training the model it's simple to get our forecasts by calling `predict()` with the test data, returning an object containing meta information for the forecasts along with the prediction and test data in columns `truth` and `response`, respectively.

```
train.tbats= train(learner = tbats.mod, task = climate.task )
train.tbats

## Model for learner.id=fcregr.tbats; learner.class=fcregr.tbats
## Trained on: task.id = M4 Climate Data; obs = 640; features = 0
## Hyperparameters: use.box.cox=TRUE,use.trend=TRUE,seasonal.periods=TRUE,max.p=60,max.q=60,stati

predict.tbats = predict(train.tbats, newdata = m4.test)
```

To measure the performane of TBATS we call `performance()` with the Mean Absolute Scaled Error (MASE) [19] measure.

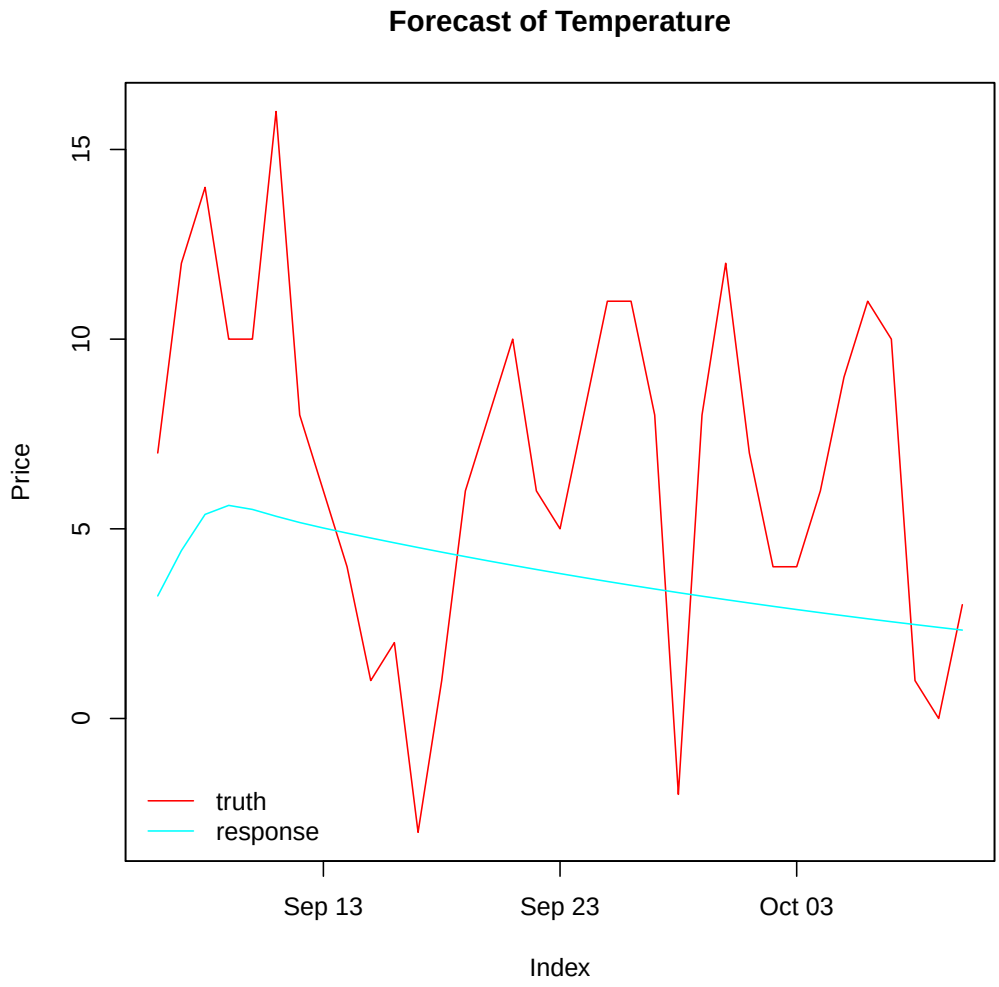
```
performance(predict.tbats, mase, task = climate.task)

##      mase
## 0.0676601
```

MASE has favorable properties for calculating forecast errors relative to measures such as mean root mean squared error or median relative absolute error. Arguably one of the most important features, it's very interpretable. Let y_t and $\tilde{y_t}$ be the target variable and prediction at time t until the final time T with $\epsilon_t = y_t - \tilde{y_t}$ being the forecast error.

$$\text{MASE} = \frac{\sum_{t=1}^{T} |\epsilon_t|}{\frac{T}{T-1} \sum_{t=2}^{T} |y_t - y_{t-1}|} \tag{11}$$

Where the denominator is the one step ahead naive forecast from the training data. When the numerator is equal to the denominator the model performed as good as a simple naive forecast method. Scores greater than one mean you are performing worse and scores less than one mean you are performing better.

The scale invariance of MASE means that it is independent of the scale of the data which allows models to be compared across data sets. The scale invariance of MASE has made it a favorite for comparing the accuracy of forecasts methods [8]. While scaling in measures such as the Mean Absolute Percentage Error can cause poor behavior as the target variable goes to zero, MASE does not become skewed when the target variable goes to zero. This allows MASE to be use in situations in which zeros occur frequently or zero is not meaningful such as predicting temperature.



Need to talk about results

Because the forecasts are only for the next 35 periods it's useful to be able to update the model continuously without retraining the model each time we want new forecasts. Univariate forecasting models in **mlr** can be updated using `updateModel()`.

```
update.tbats = updateModel(train.tbats, climate.task, newdata = m4.test)
update.tbats

## Model for learner.id=fcregr.tbats; learner.class=fcregr.tbats
## Trained on: task.id = M4 Climate Data; obs = 35; features = 0
## Hyperparameters: use.box.cox=TRUE,use.trend=TRUE,seasonal.periods=TRUE,max.p=60,max.q=60,stati

predict(update.tbats, task = climate.task)

## Prediction: 35 observations
```

```
## predict.type: response
## threshold:
## time: 0.00
##                     response
## 2009-09-04 23:59:54 6.319959
## 2009-09-05 23:59:48 7.875401
## 2009-09-06 23:59:43 8.209328
## 2009-09-07 23:59:37 8.157638
## 2009-09-08 23:59:31 8.083591
## 2009-09-09 23:59:26 8.055280
## ... (35 rows, 1 cols)
```

4.2 Multivariate Forecasting

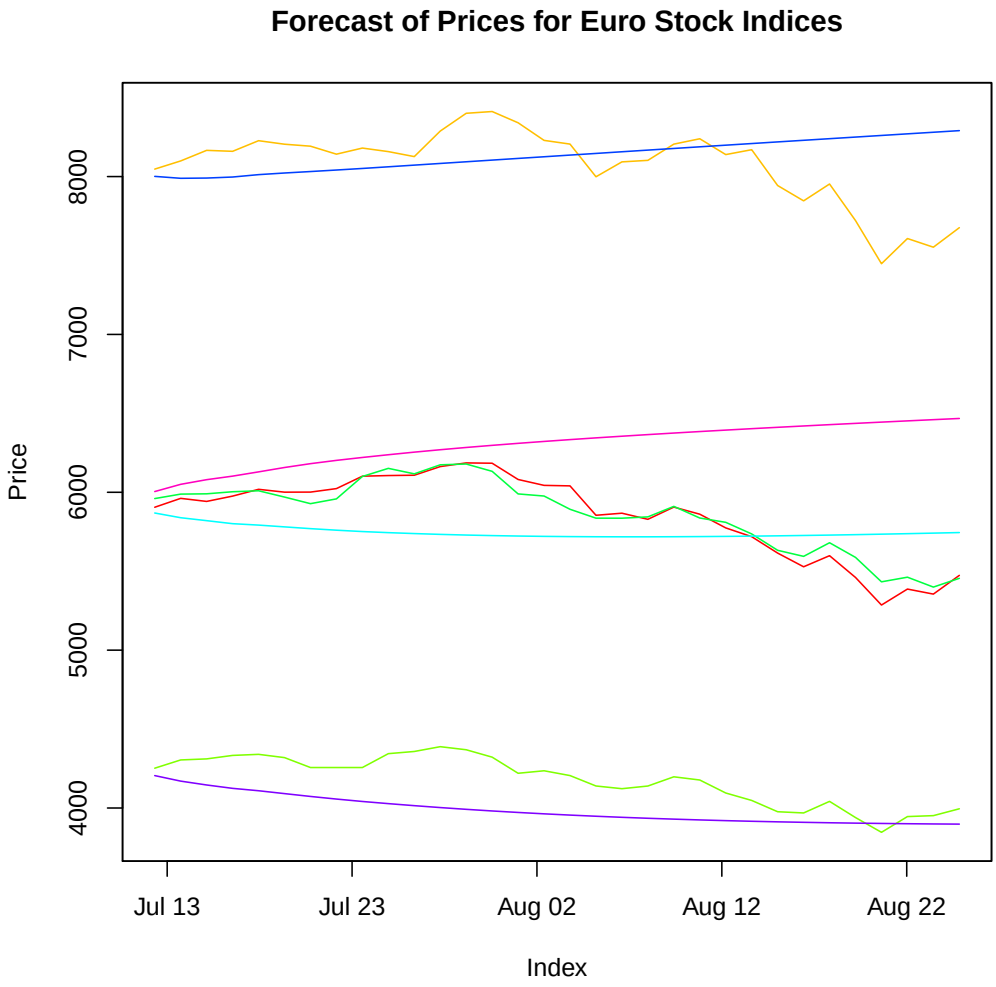
Need to write things for this

```
bigvar.mod = makeLearner("mfcregr.BigVAR",p = 5, struct = "SparseLag",
                         gran = c(50, 60),h = 35, n.ahead = 35)
```

```
train.bigvar = train(learner = bigvar.mod, task = mfcregr.all.task )
train.bigvar
```

```
## Model for learner.id=mfcregr.BigVAR; learner.class=mfcregr.BigVAR
## Trained on: task.id = bigvar; obs = 1828; features = 0
## Hyperparameters: p=5,struct=SparseLag,gran=50,60,h=35,n.ahead=35
```

```
predict.bigvar = predict(train.bigvar, newdata = eu.test)
performance(predict.bigvar, multivar.mase, task = mfcregr.all.task)

## multivar.mase
##     0.2340861
```



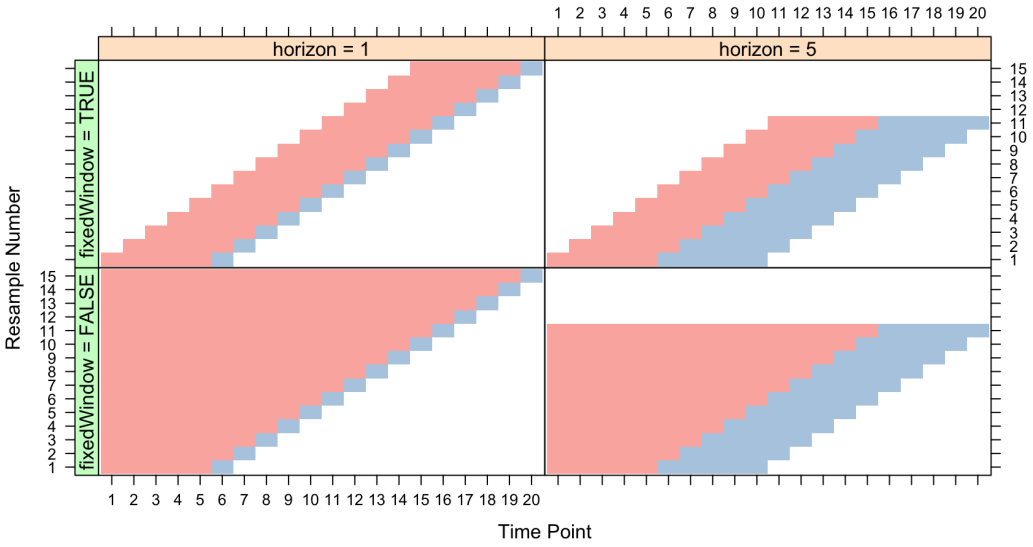
5 Resampling with Time

While TBATS is one of the most well known time series models, the order selection process or the ARIMA errors and whether to include trend, damped trend, or seasonal periods can be a subjective process that makes finding the best model difficult for users. One of the first proposals for automated forecasting methods comes from [7] for automatic order selection of ARIMA models. Innovations are obtained by fitting high order autoregressive models 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 [28] and Autobox [27] are well known and have performed to high standards in competitions such as the M3 forecasting competition [22]. One of the most well known R packages for automated forecast is **forecast** [18] 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. Resampling schemes such as cross-validation, bootstrapping, etc. are common in machine learning for dealing with the bias-variance tradeoff [9] [29]. 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. Figure one gives an example of fixed and growing windows. 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 [17] are now available in the `resampling()` function of **mlr**.

Figure 1: Resampling with a window scheme as exampled by caret [20]. The top graphs are fixed window cross validation while the bottom graphs are growing window cross validation.



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 argument to skip over some windows for the sake of time.

```
resampDesc = makeResampleDesc("GrowingCV", horizon = 35L,
                              initial.window = .7,
                              size = nrow(getTaskData(climate.task)),
                              skip = .02)
resampDesc

## Window description:
##  growing with 14 iterations:
##  448 observations in initial window and 35 horizon.
## Predict: test
## Stratification: FALSE
```

To make a parameter set to tune over **mlr** uses **ParamHelpers** [4]. There are several types of tools to help us search our parameter space including grid search, random search [3], 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 = 10, lower = 0),
                      makeIntegerParam(id = "start.p", upper = 10, lower = 1,
                                       trafo = function(x) x*2),
                      makeIntegerParam(id = "max.q", upper = 20, lower = 10),
                      makeIntegerParam(id = "start.q", upper = 5, lower = 0,
                                       trafo = function(x) x*2),
                      makeIntegerParam("max.P", lower = 0, upper = 3),
                      makeIntegerParam("max.Q", lower = 0, upper = 2),
                      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)
                      )

#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 [19] as a measure of performance [1].

```
#
library("parallelMap")
parallelStartSocket(6)
configureMlr(on.learner.error = "warn")
set.seed(1234)
tbatsTune = tuneParams(makeLearner("fcregr.tbats", h = 35),
                       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.

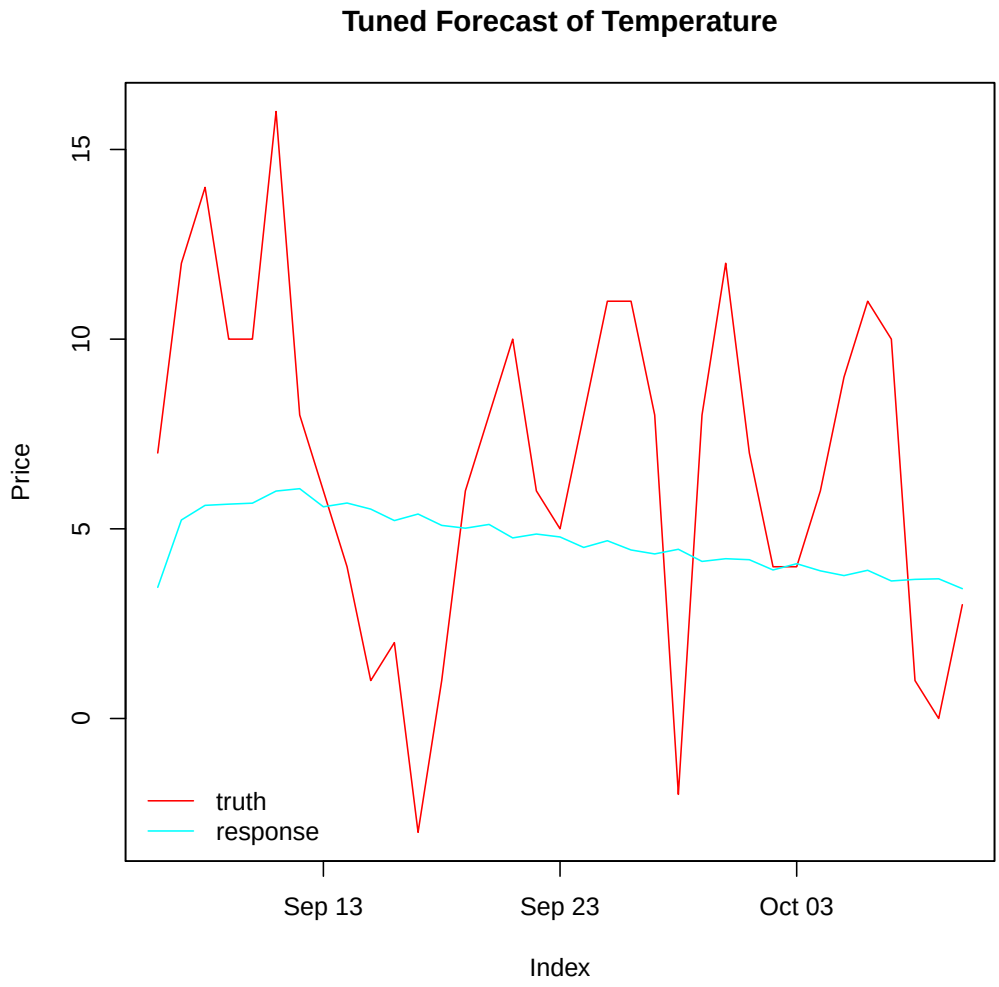
```
lrn = setHyperPars(makeLearner("fcregr.tbats", h = 35),
                   par.vals = tbatsTune$x)
m = train(lrn, climate.task)
```

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

```
climate.pred = predict(m, newdata = m4.test)
performance(climate.pred, measures = mase, task = climate.task)
```

```
##       mase
## 0.06145902
```

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

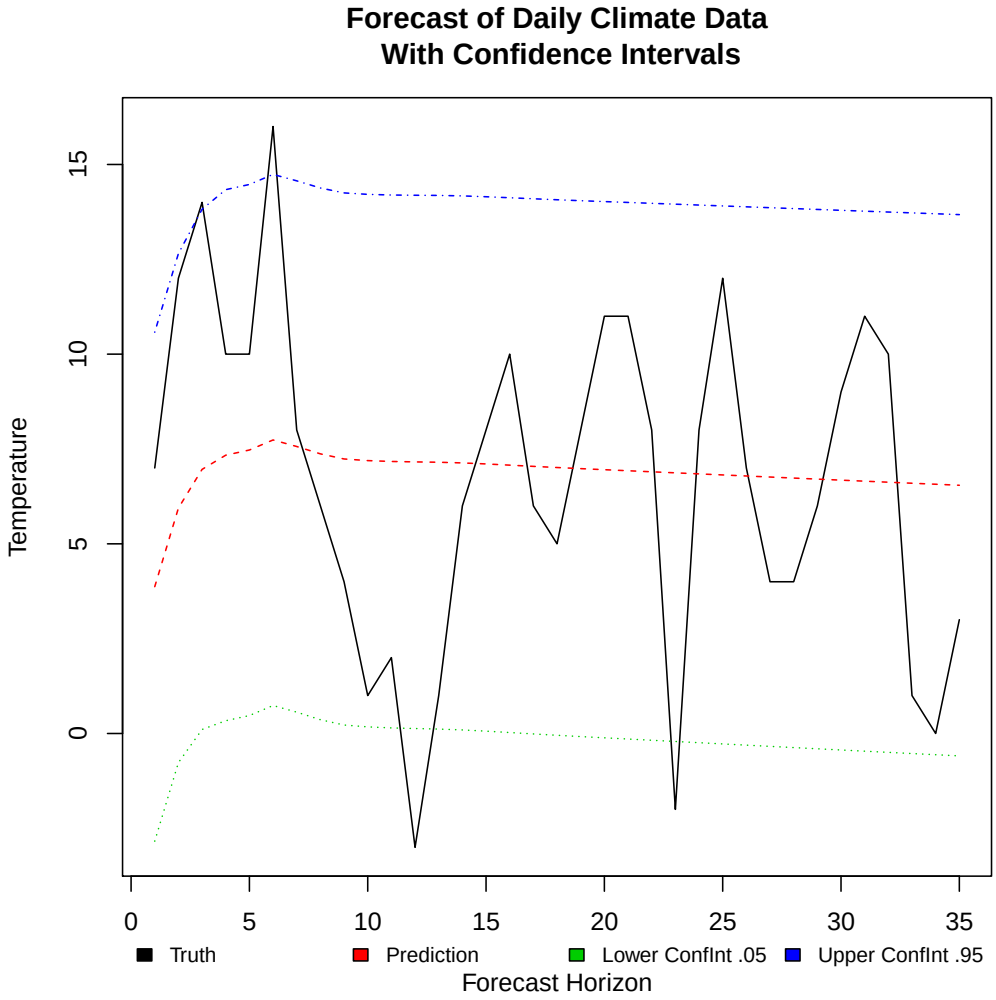



```
par_set = makeParamSet(
  makeDiscreteParam(id = "model", values = c("sGARCH", "csGARCH")),
  makeIntegerVectorParam(id = "garchOrder", len = 2L, lower = c(1,1),
                         upper = c(4,4)),
  makeIntegerVectorParam(id = "armaOrder", len = 2L, lower = c(5,1),
                       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



6 Forecasting with Machine Learning Models

6.1 Forecasting with Regression Tasks

The forecasting extension of **mlr** includes a preprocessing function that allows supervised machine learning models.

```
climate.regr.task = makeRegrTask(id = "lagged gbm",
                                 data = as.data.frame(m4.train),
                                 target = "target_var")
climate.task.lag = createLagDiffFeatures(climate.regr.task,
                                         lag = 1L:100L, difference = 1L,
                                         seasonal.lag = 1L:2L, na.pad=FALSE)
```

```
climate.task.lag

## Supervised task: lagged gbm
## Type: regr
## Target: target_var
## Observations: 539
## Features:
## numerics  factors  ordered
##      100        0        0
## Missings: FALSE
## Has weights: FALSE
## Has blocking: FALSE
```

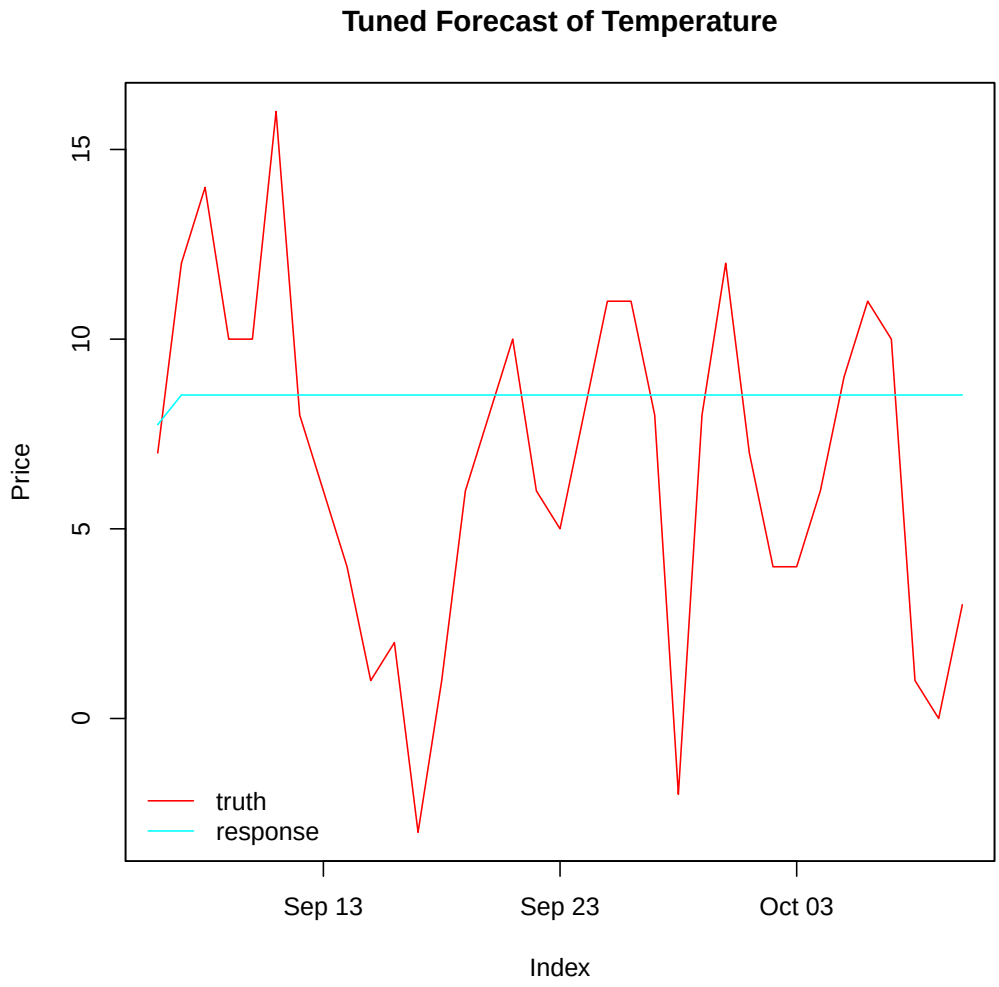
Notice that `createLagDiffFeatures()` returns a new task with the lagged variables as the new features. Once the lagged task is created the model is trained or tuned like any other.

```
lag.gbm = makeLearner("regr.gbm", par.vals = list(n.trees = 100))
gbm.train = train(lag.gbm, climate.task.lag)
```

The `forecast()` function allows machine learning models to do arbitary n-step ahead forecasts[2].

```
m4.test.df = as.data.frame(m4.test,row.names = index(m4.test))
gbm.forecast = forecast(gbm.train, h = 35L,
                        newdata = m4.test.df)
performance(gbm.forecast,mase,climate.regr.task)

##      mase
## 0.0571994
```

[2]Maybe explain what that means?



6.2 Forecasting with Classification Tasks

For developing trading strategies, we normally have a discrete set of choices such as to buy, sell, or hold onto a stock. Using forecasting in mlr we can now train classification models that forecast these choices. To example this, a simple buy, sell, or hold trading strategy will be built Using the `EuStockMarkets`'s DAX index. If the stock goes up by 5% in a day we will buy, down 5% we will sell, and otherwise we will hold onto the current stocks we have.

```
DAX = EuStockMarkets$DAX/lag(EuStockMarkets$DAX,
                             7,nap.pad = FALSE) - 1
trade.strat = ifelse(DAX > .02, "Buy",
                     ifelse(DAX < -.01, "Sell", "Hold"))
trade.strat = trade.strat[8:1860]
euro.classif.data = data.frame(trade.strat = trade.strat ,
```

```
                                row.names = index(trade.strat))

euro.classif.train = euro.classif.data[1:1821,,drop = FALSE]
euro.classif.test  = euro.classif.data[1822:1853,,drop = FALSE]
classif.task = makeClassifTask(data = euro.classif.train,
                                target = "DAX")
classif.task.lag = createLagDiffFeatures(classif.task,
                                         lag = 1L:365L,
                                         na.pad = FALSE)
classif.learn = makeLearner("classif.boosting", xval = 1,
                            mfinal = 200, minsplit = 10)
```

```
classif.train = train(classif.learn, classif.task.lag)
classif.fc = forecast(classif.train, h=32, newdata = euro.classif.test)
performance(classif.fc)
```

```
## Prediction: 32 observations
## predict.type: response
## threshold:
## time: 125.91
##   truth response
## 1   Buy      Buy
## 2   Buy      Buy
## 3  Hold      Buy
## 4  Hold      Buy
## 5  Hold      Buy
## 6   Buy      Buy
## ... (32 rows, 2 cols)
##   mmce
## 0.6875
```

7 Lambert W Transforms

Many machine learning and time series models rely one the assumption that our data or errors fit a normal distribution. This assumption becomes precarious when modeling the asymmetric and fat-tailed data of the real world. Lambert W Transforms are a family of generalized skewed distributions [12] that have bijecetive and parametric functions that allow heavy tailed and asymettric data to appear more Gaussian [13].

Let U be a continuous random variable with cdf $F_U(u|\beta)$ and pdf $f_U(u|\beta)$ given β is a parameter vector. Define a continuous location-scale random variable $X \sim F_X(x|\beta)$. A locaton-scale skewed Lambert $W \times F_X$ random variable is defined as

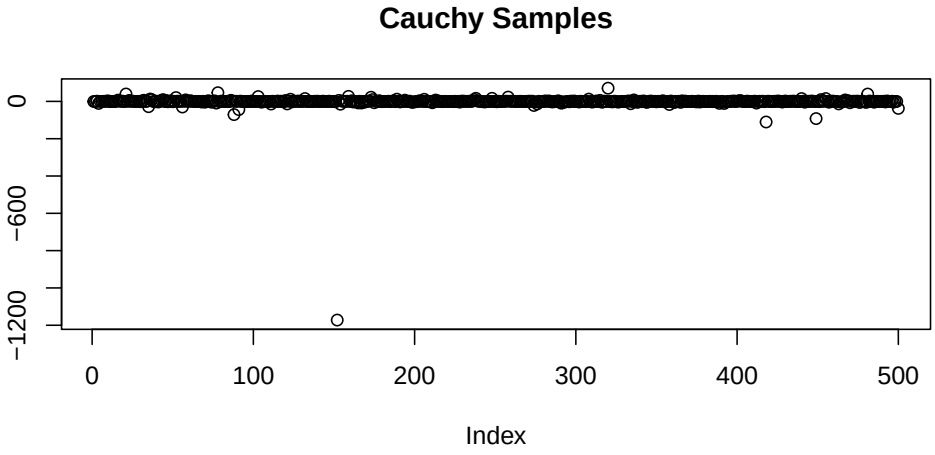
$$Z = U \exp\left(\frac{\delta}{2}(U^2)\right), \ \delta \geq 0 \tag{12}$$

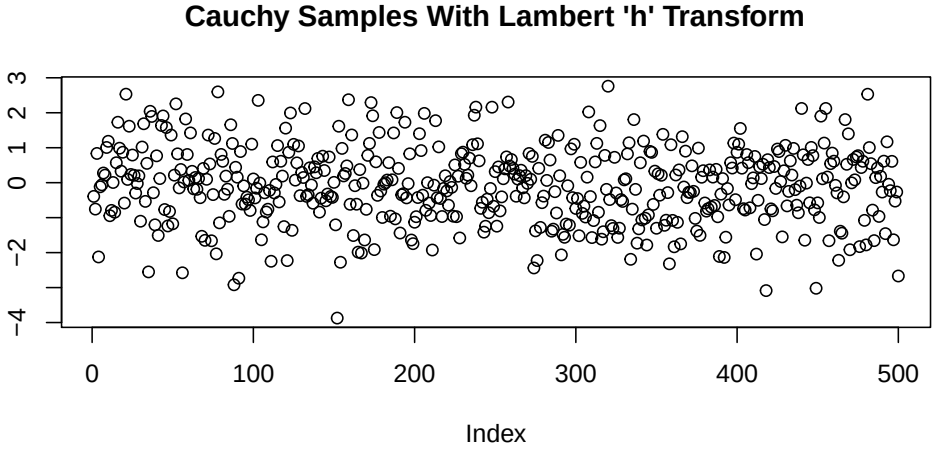
And the heavy-tailed Lambert $W \times F_X$ random variable can be defined as

$$Z = U \exp\left(\frac{\delta}{2}(U^2)^{\alpha}\right),\ \delta \geq 0\ \alpha > 0 \tag{13}$$

Given that $U = (X - \mu_X)/\sigma_X$ where μ_X, σ_X, δ, and α are the mean and standard deviation of X and the parameters to control skewness and asymetry , respectively. When $\delta = 0$, equation 12 reduces to a standard normal distribution. Equation 12 is the general form of Tukey's h distribution [16] and the basis for Morgenthaler and Tukey's [23] skewed, heavy tailed family of hh random variables.

$$Z = \begin{cases} U \exp\left(\frac{\delta_l}{2}(U^2)_l^{\alpha}\right),\ \delta_l \geq 0\ \alpha_l > 0 \\ U \exp\left(\frac{\delta_r}{2}(U^2)_r^{\alpha}\right),\ \delta_r \geq 0 \alpha_r > 0 \end{cases} \tag{14}$$





The function `Gaussianize` is available in the package **LambertW** and has been made into a preprocessing function in **mlr**.

```
# Need to make this more dramatic
lamb.lrn = makePreprocWrapperLambert("classif.lda", type = "h")
lamb.lrn

## Learner classif.lda.preproc from package MASS
## Type: classif
## Name: ; Short name:
## Class: PreprocWrapperLambert
## Properties: numerics,factors,prob,twoclass,multiclass
## Predict-Type: response
## Hyperparameters: type=h,methods=IGMM,verbose=FALSE

lamb.trn = train(lamb.lrn,iris.task, subset = 1:120)
lamb.pred = predict(lamb.trn, iris.task, subset = 121:150)

# Do the non-LW version
trn = train(makeLearner("classif.lda"),iris.task, subset = 1:120)
pred = predict(trn, iris.task, subset = 121:150)
performance(lamb.pred)

## mmce
##  0.1

performance(pred)

## mmce
##  0.1
```

8 Ensembles of Forecasting Models

```
resamp.sub = makeResampleDesc("GrowingCV",
                          horizon = 35L,
                          initial.window = .90,
                          size = nrow(getTaskData(climate.task)),
                          skip = .01
                          )

lrns = makeLearners(c("fcregr.tbats","fcregr.garch", "fcregr.arfima"))


stack.forecast = makeStackedLearner(base.learners = lrns,
                        predict.type = "response",
                        method = "average")

# Simple param set for tuning sub learners
ps = makeParamSet(
```

```
  makeDiscreteParam("fcregr.tbats.h", values = 32),
  makeDiscreteParam("fcregr.garch.n.ahead", values = 32),
  makeDiscreteParam("fcregr.arfima.h", values = 32),
  makeDiscreteParam("fcregr.arfima.estim", values = "ls"),
  makeIntegerParam("fcregr.arfima.max.P", lower = 0, upper = 3),
  makeIntegerParam("fcregr.arfima.max.Q", lower = 0, upper = 2),
  makeDiscreteParam("fcregr.arfima.ic", values = c("aicc","aic","bic")),
  makeDiscreteParam(id = "model", values = c("sGARCH", "csGARCH")),
  makeIntegerVectorParam(id = "fcregr.garch.garchOrder",
                         len = 2L, lower = c(1,1),
                         upper = c(4,4)),
  makeIntegerVectorParam(id = "fcregr.garch.armaOrder",
                         len = 2L, lower = c(5,1),
                         upper = c(8,3)),
  makeLogicalParam(id = "fcregr.garch.archm"),
  makeDiscreteParam(id = "fcregr.garch.distribution.model",
                    values = c("norm","std")),
  makeLogicalParam(id = "fcregr.tbats.use.box.cox",
                                default = FALSE,tunable = TRUE),
  makeIntegerParam("fcregr.tbats.max.P", lower = 0, upper = 3),
  makeIntegerParam("fcregr.tbats.max.Q", lower = 0, upper = 2),
  makeDiscreteParam("fcregr.tbats.ic", values = c("aicc","aic","bic")),
  makeDiscreteParam("fcregr.tbats.test", values = c("kpss","adf","pp"))
)

## tuning
parallelStartSocket(6)
configureMlr(on.learner.error = "warn")
set.seed(1234)
fore.tune = tuneParams(stack.forecast, fcregr.task, resampling = resamp.sub,
                   par.set = ps, control = makeTuneControlGrid(),
                   measures = mase, show.info = FALSE)
parallelStop()
fore.tune

# get hyper params
stack.forecast.tune  = setHyperPars2(stack.forecast,fore.tune$x)
stack.forecast.tune

# Train the final best models and predict
stack.forecast.mod = train(stack.forecast.tune,fcregr.task)
stack.forecast.pred = predict(stack.forecast.mod, newdata = dat.test)
stack.forecast.pred
```

References

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

[2] Souhaib BenTaieb. *M4comp: Data from the M4 Time Series Forecasting Competition*, 2016. R package version 0.0.1.

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

[4] 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.

[5] 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.

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

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

[8] Philip Hans Franses. A note on the Mean Absolute Scaled Error. *International Journal of Forecasting*, 32(1):20–22, 2016.

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

[10] 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.

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

[12] Georg M. Goerg. Lambert w random variables - a new family of generalized skewed distributions with applications to risk estimation. *Annals of Applied Statistics*, 5(3):2197–2230, 2011.

[13] Georg M. Goerg. The lambert way to gaussianize heavy-tailed data with the inverse of tukey's h transformation as a special case. *The Scientific World Journal: Special Issue on Probability and Statistics with Applications in Finance and Economics*, 2015(2015):16, 2015.

[14] 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.

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

[16] David C. Hoaglin. *Summarizing Shape Numerically: The g-and-h Distributions*, pages 461–513. John Wiley Sons, Inc., 2006.

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

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

[19] 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.

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

[21] Alysha M. De Livera, Rob J. Hyndman, and Ralph D. Snyder. Forecasting time series with complex seasonal patterns using exponential smoothing. *Journal of the American Statistical Association*, 106(496):1513–1527, 2011.

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

[23] Stephan Morgenthaler and John W. Tukey. Fitting quantiles: Doubling, hr, hq, and hhh distributions. *Journal of Computational and Graphical Statistics*, 9(1):180–195, 2000.

[24] Will Nicholson, David Matteson, and Jacob Bien. *BigVAR: Dimension Reduction Methods for Multivariate Time Series*, 2016. R package version 1.0.1.

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

[26] R Core Team. *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing, Vienna, Austria, 2016. The data were kindly provided by Erste Bank AG, Vienna, Austria.

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

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

[29] 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.

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

[31] Mike West and Jeff Harrison. *Bayesian Forecasting and Dynamic Models (2Nd Ed.)*. Springer-Verlag New York, Inc., New York, NY, USA, 1997.

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