Time Series Analysis Methods and Applications

by Igor Baranov, Michael Parravani, Ariana Biagi, Hui Fang Cai

Abstract The goal of this project is to discover Time Series analysis. We start with the data loading and creation of different types of object including ts, xts and zoo and proceed with data manipulation, conversion and visualization. Then we discover different algorithms of time series analysis like decomposition, forecasting and clustering. At the end we develop a practical example and build a Shiny application.

Introduction

The goal of this project is to discover Time Series analysis. We start with the data loading and creation of different types of object including ts, xts and zoo and proceed with data manipulation, conversion and visualization. Then we discover different algorithms of time series analysis like decomposition, forecasting and clustering. At the end we develop a practical example and build a Shiny application.

Background

The function **ts** for the core package **stats** (R Core Team, 2012) is used to create time-series objects. These are vector or matrices with class of "ts" (and additional attributes) which represent data which has been sampled at equispaced points in time. In the matrix case, each column of the matrix data is assumed to contain a single (univariate) time series. Time series must have at least one observation, and although they need not be numeric there is very limited support for non-numeric series.

An xts object from package **xts** (Ryan and Ulrich, 2018) extends the S3 class **zoo** from the package of the same name (Zeileis and Grothendieck, 2005). Package **zoo** is the creator for an **S3** class of indexed totally ordered observations which includes irregular time series.

Similar to zoo objects, xts objects must have an ordered index. While zoo indexes cannot contain duplicate values, xts objects have optionally supported duplicate index elements since version 0.5-0. The xts class has one additional requirement, the index must be a time-based class. Currently supported classes include: 'Date', 'POSIXct', 'timeDate', as well as 'yearmon' and 'yearqtr' where the index values remain unique.

Ethical Consideration for the Time Series ML Framework

As the goal of this report is only to research the time series methods, many aspects of the ethical ML framework do not directly apply. The time series data used here is open source and we can assume was collected in transparent ways. That being said, there is likely a large segment of the populace that can't take advantage of analysis presented in this report - we assume a low income population with limited access to internet and not STEM educated. If the outcome of this system were to be of more social impact, this would need to be use more appropriate datasets, analysis of which could be more advantageous to that population.

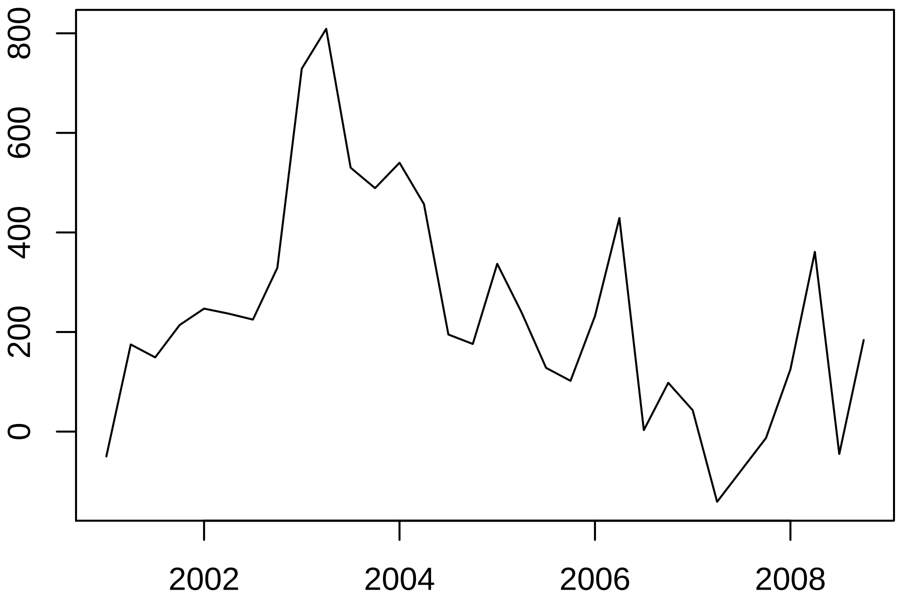

Figure 1: Time Series ts class Example Plot

Time Series Data Manipulating and Visualizing

Constructing TS object

In the following example (Fig 1) we construct and plot a simple TS class:

```
simpleTS <- c(-50, 175, 149, 214, 247, 237, 225, 329, 729, 809,
       530, 489, 540, 457, 195, 176, 337, 239, 128, 102, 232, 429, 3,
       98, 43, -141, -77, -13, 125, 361, -45, 184)
simpleTS <- ts(simpleTS, start = c(2001, 1), end = c(2008, 4), frequency = 4)
print(simpleTS)
```

```
#>      Qtr1 Qtr2 Qtr3 Qtr4
#> 2001  -50  175  149  214
#> 2002  247  237  225  329
#> 2003  729  809  530  489
#> 2004  540  457  195  176
#> 2005  337  239  128  102
#> 2006  232  429    3   98
#> 2007   43 -141  -77  -13
#> 2008  125  361  -45  184
```

```
plot(simpleTS, ylab="", xlab="")
```

Loading Stock Data

There are many ways to load times data, but the fastest is to use **zoo** that has many convinient utilities to manipulate times series data. This is especially convinient when dealing with complex stock data.

This is an example of loading and presenting of stock data using Historical stock data for all current S&P 500 companies. Stock market data can be interesting to analyze and as a further incentive, strong predictive models can have large financial payoff. Data has the following columns:

- Date - in format: yy-mm-dd
- Open - price of the stock at market open (this is NYSE data so all in USD)
- High - Highest price reached in the day
- Low Close - Lowest price reached in the day
- Volume - Number of shares traded
- Name - the stock's ticker name

First we load the whole dataset as data frame:

```
stocks5 <- read_csv(file="../../../data/all_stocks_5yr.csv.zip",col_names = TRUE)
stocks5$Name <- as.factor(stocks5$Name)
head(stocks5$Name)
```

```
#> [1] AAL AAL AAL AAL AAL AAL
#> 505 Levels: A AAL AAP AAPL ABBV ABC ABT ACN ADBE ADI ADM ADP ADS ... ZTS
```

As the name suggests we should have 500 (505 to be correct) stock names in the dataset. The code below extracts stock data of AAL (American Airlines Group Inc), converts it to **zoo** object and plots it as multi-variate time series (Fig 2).

```
stocks5_aal <- stocks5[stocks5$Name=="AAL",c(1:6)]
stocks5_aal <- zoo(stocks5_aal[,2:6], stocks5_aal$date)
print(paste("Start date: ", start(stocks5_aal)))
```

```
#> [1] "Start date:  2013-02-08"
```

```
print(paste("Last date: ", end(stocks5_aal)))
```

```
#> [1] "Last date:  2018-02-07"
```

```
str(stocks5_aal)
```

```
#> 'zoo' series from 2013-02-08 to 2018-02-07
#>   Data: num [1:1259, 1:5] 15.1 14.9 14.4 14.3 14.9 ...
#>  - attr(*, "dimnames")=List of 2
#>   ..$ : NULL
#>   ..$ : chr [1:5] "open" "high" "low" "close" ...
#>   Index:  Date[1:1259], format: "2013-02-08" "2013-02-11" "2013-02-12" "2013-02-13" "2013-02-14" ...
```

```
plot(stocks5_aal, xlab = "", nc = 1, main = "")
```

Additional time series data sets

Yahoo Science labeled time series

Yahoo Science labeled time series is a big Synthetic and real time-series with labeled anomalies, it should be downloaded and used locally. Yahoo Science labeled time series

NAB Data Corpus

NAB Data Corpus is a large dataset of ordered, timestamped, single-valued metrics. All data files contain anomalies, unless otherwise noted. This data better be loaded from github directly to R script.

Time Series Methods Showcase

Time series decomposition

Time series decomposition is to decompose a time series into trend, seasonal, cyclical and irregular components. Frequency represents data which has been sampled at equispaced points in time:

- frequency=7: a weekly series
- frequency=12: a monthly series
- frequency=4: a quarterly series

To decompose a time series into components:

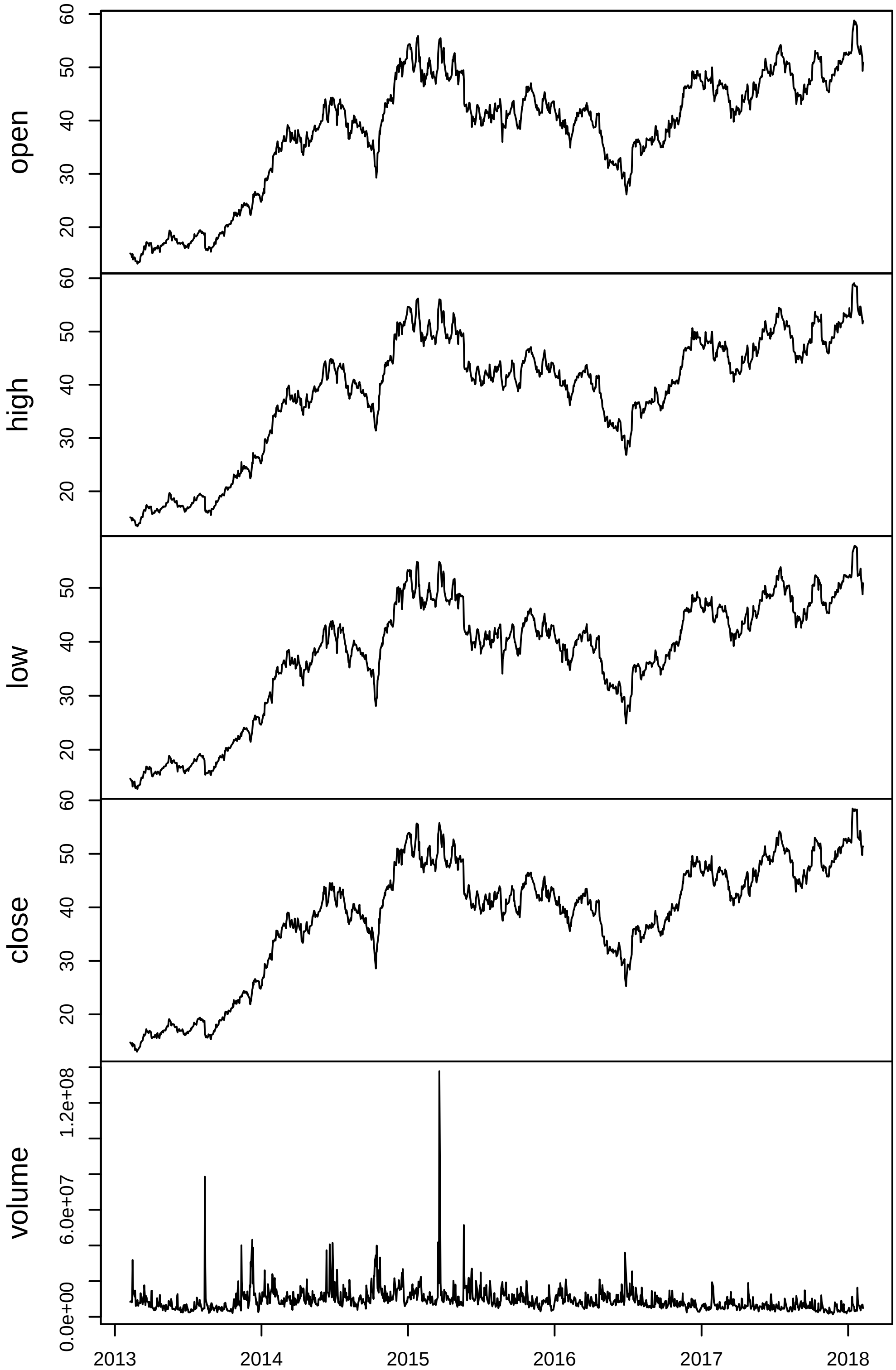

Figure 2: Multi-variate time series visualization of AAL stock

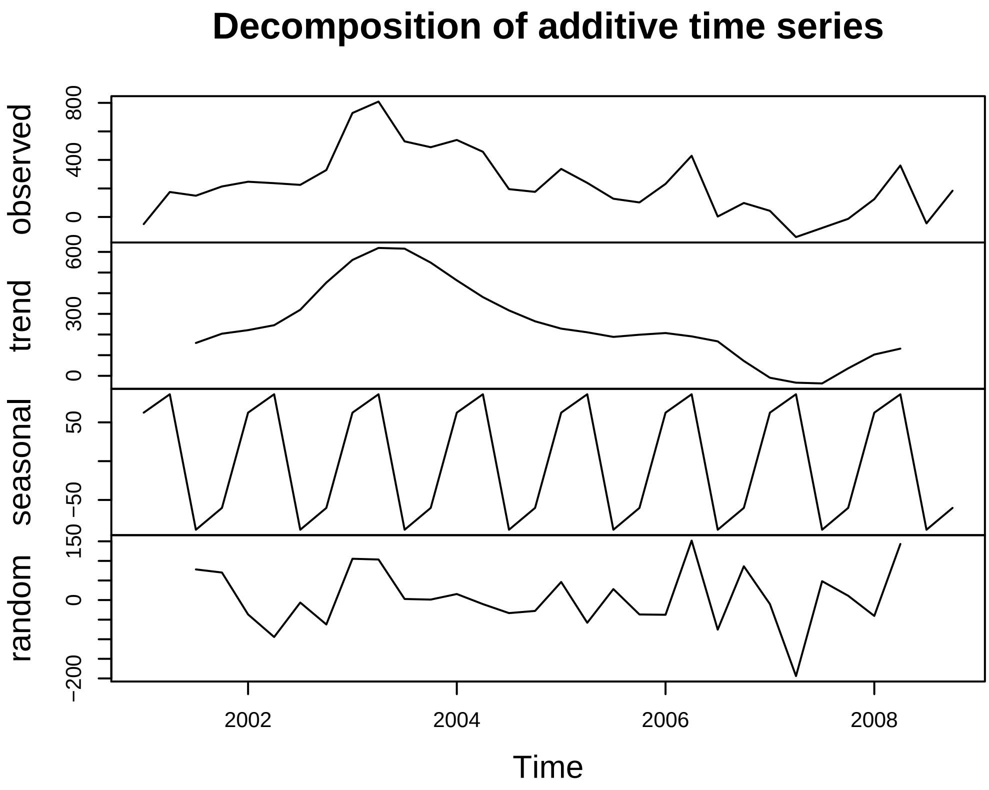

Figure 3: Decomposition of cyclic object example

- Trend component: long term trend
- Seasonal component: seasonal variation
- Cyclical component: repeated but non-periodic fluctuations
- Irregular component: the residuals

A **simpleTS** time series object was constructed in section Constructing TS object. It is used below as an example to demonstrate time series decomposition (Fig 3). It was constructed to have quarterly data and will be decomposed with frequency 4.

```
m <- decompose(simpleTS)
plot(m)
```

A more complex an realistic example of time series manipulation and decomposition presented below. We will use 'open' series of the object AAL stock taken directly from Yahoo Finance using **tseries** (Trapletti and Hornik, 2018) package instead of data created in section Loading Stock Data. The full daily chart of this series presented on (Fig. 4). To decompose the series, code below calculates yearly cycles of the last years since 2010, data aggregated monthly (Fig. 5).

```
library("tseries")
AAL <- get.hist.quote(instrument = "AAL", start = "2010-01-01")
```

```
#> time series starts 2010-01-04
#> time series ends   2019-02-22
```

```
plot(AAL, xlab = "")


require(xts)
last8 <- window(AAL$Open, start=as.Date("2010-01-01"), end=as.Date("2017-12-31"))
last8_mo <- aggregate(last8, as.yearmon, tail, 1)
m <- decompose(ts(last8_mo, frequency = 12))
years <- seq(2010, 2020, 1)
plot(m, xlab="", xaxt="n")
axis(1, at=1:9, labels=years[1:9], pos = -6.6)
```

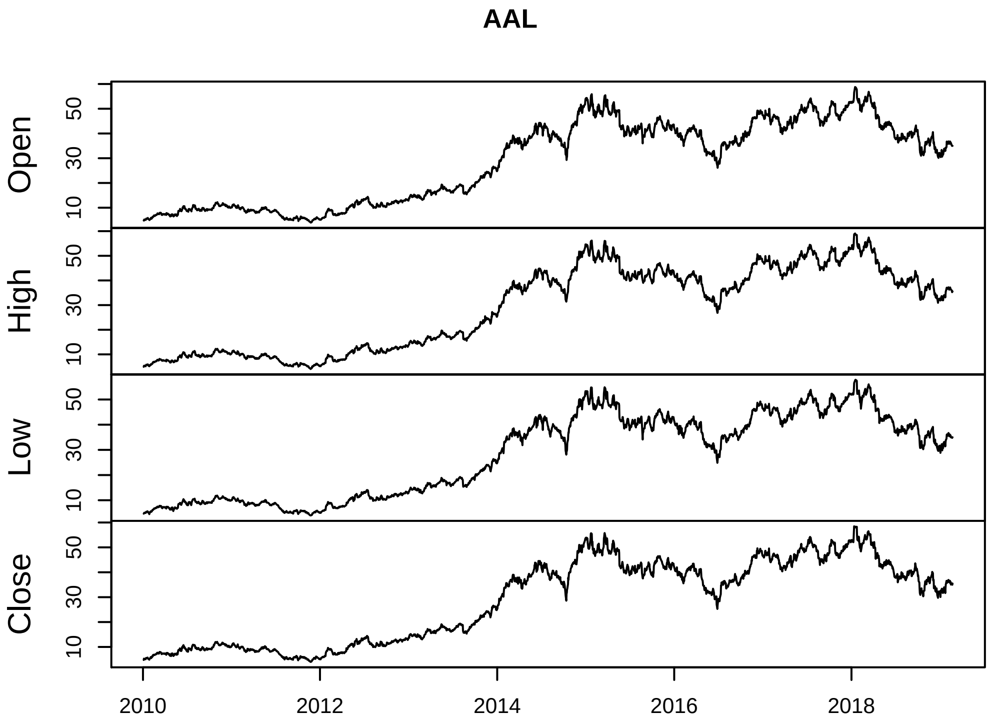

Figure 4: AAL dayly stock data since 2010

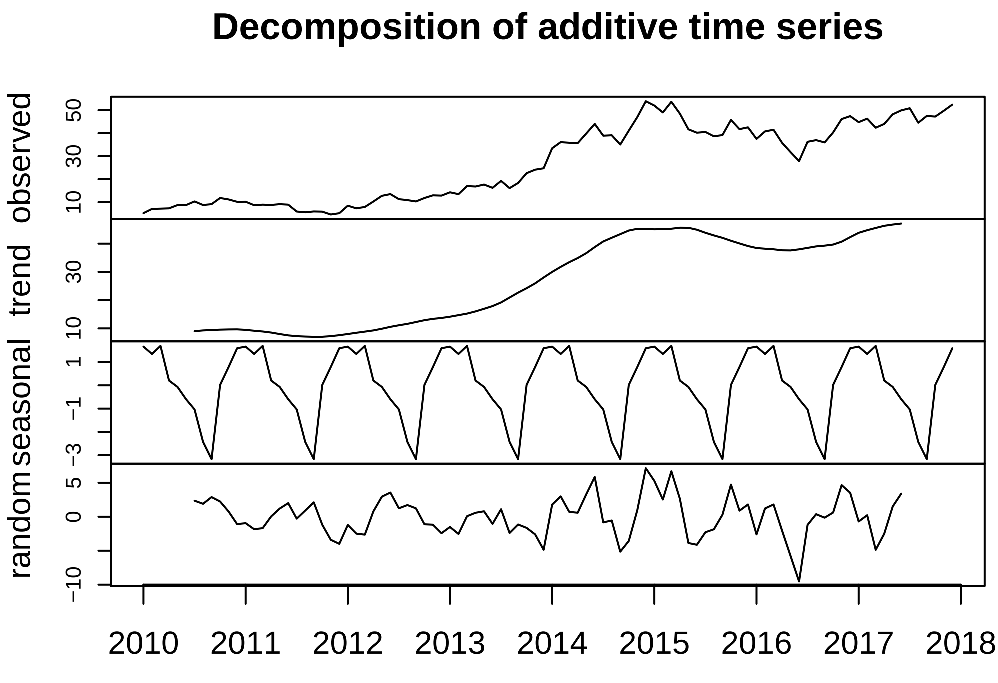

Figure 5: Decomposition of 8 years AAL 'open' series

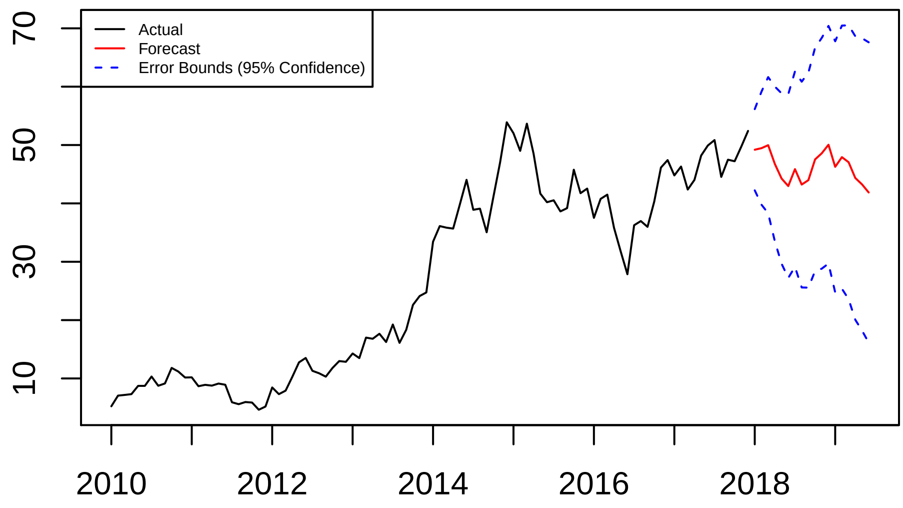

Figure 6: Cyclic Time Series Forecasting with ARIMA model

Time series forecasting

Time series forecasting is to forecast future events based on known past data. To forecast future events based on known past data For example, to predict the price of a stock based on its past performance. Popular models are:

- Autoregressive moving average (ARMA)
- Autoregressive integrated moving average (ARIMA)

Example on Fig. 6 shows forecasting using ARIMA model.

```
ff <- ts(last8_mo)
fit <- arima(ff, order=c(1,0,0), list(order=c(2,1,0), period=12))
fore <- predict(fit, n.ahead=18)

# error bounds at 95% confidence level
U <- fore$pred + 2*fore$se
L <- fore$pred - 2*fore$se

ts.plot(ff, fore$pred, U, L, col=c(1,2,4,4), lty = c(1,1,2,2),
        xlab="", gpars = list(xaxt="n"))
axis(1, at=seq(from=1, to=121, by=12), labels=years[1:11])
legend("topleft", c("Actual", "Forecast", "Error Bounds (95% Confidence)"),
       col=c(1,2,4), lty=c(1,1,2), cex=0.5)
```

Time series forecasting with forecast package

It is common practice to split the data into two parts, training and test data, where the training data is used to estimate any parameters of a forecasting method and the test data is used to evaluate its accuracy. To perform this task and also to test several additional forrecasting methods we used package **forecast** (Hyndman and Khandakar, 2008).

Code below creates train and test data from the the previously obtained 'open' stock data for the last 8 years aggregated monthly. Then it fits the train data using several models and plots the fitting results in the Fig.7. It also plots the test portion in red for easy comparision of prediction and real data.

```
library(forecast)

# test data
test_x <- window(last8, start=as.Date("2016-12-20"))
test_x <- test_x[endpoints(test_x, "month")]
test_x <- ts(test_x, frequency = 12)
str(test_x)
```

```
#>  Time-Series [1:13] from 1 to 2: 47.4 44.8 46.3 42.4 44 ...
#>  - attr(*, "index")= Date[1:13], format: "2016-12-30" "2017-01-31" ...
```

```
#train data
x <- window(last8, end=as.Date("2016-12-31"))
x <- x[endpoints(x, "month")]
x <- ts(x, frequency = 12)
str(x)
```

```
#>  Time-Series [1:84] from 1 to 7.92: 5.23 7.06 7.18 7.31 8.73 ...
#>  - attr(*, "index")= Date[1:84], format: "2010-01-29" "2010-02-26" ...
```

```
library(forecast)
if(!file.exists("./models.Rds")) {
  models <- list(
    mod_arima = auto.arima(x, ic='aicc', stepwise=FALSE),
    mod_exp = ets(x, ic='aicc', restrict=FALSE),
    mod_neural = nnetar(x, p=12, size=25),
    mod_tbats = tbats(x, ic='aicc', seasonal.periods=12),
    mod_bats = bats(x, ic='aicc', seasonal.periods=12),
    mod_stl = stlm(x, s.window=12, ic='aicc', robust=TRUE, method='ets'),
    mod_sts = StructTS(x)
    )
  saveRDS(models, file = "./models.Rds")
} else  {
  models <- readRDS("./models.Rds")
}

forecasts <- lapply(models, forecast, 12)
forecasts$naive <- naive(x, 12)
par(mfrow=c(4, 2))
for(f in forecasts){
  plot(f, xlab="", xaxt="n")
  lines(y=test_x[1:13], x=seq(from=(8-1/12), to=9, by=1/12)[1:13], col='red')
  axis(1, at=1:10, labels=years[1:10])
}
```

Evaluating forecast accuracy

Because the test data is not used in determining the forecasts, it should provide a reliable indication of how well the model is likely to forecast on new data. he metrics used below are described in (Hyndman and Athanasopoulos, 2014), also available online here. The metrics are:

- ME - Mean absolute error
- RMSE - Root mean squared error
- MAE - mean absolute erro
- MPE - mean percentage error
- MAPE - Mean absolute percentage error
- MASE - mean absolute scaled error
- Theil's U - Uncertainty coefficient

Code below (adaptation of Timeseries analysis procedure and methods using R) calculates common forecast errors of the predictions made in the previous section.

```
test_xx <- ts(test_x[2:13], start=c(8,1), frequency = 12)
acc <- lapply(forecasts, function(f){
  accuracy(f, test_xx)[2,,drop=FALSE]
```

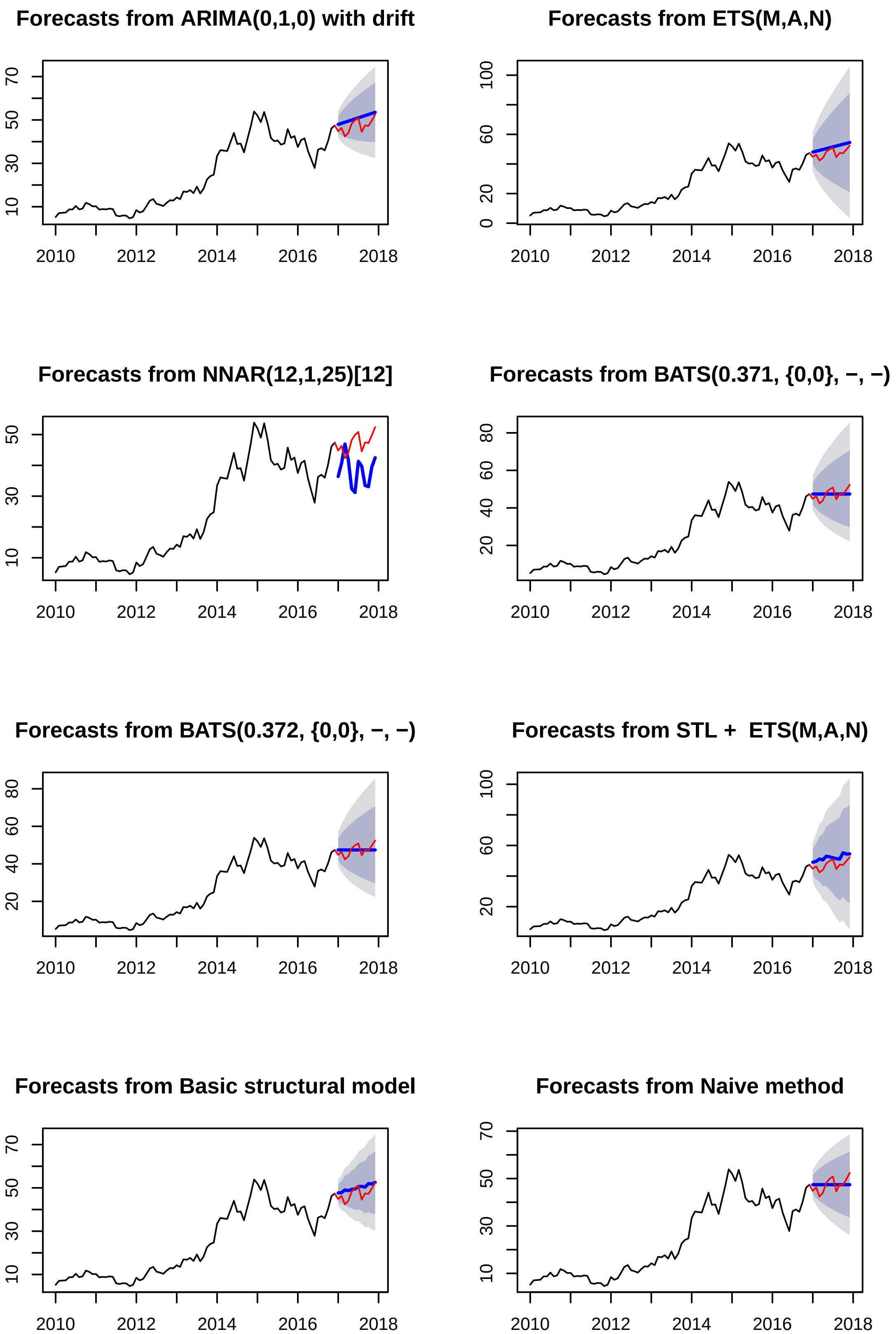

Figure 7: AAL stock prediction vs reality using different forecasting methods

```
})
acc <- Reduce(rbind, acc)
row.names(acc) <- names(forecasts)
acc <- acc[order(acc[,'MASE']),]
round(acc, 2)
```

```
#>                ME  RMSE  MAE    MPE  MAPE MASE  ACF1 Theil's U
#> mod_tbats  -0.08  2.92 2.45  -0.56  5.21 0.28  0.33      0.96
#> mod_bats   -0.08  2.92 2.45  -0.56  5.21 0.28  0.33      0.96
#> naive      -0.09  2.92 2.45  -0.58  5.21 0.28  0.33      0.96
#> mod_sts    -2.68  3.55 2.79  -5.94  6.16 0.32  0.08      1.19
#> mod_arima  -3.40  4.07 3.40  -7.45  7.45 0.39  0.13      1.37
#> mod_exp    -3.93  4.52 3.93  -8.55  8.55 0.45  0.15      1.53
#> mod_stl    -4.70  5.23 4.70 -10.20 10.20 0.54 -0.06      1.77
#> mod_neural  9.08 10.97 9.84  18.68 20.47 1.13  0.38      3.69
```

Intoduction in Time series clustering

In many real applications, the cluster analysis must be performed on time series data. Clustering is an unsupervised learning task aimed to partition a set of unlabeled data objects into homogeneous groups or clusters. Partition is performed in such a way that objects in the same cluster are more similar to each other than objects in different clusters according to some defined criterion.

Those are some measures of distance/dissimilarity - Euclidean distance - Manhattan distance - Maximum norm - Hamming distance - The angle between two vectors (inner product) - Dynamic Time Warping (DTW) distance

Example: Grouping together time series of similar types

The dataset contains 600 examples of control charts synthetically generated by the process developed by Alcock and Manolopoulos (1999). Each control chart is a time series with 60 values. The classes are organized as follows:

- 1-100 Normal
- 101-200 Cyclic
- 201-300 Increasing trend
- 301-400 Decreasing trend
- 401-500 Upward shift
- 501-600 Downward shift

Code below reads the series and shows example of each type of series (Fig. 8):

```
controlCharts <- read.csv(
  "http://kdd.ics.uci.edu/databases/synthetic_control/synthetic_control.data",
  header=F, sep="")

idx <- c(1, 101, 201, 301, 401, 501)
sample1 <- t(controlCharts[idx, ])
plot.ts(sample1, main = "")
```

Now we randomly sample n cases from each class, to make it easy for plotting:

```
n <- 5
s <- sample(1:100, n)
idx <- c(s, 100+s, 200+s, 300+s, 400+s, 500+s)
sample2 <- controlCharts[idx,]
observedLabels <- c(rep(1,n), rep(2,n), rep(3,n), rep(4,n), rep(5,n), rep(6,n))
```

Next step is to compute DTW distances between the sample series and perform hierarchical clustering (Fig. 9).

```
library(dtw)
distMatrix <- dist(sample2, method="DTW")
hc <- hclust(distMatrix, method="average")
plot(hc, labels=observedLabels, main="")
```

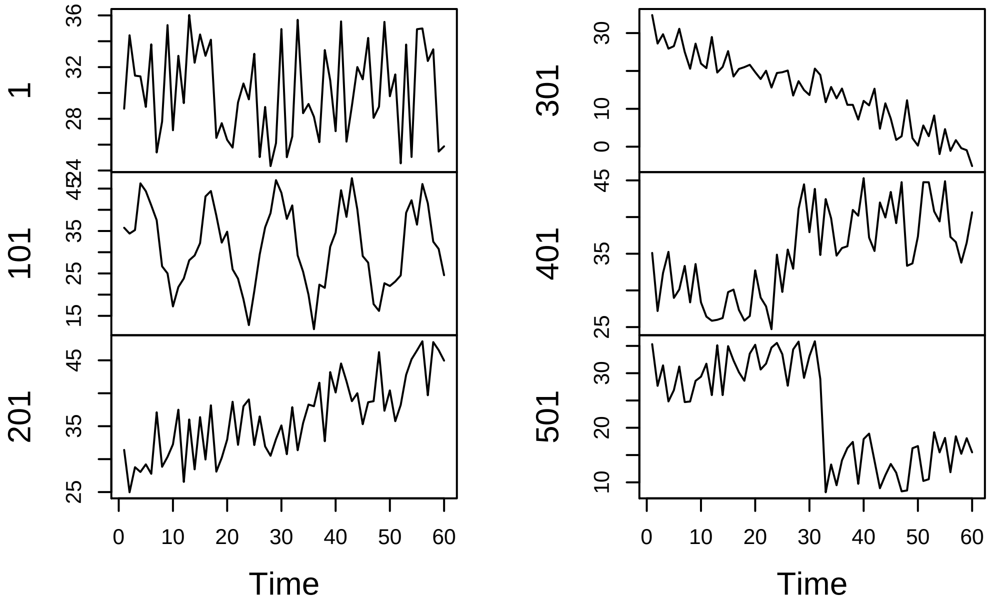

Figure 8: Different types of syntetic time series

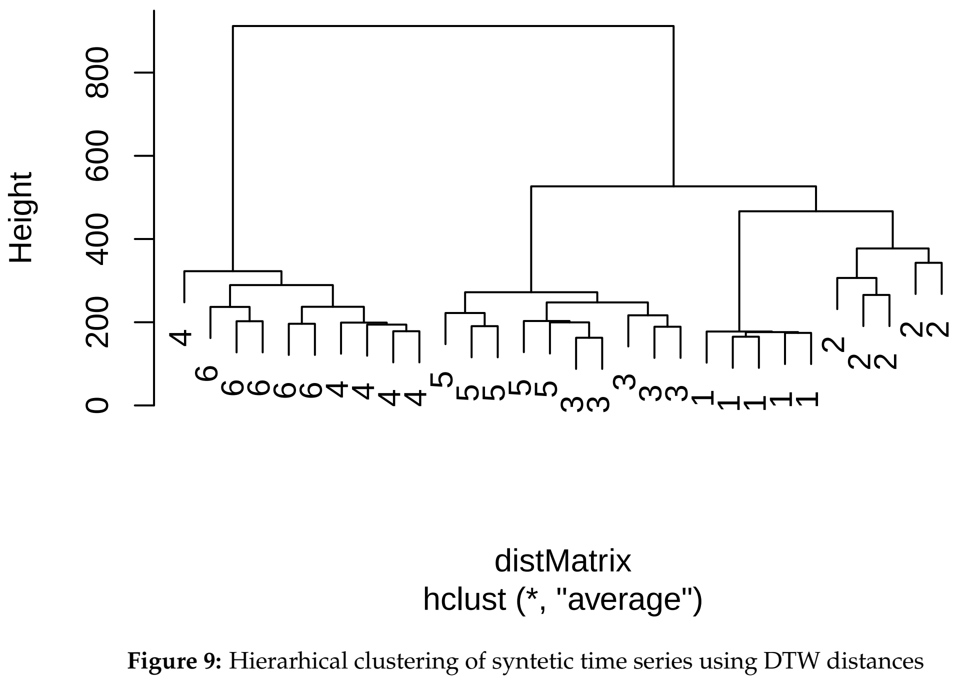

Figure 9: Hierarhical clustering of syntetic time series using DTW distances

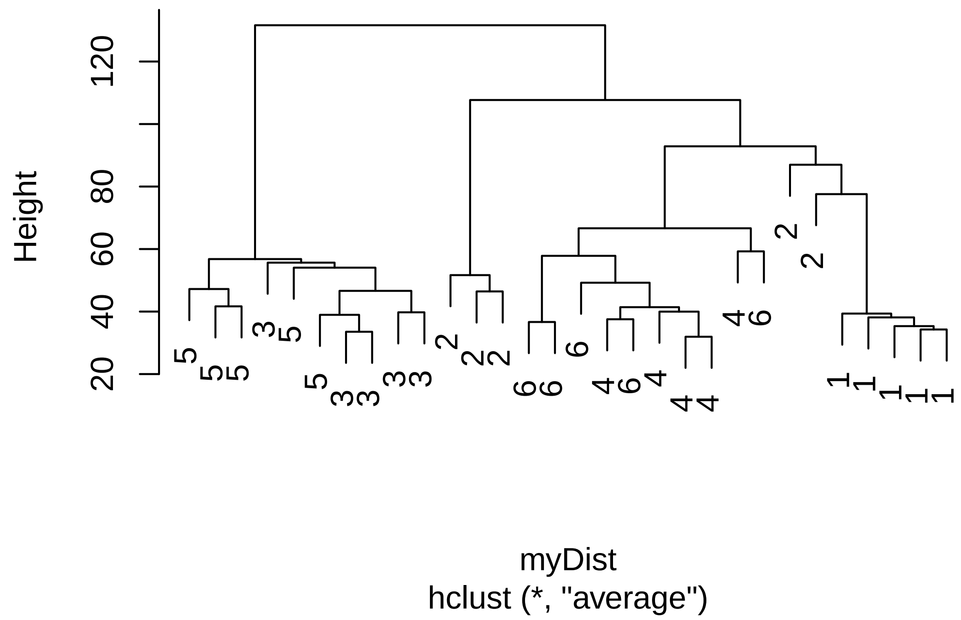

Figure 10: Hierarhical clustering of syntetic time series using Euclidean distances

```
memb <- cutree(hc, k = 8)
table(observedLabels, memb)
```

```
#>              memb
#> observedLabels 1 2 3 4 5 6 7 8
#>             1 5 0 0 0 0 0 0 0
#>             2 0 1 1 2 1 0 0 0
#>             3 0 0 0 0 0 5 0 0
#>             4 0 0 0 0 0 0 1 4
#>             5 0 0 0 0 0 5 0 0
#>             6 0 0 0 0 0 0 0 5
```

Example: Hierarchical clustering with Euclidean distance

```
myDist <- dist(sample2)
hc <- hclust(myDist, method = "ave")
plot(hc, labels = observedLabels, main = "")
```

Cut tree to get 8 clusters

```
memb <- cutree(hc, k = 8)
table(observedLabels, memb)
```

```
#>              memb
#> observedLabels 1 2 3 4 5 6 7 8
#>             1 5 0 0 0 0 0 0 0
#>             2 0 1 1 3 0 0 0 0
#>             3 0 0 0 0 5 0 0 0
#>             4 0 0 0 0 0 1 4 0
#>             5 0 0 0 0 5 0 0 0
#>             6 0 0 0 0 0 0 4 1
```

Advanced Time Series Clustering with TSclust Package

In Time Series anslysis the grouping of series plays a central role in many applications. Finding stocks that behave in a similar way, determining products with similar selling patterns, identifying countries with similar population growth or regions with similar temperature are some typical applications.Package **TSclust** (Montero and Vilar, 2014) was developed for Time Series clustering based on measures of dissimilarity. The package implemernts a variety of such measures that split in the following groups:

1. Model-free approaches that measure proximity between TS based on closeness of values:

- Minkowski distance.
- Frechet distance.
- Dynamic time warping distance (DTW).
- ... many others.

2. Model-based approaches that assume that the underlying models are generated from specific parametric structures:

- Piccolo distance - defines a dissimilarity measure in the class of invertible ARIMA processes as the Euclidean distance.
- Maharaj distance - based on hypotheses testing to determine whether or not two time series have significantly different generating processes.

3. Complexity-based approaches that based on comparing levels of complexity of time series are presented in this section:

- Normalized compression distance (NCD) - based on measuring of compression rates of the time series.
- Permutation distribution clustering (PDC) - described in terms of divergence between permutation distributions of order patterns in m-embedding of the original series.
- Complexity-invariant dissimilarity measure (CID) - intuitive, parameter-free, invariant to the complexity of time series, computationally more efficient and more accurate than methods above.

4. Prediction-based approaches - dissimilarity measures based on comparing the forecast densities for each series at a future horizon of interest, where the forecast are unknown and must be approximated from the data.

The full list of the dissimilarity methods, implemented in **TCslust** borrowed from page 25 of (Montero and Vilar, 2014) is presented on Fig. 11.

Example: Clustering based on model free approach

Let's test the diss.CORT dsstance method to separate 18 series from the syntetic dataset we used before. The resulting clusters presented on (Fig. 12):

```
library(TSclust)

set.seed(1234)
n <- 3
s <- sample(1:100, n)
idx <- c(s, 100+s, 200+s, 300+s, 400+s, 500+s)
sample3 <- ts(t(controlCharts[idx,]))

# create the true cluster solution for comparision
true_cluster <- rep(1:6, each = n)

# create distance matrix covering both proximity on values and behavior
IP.dis <- diss(sample3, "CORT")

# hierarchical cluster solution
clust <- hclust(IP.dis)
plot(clust, labels = true_cluster, main = "")
```

Dissimilarity measures	Function in **TSclust**
Model free approaches	
Based on raw data	
d_{L_2}(Euclidean)	`diss.EUCL`
d_F	`diss.FRECHET`
d_{DTW}	`diss.DTW`
Covering both proximity on values and on behavior	
d_{CORT}	`diss.CORT`
Based on correlations	
$d_{COR.1}$	`diss.COR`, `beta = NULL`
$d_{COR.2}$	`diss.COR` and a specified value for `beta`
Based on simple and partial autocorrelations	
d_{ACF}	`diss.ACF`
d_{ACFU}	`diss.ACF` without parameters
d_{ACFG}	`diss.ACF`, `p` $\neq 0$
d_{PACF}	`diss.PACF`
d_{PACFU}	`diss.PACF` without parameters
d_{PACFG}	`diss.PACF`, `p` $\neq 0$
Based on periodograms	
d_P	`diss.PER`
d_{NP}	`diss.PER`, `normalize = TRUE`
d_{LNP}	`diss.PER`, `normalize = TRUE`, `logarithm = TRUE`
d_{IP}	`diss.INT.PER`
Based on nonparametric spectral estimators	
$d_{W(LS)}$	`diss.SPEC.LLR`, `method = "LS"`
$d_{W(LK)}$	`diss.SPEC.LLR`, `method = "LK"`
d_{GLK}	`diss.SPEC.GLK`
d_{ISD}	`diss.SPEC.IDS`
Based on the discrete wavelet transform	
d_{DWT}	`diss.DWT`
Based on symbolic representation	
$d_{MINDIST.SAX}$	`diss.MINDIST.SAX`
Model-based approaches	
d_{PIC}	`diss.AR.PIC`
d_{MAH}	`diss.AR.MAH`
d_{MAHext}	`diss.AR.MAH`, `dependence = TRUE`
$d_{LCP.Cep}$	`diss.AR.LPC.CEPS`
Complexity-based approaches	
d_{CID}	`diss.CID`
d_{PDC}	`diss.PDC`
d_{CDM}	`diss.CDM`
d_{NCD}	`diss.NCD`
Prediction-based approach	
$d_{PRED,h}$	`diss.PRED`

Figure 11: Dissimilarity measures implemented in TSclust

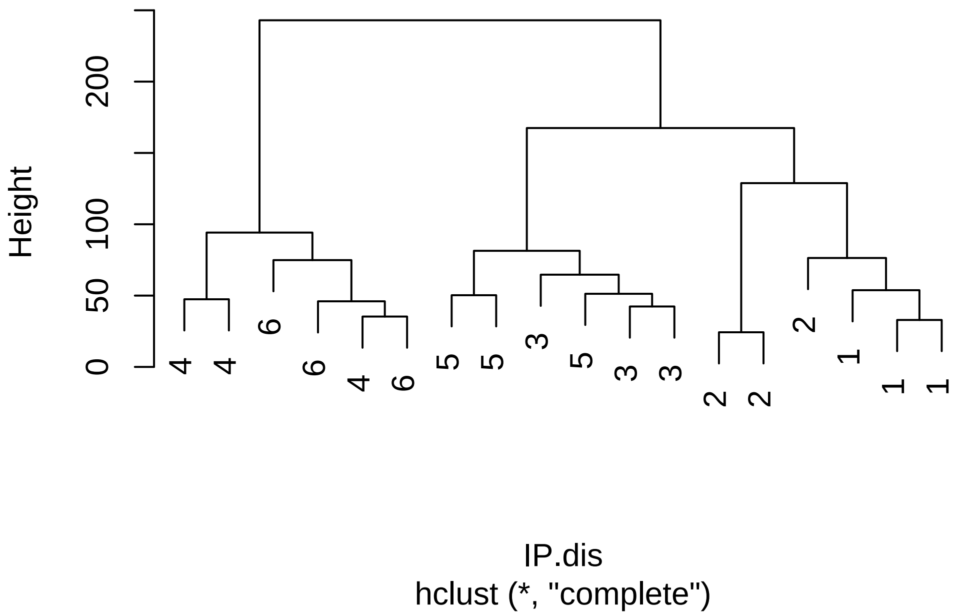

Figure 12: Custering of time series using CORT distances based on proximity and behavior

```
IP.hclus <- cutree(clust, k = 6)

# rate the solution based on the implemented cluster similarity
cluster.evaluation(true_cluster, IP.hclus)

#> [1] 0.8285714
```

Time Series classification problem

Time series classification problem is a classification model based on labelled time series and then use the model to predict the label of unlabelled time series. The way for time series classification with R is to extract and build features from time series data first, and then apply existing classification techniques, such as SVM, k-NN, neural networks, regression and decision trees, to the feature set. Feature Extraction is done by the following methods:

- Singular Value Decomposition (SVD)
- Discrete Fourier Transform (DFT)
- Discrete Wavelet Transform (DWT)
- Piecewise Aggregate Approximation (PAA)
- Perpetually Important Points (PIP)
- Piecewise Linear Representation
- Symbolic Representation

Below is the example of Decision Tree model use for time series classification (Fig. 13):

```
classId <- rep(as.character(1:6), each = 100)
newSc <- data.frame(cbind(classId, controlCharts))

library(party)
ct <- ctree(classId ~ ., data = newSc,
controls = ctree_control(minsplit = 20,
minbucket = 5, maxdepth = 5))
plot(ct)
```

Let's estimate the accuracy of the prediction:

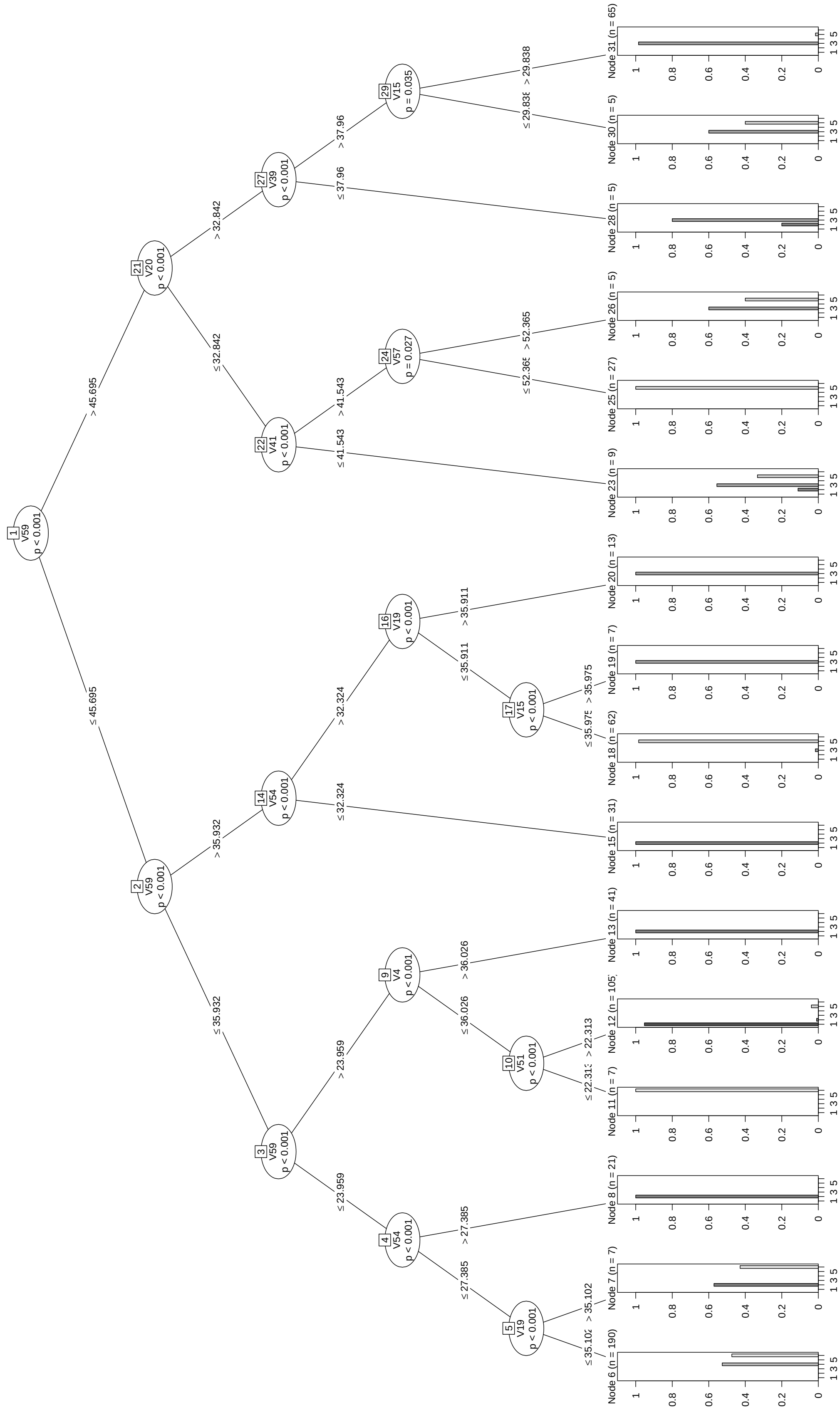

Figure 13: Decision Tree model use for time series classification

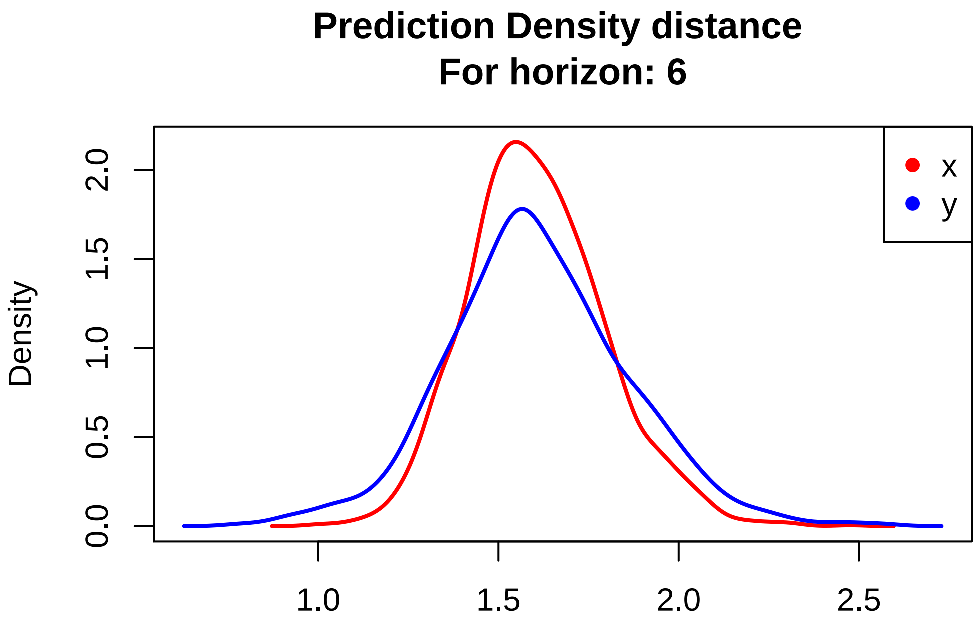

Figure 14: Sample plot of the prediction densities

```
pClassId <- predict(ct)
table(classId, pClassId)
```

```
#>        pClassId
#> classId   1   2   3   4   5   6
#>       1 100   0   0   0   0   0
#>       2   1  97   2   0   0   0
#>       3   0   0  99   0   1   0
#>       4   0   0   0 100   0   0
#>       5   4   0   8   0  88   0
#>       6   0   3   0  90   0   7
```

```
(sum(classId == pClassId))/nrow(controlCharts)
```

```
#> [1] 0.8183333
```

Example: Grouping countries by prediction of interest rates

First lets create a sample plot of the prediction densities from which the distance is calculated, the prediction is done at an horizon h=6 steps (Fig. 14):

```
data("interest.rates")
set.seed(35)

diss.PRED(interest.rates[, 13], interest.rates[, 16],
          h = 6, B = 2000, logarithm.x = TRUE,
          logarithm.y = TRUE, differences.x = 1, differences.y = 1,
          plot = T)$L1dist
```

```
#> [1] 0.2487087
```

Now lets prepare the correct differences and logarithms for all the countries involved in the dataset, then compute the distance at for the datase and printprediction dencities for all counties from the dataset (Fig. 15).

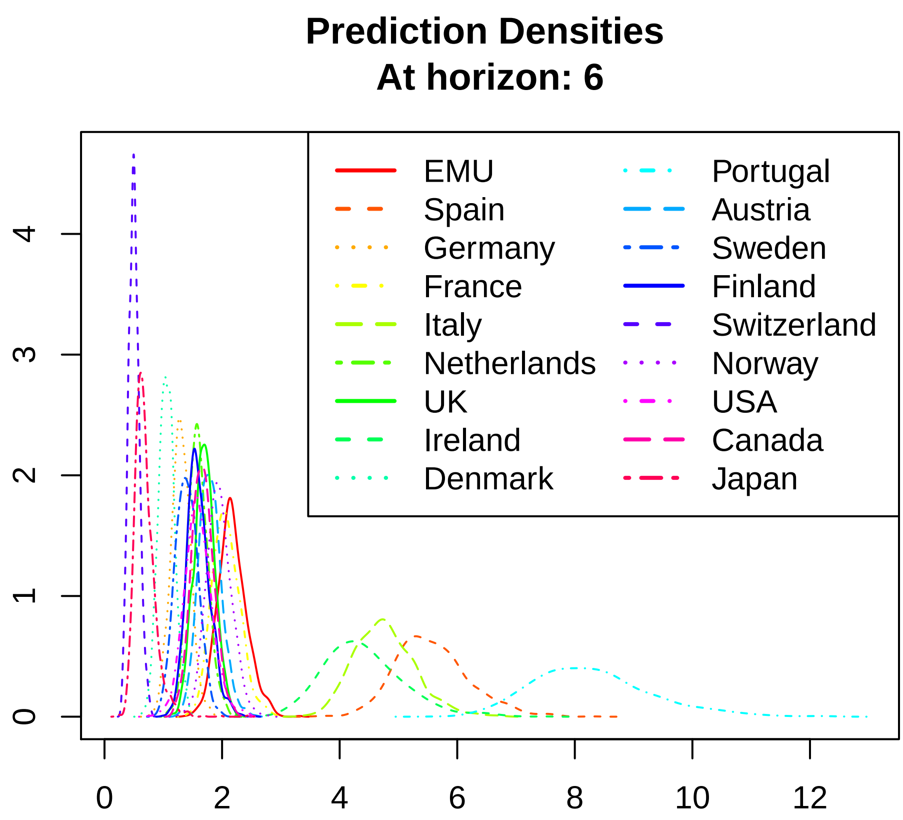

Figure 15: Grouping countries by prediction of interest rates

```
diffs <- rep(1, ncol(interest.rates))
logs <- rep(TRUE, ncol(interest.rates))

set.seed(74)
dpred <- diss(interest.rates, "PRED", h = 6, B = 1200,
              logarithms = logs, differences = diffs,  plot = T)
```

Code below perform hierarchical clustering and plots the dendogram (Fig. 16).

```
hc.dpred <- hclust(dpred$dist)
plot(hc.dpred, main = "", sub = "", xlab = "", ylab = "")
```

Time Series feature extraction with Discrete Wavelet Transform

Discrete Wavelet Transform (DWT) provides a multi-resolution representation using wavelets and is used in the example below. Another popular feature extraction technique is Discrete Fourier Transform (DFT). Haar Wavelet Transform is a simplest DWT. The code below extracting DWT coefficients (with Haar filter).

```
library(wavelets)

wtData <- NULL
for (i in 1:nrow(controlCharts)) {
  a <- t(controlCharts[i,])
  wt <- dwt(a, filter="haar", boundary="periodic")
  wtData <- rbind(wtData, unlist(c(wt@W,wt@V[[wt@level]])))
}
wtData <- as.data.frame(wtData)
```

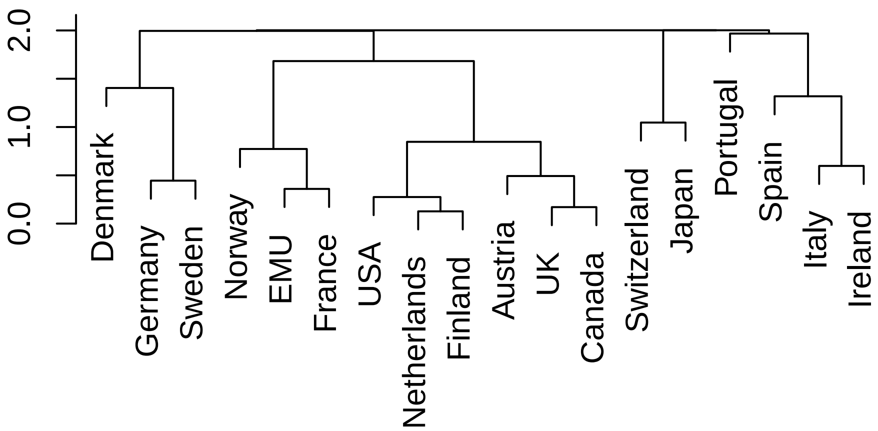

Figure 16: Grouping countries by prediction of interest rates

Now lt's set class labels into categorical values:

```
classId <- c(rep("1",100), rep("2",100), rep("3",100),
rep("4",100), rep("5",100), rep("6",100))
wtSc <- data.frame(cbind(classId, wtData))
```

Now let's build a decision tree with **ctree** from package party, check predicted classes against original class labels and calulate accuracy. The resulting decision tree in presented in Fig. 17.

```
library(party)

ct <- ctree(classId ~ ., data=wtSc,
controls = ctree_control(minsplit=30, minbucket=10, maxdepth=5))
pClassId <- predict(ct)
table(classId, pClassId)

#>        pClassId
#> classId  1  2  3  4  5  6
#>       1 97  3  0  0  0  0
#>       2  1 99  0  0  0  0
#>       3  0  0 81  0 19  0
#>       4  0  0  0 63  0 37
#>       5  0  0 16  0 84  0
#>       6  0  0  0  1  0 99

(sum(classId==pClassId)) / nrow(wtSc)

#> [1] 0.8716667

plot(ct, ip_args = list(pval = F), ep_args = list(digits = 0))
```

Finding k nearest neighbours of a specific time series

To find the k nearest neighbours first we generate a new series as a deviation series 501, then calculate and sort distancesand display 20 closest series:

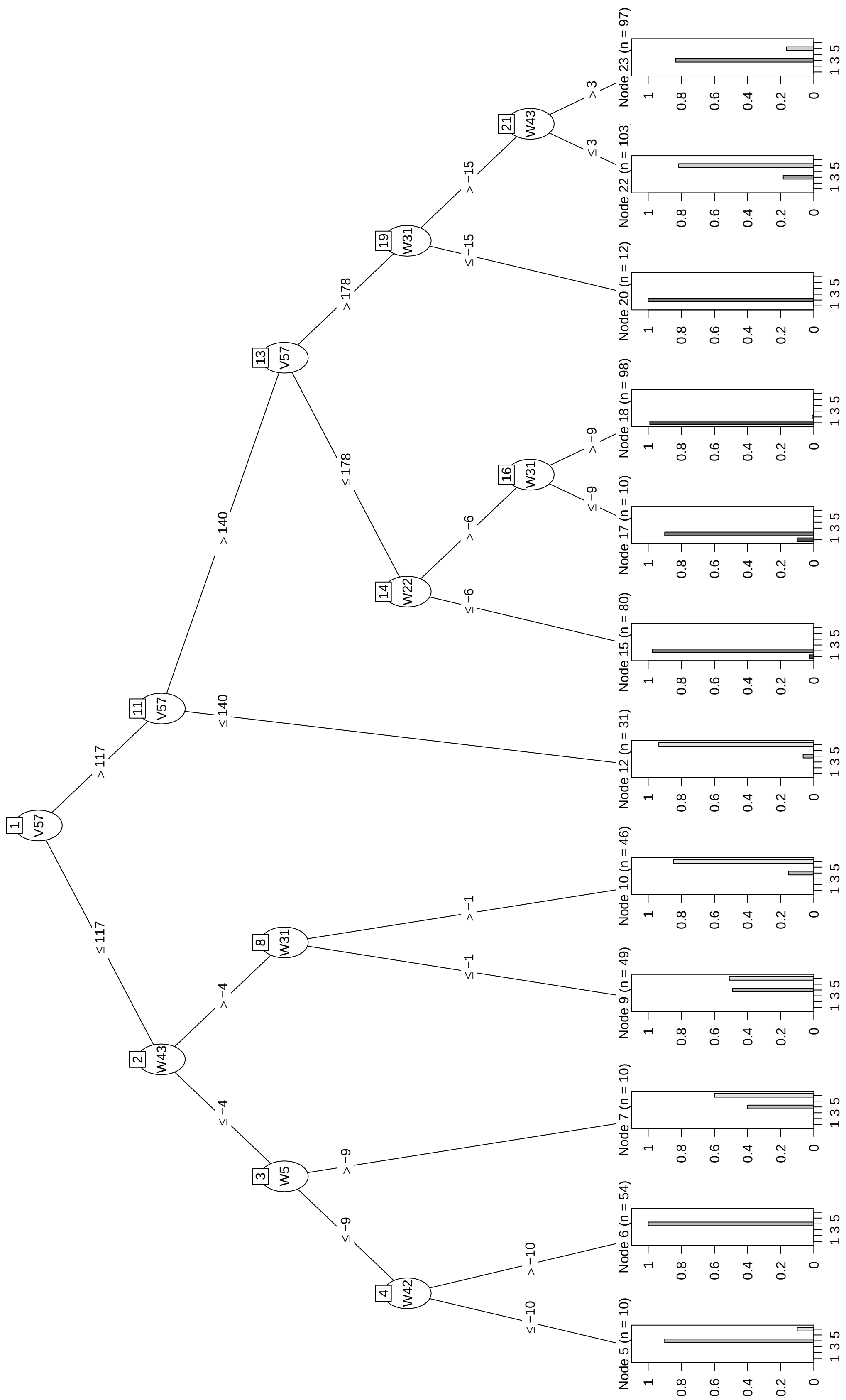

Figure 17: Hierarhical clustering of syntetic time series using Discrete Wavelet Transform

```
set.seed(22)
k <- 20
newTS <- controlCharts[501, ] + runif(100) * 15
distances <- dist(newTS, controlCharts, method = "DTW")
s <- sort(as.vector(distances), index.return = TRUE)

#Distances and indexes of the closest series
str(s)
```

```
#> List of 2
#>  $ x : num [1:600] 331 332 333 339 340 ...
#>  $ ix: int [1:600] 577 530 377 512 598 511 349 504 589 306 ...
```

```
#Cass IDs of k nearest neighbours out of 20 closest
table(classId[s$ix[1:k]])
```

```
#>
#>  4  6
#>  4 16
```

Example of Time-Series Analysis Practical Application

As a practical example of Time Series analysis we will develop a Shiny application dealing with Nasdaq stock data.

Prepare Shiny App for Deployment

Fists step is loading a list of all Nasdaq listings. The description of the data is here;

```
nasdaqTraded <- read.csv(
  file = "ftp://ftp.nasdaqtrader.com/SymbolDirectory/nasdaqtraded.txt",
  sep = "|"  )

# last line in the file is the date of creation as " File Creation Time: mmddyyyyhhmm"
creationDate <- as.list(levels(nasdaqTraded[-1,1]))[[1]]
print(creationDate)
```

```
#> [1] "File Creation Time: 0225201915:41"
```

```
nasdaqTraded <- nasdaqTraded[1:nrow(nasdaqTraded)-1,]

# remove unnecessary rows and columns
nasdaqTraded <- nasdaqTraded[
  (nasdaqTraded$Nasdaq.Traded=="Y" & nasdaqTraded$Financial.Status=="N"),
  c(2,3)]
nasdaqTraded$Symbol <- factor(nasdaqTraded$Symbol)
nasdaqTraded$Security.Name <- factor(nasdaqTraded$Security.Name)

tail(nasdaqTraded)
```

```
#>      Symbol                               Security.Name
#> 8690  ZVZZT                            NASDAQ TEST STOCK
#> 8691  ZWZZT                            NASDAQ TEST STOCK
#> 8693 ZXYZ.A          Nasdaq Symbology Test Common Stock
#> 8694  ZXZZT                            NASDAQ TEST STOCK
#> 8696   ZYNE Zynerba Pharmaceuticals, Inc. - Common Stock
#> 8697   ZYXI                   Zynex, Inc. - Common Stock
```

Now lets take 5 random stocks andrequest their montholy quotes for the last 10 years (Fig. 18):

```
set.seed(21)
library("tseries")
nStocks <- 5
stocks <- nasdaqTraded[sample(1:nrow(nasdaqTraded), nStocks),]

z <- list()
par(mfrow = c(nStocks, 1))
for (i in 1:nrow(stocks)) {
  name <- toString(stocks[i, "Symbol"][1])
  z[[i]] <- get.hist.quote(
    instrument = name,
    quote = "Close",
    compression = "m",
    start = Sys.Date() - (365*10))
  plot(z[[i]], xlab = "", ylab="", main=name)
}
```

```
#> time series starts 2013-06-01
#> time series ends   2019-02-01

#> time series starts 2013-07-01
#> time series ends   2019-02-01

#> time series starts 2009-02-01
#> time series ends   2019-02-01
```

```
#> time series starts 2016-12-01
#> time series ends   2019-02-01

#> time series starts 2013-06-01
#> time series ends   2019-02-01
```

Prediction of stock prices

The code below defines functions to read, predict and plot stocks to be used in the Shiny App:

```
library(forecast)

readStocks <- function(stocksList, startDate, quote = "Close", isMonthly=TRUE) {
  z <- list()
  compression <- if (isMonthly == TRUE) "m" else "d"
  for (i in 1:nrow(stocksList)) {
    name <- toString(stocksList[i, "Symbol"][1])
    tryCatch({
      z[[name]] <- get.hist.quote(
        instrument = name,
        quote = quote,
        compression = compression,
        start = startDate)
    },
    error=function(cond) {
      message(cond)
    })
  }
  return (z)
}

plotForecast <- function(stockZoo, stockSymbol, freq, toForecast, isMonthly=TRUE) {
  x <- ts(stockZoo, frequency=freq)
  dmin <- start(z[[n]])
  dmax <- end(z[[n]])

  tryCatch({
    models <- list(
      mod_stl = stlm(x, s.window=freq, ic='aicc', robust=TRUE, method='ets')
    )

    forecasts <- lapply(models, forecast, toForecast)
    len <- ceiling(1 + length(x)/freq)
    format <- if (isMonthly == TRUE) "%b-%Y" else "%Y-%m-%d"
    by <- if (isMonthly == TRUE) paste(freq, "months") else paste(freq, "days")
    to <- if (isMonthly == TRUE) as.Date(dmax) + toForecast*30 else as.Date(dmax) + toForecast
    for (f in forecasts) {
      ticks<-format(seq(as.Date(dmin),to=to, length.out=len), format=format)
      print (ticks)
      plot(f, xlab="", xaxt="n", main = paste(stockSymbol, "Stock Forecast"))
      axis(1, at=1:len, labels=ticks, par(las=2))
    }
  },
  error=function(cond) {
    message(cond)
  })
}
```

Let's test print forecast for the next 6 month of up to 5 random stocks (Fig. 19)). Some of the stocks might be not available and some could not be predicted by the method of choice, so the final resultmight have less than 5 charts. The stock data aggregated monthly and requested since 2016.

```
set.seed(99)
nStocks <- 5
```

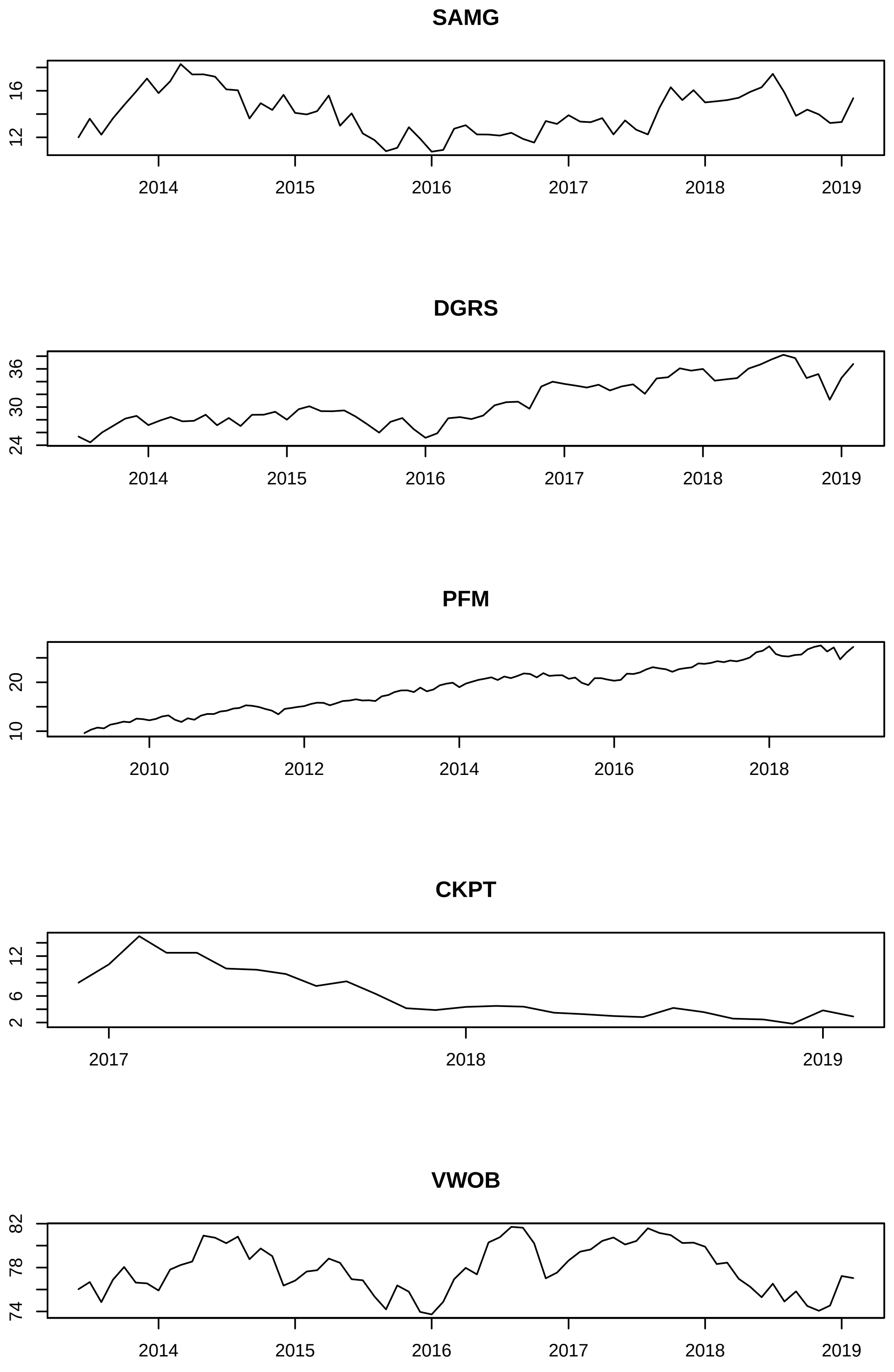

Figure 18: Random stocks 'Close' prices

```
stocks <- nasdaqTraded[sample(1:nrow(nasdaqTraded), nStocks),]

z <- readStocks(stocks, "2010-01-01", "Close", TRUE)

#> time series starts 2014-12-01
#> time series ends   2019-02-01
#> time series ends   2019-02-01
#> time series ends   2019-02-01
#> time series ends   2019-02-01

par(mfrow = c(nStocks, 1))
for (n in 1:length(z)) {
  plotForecast(z[[n]], names(z)[n], 12, 12)
}

#> [1] "Dec-2014" "Dec-2015" "Dec-2016" "Jan-2018" "Jan-2019" "Jan-2020"

#>  [1] "Jan-2010" "Jan-2011" "Jan-2012" "Jan-2013" "Jan-2014" "Jan-2015"
#>  [7] "Jan-2016" "Jan-2017" "Jan-2018" "Jan-2019" "Jan-2020"

#>  [1] "Jan-2010" "Jan-2011" "Jan-2012" "Jan-2013" "Jan-2014" "Jan-2015"
#>  [7] "Jan-2016" "Jan-2017" "Jan-2018" "Jan-2019" "Jan-2020"

#>  [1] "Jan-2010" "Jan-2011" "Jan-2012" "Jan-2013" "Jan-2014" "Jan-2015"
#>  [7] "Jan-2016" "Jan-2017" "Jan-2018" "Jan-2019" "Jan-2020"
```

Now let's test print forecast for the next 2 weeks of up to 5 random stocks (Fig. 20)). Some of the stocks might be not available and some could not be predicted by the method of choice, so the final resultmight have less than 5 charts. The stock data aggregated monthly and requested since beginning of 2018.

```
set.seed(99)
nStocks <- 5
stocks <- nasdaqTraded[sample(1:nrow(nasdaqTraded), nStocks),]

z <- readStocks(stocks, "2018-08-01", "Close", isMonthly=FALSE)

#> time series ends   2019-02-22
#> time series ends   2019-02-22
#> time series ends   2019-02-22
#> time series ends   2019-02-22

par(mfrow = c(nStocks, 1))
for (n in 1:length(z)) {
  plotForecast(z[[n]], names(z)[n], 10, 10, isMonthly=FALSE)
}

#>  [1] "2018-08-01" "2018-08-15" "2018-08-29" "2018-09-13" "2018-09-27"
#>  [6] "2018-10-11" "2018-10-26" "2018-11-09" "2018-11-23" "2018-12-08"
#> [11] "2018-12-22" "2019-01-05" "2019-01-20" "2019-02-03" "2019-02-17"
#> [16] "2019-03-04"

#>  [1] "2018-08-01" "2018-08-15" "2018-08-29" "2018-09-13" "2018-09-27"
#>  [6] "2018-10-11" "2018-10-26" "2018-11-09" "2018-11-23" "2018-12-08"
#> [11] "2018-12-22" "2019-01-05" "2019-01-20" "2019-02-03" "2019-02-17"
#> [16] "2019-03-04"

#>  [1] "2018-08-01" "2018-08-15" "2018-08-29" "2018-09-13" "2018-09-27"
#>  [6] "2018-10-11" "2018-10-26" "2018-11-09" "2018-11-23" "2018-12-08"
#> [11] "2018-12-22" "2019-01-05" "2019-01-20" "2019-02-03" "2019-02-17"
#> [16] "2019-03-04"

#>  [1] "2018-08-01" "2018-08-15" "2018-08-29" "2018-09-13" "2018-09-27"
#>  [6] "2018-10-11" "2018-10-26" "2018-11-09" "2018-11-23" "2018-12-08"
#> [11] "2018-12-22" "2019-01-05" "2019-01-20" "2019-02-03" "2019-02-17"
#> [16] "2019-03-04"
```

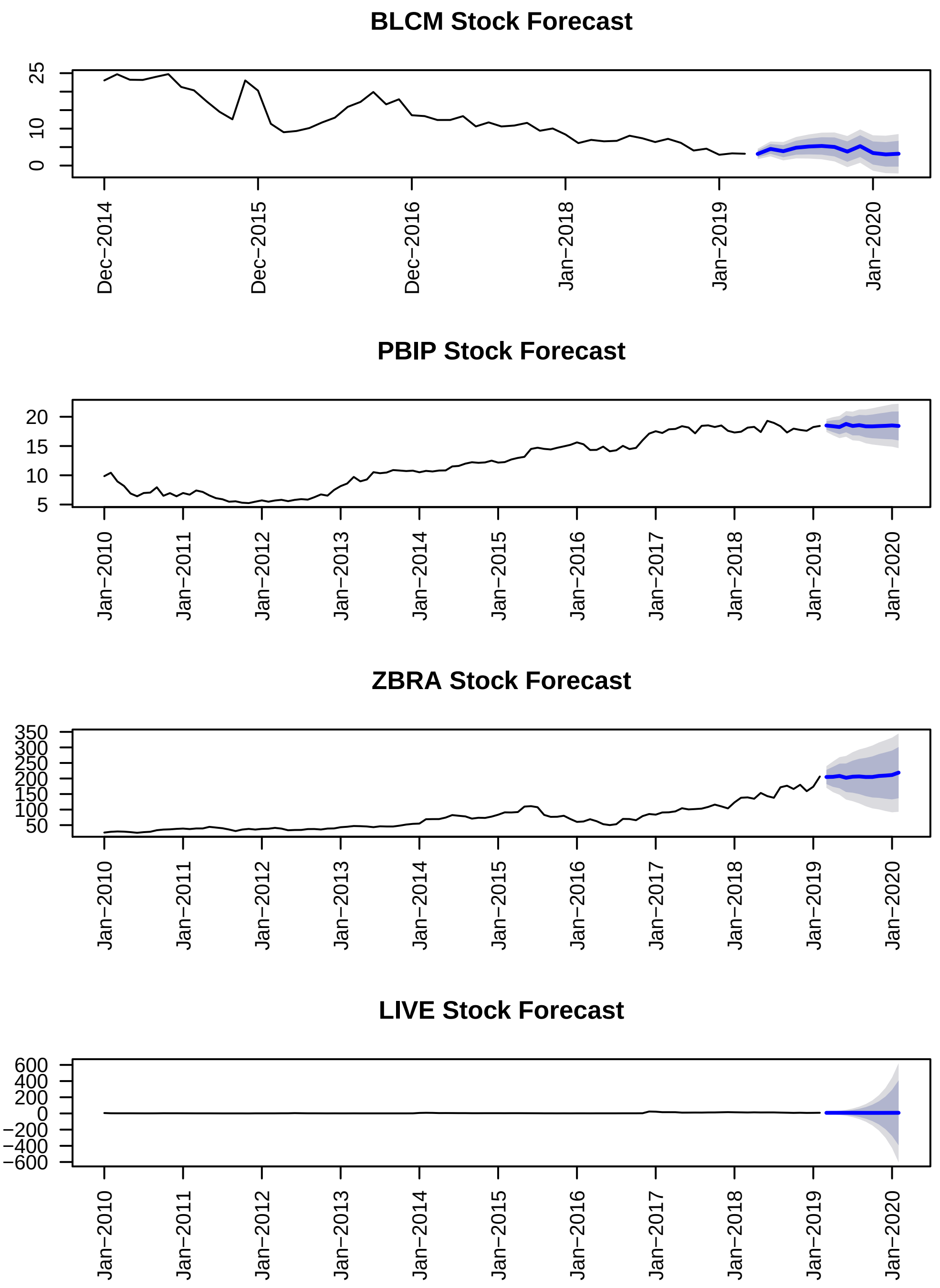

Figure 19: Prediction of monthly stock prices

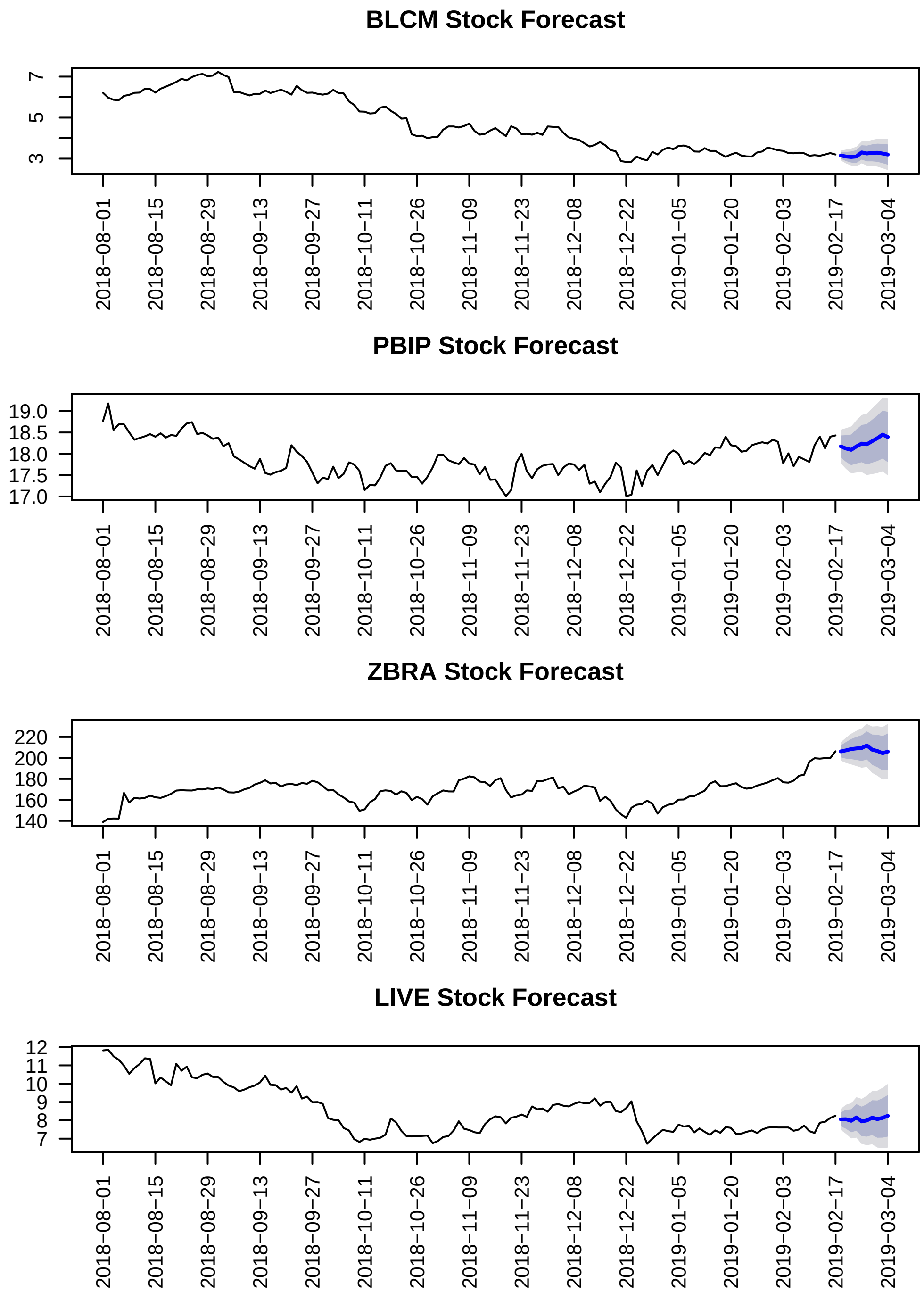

Figure 20: Prediction of daily stock prices

Deployment Discussion

The model developed in this project was used to create Shiny application currently deployed at ivbsoftware.shinyapps.io/Stocks/. Code of the application could be found in Github.

Bibliography

R. Alcock and Y. Manolopoulos. Time-Series Similarity Queries Employing a Feature-Based Approach. Ioannina,Greece, Aug. 1999. [p10]

R. J. Hyndman and G. Athanasopoulos. *Forecasting: principles and practice*. OTexts, S.l., print edition edition, 2014. ISBN 9780987507105. OCLC: 935493912. [p8]

R. J. Hyndman and Y. Khandakar. Automatic time series forecasting: the forecast package for R. *Journal of Statistical Software*, 26(3):1–22, 2008. URL http://www.jstatsoft.org/article/view/v027i03. [p7]

P. Montero and J. A. Vilar. TSclust: An R package for time series clustering. *Journal of Statistical Software*, 62(1):1–43, 2014. URL http://www.jstatsoft.org/v62/i01/. [p13]

R Core Team. *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing, Vienna, Austria, 2012. URL http://www.R-project.org/. ISBN 3-900051-07-0. [p1]

J. A. Ryan and J. M. Ulrich. *xts: eXtensible Time Series*, 2018. URL https://CRAN.R-project.org/package=xts. R package version 0.11-2. [p1]

A. Trapletti and K. Hornik. *tseries: Time Series Analysis and Computational Finance*, 2018. URL https://CRAN.R-project.org/package=tseries. R package version 0.10-46. [p5]

A. Zeileis and G. Grothendieck. zoo: S3 infrastructure for regular and irregular time series. *Journal of Statistical Software*, 14(6):1–27, 2005. doi: 10.18637/jss.v014.i06. [p1]

Note from the Authors

This file was generated using *The R Journal* style article template, additional information on how to prepare articles for submission is here - Instructions for Authors. The article itself is an executable R Markdown file that could be downloaded from Github with all the necessary artifacts.

Igor Baranov
York University School of Continuing Studies

https://learn.continue.yorku.ca/user/profile.php?id=21219

Michael Parravani
York University School of Continuing Studies

Ariana Biagi
York University School of Continuing Studies

Hui Fang Cai
York University School of Continuing Studies