Quiz 4

Question 1

For this quiz we will be using several R packages. R package versions change over time, the right answers have been checked using the following versions of the packages.

* AppliedPredictiveModeling: v1.1.6
* caret: v6.0.47
* ElemStatLearn: v2012.04-0
* pgmm: v1.1
* rpart: v4.1.8
* gbm: v2.1
* lubridate: v1.3.3
* forecast: v5.6
* e1071: v1.6.4

If you aren’t using these versions of the packages, your answers may not exactly match the right answer, but hopefully should be close.

Load the vowel.train and vowel.test data sets:

**library**(ElemStatLearn)

data(vowel.train)

data(vowel.test)

Set the variable y to be a factor variable in both the training and test set. Then set the seed to 33833. Fit (1) a random forest predictor relating the factor variable y to the remaining variables and (2) a boosted predictor using the “gbm” method. Fit these both with the train() command in the caret package.

What are the accuracies for the two approaches on the test data set? What is the accuracy among the test set samples where the two methods agree?

Answer

vowel.train$y <- as.factor(vowel.train$y)

vowel.test$y <- as.factor(vowel.test$y)

set.seed(33833)

modRF <- train(y ~ ., data=vowel.train, method="rf") *#, trControl=trainControl("cv"), number=3)*

modBoost <- train(y ~ ., data=vowel.train, method="gbm", verbose=FALSE)

predRF <- predict(modRF, vowel.test)

predBoost <- predict(modBoost, vowel.test)

agreedIndex <- predRF == predBoost

cfmRf <- confusionMatrix(vowel.test$y, predRF)

cfmBoost <- confusionMatrix(vowel.test$y, predBoost)

cfmAgreed <- confusionMatrix(vowel.test$y[agreedIndex], predRF[agreedIndex])

cfmRf$overall["Accuracy"]

## Accuracy

## 0.6147186

cfmBoost$overall["Accuracy"]

## Accuracy

## 0.5367965

cfmAgreed$overall["Accuracy"]

## Accuracy

## 0.6656051

Question 2

Load the Alzheimer’s data using the following commands

**library**(gbm)

set.seed(3433)

**library**(AppliedPredictiveModeling)

data(AlzheimerDisease)

adData = data.frame(diagnosis,predictors)

inTrain = createDataPartition(adData$diagnosis, p = 3/4)[[1]]

training = adData[ inTrain,]

testing = adData[-inTrain,]

Set the seed to 62433 and predict diagnosis with all the other variables using a random forest (“rf”), boosted trees (“gbm”) and linear discriminant analysis (“lda”) model. Stack the predictions together using random forests (“rf”). What is the resulting accuracy on the test set? Is it better or worse than each of the individual predictions?

Answer

set.seed(62433)

modRF2 <- train(diagnosis ~ ., data=training, method="rf") *#, trControl=trainControl("cv"), number=3)*

modBoost2 <- train(diagnosis ~ ., data=training, method="gbm", verbose=FALSE)

modLDA2 <- train(diagnosis ~ ., data=training, method="lda", verbose=FALSE)

predRF2 <- predict(modRF2, testing)

predBoost2 <- predict(modBoost2, testing)

predLDA2 <- predict(modLDA2, testing)

dataCombined <- data.frame(predRF2, predBoost2, predLDA2, diagnosis=testing$diagnosis)

modCombined <- train(diagnosis ~ ., data=dataCombined, method="rf", verbose=FALSE)

## note: only 2 unique complexity parameters in default grid. Truncating the grid to 2 .

predCombined <- predict(modCombined, dataCombined)

cfmRF2 <- confusionMatrix(testing$diagnosis, predRF2)

cfmBoost2 <- confusionMatrix(testing$diagnosis, predBoost2)

cfmLDA2 <- confusionMatrix(testing$diagnosis, predLDA2)

cfmCombined <- confusionMatrix(testing$diagnosis, predCombined)

cfmRF2$overall["Accuracy"]

## Accuracy

## 0.7682927

cfmBoost2$overall["Accuracy"]

## Accuracy

## 0.7926829

cfmLDA2$overall["Accuracy"]

## Accuracy

## 0.7682927

cfmCombined$overall["Accuracy"]

## Accuracy

## 0.804878

|  |
| --- |
| ## Question 3 |
| Load the concrete data with the commands: |
| r set.seed(3523) library(AppliedPredictiveModeling) data(concrete) inTrain = createDataPartition(concrete$CompressiveStrength, p = 3/4)[[1]] training = concrete[ inTrain,] testing = concrete[-inTrain,] |
| Set the seed to 233 and fit a lasso model to predict Compressive Strength. Which variable is the last coefficient to be set to zero as the penalty increases? (Hint: it may be useful to look up ?plot.enet). |
| ### Answer |
| r set.seed(233) modLasso <- train(CompressiveStrength ~ ., data=training, method="lasso") plot.enet(modLasso$finalModel, xvar="penalty", use.color=TRUE) |
| C:\Users\Dipak\AppData\Local\Microsoft\Windows\INetCache\Content.MSO\E90FB4DC.tmp |

Question 4

Load the data on the number of visitors to the instructors blog from here:

<https://d396qusza40orc.cloudfront.net/predmachlearn/gaData.csv>

**library**(lubridate) *# For year() function below*

dat = read.csv("gaData.csv")

training = dat[year(dat$date) < 2012,]

testing = dat[(year(dat$date)) > 2011,]

tstrain = ts(training$visitsTumblr)

Fit a model using the bats() function in the forecast package to the training time series. Then forecast this model for the remaining time points. For how many of the testing points is the true value within the 95% prediction interval bounds?

Answer

modBats <- bats(tstrain)

forecastObj <- forecast(modBats, level=95, h=nrow(testing))

betweenVal <- sum(testing$visitsTumblr > forecastObj$lower & testing$visitsTumblr < forecastObj$upper)

betweenVal / nrow(testing) \* 100

## [1] 96.17021

Question 5

Load the concrete data with the commands:

set.seed(3523)

**library**(AppliedPredictiveModeling)

data(concrete)

inTrain = createDataPartition(concrete$CompressiveStrength, p = 3/4)[[1]]

training = concrete[ inTrain,]

testing = concrete[-inTrain,]

Set the seed to 325 and 􀃒t a support vector machine using the e1071 package to predict Compressive Strength using the default settings. Predict on the testing set. What is the RMSE?

Answer

set.seed(325)

modSvm <- svm(CompressiveStrength ~ ., data = training)

predSvm <- predict(modSvm, testing)

accSvm <- accuracy(predSvm, testing$CompressiveStrength)

data.frame(accSvm)["RMSE"]

## RMSE

## Test set 6.715009

* [Regularised Regression](https://datascience.study.foletta.org/C8_Practical_Machine_Learning/w4_notes.html#regularised-regression)
  + [Model Selection Approach](https://datascience.study.foletta.org/C8_Practical_Machine_Learning/w4_notes.html#model-selection-approach)
  + [Decomposing Prediction Error](https://datascience.study.foletta.org/C8_Practical_Machine_Learning/w4_notes.html#decomposing-prediction-error)
  + [High Dimensional Data](https://datascience.study.foletta.org/C8_Practical_Machine_Learning/w4_notes.html#high-dimensional-data)
    - [Hard Thresholding](https://datascience.study.foletta.org/C8_Practical_Machine_Learning/w4_notes.html#hard-thresholding)
    - [Regularised Regression](https://datascience.study.foletta.org/C8_Practical_Machine_Learning/w4_notes.html#regularised-regression-1)
    - [Ridge Regression](https://datascience.study.foletta.org/C8_Practical_Machine_Learning/w4_notes.html#ridge-regression)
    - [Lasso](https://datascience.study.foletta.org/C8_Practical_Machine_Learning/w4_notes.html#lasso)
* [Combining Predictors](https://datascience.study.foletta.org/C8_Practical_Machine_Learning/w4_notes.html#combining-predictors)
  + [Intuition](https://datascience.study.foletta.org/C8_Practical_Machine_Learning/w4_notes.html#intuition)
  + [Example](https://datascience.study.foletta.org/C8_Practical_Machine_Learning/w4_notes.html#example)
  + [Notes](https://datascience.study.foletta.org/C8_Practical_Machine_Learning/w4_notes.html#notes)
* [Forecasting](https://datascience.study.foletta.org/C8_Practical_Machine_Learning/w4_notes.html#forecasting)
  + [What is different?](https://datascience.study.foletta.org/C8_Practical_Machine_Learning/w4_notes.html#what-is-different)
  + [Example](https://datascience.study.foletta.org/C8_Practical_Machine_Learning/w4_notes.html#example-1)
  + [Time Series Decomposition](https://datascience.study.foletta.org/C8_Practical_Machine_Learning/w4_notes.html#time-series-decomposition)
  + [Training and Test Sets](https://datascience.study.foletta.org/C8_Practical_Machine_Learning/w4_notes.html#training-and-test-sets)
  + [Simple Moving Average](https://datascience.study.foletta.org/C8_Practical_Machine_Learning/w4_notes.html#simple-moving-average)
  + [Exponential Smoothing](https://datascience.study.foletta.org/C8_Practical_Machine_Learning/w4_notes.html#exponential-smoothing)
* [Unsupervised Prediction](https://datascience.study.foletta.org/C8_Practical_Machine_Learning/w4_notes.html#unsupervised-prediction)
  + [Example](https://datascience.study.foletta.org/C8_Practical_Machine_Learning/w4_notes.html#example-2)
  + [Notes](https://datascience.study.foletta.org/C8_Practical_Machine_Learning/w4_notes.html#notes-1)
* [Quiz](https://datascience.study.foletta.org/C8_Practical_Machine_Learning/w4_notes.html#quiz)
  + [Question 1](https://datascience.study.foletta.org/C8_Practical_Machine_Learning/w4_notes.html#question-1)
  + [Question 2](https://datascience.study.foletta.org/C8_Practical_Machine_Learning/w4_notes.html#question-2)
  + [Question 3](https://datascience.study.foletta.org/C8_Practical_Machine_Learning/w4_notes.html#question-3)
  + [Question 4](https://datascience.study.foletta.org/C8_Practical_Machine_Learning/w4_notes.html#question-4)
  + [Question 5](https://datascience.study.foletta.org/C8_Practical_Machine_Learning/w4_notes.html#question-5)

**Regularised Regression**

The basic idea is to fit a regression model (LM or GLM) then penalise or shrink the large coefficients corresponding with of the predictor values. We do this as it might help with the bias-variance tradeoff.

As an example, take:

Y=β0+β1X1+β2X2+ϵY=β0+β1X1+β2X2+ϵ

where X1X1 and X2X2 are nearly perfectly correlated (co-linear). You can approximate this model with:

Y=β0+(β1+β2)X1+ϵY=β0+(β1+β2)X1+ϵ

The result is:

* A good estimate of YY.
* The estimate of YY will be biased.
* We may reduce the variance in the estimate.

**Model Selection Approach**

* Divide data into train/test/validation.
* Treat validation as test data.
* Train all competing models on the train data.
* Pick the best one on validation.
* To appropriately assess performance on new data apply to test set.

Common problems:

* You may have limited data
* There may be computational complexity

**Decomposing Prediction Error**

Assume Yi=f(Xi)+ϵiYi=f(Xi)+ϵi, then the expected prediction error is the differentce between the outcome and the prediction of the outcome squared:

EPE(λ)=E[{Y−f^λ(X)}2]EPE(λ)=E[{Y−f^λ(X)}2]

Suppose f^λf^λ is the estimate from the training data and we look at a new data point X=x∗X=x∗. The difference between the outcome and the new data point can be decompsed into:

* Irreducible error σ2σ2
* The bias, which is the difference between our expected prediction and the truth, and
* The varaince of the estimate

E[{Y−f^λ(x∗)}2]=σ2+{E[f^λ(x∗)]−f(x∗)}2+var[f^λ(x0)]E[{Y−f^λ(x∗)}2]=σ2+{E[f^λ(x∗)]−f(x∗)}2+var[f^λ(x0)]

The goal is to reduce this overall quantity. You can’t reduce the irreducible erorr, but you can trade-off the bias and variance.

**High Dimensional Data**

Another issue is with high-dimensional data. If you’ve got more variables than there are observations, there is no single matrix that represents them:

library(ElemStatLearn)

prostate %>%

slice(1:5) %>%

lm(lpsa ~ ., data = .) %>%

summary()

##

## Call:

## lm(formula = lpsa ~ ., data = .)

##

## Residuals:

## ALL 5 residuals are 0: no residual degrees of freedom!

##

## Coefficients: (5 not defined because of singularities)

## Estimate Std. Error t value Pr(>|t|)

## (Intercept) 9.60615 NA NA NA

## lcavol 0.13901 NA NA NA

## lweight -0.79142 NA NA NA

## age 0.09516 NA NA NA

## lbph NA NA NA NA

## svi NA NA NA NA

## lcp NA NA NA NA

## gleason -2.08710 NA NA NA

## pgg45 NA NA NA NA

## trainTRUE NA NA NA NA

##

## Residual standard error: NaN on 0 degrees of freedom

## Multiple R-squared: 1, Adjusted R-squared: NaN

## F-statistic: NaN on 4 and 0 DF, p-value: NA

You see on the right that some of the variables are NA.

**Hard Thresholding**

One way to try and resolve this is to use **hard thresholding**:

* Model Y=f(X)+ϵY=f(X)+ϵ
* ASsume that it has a linear form f^λ(x)=x′βf^λ(x)=x′β
* Constrain λλ coefficients to be nonzero.
* Selection problem is after chosing λλ, figure out which p−λp−λ coefficients to make nonzero

**Regularised Regression**

This is another option. If the βjβjs are unconstrained, i.e we don’t claim that they have any particular form, the may explode if you have highly correlated variables, and they can be susceptible to high variance.

To control we might regularise/shrink the coefficients with a penalised residual sum of squares:

PRSS(β)=∑j=1n(Yj−∑i=1mβ1iXij)2+P(λ;β)PRSS(β)=∑j=1n(Yj−∑i=1mβ1iXij)2+P(λ;β)

The first part above is the minimisation of the outcome and the linear model fit squared, which is the standard RSS.

The second term is a penalty term which says if the penality term is too big it will shrink them back down.

**Ridge Regression**

Solve:

∑i=1N(yi−β0+∑j=1pxijβj)2+λ∑j=1pβ2j∑i=1N(yi−β0+∑j=1pxijβj)2+λ∑j=1pβj2

which is equivalanet to minimising the standard RSS subject to ∑pj=1β2j≤s∑j=1pβj2≤s where ss is inversey proportonal to λλ

Inclusion of the λλ may make the problem *non-singular* even if XTXXTX is not invertible.

* The tuning parameter λλ controls the size of the coefficients.
* As λ→0λ→0 we obtain the least sqaure solution.
* As λ→∞λ→∞ we have β^ridgeλ=∞=0β^λ=∞ridge=0
  + i.e. all of the coefficients go to zero.

**Lasso**

∑i=1N(yi−β0+∑j=1pxijβj)2+λ∑j=1p|βj|∑i=1N(yi−β0+∑j=1pxijβj)2+λ∑j=1p|βj|

The lasso shrinks coefficients but also sets some to zero, rather than the ridge regression which makes the coefficients approach zero. Thus it performs model selection.

**Combining Predictors**

* Combine classifiers by averaging/voting.
  + Combine boosting with a random forest.
* Combining classifiers improves accuracy, however it can reduce interpretability.
* Boosting, bagging and random forests are variants on this theme.

**Intuition**

If we have 5 completely independent classifiers, and if the accuracy is 70% for each, then

10×(0.7)3(0.3)2+5×(0.7)4(0.3)2+(0.7)510×(0.7)3(0.3)2+5×(0.7)4(0.3)2+(0.7)5

which is 83.7 majority vote accuracy.

**Example**

* Bagging, boosting and random forests
  + Usually combining similar classifiers
* Combining different classifiers
  + Model stacking
  + Model ensembling

library(ISLR)

library(modelr)

##

## Attaching package: 'modelr'

## The following object is masked from 'package:broom':

##

## bootstrap

library(caret)

## Loading required package: lattice

##

## Attaching package: 'caret'

## The following object is masked from 'package:purrr':

##

## lift

Wage %>%

select(-logwage) %>%

resample\_partition(c(train = .5, test = .2, validation = .3)) ->

wage\_smpl

*# Build two different models*

wage\_smpl$train %>%

as\_tibble() %>%

train(wage ~ ., method = 'glm', data = .) ->

wage\_glm

wage\_smpl$train %>%

as\_tibble() %>%

train(

wage ~ ., method = 'rf', data = .,

trControl = trainControl(method = 'cv'), number = 3

) ->

wage\_rf

wage\_smpl$test %>%

as\_tibble() %>%

mutate(

glm\_pred = predict(wage\_glm, newdata = .),

rf\_pred = predict(wage\_rf, newdata = .)

) -> test\_predictions

## Warning in model.matrix.default(Terms, m, contrasts = object$contrasts):

## partial argument match of 'contrasts' to 'contrasts.arg'

## Warning in predict.lm(object, newdata, se.fit, scale = 1, type = if (type

## == : prediction from a rank-deficient fit may be misleading

## Warning in model.matrix.default(Terms, m, contrasts = object$contrasts):

## partial argument match of 'contrasts' to 'contrasts.arg'

test\_predictions %>%

ggplot() +

geom\_point(aes(glm\_pred, rf\_pred, colour = wage)) +

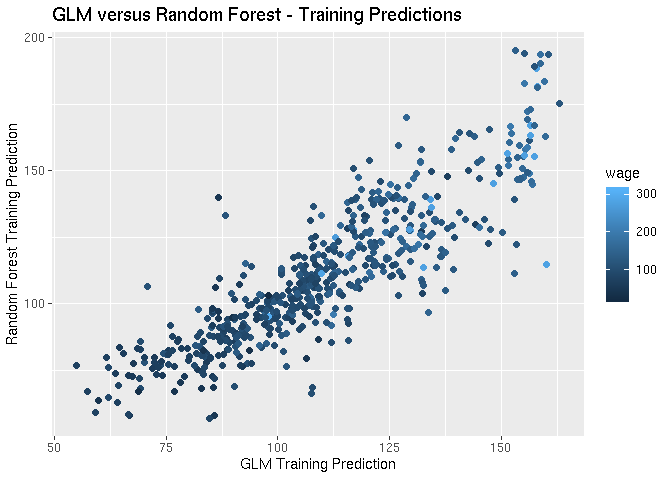
labs(

title = 'GLM versus Random Forest - Training Predictions',

x = 'GLM Training Prediction',

y = 'Random Forest Training Prediction'

)



Predictions are close but don’t line up completely. We then create a data frame consisting of the predictions from the two models as well as the repsonse variable from the **test set**:

test\_predictions %>%

select(glm\_pred, rf\_pred, wage) %>%

train(wage ~ ., data = ., method = 'gam') ->

combo\_model

## Loading required package: mgcv

## Loading required package: nlme

##

## Attaching package: 'nlme'

## The following object is masked from 'package:dplyr':

##

## collapse

## This is mgcv 1.8-28. For overview type 'help("mgcv-package")'.

*# Looking at the RMSE of the testing set with the GLM and the RF*

test\_predictions %>%

mutate(combo\_pred = predict(combo\_model)) %>%

summarise(

GLM\_RMSE = sqrt(sum((wage - glm\_pred)^2)),

RF\_RMSE = sqrt(sum((wage - rf\_pred)^2)),

COMBO\_RMSE = sqrt(sum((wage - combo\_pred)^2))

) %>%

gather('model', 'rmse')

## # A tibble: 3 x 2

## model rmse

## <chr> <dbl>

## 1 GLM\_RMSE 885.

## 2 RF\_RMSE 904.

## 3 COMBO\_RMSE 858.

Now we need to validate, as we’ve used the test set to blend the two models together:

wage\_smpl$validation %>%

as\_tibble() %>%

mutate(

glm\_pred = predict(wage\_glm, newdata = .),

rf\_pred = predict(wage\_rf, newdata = .)

) %>%

mutate(

combo\_pred = predict(combo\_model, newdata = .)

) %>%

summarise(

GLM\_RMSE = sqrt(sum((wage - glm\_pred)^2)),

RF\_RMSE = sqrt(sum((wage - rf\_pred)^2)),

COMBO\_RMSE = sqrt(sum((wage - combo\_pred)^2))

) %>%

gather('model', 'rmse')

## Warning in model.matrix.default(Terms, m, contrasts = object$contrasts):

## partial argument match of 'contrasts' to 'contrasts.arg'

## Warning in predict.lm(object, newdata, se.fit, scale = 1, type = if (type

## == : prediction from a rank-deficient fit may be misleading

## Warning in model.matrix.default(Terms, m, contrasts = object$contrasts):

## partial argument match of 'contrasts' to 'contrasts.arg'

## Warning in model.matrix.default(Terms, m, contrasts = object$contrasts):

## partial argument match of 'contrasts' to 'contrasts.arg'

## Warning in model.matrix.default(Terms[[i]], mf, contrasts =

## object$contrasts): partial argument match of 'contrasts' to 'contrasts.arg'

## # A tibble: 3 x 2

## model rmse

## <chr> <dbl>

## 1 GLM\_RMSE 1072.

## 2 RF\_RMSE 1107.

## 3 COMBO\_RMSE 1072.

We see the blended model also has a lower RMSE on the validation as well.

**Notes**

* Even simple blending can be useful.
* Typical model for binary/multiclass data:
  + Build an odd number of models
  + Predict with each model
  + Predict the class by the majority vote
* This can get dramatically more complicated.

One important note is that computational complexity can go through the roof.

**Forecasting**

This a very specific type of prediction problem, generally applied to time series data.

**What is different?**

* Data are dependent over time.
* Specific pattern types:
  + Trends: long term increase or decrease
  + Seasonal patterns: patterns related to time of week, month, year, etc.
  + Cycles: patterns that rise and fall periodically.
* Subsampling into training/test is more complicated
  + You can’t just randomly assign observations into training/test
  + Have to take into account the time
* Similar issues arise in spatial data
  + Dependency between nearby observations
  + Location specific effects
* Typically goal is to predict one or more observations into the future.
* All standard predictions can be used - **with caution**.

Need to beware of spurious correlations.

**Example**

Using the quantmod package:

library(ggfortify)

library(quantmod)

## Loading required package: xts

## Loading required package: zoo

##

## Attaching package: 'zoo'

## The following objects are masked from 'package:base':

##

## as.Date, as.Date.numeric

## Registered S3 method overwritten by 'xts':

## method from

## as.zoo.xts zoo

##

## Attaching package: 'xts'

## The following objects are masked from 'package:dplyr':

##

## first, last

## Loading required package: TTR

## Registered S3 method overwritten by 'quantmod':

## method from

## as.zoo.data.frame zoo

## Version 0.4-0 included new data defaults. See ?getSymbols.

*# Loads into the global environment as the symbol*

getSymbols('GOOG', src = "yahoo", from = Sys.Date() - 365 \* 2, to = Sys.Date())

## 'getSymbols' currently uses auto.assign=TRUE by default, but will

## use auto.assign=FALSE in 0.5-0. You will still be able to use

## 'loadSymbols' to automatically load data. getOption("getSymbols.env")

## and getOption("getSymbols.auto.assign") will still be checked for

## alternate defaults.

##

## This message is shown once per session and may be disabled by setting

## options("getSymbols.warning4.0"=FALSE). See ?getSymbols for details.

## [1] "GOOG"

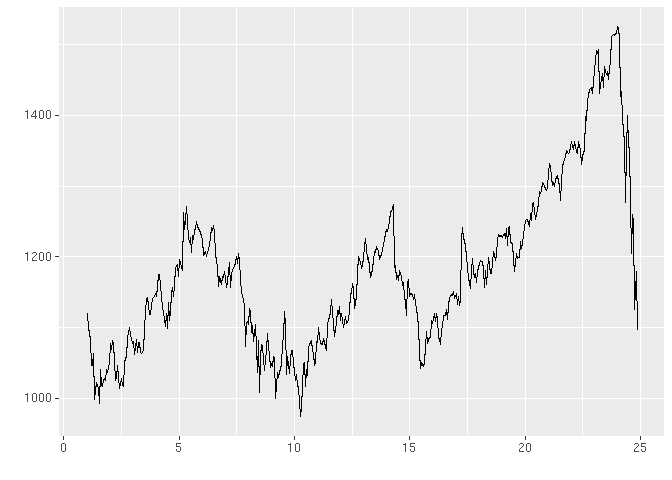
GOOG %>%

Op() %>%

ts(frequency = 21) ->

time\_series

time\_series %>% autoplot()



**Time Series Decomposition**

* **Trend**: consistently increasing pattern over time.
* **Seasonal**: when there is a pattern over a fixed period of time that recurs.
* **Cyclic**: when data rises and falls over non-fixed periods.

Can use the decompose() function in R:

time\_series %>%

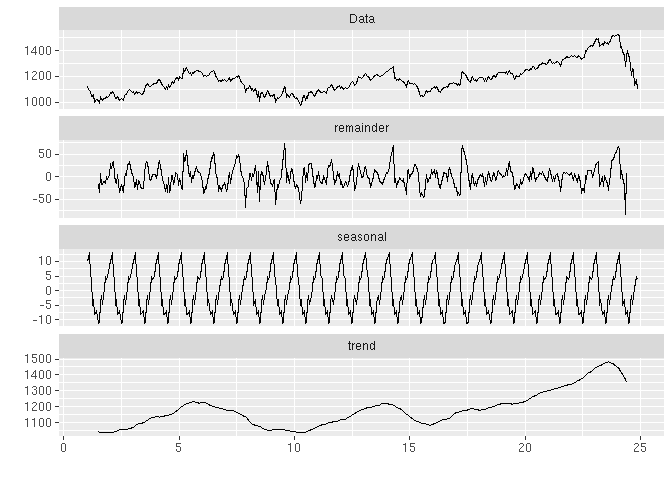
decompose() %>%

autoplot()

## Warning: attributes are not identical across measure variables;

## they will be dropped

## Warning: Removed 40 rows containing missing values (geom\_path).



**Training and Test Sets**

We build the training set for a particular window of time, then the test set is the next consecutive sets of points after that.

time\_series %>%

window(start = 1, end = 5) ->

ts\_train

time\_series %>%

window(start = 5, end = (7 - 0.01)) ->

ts\_test

**Simple Moving Average**

Yt=12×k+1∑j=−kkyt+jYt=12×k+1∑j=−kkyt+j

library(forecast)

## Registered S3 methods overwritten by 'forecast':

## method from

## autoplot.Arima ggfortify

## autoplot.acf ggfortify

## autoplot.ar ggfortify

## autoplot.bats ggfortify

## autoplot.decomposed.ts ggfortify

## autoplot.ets ggfortify

## autoplot.forecast ggfortify

## autoplot.stl ggfortify

## autoplot.ts ggfortify

## fitted.ar ggfortify

## fortify.ts ggfortify

## residuals.ar ggfortify

##

## Attaching package: 'forecast'

## The following object is masked from 'package:nlme':

##

## getResponse

ts\_train %>%

ma(order = 3) %>%

autoplot()



**Exponential Smoothing**

Weight nearby time points more heavily than time points that are farther away.

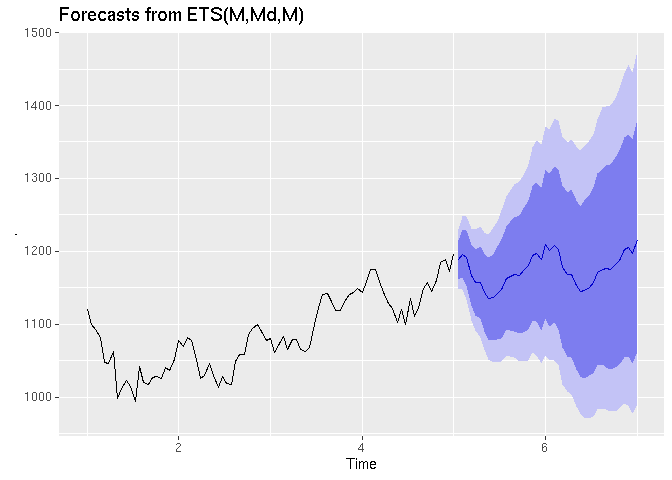
ts\_train %>%

ets(model = 'MMM') %>%

forecast() ->

ts\_train\_forecast

ts\_train\_forecast %>% autoplot()



ts\_train\_forecast %>% accuracy(ts\_test)

## ME RMSE MAE MPE MAPE MASE

## Training set 1.394682 15.84927 12.52706 0.1129408 1.158254 0.2961316

## Test set 41.819808 60.00343 49.44493 3.3678686 4.018683 1.1688463

## ACF1 Theil's U

## Training set 0.0030011 NA

## Test set 0.8162444 3.082

Best resource: [Forecasting: Principles and Practice](https://otexts.com/fpp2/)

**Unsupervised Prediction**

Sometimes you don’t know the labels for prediction.

To build a predictor:

* Create clusters
* Name clusters
* Build predictors for those clusters

Then in the new data set:

* Predict clusters

This adds several layers of complexity: creating the clusters is not perfectly noiseless, and coming up with the right names (interpretation) is challenging.

**Example**

library(modelr)

library(magrittr)

##

## Attaching package: 'magrittr'

## The following object is masked from 'package:purrr':

##

## set\_names

## The following object is masked from 'package:tidyr':

##

## extract

library(caret)

set.seed(1)

iris %>%

resample\_partition(c(train = .75, test = .25)) ->

iris\_smpl

iris\_smpl$train %>%

as\_tibble() %>%

select(-Species) %>%

kmeans(centers = 3) %>%

magrittr::extract2('cluster') %>%

factor() ->

iris\_train\_cluster

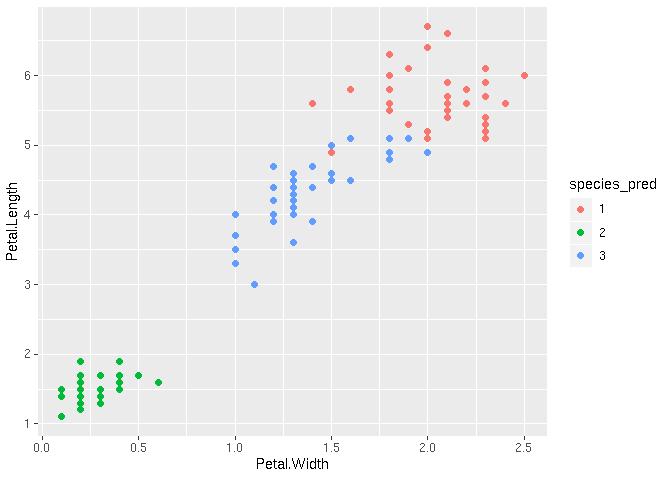
iris\_smpl$train %>%

as\_tibble() %>%

mutate(species\_pred = iris\_train\_cluster) %>%

ggplot() +

geom\_point(aes(Petal.Width, Petal.Length, colour = species\_pred))



labs(

title = 'Species K-Means',

x = 'Petal Width',

y = 'Petal Height',

colour = 'Species K-Means Prediction'

)

## $x

## [1] "Petal Width"

##

## $y

## [1] "Petal Height"

##

## $colour

## [1] "Species K-Means Prediction"

##

## $title

## [1] "Species K-Means"

##

## attr(,"class")

## [1] "labels"

We can then build a predictor based on these clusters

iris\_smpl$train %>%

as\_tibble() %>%

select(-Species) %>%

mutate(species\_pred = iris\_train\_cluster) %>%

train(species\_pred ~ ., method = 'rpart', data = .) ->

iris\_rpart

iris\_smpl$train %>%

as\_tibble() %>%

mutate(pred = predict.train(iris\_rpart)) %>%

{ table( .[['Species']], .[['pred']] ) }

##

## 1 2 3

## setosa 0 39 0

## versicolor 0 0 33

## virginica 29 0 11

We then apply on the test dataset:

iris\_smpl$test %>%

as\_tibble() %>%

mutate(pred = predict(iris\_rpart, newdata = .)) %>%

{ table( .$Species, .$pred) }

##

## 1 2 3

## setosa 0 11 0

## versicolor 0 0 17

## virginica 5 0 5

**Notes**

* The cl\_predict() function in the clue package provides similar functionality.
* Beware of over-interpretation of clusters.

**Quiz**

**Question 1**

For this quiz we will be using several R packages. R package versions change over time, the right answers have been checked using the following versions of the packages.

AppliedPredictiveModeling: v1.1.6

caret: v6.0.47

ElemStatLearn: v2012.04-0

pgmm: v1.1

rpart: v4.1.8

gbm: v2.1

lubridate: v1.3.3

forecast: v5.6

e1071: v1.6.4

If you aren’t using these versions of the packages, your answers may not exactly match the right answer, but hopefully should be close.

Load the vowel.train and vowel.test data sets:

library(ElemStatLearn)

data(vowel.train)

data(vowel.test)

Set the variable y to be a factor variable in both the training and test set. Then set the seed to 33833. Fit (1) a random forest predictor relating the factor variable y to the remaining variables and (2) a boosted predictor using the “gbm” method. Fit these both with the train() command in the caret package.

What are the accuracies for the two approaches on the test data set? What is the accuracy among the test set samples where the two methods agree?

* RF Accuracy = 0.3233 GBM Accuracy = 0.8371 Agreement Accuracy = 0.9983
* RF Accuracy = 0.6082 GBM Accuracy = 0.5152 Agreement Accuracy = 0.6361
* RF Accuracy = 0.9987 GBM Accuracy = 0.5152 Agreement Accuracy = 0.9985
* RF Accuracy = 0.6082 GBM Accuracy = 0.5152 Agreement Accuracy = 0.5325

vowel.train %>%

mutate(y = factor(y)) ->

vowel.train

vowel.test %>%

mutate(y = factor(y)) ->

vowel.test

set.seed(33833)

**Question 2**

Load the Alzheimer’s data using the following commands:

library(caret)

library(gbm)

## Loaded gbm 2.1.5

set.seed(3433)

library(AppliedPredictiveModeling)

data(AlzheimerDisease)

adData = data.frame(diagnosis,predictors)

inTrain = createDataPartition(adData$diagnosis, p = 3/4)[[1]]

## Warning in seq.default(along = y): partial argument match of 'along' to

## 'along.with'

## Warning in seq.default(along = x): partial argument match of 'along' to

## 'along.with'

training = adData[ inTrain,]

testing = adData[-inTrain,]

Set the seed to 62433 and predict diagnosis with all the other variables using a random forest (“rf”), boosted trees (“gbm”) and linear discriminant analysis (“lda”) model. Stack the predictions together using random forests (“rf”). What is the resulting accuracy on the test set? Is it better or worse than each of the individual predictions?

* Stacked Accuracy: 0.80 is better than all three other methods
* Stacked Accuracy: 0.80 is better than random forests and lda and the same as boosting.
* Stacked Accuracy: 0.76 is better than random forests and boosting, but not lda.
* Stacked Accuracy: 0.76 is better than lda but not random forests or boosting.

**Question 3**

Load the concrete data with the commands:

set.seed(3523)

library(AppliedPredictiveModeling)

data(concrete)

inTrain = createDataPartition(concrete$CompressiveStrength, p = 3/4)[[1]]

## Warning in seq.default(0, 1, length = groups): partial argument match of

## 'length' to 'length.out'

## Warning in seq.default(along = y): partial argument match of 'along' to

## 'along.with'

## Warning in seq.default(along = x): partial argument match of 'along' to

## 'along.with'

training = concrete[ inTrain,]

testing = concrete[-inTrain,]

Set the seed to 233 and fit a lasso model to predict Compressive Strength. Which variable is the last coefficient to be set to zero as the penalty increases? (Hint: it may be useful to look up ?plot.enet).

* CoarseAggregate
* Cement
* Water
* Age

**Question 4**

Load the data on the number of visitors to the instructors blog from here:

<https://d396qusza40orc.cloudfront.net/predmachlearn/gaData.csv>

Using the commands:

library(lubridate) *# For year() function below*

##

## Attaching package: 'lubridate'

## The following object is masked from 'package:base':

##

## date

dat = read.csv("https://d396qusza40orc.cloudfront.net/predmachlearn/gaData.csv")

training = dat[year(dat$date) < 2012,]

testing = dat[(year(dat$date)) > 2011,]

tstrain = ts(training$visitsTumblr)

Fit a model using the bats() function in the forecast package to the training time series. Then forecast this model for the remaining time points. For how many of the testing points is the true value within the 95% prediction interval bounds?

* 94%
* 98%
* 96%
* 93%

**Question 5**

Load the concrete data with the commands:

set.seed(3523)

library(AppliedPredictiveModeling)

data(concrete)

inTrain = createDataPartition(concrete$CompressiveStrength, p = 3/4)[[1]]

## Warning in seq.default(0, 1, length = groups): partial argument match of

## 'length' to 'length.out'

## Warning in seq.default(along = y): partial argument match of 'along' to

## 'along.with'

## Warning in seq.default(along = x): partial argument match of 'along' to

## 'along.with'

training = concrete[ inTrain,]

testing = concrete[-inTrain,]

Set the seed to 325 and fit a support vector machine using the e1071 package to predict Compressive Strength using the default settings. Predict on the testing set. What is the RMSE?

* 35.59
* 6.72
* 6.93
* 107.44