**Setup**

All analyses are done in R using RStudio. For detailed session information including R version, operating system and package versions, see the sessionInfo() output at the end of this document.

All figures are produced with ggplot2.

* libraries

library(tidyverse) # for tidy data analysis

library(readr) # for fast reading of input files

library(mice) # mice package for Multivariate Imputation by Chained Equations (MICE)

**Data preparation**

**The dataset**

The dataset I am using in these example analyses, is the **Breast Cancer Wisconsin (Diagnostic) Dataset**. The data was downloaded from the [UC Irvine Machine Learning Repository](http://archive.ics.uci.edu/ml/datasets/Breast+Cancer+Wisconsin+%28Diagnostic%29).

The first dataset looks at the predictor classes:

* malignant or
* benign breast mass.

The features characterise cell nucleus properties and were generated from image analysis of [fine needle aspirates (FNA)](https://en.wikipedia.org/wiki/Fine-needle_aspiration) of breast masses:

* Sample ID (code number)
* Clump thickness
* Uniformity of cell size
* Uniformity of cell shape
* Marginal adhesion
* Single epithelial cell size
* Number of bare nuclei
* Bland chromatin
* Number of normal nuclei
* Mitosis
* Classes, i.e. diagnosis

bc\_data <- read\_delim("/Users/shiringlander/Documents/Github/intro\_to\_ml\_workshop/intro\_to\_ml\_uni\_heidelberg/datasets/breast-cancer-wisconsin.data.txt",

delim = ",",

col\_names = c("sample\_code\_number",

"clump\_thickness",

"uniformity\_of\_cell\_size",

"uniformity\_of\_cell\_shape",

"marginal\_adhesion",

"single\_epithelial\_cell\_size",

"bare\_nuclei",

"bland\_chromatin",

"normal\_nucleoli",

"mitosis",

"classes")) %>%

mutate(bare\_nuclei = as.numeric(bare\_nuclei),

classes = ifelse(classes == "2", "benign",

ifelse(classes == "4", "malignant", NA)))

summary(bc\_data)

## sample\_code\_number clump\_thickness uniformity\_of\_cell\_size

## Min. : 61634 Min. : 1.000 Min. : 1.000

## 1st Qu.: 870688 1st Qu.: 2.000 1st Qu.: 1.000

## Median : 1171710 Median : 4.000 Median : 1.000

## Mean : 1071704 Mean : 4.418 Mean : 3.134

## 3rd Qu.: 1238298 3rd Qu.: 6.000 3rd Qu.: 5.000

## Max. :13454352 Max. :10.000 Max. :10.000

##

## uniformity\_of\_cell\_shape marginal\_adhesion single\_epithelial\_cell\_size

## Min. : 1.000 Min. : 1.000 Min. : 1.000

## 1st Qu.: 1.000 1st Qu.: 1.000 1st Qu.: 2.000

## Median : 1.000 Median : 1.000 Median : 2.000

## Mean : 3.207 Mean : 2.807 Mean : 3.216

## 3rd Qu.: 5.000 3rd Qu.: 4.000 3rd Qu.: 4.000

## Max. :10.000 Max. :10.000 Max. :10.000

##

## bare\_nuclei bland\_chromatin normal\_nucleoli mitosis

## Min. : 1.000 Min. : 1.000 Min. : 1.000 Min. : 1.000

## 1st Qu.: 1.000 1st Qu.: 2.000 1st Qu.: 1.000 1st Qu.: 1.000

## Median : 1.000 Median : 3.000 Median : 1.000 Median : 1.000

## Mean : 3.545 Mean : 3.438 Mean : 2.867 Mean : 1.589

## 3rd Qu.: 6.000 3rd Qu.: 5.000 3rd Qu.: 4.000 3rd Qu.: 1.000

## Max. :10.000 Max. :10.000 Max. :10.000 Max. :10.000

## NA's :16

## classes

## Length:699

## Class :character

## Mode :character

##

##

##

##

**Missing data**

# how many NAs are in the data

md.pattern(bc\_data, plot = FALSE)

## sample\_code\_number clump\_thickness uniformity\_of\_cell\_size

## 683 1 1 1

## 16 1 1 1

## 0 0 0

## uniformity\_of\_cell\_shape marginal\_adhesion single\_epithelial\_cell\_size

## 683 1 1 1

## 16 1 1 1

## 0 0 0

## bland\_chromatin normal\_nucleoli mitosis classes bare\_nuclei

## 683 1 1 1 1 1 0

## 16 1 1 1 1 0 1

## 0 0 0 0 16 16

bc\_data <- bc\_data %>%

drop\_na() %>%

select(classes, everything(), -sample\_code\_number)

head(bc\_data)

## # A tibble: 6 x 10

## classes clump\_thickness uniformity\_of\_cell\_si… uniformity\_of\_cell\_sha…

##

## 1 benign 5 1 1

## 2 benign 5 4 4

## 3 benign 3 1 1

## 4 benign 6 8 8

## 5 benign 4 1 1

## 6 malignant 8 10 10

## # ... with 6 more variables: marginal\_adhesion ,

## # single\_epithelial\_cell\_size , bare\_nuclei ,

## # bland\_chromatin , normal\_nucleoli , mitosis

Missing values can be imputed with the *mice* package.

More info and tutorial with code: <https://shirinsplayground.netlify.com/2018/04/flu_prediction/>

**Data exploration**

* Response variable for classification

ggplot(bc\_data, aes(x = classes, fill = classes)) +

geom\_bar()



Code on dealing with unbalanced classes:

But because I had gotten a few questions regarding this, I thought it would be worthwhile to explain over- and under-sampling techniques in more detail and show how you can very easily implement them with caret.

library(caret)

### Unbalanced data

In this context, unbalanced data refers to classification problems where we have unequal instances for different classes. Having unbalanced data is actually very common in general, but it is especially prevalent when working with disease data where we usually have more healthy control samples than disease cases. Even more extreme unbalance is seen with fraud detection, where e.g. most credit card uses are okay and only very few will be fraudulent.

summary(bc\_data$classes)

## benign malignant

## 458 241

### Why is unbalanced data a problem in machine learning?

Most machine learning classification algorithms are sensitive to unbalance in the predictor classes. Let’s consider an even more extreme example than our breast cancer dataset: assume we had 10 malignant vs 90 benign samples. A machine learning model that has been trained and tested on such a dataset could now predict “benign” for all samples and still gain a very high accuracy. An unbalanced dataset will bias the prediction model towards the more common class!

### How to balance data for modeling

The basic theoretical concepts behind over- and under-sampling are very simple:

* With under-sampling, we randomly select a subset of samples from the class with more instances to match the number of samples coming from each class. In our example, we would randomly pick 241 out of the 458 benign cases. The main disadvantage of under-sampling is that we lose potentially relevant information from the left-out samples.
* With oversampling, we randomly duplicate samples from the class with fewer instances or we generate additional instances based on the data that we have, so as to match the number of samples in each class. While we avoid losing information with this approach, we also run the risk of overfitting our model as we are more likely to get the same samples in the training and in the test data, i.e. the test data is no longer independent from training data. This would lead to an overestimation of our model’s performance and generalizability.

In reality though, we should not simply perform over- or under-sampling on our training data and then run the model. We need to account for cross-validation and perform over- or under-sampling on each fold independently to get an honest estimate of model performance!

#### Modeling the original unbalanced data

Here is the same model I used in my webinar example: I randomly divide the data into training and test sets (stratified by class) and perform Random Forest modeling with 10 x 10 repeated cross-validation. Final model performance is then measured on the test set.

set.seed(42)

index <- createDataPartition(bc\_data$classes, p = 0.7, list = FALSE)

train\_data <- bc\_data[index, ]

test\_data <- bc\_data[-index, ]

set.seed(42)

model\_rf <- caret::train(classes ~ .,

data = train\_data,

method = "rf",

preProcess = c("scale", "center"),

trControl = trainControl(method = "repeatedcv",

number = 10,

repeats = 10,

verboseIter = FALSE))

final <- data.frame(actual = test\_data$classes,

predict(model\_rf, newdata = test\_data, type = "prob"))

final$predict <- ifelse(final$benign > 0.5, "benign", "malignant")

cm\_original <- confusionMatrix(final$predict, test\_data$classes)

#### Under-sampling

Luckily, caret makes it very easy to incorporate over- and under-sampling techniques with cross-validation resampling. We can simply add the sampling option to our trainControl and choose down for under- (also called down-) sampling. The rest stays the same as with our original model.

ctrl <- trainControl(method = "repeatedcv",

number = 10,

repeats = 10,

verboseIter = FALSE,

sampling = "down")

set.seed(42)

model\_rf\_under <- caret::train(classes ~ .,

data = train\_data,

method = "rf",

preProcess = c("scale", "center"),

trControl = ctrl)

final\_under <- data.frame(actual = test\_data$classes,

predict(model\_rf\_under, newdata = test\_data, type = "prob"))

final\_under$predict <- ifelse(final\_under$benign > 0.5, "benign", "malignant")

cm\_under <- confusionMatrix(final\_under$predict, test\_data$classes)

#### Oversampling

For over- (also called up-) sampling we simply specify sampling = "up".

ctrl <- trainControl(method = "repeatedcv",

number = 10,

repeats = 10,

verboseIter = FALSE,

sampling = "up")

set.seed(42)

model\_rf\_over <- caret::train(classes ~ .,

data = train\_data,

method = "rf",

preProcess = c("scale", "center"),

trControl = ctrl)

final\_over <- data.frame(actual = test\_data$classes,

predict(model\_rf\_over, newdata = test\_data, type = "prob"))

final\_over$predict <- ifelse(final\_over$benign > 0.5, "benign", "malignant")

cm\_over <- confusionMatrix(final\_over$predict, test\_data$classes)

#### ROSE

Besides over- and under-sampling, there are hybrid methods that combine under-sampling with the generation of additional data. Two of the most popular are ROSE and SMOTE.

From Nicola Lunardon, Giovanna Menardi and Nicola Torelli’s **“ROSE: A Package for Binary Imbalanced Learning”** (R Journal, 2014, Vol. 6 Issue 1, p. 79): “The ROSE package provides functions to deal with binary classification problems in the presence of imbalanced classes. Artificial balanced samples are generated according to a smoothed bootstrap approach and allow for aiding both the phases of estimation and accuracy evaluation of a binary classifier in the presence of a rare class. Functions that implement more traditional remedies for the class imbalance and different metrics to evaluate accuracy are also provided. These are estimated by holdout, bootstrap, or cross-validation methods.”

You implement them the same way as before, this time choosing sampling = "rose"…

ctrl <- trainControl(method = "repeatedcv",

number = 10,

repeats = 10,

verboseIter = FALSE,

sampling = "rose")

set.seed(42)

model\_rf\_rose <- caret::train(classes ~ .,

data = train\_data,

method = "rf",

preProcess = c("scale", "center"),

trControl = ctrl)

final\_rose <- data.frame(actual = test\_data$classes,

predict(model\_rf\_rose, newdata = test\_data, type = "prob"))

final\_rose$predict <- ifelse(final\_rose$benign > 0.5, "benign", "malignant")

cm\_rose <- confusionMatrix(final\_rose$predict, test\_data$classes)

#### SMOTE

… or by choosing sampling = "smote" in the trainControl settings.

From Nitesh V. Chawla, Kevin W. Bowyer, Lawrence O. Hall and W. Philip Kegelmeyer’s **“SMOTE: Synthetic Minority Over-sampling Technique”** (Journal of Artificial Intelligence Research, 2002, Vol. 16, pp. 321–357): “This paper shows that a combination of our method of over-sampling the minority (abnormal) class and under-sampling the majority (normal) class can achieve better classifier performance (in ROC space) than only under-sampling the majority class. This paper also shows that a combination of our method of over-sampling the minority class and under-sampling the majority class can achieve better classifier performance (in ROC space) than varying the loss ratios in Ripper or class priors in Naive Bayes. Our method of over-sampling the minority class involves creating synthetic minority class examples.”

ctrl <- trainControl(method = "repeatedcv",

number = 10,

repeats = 10,

verboseIter = FALSE,

sampling = "smote")

set.seed(42)

model\_rf\_smote <- caret::train(classes ~ .,

data = train\_data,

method = "rf",

preProcess = c("scale", "center"),

trControl = ctrl)

final\_smote <- data.frame(actual = test\_data$classes,

predict(model\_rf\_smote, newdata = test\_data, type = "prob"))

final\_smote$predict <- ifelse(final\_smote$benign > 0.5, "benign", "malignant")

cm\_smote <- confusionMatrix(final\_smote$predict, test\_data$classes)

### Predictions

Now let’s compare the predictions of all these models:

models <- list(original = model\_rf,

under = model\_rf\_under,

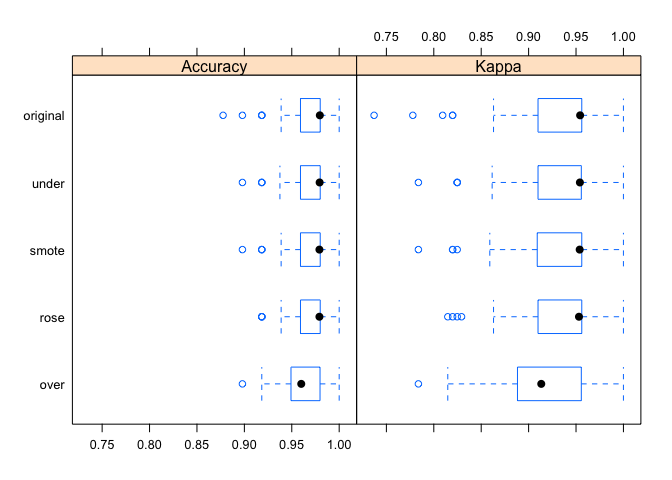
over = model\_rf\_over,

smote = model\_rf\_smote,

rose = model\_rf\_rose)

resampling <- resamples(models)

bwplot(resampling)



library(dplyr)

comparison <- data.frame(model = names(models),

Sensitivity = rep(NA, length(models)),

Specificity = rep(NA, length(models)),

Precision = rep(NA, length(models)),

Recall = rep(NA, length(models)),

F1 = rep(NA, length(models)))

for (name in names(models)) {

model <- get(paste0("cm\_", name))

comparison[comparison$model == name, ] <- filter(comparison, model == name) %>%

mutate(Sensitivity = model$byClass["Sensitivity"],

Specificity = model$byClass["Specificity"],

Precision = model$byClass["Precision"],

Recall = model$byClass["Recall"],

F1 = model$byClass["F1"])

}

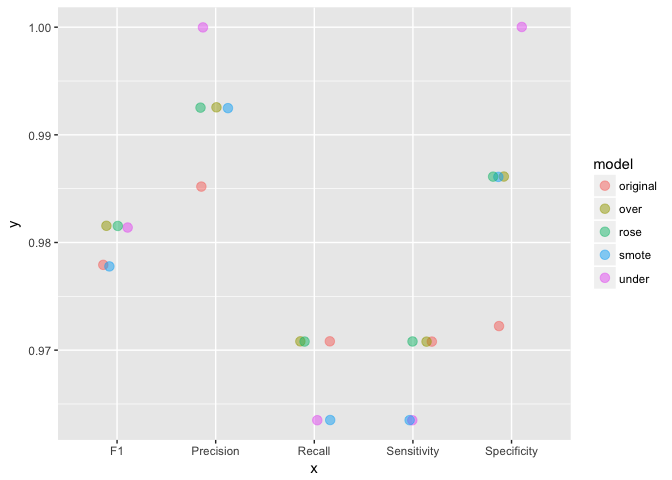
library(tidyr)

comparison %>%

gather(x, y, Sensitivity:F1) %>%

ggplot(aes(x = x, y = y, color = model)) +

geom\_jitter(width = 0.2, alpha = 0.5, size = 3)



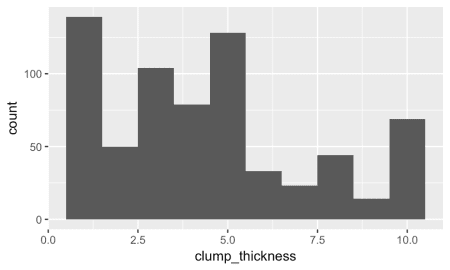
With this small dataset, we can already see how the different techniques can influence model performance. Sensitivity (or recall) describes the proportion of benign cases that have been predicted correctly, while specificity describes the proportion of malignant cases that have been predicted correctly. Precision describes the true positives, i.e. the proportion of benign predictions that were actual from benign samples. F1 is the weighted average of precision and sensitivity/ recall.

Here, all four methods improved specificity and precision compared to the original model. Under-sampling, over-sampling and ROSE additionally improved precision and the F1 score.

Response variable for regression

ggplot(bc\_data, aes(x = clump\_thickness)) +

geom\_histogram(bins = 10)



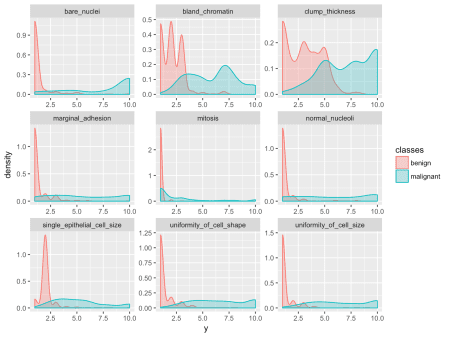
* Features

gather(bc\_data, x, y, clump\_thickness:mitosis) %>%

ggplot(aes(x = y, color = classes, fill = classes)) +

geom\_density(alpha = 0.3) +

facet\_wrap( ~ x, scales = "free", ncol = 3)



* Correlation graphs

co\_mat\_benign <- filter(bc\_data, classes == "benign") %>%

select(-1) %>%

cor()

co\_mat\_malignant <- filter(bc\_data, classes == "malignant") %>%

select(-1) %>%

cor()

library(igraph)

g\_benign <- graph.adjacency(co\_mat\_benign,

weighted = TRUE,

diag = FALSE,

mode = "upper")

g\_malignant <- graph.adjacency(co\_mat\_malignant,

weighted = TRUE,

diag = FALSE,

mode = "upper")

# http://kateto.net/networks-r-igraph

cut.off\_b <- mean(E(g\_benign)$weight)

cut.off\_m <- mean(E(g\_malignant)$weight)

g\_benign\_2 <- delete\_edges(g\_benign, E(g\_benign)[weight < cut.off\_b])

g\_malignant\_2 <- delete\_edges(g\_malignant, E(g\_malignant)[weight < cut.off\_m])

c\_g\_benign\_2 <- cluster\_fast\_greedy(g\_benign\_2)

c\_g\_malignant\_2 <- cluster\_fast\_greedy(g\_malignant\_2)

par(mfrow = c(1,2))

plot(c\_g\_benign\_2, g\_benign\_2,

vertex.size = colSums(co\_mat\_benign) \* 10,

vertex.frame.color = NA,

vertex.label.color = "black",

vertex.label.cex = 0.8,

edge.width = E(g\_benign\_2)$weight \* 15,

layout = layout\_with\_fr(g\_benign\_2),

main = "Benign tumors")

plot(c\_g\_malignant\_2, g\_malignant\_2,

vertex.size = colSums(co\_mat\_malignant) \* 10,

vertex.frame.color = NA,

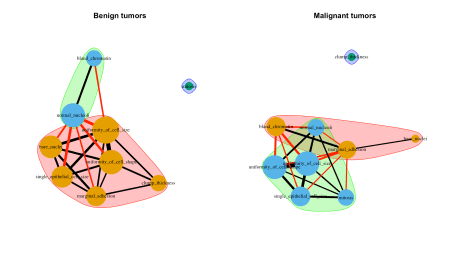
vertex.label.color = "black",

vertex.label.cex = 0.8,

edge.width = E(g\_malignant\_2)$weight \* 15,

layout = layout\_with\_fr(g\_malignant\_2),

main = "Malignant tumors")



**Principal Component Analysis**

library(ellipse)

# perform pca and extract scores

pcaOutput <- prcomp(as.matrix(bc\_data[, -1]), scale = TRUE, center = TRUE)

pcaOutput2 <- as.data.frame(pcaOutput$x)

# define groups for plotting

pcaOutput2$groups <- bc\_data$classes

centroids <- aggregate(cbind(PC1, PC2) ~ groups, pcaOutput2, mean)

conf.rgn <- do.call(rbind, lapply(unique(pcaOutput2$groups), function(t)

data.frame(groups = as.character(t),

ellipse(cov(pcaOutput2[pcaOutput2$groups == t, 1:2]),

centre = as.matrix(centroids[centroids$groups == t, 2:3]),

level = 0.95),

stringsAsFactors = FALSE)))

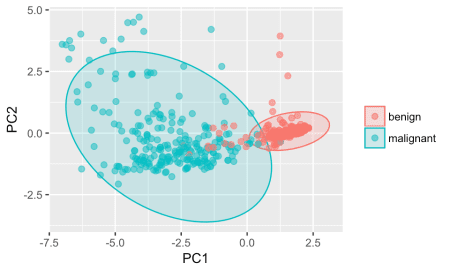
ggplot(data = pcaOutput2, aes(x = PC1, y = PC2, group = groups, color = groups)) +

geom\_polygon(data = conf.rgn, aes(fill = groups), alpha = 0.2) +

geom\_point(size = 2, alpha = 0.6) +

labs(color = "",

fill = "")



**Multidimensional Scaling**

select(bc\_data, -1) %>%

dist() %>%

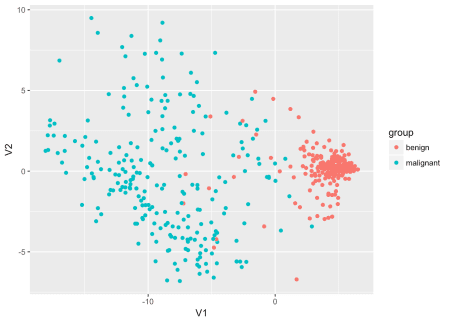
cmdscale %>%

as.data.frame() %>%

mutate(group = bc\_data$classes) %>%

ggplot(aes(x = V1, y = V2, color = group)) +

geom\_point()



**t-SNE dimensionality reduction**

library(tsne)

select(bc\_data, -1) %>%

dist() %>%

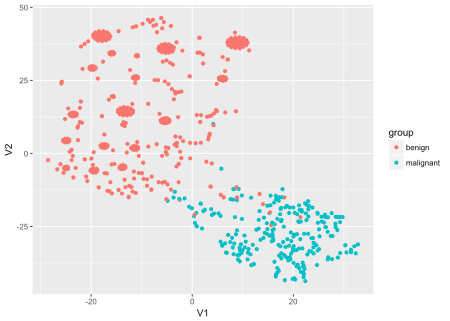
tsne() %>%

as.data.frame() %>%

mutate(group = bc\_data$classes) %>%

ggplot(aes(x = V1, y = V2, color = group)) +

geom\_point()



**Machine Learning packages for R**

# configure multicore

library(doParallel)

cl <- makeCluster(detectCores())

registerDoParallel(cl)

library(caret)

**Training, validation and test data**

set.seed(42)

index <- createDataPartition(bc\_data$classes, p = 0.7, list = FALSE)

train\_data <- bc\_data[index, ]

test\_data <- bc\_data[-index, ]

bind\_rows(data.frame(group = "train", train\_data),

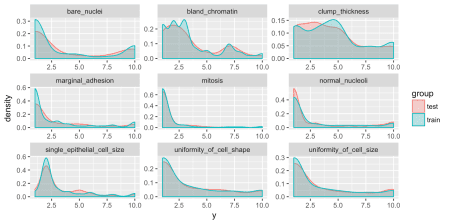
data.frame(group = "test", test\_data)) %>%

gather(x, y, clump\_thickness:mitosis) %>%

ggplot(aes(x = y, color = group, fill = group)) +

geom\_density(alpha = 0.3) +

facet\_wrap( ~ x, scales = "free", ncol = 3)



**Regression**

set.seed(42)

model\_glm <- caret::train(clump\_thickness ~ .,

data = train\_data,

method = "glm",

preProcess = c("scale", "center"),

trControl = trainControl(method = "repeatedcv",

number = 10,

repeats = 10,

savePredictions = TRUE,

verboseIter = FALSE))

model\_glm

## Generalized Linear Model

##

## 479 samples

## 9 predictor

##

## Pre-processing: scaled (9), centered (9)

## Resampling: Cross-Validated (10 fold, repeated 10 times)

## Summary of sample sizes: 432, 431, 432, 431, 431, 431, ...

## Resampling results:

##

## RMSE Rsquared MAE

## 1.972314 0.5254215 1.648832

predictions <- predict(model\_glm, test\_data)

# model\_glm$finalModel$linear.predictors == model\_glm$finalModel$fitted.values

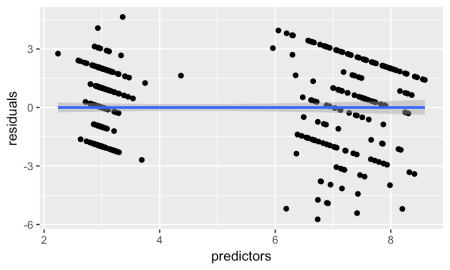
data.frame(residuals = resid(model\_glm),

predictors = model\_glm$finalModel$linear.predictors) %>%

ggplot(aes(x = predictors, y = residuals)) +

geom\_jitter() +

geom\_smooth(method = "lm")



# y == train\_data$clump\_thickness

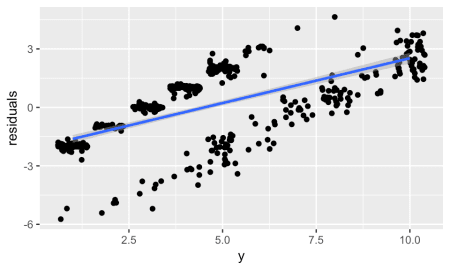
data.frame(residuals = resid(model\_glm),

y = model\_glm$finalModel$y) %>%

ggplot(aes(x = y, y = residuals)) +

geom\_jitter() +

geom\_smooth(method = "lm")



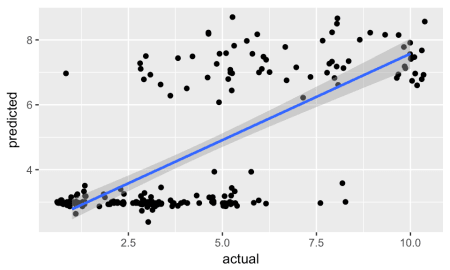
data.frame(actual = test\_data$clump\_thickness,

predicted = predictions) %>%

ggplot(aes(x = actual, y = predicted)) +

geom\_jitter() +

geom\_smooth(method = "lm")



**Classification**

**Decision trees**

library(rpart)

library(rpart.plot)

set.seed(42)

fit <- rpart(classes ~ .,

data = train\_data,

method = "class",

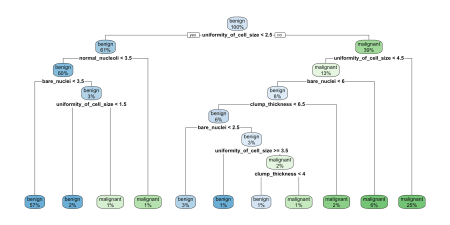
control = rpart.control(xval = 10,

minbucket = 2,

cp = 0),

parms = list(split = "information"))

rpart.plot(fit, extra = 100)



**Random Forests**

Random Forests predictions are based on the generation of multiple classification trees. They can be used for both, classification and regression tasks. Here, I show a classification task.

set.seed(42)

model\_rf <- caret::train(classes ~ .,

data = train\_data,

method = "rf",

preProcess = c("scale", "center"),

trControl = trainControl(method = "repeatedcv",

number = 5,

repeats = 3,

savePredictions = TRUE,

verboseIter = FALSE))

When you specify savePredictions = TRUE, you can access the cross-validation resuls with model\_rf$pred.

model\_rf

## Random Forest

##

## 479 samples

## 9 predictor

## 2 classes: 'benign', 'malignant'

##

## Pre-processing: scaled (9), centered (9)

## Resampling: Cross-Validated (10 fold, repeated 10 times)

## Summary of sample sizes: 432, 431, 431, 431, 431, 431, ...

## Resampling results across tuning parameters:

##

## mtry Accuracy Kappa

## 2 0.9776753 0.9513499

## 5 0.9757957 0.9469999

## 9 0.9714200 0.9370285

##

## Accuracy was used to select the optimal model using the largest value.

## The final value used for the model was mtry = 2.

model\_rf$finalModel$confusion

## benign malignant class.error

## benign 304 7 0.02250804

## malignant 5 163 0.02976190

**Dealing with unbalanced data**

Luckily, caret makes it very easy to incorporate over- and under-sampling techniques with cross-validation resampling. We can simply add the sampling option to our trainControl and choose down for under- (also called down-) sampling. The rest stays the same as with our original model.

set.seed(42)

model\_rf\_down <- caret::train(classes ~ .,

data = train\_data,

method = "rf",

preProcess = c("scale", "center"),

trControl = trainControl(method = "repeatedcv",

number = 10,

repeats = 10,

savePredictions = TRUE,

verboseIter = FALSE,

sampling = "down"))

model\_rf\_down

## Random Forest

##

## 479 samples

## 9 predictor

## 2 classes: 'benign', 'malignant'

##

## Pre-processing: scaled (9), centered (9)

## Resampling: Cross-Validated (10 fold, repeated 10 times)

## Summary of sample sizes: 432, 431, 431, 431, 431, 431, ...

## Addtional sampling using down-sampling prior to pre-processing

##

## Resampling results across tuning parameters:

##

## mtry Accuracy Kappa

## 2 0.9797503 0.9563138

## 5 0.9741198 0.9438326

## 9 0.9699578 0.9346310

##

## Accuracy was used to select the optimal model using the largest value.

## The final value used for the model was mtry = 2.

**Feature Importance**

imp <- model\_rf$finalModel$importance

imp[order(imp, decreasing = TRUE), ]

## uniformity\_of\_cell\_size uniformity\_of\_cell\_shape

## 43.936945 39.840595

## bare\_nuclei bland\_chromatin

## 33.820345 31.984813

## normal\_nucleoli single\_epithelial\_cell\_size

## 21.686039 17.761202

## clump\_thickness marginal\_adhesion

## 16.318817 9.518437

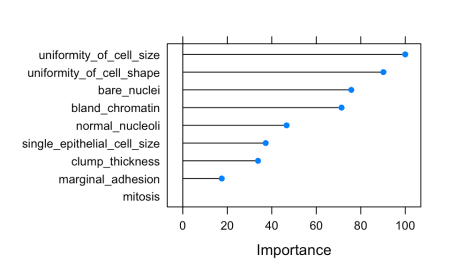
## mitosis

## 2.220633

# estimate variable importance

importance <- varImp(model\_rf, scale = TRUE)

plot(importance)



* predicting test data

confusionMatrix(predict(model\_rf, test\_data), as.factor(test\_data$classes))

## Confusion Matrix and Statistics

##

## Reference

## Prediction benign malignant

## benign 128 4

## malignant 5 67

##

## Accuracy : 0.9559

## 95% CI : (0.9179, 0.9796)

## No Information Rate : 0.652

## P-Value [Acc > NIR] : <2e-16

##

## Kappa : 0.9031

## Mcnemar's Test P-Value : 1

##

## Sensitivity : 0.9624

## Specificity : 0.9437

## Pos Pred Value : 0.9697

## Neg Pred Value : 0.9306

## Prevalence : 0.6520

## Detection Rate : 0.6275

## Detection Prevalence : 0.6471

## Balanced Accuracy : 0.9530

##

## 'Positive' Class : benign

##

results <- data.frame(actual = test\_data$classes,

predict(model\_rf, test\_data, type = "prob"))

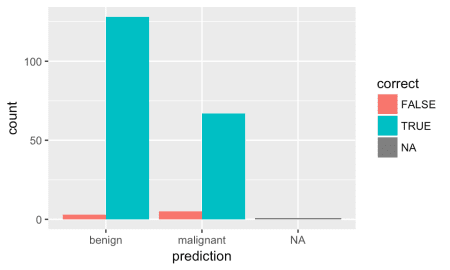
results$prediction <- ifelse(results$benign > 0.5, "benign",

ifelse(results$malignant > 0.5, "malignant", NA))

results$correct <- ifelse(results$actual == results$prediction, TRUE, FALSE)

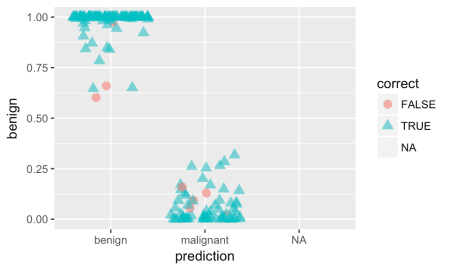
ggplot(results, aes(x = prediction, fill = correct)) +

geom\_bar(position = "dodge")



ggplot(results, aes(x = prediction, y = benign, color = correct, shape = correct)) +

geom\_jitter(size = 3, alpha = 0.6)



**Extreme gradient boosting trees**

Extreme gradient boosting (XGBoost) is a faster and improved implementation of [gradient boosting](https://en.wikipedia.org/wiki/Gradient_boosting) for supervised learning.

“XGBoost uses a more regularized model formalization to control over-fitting, which gives it better performance.” Tianqi Chen, developer of xgboost

XGBoost is a tree ensemble model, which means the sum of predictions from a set of classification and regression trees (CART). In that, XGBoost is similar to Random Forests but it uses a different approach to model training. Can be used for classification and regression tasks. Here, I show a classification task.

set.seed(42)

model\_xgb <- caret::train(classes ~ .,

data = train\_data,

method = "xgbTree",

preProcess = c("scale", "center"),

trControl = trainControl(method = "repeatedcv",

number = 5,

repeats = 3,

savePredictions = TRUE,

verboseIter = FALSE))

model\_xgb

## eXtreme Gradient Boosting

##

## 479 samples

## 9 predictor

## 2 classes: 'benign', 'malignant'

##

## Pre-processing: scaled (9), centered (9)

## Resampling: Cross-Validated (10 fold, repeated 10 times)

## Summary of sample sizes: 432, 431, 431, 431, 431, 431, ...

## Resampling results across tuning parameters:

##

## eta max\_depth colsample\_bytree subsample nrounds Accuracy

## 0.3 1 0.6 0.50 50 0.9567788

## 0.3 1 0.6 0.50 100 0.9544912

## 0.3 1 0.6 0.50 150 0.9513572

## 0.3 1 0.6 0.75 50 0.9576164

## 0.3 1 0.6 0.75 100 0.9536448

## 0.3 1 0.6 0.75 150 0.9525987

## 0.3 1 0.6 1.00 50 0.9559409

## 0.3 1 0.6 1.00 100 0.9555242

## 0.3 1 0.6 1.00 150 0.9551031

## 0.3 1 0.8 0.50 50 0.9718588

## 0.3 1 0.8 0.50 100 0.9720583

## 0.3 1 0.8 0.50 150 0.9699879

## 0.3 1 0.8 0.75 50 0.9726964

## 0.3 1 0.8 0.75 100 0.9724664

## 0.3 1 0.8 0.75 150 0.9705868

## 0.3 1 0.8 1.00 50 0.9714202

## 0.3 1 0.8 1.00 100 0.9710035

## 0.3 1 0.8 1.00 150 0.9705866

## 0.3 2 0.6 0.50 50 0.9559448

## 0.3 2 0.6 0.50 100 0.9565397

## 0.3 2 0.6 0.50 150 0.9555063

## 0.3 2 0.6 0.75 50 0.9530150

## 0.3 2 0.6 0.75 100 0.9550985

## 0.3 2 0.6 0.75 150 0.9551070

## 0.3 2 0.6 1.00 50 0.9532320

## 0.3 2 0.6 1.00 100 0.9551072

## 0.3 2 0.6 1.00 150 0.9557237

## 0.3 2 0.8 0.50 50 0.9720583

## 0.3 2 0.8 0.50 100 0.9735166

## 0.3 2 0.8 0.50 150 0.9720540

## 0.3 2 0.8 0.75 50 0.9722494

## 0.3 2 0.8 0.75 100 0.9726703

## 0.3 2 0.8 0.75 150 0.9716374

## 0.3 2 0.8 1.00 50 0.9716327

## 0.3 2 0.8 1.00 100 0.9724622

## 0.3 2 0.8 1.00 150 0.9718416

## 0.3 3 0.6 0.50 50 0.9548905

## 0.3 3 0.6 0.50 100 0.9557237

## 0.3 3 0.6 0.50 150 0.9555198

## 0.3 3 0.6 0.75 50 0.9561404

## 0.3 3 0.6 0.75 100 0.9546820

## 0.3 3 0.6 0.75 150 0.9552982

## 0.3 3 0.6 1.00 50 0.9577983

## 0.3 3 0.6 1.00 100 0.9573819

## 0.3 3 0.6 1.00 150 0.9567655

## 0.3 3 0.8 0.50 50 0.9733131

## 0.3 3 0.8 0.50 100 0.9728829

## 0.3 3 0.8 0.50 150 0.9718499

## 0.3 3 0.8 0.75 50 0.9751879

## 0.3 3 0.8 0.75 100 0.9743546

## 0.3 3 0.8 0.75 150 0.9735212

## 0.3 3 0.8 1.00 50 0.9743372

## 0.3 3 0.8 1.00 100 0.9737122

## 0.3 3 0.8 1.00 150 0.9743461

## 0.4 1 0.6 0.50 50 0.9548861

## 0.4 1 0.6 0.50 100 0.9528290

## 0.4 1 0.6 0.50 150 0.9498772

## 0.4 1 0.6 0.75 50 0.9557239

## 0.4 1 0.6 0.75 100 0.9513529

## 0.4 1 0.6 0.75 150 0.9492779

## 0.4 1 0.6 1.00 50 0.9559365

## 0.4 1 0.6 1.00 100 0.9551031

## 0.4 1 0.6 1.00 150 0.9536361

## 0.4 1 0.8 0.50 50 0.9710164

## 0.4 1 0.8 0.50 100 0.9697577

## 0.4 1 0.8 0.50 150 0.9687074

## 0.4 1 0.8 0.75 50 0.9710122

## 0.4 1 0.8 0.75 100 0.9707996

## 0.4 1 0.8 0.75 150 0.9691455

## 0.4 1 0.8 1.00 50 0.9705911

## 0.4 1 0.8 1.00 100 0.9697446

## 0.4 1 0.8 1.00 150 0.9697576

## 0.4 2 0.6 0.50 50 0.9544866

## 0.4 2 0.6 0.50 100 0.9542694

## 0.4 2 0.6 0.50 150 0.9536357

## 0.4 2 0.6 0.75 50 0.9540611

## 0.4 2 0.6 0.75 100 0.9542694

## 0.4 2 0.6 0.75 150 0.9549033

## 0.4 2 0.6 1.00 50 0.9540653

## 0.4 2 0.6 1.00 100 0.9555239

## 0.4 2 0.6 1.00 150 0.9546818

## 0.4 2 0.8 0.50 50 0.9720670

## 0.4 2 0.8 0.50 100 0.9695629

## 0.4 2 0.8 0.50 150 0.9702006

## 0.4 2 0.8 0.75 50 0.9722627

## 0.4 2 0.8 0.75 100 0.9720500

## 0.4 2 0.8 0.75 150 0.9716289

## 0.4 2 0.8 1.00 50 0.9726705

## 0.4 2 0.8 1.00 100 0.9708042

## 0.4 2 0.8 1.00 150 0.9708129

## 0.4 3 0.6 0.50 50 0.9555150

## 0.4 3 0.6 0.50 100 0.9553021

## 0.4 3 0.6 0.50 150 0.9548943

## 0.4 3 0.6 0.75 50 0.9555281

## 0.4 3 0.6 0.75 100 0.9563662

## 0.4 3 0.6 0.75 150 0.9555324

## 0.4 3 0.6 1.00 50 0.9575900

## 0.4 3 0.6 1.00 100 0.9571735

## 0.4 3 0.6 1.00 150 0.9559104

## 0.4 3 0.8 0.50 50 0.9737255

## 0.4 3 0.8 0.50 100 0.9745501

## 0.4 3 0.8 0.50 150 0.9730874

## 0.4 3 0.8 0.75 50 0.9747539

## 0.4 3 0.8 0.75 100 0.9724664

## 0.4 3 0.8 0.75 150 0.9720498

## 0.4 3 0.8 1.00 50 0.9747539

## 0.4 3 0.8 1.00 100 0.9749624

## 0.4 3 0.8 1.00 150 0.9734996

## Kappa

## 0.9050828

## 0.8999999

## 0.8930637

## 0.9067208

## 0.8982284

## 0.8959903

## 0.9028825

## 0.9022543

## 0.9014018

## 0.9382467

## 0.9386326

## 0.9340573

## 0.9400323

## 0.9395968

## 0.9353783

## 0.9372262

## 0.9362148

## 0.9353247

## 0.9032270

## 0.9047203

## 0.9024465

## 0.8968511

## 0.9015282

## 0.9016169

## 0.8971329

## 0.9015111

## 0.9028614

## 0.9387022

## 0.9419143

## 0.9387792

## 0.9391933

## 0.9401872

## 0.9379714

## 0.9377309

## 0.9397601

## 0.9384827

## 0.9008861

## 0.9029797

## 0.9024531

## 0.9037859

## 0.9004226

## 0.9019909

## 0.9074584

## 0.9064701

## 0.9051441

## 0.9414031

## 0.9405025

## 0.9380734

## 0.9456856

## 0.9438986

## 0.9419994

## 0.9438642

## 0.9426000

## 0.9439780

## 0.9007223

## 0.8964381

## 0.8897615

## 0.9027951

## 0.8931520

## 0.8886910

## 0.9030461

## 0.9014362

## 0.8982364

## 0.9363059

## 0.9334254

## 0.9311383

## 0.9361883

## 0.9357131

## 0.9320657

## 0.9353688

## 0.9333607

## 0.9334467

## 0.8999756

## 0.8997888

## 0.8983861

## 0.8991356

## 0.8998960

## 0.9013529

## 0.8990428

## 0.9023340

## 0.9004889

## 0.9387165

## 0.9332663

## 0.9345567

## 0.9393855

## 0.9389455

## 0.9380863

## 0.9401366

## 0.9361847

## 0.9361724

## 0.9021263

## 0.9017938

## 0.9010613

## 0.9025263

## 0.9043436

## 0.9024744

## 0.9069828

## 0.9059579

## 0.9031829

## 0.9424523

## 0.9442537

## 0.9410193

## 0.9447486

## 0.9397683

## 0.9388701

## 0.9449064

## 0.9454375

## 0.9422358

##

## Tuning parameter 'gamma' was held constant at a value of 0

##

## Tuning parameter 'min\_child\_weight' was held constant at a value of 1

## Accuracy was used to select the optimal model using the largest value.

## The final values used for the model were nrounds = 50, max\_depth = 3,

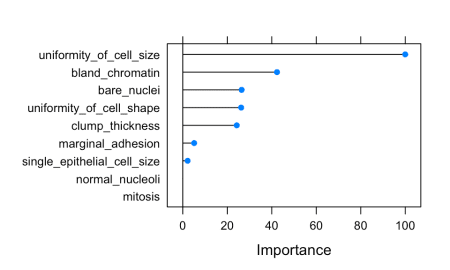
## eta = 0.3, gamma = 0, colsample\_bytree = 0.8, min\_child\_weight = 1

## and subsample = 0.75.

* Feature Importance

importance <- varImp(model\_xgb, scale = TRUE)

plot(importance)



* predicting test data

confusionMatrix(predict(model\_xgb, test\_data), as.factor(test\_data$classes))

## Confusion Matrix and Statistics

##

## Reference

## Prediction benign malignant

## benign 128 3

## malignant 5 68

##

## Accuracy : 0.9608

## 95% CI : (0.9242, 0.9829)

## No Information Rate : 0.652

## P-Value [Acc > NIR] : <2e-16

##

## Kappa : 0.9142

## Mcnemar's Test P-Value : 0.7237

##

## Sensitivity : 0.9624

## Specificity : 0.9577

## Pos Pred Value : 0.9771

## Neg Pred Value : 0.9315

## Prevalence : 0.6520

## Detection Rate : 0.6275

## Detection Prevalence : 0.6422

## Balanced Accuracy : 0.9601

##

## 'Positive' Class : benign

##

results <- data.frame(actual = test\_data$classes,

predict(model\_xgb, test\_data, type = "prob"))

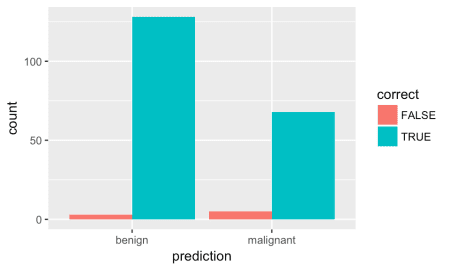
results$prediction <- ifelse(results$benign > 0.5, "benign",

ifelse(results$malignant > 0.5, "malignant", NA))

results$correct <- ifelse(results$actual == results$prediction, TRUE, FALSE)

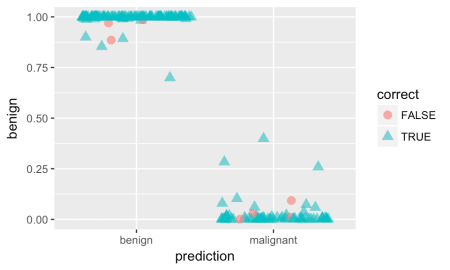
ggplot(results, aes(x = prediction, fill = correct)) +

geom\_bar(position = "dodge")



ggplot(results, aes(x = prediction, y = benign, color = correct, shape = correct)) +

geom\_jitter(size = 3, alpha = 0.6)



**Available models in caret**

**Feature Selection**

Performing feature selection on the whole dataset would lead to prediction bias, we therefore need to run the whole modeling process on the training data alone!

* Correlation

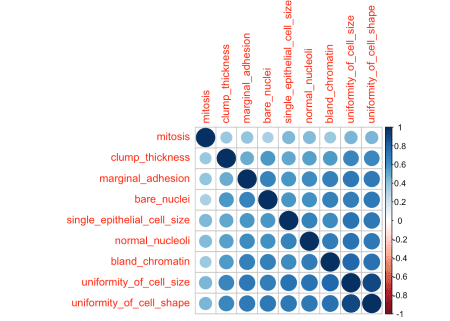
Correlations between all features are calculated and visualised with the *corrplot* package. I am then removing all features with a correlation higher than 0.7, keeping the feature with the lower mean.

library(corrplot)

# calculate correlation matrix

corMatMy <- cor(train\_data[, -1])

corrplot(corMatMy, order = "hclust")



#Apply correlation filter at 0.70,

highlyCor <- colnames(train\_data[, -1])[findCorrelation(corMatMy, cutoff = 0.7, verbose = TRUE)]

## Compare row 2 and column 3 with corr 0.908

## Means: 0.709 vs 0.594 so flagging column 2

## Compare row 3 and column 7 with corr 0.749

## Means: 0.67 vs 0.569 so flagging column 3

## All correlations <= 0.7

# which variables are flagged for removal?

highlyCor

## [1] "uniformity\_of\_cell\_size" "uniformity\_of\_cell\_shape"

#then we remove these variables

train\_data\_cor <- train\_data[, which(!colnames(train\_data) %in% highlyCor)]

* Recursive Feature Elimination (RFE)

Another way to choose features is with Recursive Feature Elimination. RFE uses a Random Forest algorithm to test combinations of features and rate each with an accuracy score. The combination with the highest score is usually preferential.

set.seed(7)

results\_rfe <- rfe(x = train\_data[, -1],

y = as.factor(train\_data$classes),

sizes = c(1:9),

rfeControl = rfeControl(functions = rfFuncs, method = "cv", number = 10))

# chosen features

predictors(results\_rfe)

## [1] "bare\_nuclei" "clump\_thickness"

## [3] "uniformity\_of\_cell\_size" "uniformity\_of\_cell\_shape"

## [5] "bland\_chromatin" "normal\_nucleoli"

## [7] "marginal\_adhesion" "single\_epithelial\_cell\_size"

train\_data\_rfe <- train\_data[, c(1, which(colnames(train\_data) %in% predictors(results\_rfe)))]

* Genetic Algorithm (GA)

The Genetic Algorithm (GA) has been developed based on evolutionary principles of natural selection: It aims to optimize a population of individuals with a given set of genotypes by modeling selection over time. In each generation (i.e. iteration), each individual’s fitness is calculated based on their genotypes. Then, the fittest individuals are chosen to produce the next generation. This subsequent generation of individuals will have genotypes resulting from (re-) combinations of the parental alleles. These new genotypes will again determine each individual’s fitness. This selection process is iterated for a specified number of generations and (ideally) leads to fixation of the fittest alleles in the gene pool.

This concept of optimization can be applied to non-evolutionary models as well, like feature selection processes in machine learning.

set.seed(27)

model\_ga <- gafs(x = train\_data[, -1],

y = as.factor(train\_data$classes),

iters = 10, # generations of algorithm

popSize = 10, # population size for each generation

levels = c("malignant", "benign"),

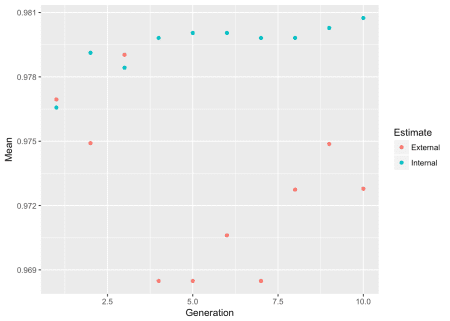
gafsControl = gafsControl(functions = rfGA, # Assess fitness with RF

method = "cv", # 10 fold cross validation

genParallel = TRUE, # Use parallel programming

allowParallel = TRUE))

plot(model\_ga) # Plot mean fitness (AUC) by generation



train\_data\_ga <- train\_data[, c(1, which(colnames(train\_data) %in% model\_ga$ga$final))]

**Hyperparameter tuning with caret**

* Cartesian Grid
* mtry: Number of variables randomly sampled as candidates at each split.

set.seed(42)

grid <- expand.grid(mtry = c(1:10))

model\_rf\_tune\_man <- caret::train(classes ~ .,

data = train\_data,

method = "rf",

preProcess = c("scale", "center"),

trControl = trainControl(method = "repeatedcv",

number = 10,

repeats = 10,

savePredictions = TRUE,

verboseIter = FALSE),

tuneGrid = grid)

model\_rf\_tune\_man

## Random Forest

##

## 479 samples

## 9 predictor

## 2 classes: 'benign', 'malignant'

##

## Pre-processing: scaled (9), centered (9)

## Resampling: Cross-Validated (10 fold, repeated 10 times)

## Summary of sample sizes: 432, 431, 431, 431, 431, 431, ...

## Resampling results across tuning parameters:

##

## mtry Accuracy Kappa

## 1 0.9785044 0.9532161

## 2 0.9772586 0.9504377

## 3 0.9774625 0.9508246

## 4 0.9766333 0.9488778

## 5 0.9753789 0.9460274

## 6 0.9737078 0.9422613

## 7 0.9730957 0.9408547

## 8 0.9714155 0.9371611

## 9 0.9718280 0.9380578

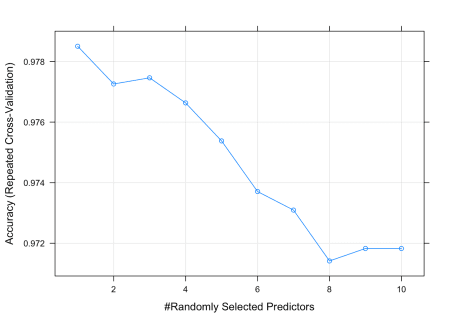
## 10 0.9718280 0.9380135

##

## Accuracy was used to select the optimal model using the largest value.

## The final value used for the model was mtry = 1.

plot(model\_rf\_tune\_man)



* Random Search

set.seed(42)

model\_rf\_tune\_auto <- caret::train(classes ~ .,

data = train\_data,

method = "rf",

preProcess = c("scale", "center"),

trControl = trainControl(method = "repeatedcv",

number = 10,

repeats = 10,

savePredictions = TRUE,

verboseIter = FALSE,

search = "random"),

tuneGrid = grid,

tuneLength = 15)

model\_rf\_tune\_auto

## Random Forest

##

## 479 samples

## 9 predictor

## 2 classes: 'benign', 'malignant'

##

## Pre-processing: scaled (9), centered (9)

## Resampling: Cross-Validated (10 fold, repeated 10 times)

## Summary of sample sizes: 432, 431, 431, 431, 431, 431, ...

## Resampling results across tuning parameters:

##

## mtry Accuracy Kappa

## 1 0.9785044 0.9532161

## 2 0.9772586 0.9504377

## 3 0.9774625 0.9508246

## 4 0.9766333 0.9488778

## 5 0.9753789 0.9460274

## 6 0.9737078 0.9422613

## 7 0.9730957 0.9408547

## 8 0.9714155 0.9371611

## 9 0.9718280 0.9380578

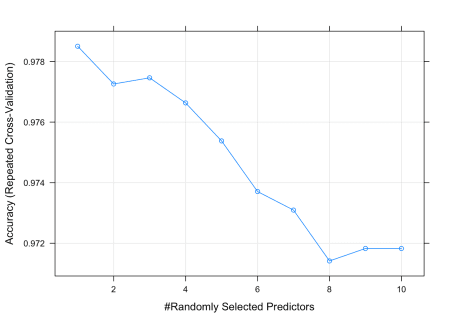
## 10 0.9718280 0.9380135

##

## Accuracy was used to select the optimal model using the largest value.

## The final value used for the model was mtry = 1.

plot(model\_rf\_tune\_auto)



**Grid search with h2o**

The R package h2o provides a convenient interface to [H2O](http://www.h2o.ai/h2o/), which is an open-source machine learning and deep learning platform. H2O distributes a wide range of common machine learning algorithms for classification, regression and deep learning.

library(h2o)

h2o.init(nthreads = -1)

## Connection successful!

##

## R is connected to the H2O cluster:

## H2O cluster uptime: 26 minutes 45 seconds

## H2O cluster timezone: Europe/Berlin

## H2O data parsing timezone: UTC

## H2O cluster version: 3.20.0.2

## H2O cluster version age: 13 days

## H2O cluster name: H2O\_started\_from\_R\_shiringlander\_jrj894

## H2O cluster total nodes: 1

## H2O cluster total memory: 3.24 GB

## H2O cluster total cores: 8

## H2O cluster allowed cores: 8

## H2O cluster healthy: TRUE

## H2O Connection ip: localhost

## H2O Connection port: 54321

## H2O Connection proxy: NA

## H2O Internal Security: FALSE

## H2O API Extensions: XGBoost, Algos, AutoML, Core V3, Core V4

## R Version: R version 3.5.0 (2018-04-23)

h2o.no\_progress()

bc\_data\_hf <- as.h2o(bc\_data)

h2o.describe(bc\_data\_hf) %>%

gather(x, y, Zeros:Sigma) %>%

mutate(group = ifelse(x %in% c("Min", "Max", "Mean"), "min, mean, max",

ifelse(x %in% c("NegInf", "PosInf"), "Inf", "sigma, zeros"))) %>%

ggplot(aes(x = Label, y = as.numeric(y), color = x)) +

geom\_point(size = 4, alpha = 0.6) +

scale\_color\_brewer(palette = "Set1") +

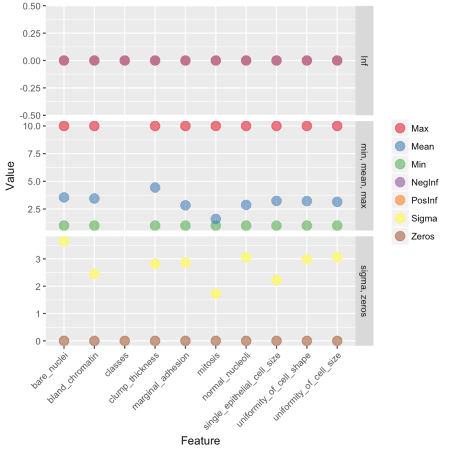
theme(axis.text.x = element\_text(angle = 45, vjust = 1, hjust = 1)) +

facet\_grid(group ~ ., scales = "free") +

labs(x = "Feature",

y = "Value",

color = "")



library(reshape2) # for melting

bc\_data\_hf[, 1] <- h2o.asfactor(bc\_data\_hf[, 1])

cor <- h2o.cor(bc\_data\_hf)

rownames(cor) <- colnames(cor)

melt(cor) %>%

mutate(Var2 = rep(rownames(cor), nrow(cor))) %>%

mutate(Var2 = factor(Var2, levels = colnames(cor))) %>%

mutate(variable = factor(variable, levels = colnames(cor))) %>%

ggplot(aes(x = variable, y = Var2, fill = value)) +

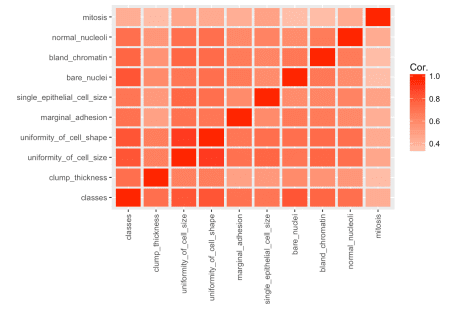
geom\_tile(width = 0.9, height = 0.9) +

scale\_fill\_gradient2(low = "white", high = "red", name = "Cor.") +

theme(axis.text.x = element\_text(angle = 90, vjust = 0.5, hjust = 1)) +

labs(x = "",

y = "")



**Training, validation and test data**

splits <- h2o.splitFrame(bc\_data\_hf,

ratios = c(0.7, 0.15),

seed = 1)

train <- splits[[1]]

valid <- splits[[2]]

test <- splits[[3]]

response <- "classes"

features <- setdiff(colnames(train), response)

summary(as.factor(train$classes), exact\_quantiles = TRUE)

## classes

## benign :313

## malignant:167

summary(as.factor(valid$classes), exact\_quantiles = TRUE)

## classes

## benign :64

## malignant:38

summary(as.factor(test$classes), exact\_quantiles = TRUE)

## classes

## benign :67

## malignant:34

pca <- h2o.prcomp(training\_frame = train,

x = features,

validation\_frame = valid,

transform = "NORMALIZE",

impute\_missing = TRUE,

k = 3,

seed = 42)

eigenvec <- as.data.frame(pca@model$eigenvectors)

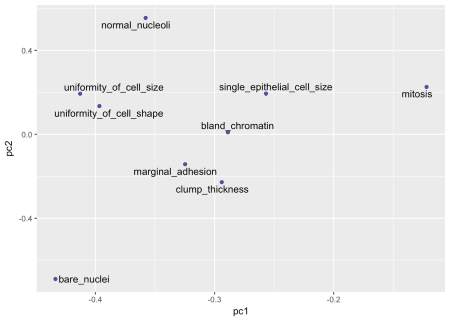
eigenvec$label <- features

library(ggrepel)

ggplot(eigenvec, aes(x = pc1, y = pc2, label = label)) +

geom\_point(color = "navy", alpha = 0.7) +

geom\_text\_repel()



**Classification**

**Random Forest**

hyper\_params <- list(

ntrees = c(25, 50, 75, 100),

max\_depth = c(10, 20, 30),

min\_rows = c(1, 3, 5)

)

search\_criteria <- list(

strategy = "RandomDiscrete",

max\_models = 50,

max\_runtime\_secs = 360,

stopping\_rounds = 5,

stopping\_metric = "AUC",

stopping\_tolerance = 0.0005,

seed = 42

)

rf\_grid <- h2o.grid(algorithm = "randomForest", # h2o.randomForest,

# alternatively h2o.gbm

# for Gradient boosting trees

x = features,

y = response,

grid\_id = "rf\_grid",

training\_frame = train,

validation\_frame = valid,

nfolds = 25,

fold\_assignment = "Stratified",

hyper\_params = hyper\_params,

search\_criteria = search\_criteria,

seed = 42

)

# performance metrics where smaller is better -> order with decreasing = FALSE

sort\_options\_1 <- c("mean\_per\_class\_error", "mse", "err", "logloss")

for (sort\_by\_1 in sort\_options\_1) {

grid <- h2o.getGrid("rf\_grid", sort\_by = sort\_by\_1, decreasing = FALSE)

model\_ids <- grid@model\_ids

best\_model <- h2o.getModel(model\_ids[[1]])

h2o.saveModel(best\_model, path="models", force = TRUE)

}

# performance metrics where bigger is better -> order with decreasing = TRUE

sort\_options\_2 <- c("auc", "precision", "accuracy", "recall", "specificity")

for (sort\_by\_2 in sort\_options\_2) {

grid <- h2o.getGrid("rf\_grid", sort\_by = sort\_by\_2, decreasing = TRUE)

model\_ids <- grid@model\_ids

best\_model <- h2o.getModel(model\_ids[[1]])

h2o.saveModel(best\_model, path = "models", force = TRUE)

}

files <- list.files(path = "/Users/shiringlander/Documents/Github/intro\_to\_ml\_workshop/intro\_to\_ml\_uni\_heidelberg/models")

rf\_models <- files[grep("rf\_grid\_model", files)]

for (model\_id in rf\_models) {

path <- paste0("/Users/shiringlander/Documents/Github/intro\_to\_ml\_workshop/intro\_to\_ml\_uni\_heidelberg", "/models/", model\_id)

best\_model <- h2o.loadModel(path)

mse\_auc\_test <- data.frame(model\_id = model\_id,

mse = h2o.mse(h2o.performance(best\_model, test)),

auc = h2o.auc(h2o.performance(best\_model, test)))

if (model\_id == rf\_models[[1]]) {

mse\_auc\_test\_comb <- mse\_auc\_test

} else {

mse\_auc\_test\_comb <- rbind(mse\_auc\_test\_comb, mse\_auc\_test)

}

}

mse\_auc\_test\_comb %>%

gather(x, y, mse:auc) %>%

ggplot(aes(x = model\_id, y = y, fill = model\_id)) +

facet\_grid(x ~ ., scales = "free") +

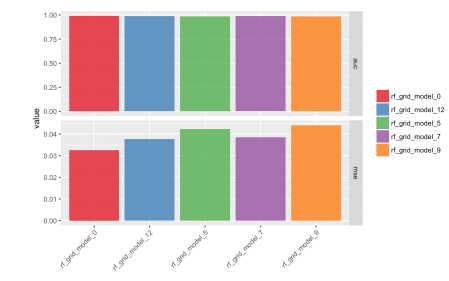
geom\_bar(stat = "identity", alpha = 0.8, position = "dodge") +

scale\_fill\_brewer(palette = "Set1") +

theme(axis.text.x = element\_text(angle = 45, vjust = 1, hjust = 1),

plot.margin = unit(c(0.5, 0, 0, 1.5), "cm")) +

labs(x = "", y = "value", fill = "")



for (model\_id in rf\_models) {

best\_model <- h2o.getModel(model\_id)

finalRf\_predictions <- data.frame(model\_id = rep(best\_model@model\_id,

nrow(test)),

actual = as.vector(test$classes),

as.data.frame(h2o.predict(object = best\_model,

newdata = test)))

finalRf\_predictions$accurate <- ifelse(finalRf\_predictions$actual ==

finalRf\_predictions$predict,

"yes", "no")

finalRf\_predictions$predict\_stringent <- ifelse(finalRf\_predictions$benign > 0.8,

"benign",

ifelse(finalRf\_predictions$malignant

> 0.8, "malignant", "uncertain"))

finalRf\_predictions$accurate\_stringent <- ifelse(finalRf\_predictions$actual ==

finalRf\_predictions$predict\_stringent, "yes",

ifelse(finalRf\_predictions$predict\_stringent ==

"uncertain", "na", "no"))

if (model\_id == rf\_models[[1]]) {

finalRf\_predictions\_comb <- finalRf\_predictions

} else {

finalRf\_predictions\_comb <- rbind(finalRf\_predictions\_comb, finalRf\_predictions)

}

}

finalRf\_predictions\_comb %>%

ggplot(aes(x = actual, fill = accurate)) +

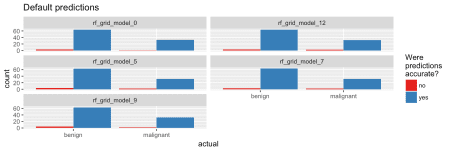
geom\_bar(position = "dodge") +

scale\_fill\_brewer(palette = "Set1") +

facet\_wrap(~ model\_id, ncol = 2) +

labs(fill = "Were\npredictions\naccurate?",

title = "Default predictions")



finalRf\_predictions\_comb %>%

subset(accurate\_stringent != "na") %>%

ggplot(aes(x = actual, fill = accurate\_stringent)) +

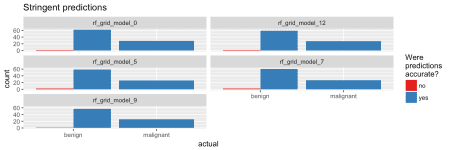
geom\_bar(position = "dodge") +

scale\_fill\_brewer(palette = "Set1") +

facet\_wrap(~ model\_id, ncol = 2) +

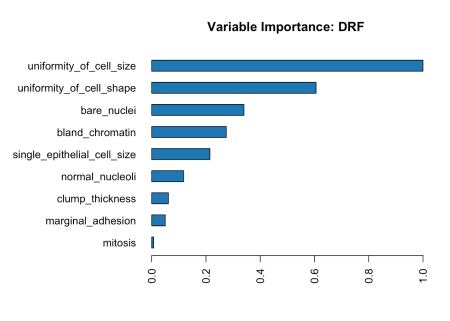
labs(fill = "Were\npredictions\naccurate?",

title = "Stringent predictions")



rf\_model <- h2o.loadModel("/Users/shiringlander/Documents/Github/intro\_to\_ml\_workshop/intro\_to\_ml\_uni\_heidelberg/models/rf\_grid\_model\_0")

h2o.varimp\_plot(rf\_model)



#h2o.varimp(rf\_model)

h2o.mean\_per\_class\_error(rf\_model, train = TRUE, valid = TRUE, xval = TRUE)

## train valid xval

## 0.02196246 0.02343750 0.02515735

h2o.confusionMatrix(rf\_model, valid = TRUE)

## Confusion Matrix (vertical: actual; across: predicted) for max f1 @ threshold = 0.533333333333333:

## benign malignant Error Rate

## benign 61 3 0.046875 =3/64

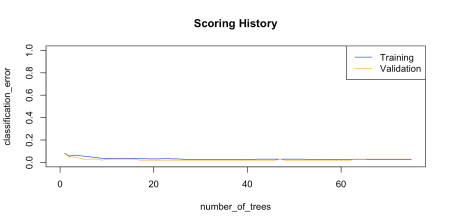
## malignant 0 38 0.000000 =0/38

## Totals 61 41 0.029412 =3/102

plot(rf\_model,

timestep = "number\_of\_trees",

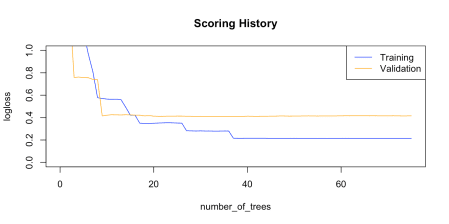
metric = "classification\_error")



plot(rf\_model,

timestep = "number\_of\_trees",

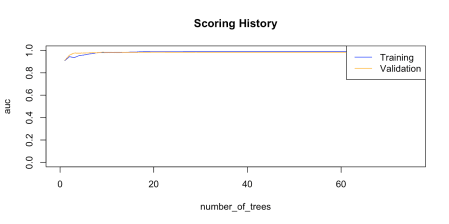
metric = "logloss")



plot(rf\_model,

timestep = "number\_of\_trees",

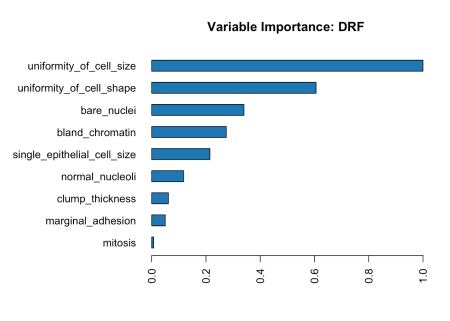
metric = "AUC")



plot(rf\_model,

timestep = "number\_of\_trees",

metric = "rmse")



h2o.auc(rf\_model, train = TRUE)

## [1] 0.9907214

h2o.auc(rf\_model, valid = TRUE)

## [1] 0.9829359

h2o.auc(rf\_model, xval = TRUE)

## [1] 0.9903005

perf <- h2o.performance(rf\_model, test)

perf

## H2OBinomialMetrics: drf

##

## MSE: 0.03258482

## RMSE: 0.1805127

## LogLoss: 0.1072519

## Mean Per-Class Error: 0.02985075

## AUC: 0.9916594

## Gini: 0.9833187

##

## Confusion Matrix (vertical: actual; across: predicted) for F1-optimal threshold:

## benign malignant Error Rate

## benign 63 4 0.059701 =4/67

## malignant 0 34 0.000000 =0/34

## Totals 63 38 0.039604 =4/101

##

## Maximum Metrics: Maximum metrics at their respective thresholds

## metric threshold value idx

## 1 max f1 0.306667 0.944444 18

## 2 max f2 0.306667 0.977011 18

## 3 max f0point5 0.720000 0.933735 13

## 4 max accuracy 0.533333 0.960396 16

## 5 max precision 1.000000 1.000000 0

## 6 max recall 0.306667 1.000000 18

## 7 max specificity 1.000000 1.000000 0

## 8 max absolute\_mcc 0.306667 0.917235 18

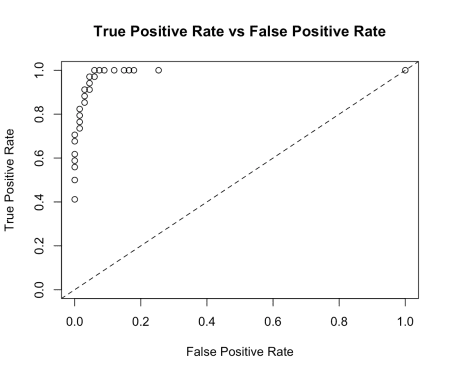
## 9 max min\_per\_class\_accuracy 0.533333 0.955224 16

## 10 max mean\_per\_class\_accuracy 0.306667 0.970149 18

##

## Gains/Lift Table: Extract with `h2o.gainsLift(, )` or `h2o.gainsLift(, valid=, xval=)`

plot(perf)



perf@metrics$thresholds\_and\_metric\_scores %>%

ggplot(aes(x = fpr, y = tpr)) +

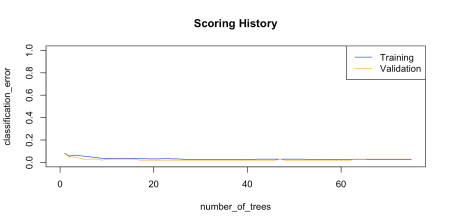
geom\_point() +

geom\_line() +

geom\_abline(slope = 1, intercept = 0) +

labs(x = "False Positive Rate",

y = "True Positive Rate")



h2o.logloss(perf)

## [1] 0.1072519

h2o.mse(perf)

## [1] 0.03258482

h2o.auc(perf)

## [1] 0.9916594

head(h2o.metric(perf))

## Metrics for Thresholds: Binomial metrics as a function of classification thresholds

## threshold f1 f2 f0point5 accuracy precision recall

## 1 1.000000 0.583333 0.466667 0.777778 0.801980 1.000000 0.411765

## 2 0.986667 0.666667 0.555556 0.833333 0.831683 1.000000 0.500000

## 3 0.973333 0.716981 0.612903 0.863636 0.851485 1.000000 0.558824

## 4 0.960000 0.740741 0.641026 0.877193 0.861386 1.000000 0.588235

## 5 0.946667 0.763636 0.668790 0.889831 0.871287 1.000000 0.617647

## 6 0.920000 0.807018 0.723270 0.912698 0.891089 1.000000 0.676471

## specificity absolute\_mcc min\_per\_class\_accuracy mean\_per\_class\_accuracy

## 1 1.000000 0.563122 0.411765 0.705882

## 2 1.000000 0.631514 0.500000 0.750000

## 3 1.000000 0.675722 0.558824 0.779412

## 4 1.000000 0.697542 0.588235 0.794118

## 5 1.000000 0.719221 0.617647 0.808824

## 6 1.000000 0.762280 0.676471 0.838235

## tns fns fps tps tnr fnr fpr tpr idx

## 1 67 20 0 14 1.000000 0.588235 0.000000 0.411765 0

## 2 67 17 0 17 1.000000 0.500000 0.000000 0.500000 1

## 3 67 15 0 19 1.000000 0.441176 0.000000 0.558824 2

## 4 67 14 0 20 1.000000 0.411765 0.000000 0.588235 3

## 5 67 13 0 21 1.000000 0.382353 0.000000 0.617647 4

## 6 67 11 0 23 1.000000 0.323529 0.000000 0.676471 5

finalRf\_predictions <- data.frame(actual = as.vector(test$classes),

as.data.frame(h2o.predict(object = rf\_model,

newdata = test)))

finalRf\_predictions$accurate <- ifelse(finalRf\_predictions$actual ==

finalRf\_predictions$predict, "yes", "no")

finalRf\_predictions$predict\_stringent <- ifelse(finalRf\_predictions$benign > 0.8, "benign",

ifelse(finalRf\_predictions$malignant

> 0.8, "malignant", "uncertain"))

finalRf\_predictions$accurate\_stringent <- ifelse(finalRf\_predictions$actual ==

finalRf\_predictions$predict\_stringent, "yes",

ifelse(finalRf\_predictions$predict\_stringent ==

"uncertain", "na", "no"))

finalRf\_predictions %>%

group\_by(actual, predict) %>%

dplyr::summarise(n = n())

## # A tibble: 4 x 3

## # Groups: actual [?]

## actual predict n

##

## 1 benign benign 64

## 2 benign malignant 3

## 3 malignant benign 1

## 4 malignant malignant 33

finalRf\_predictions %>%

group\_by(actual, predict\_stringent) %>%

dplyr::summarise(n = n())

## # A tibble: 5 x 3

## # Groups: actual [?]

## actual predict\_stringent n

##

## 1 benign benign 62

## 2 benign malignant 2

## 3 benign uncertain 3

## 4 malignant malignant 29

## 5 malignant uncertain 5

finalRf\_predictions %>%

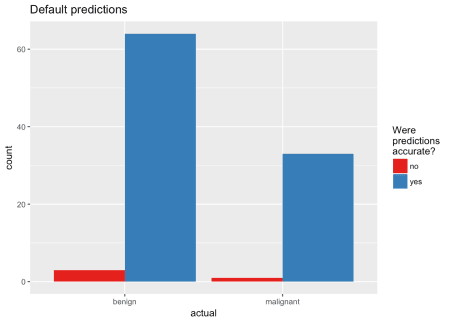
ggplot(aes(x = actual, fill = accurate)) +

geom\_bar(position = "dodge") +

scale\_fill\_brewer(palette = "Set1") +

labs(fill = "Were\npredictions\naccurate?",

title = "Default predictions")



finalRf\_predictions %>%

subset(accurate\_stringent != "na") %>%

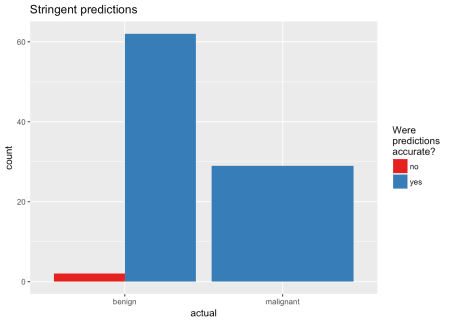
ggplot(aes(x = actual, fill = accurate\_stringent)) +

geom\_bar(position = "dodge") +

scale\_fill\_brewer(palette = "Set1") +

labs(fill = "Were\npredictions\naccurate?",

title = "Stringent predictions")



df <- finalRf\_predictions[, c(1, 3, 4)]

thresholds <- seq(from = 0, to = 1, by = 0.1)

prop\_table <- data.frame(threshold = thresholds, prop\_true\_b = NA, prop\_true\_m = NA)

for (threshold in thresholds) {

pred <- ifelse(df$benign > threshold, "benign", "malignant")

pred\_t <- ifelse(pred == df$actual, TRUE, FALSE)

group <- data.frame(df, "pred" = pred\_t) %>%

group\_by(actual, pred) %>%

dplyr::summarise(n = n())

group\_b <- filter(group, actual == "benign")

prop\_b <- sum(filter(group\_b, pred == TRUE)$n) / sum(group\_b$n)

prop\_table[prop\_table$threshold == threshold, "prop\_true\_b"] <- prop\_b

group\_m <- filter(group, actual == "malignant")

prop\_m <- sum(filter(group\_m, pred == TRUE)$n) / sum(group\_m$n)

prop\_table[prop\_table$threshold == threshold, "prop\_true\_m"] <- prop\_m

}

prop\_table %>%

gather(x, y, prop\_true\_b:prop\_true\_m) %>%

ggplot(aes(x = threshold, y = y, color = x)) +

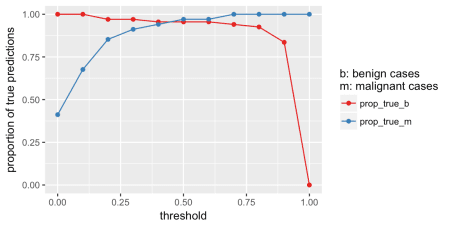
geom\_point() +

geom\_line() +

scale\_color\_brewer(palette = "Set1") +

labs(y = "proportion of true predictions",

color = "b: benign cases\nm: malignant cases")



stopCluster(cl)

h2o.shutdown()

## Are you sure you want to shutdown the H2O instance running at http://localhost:54321/ (Y/N)?

sessionInfo()

## R version 3.5.0 (2018-04-23)

## Platform: x86\_64-apple-darwin15.6.0 (64-bit)

## Running under: macOS High Sierra 10.13.5

##

## Matrix products: default

## BLAS: /Library/Frameworks/R.framework/Versions/3.5/Resources/lib/libRblas.0.dylib

## LAPACK: /Library/Frameworks/R.framework/Versions/3.5/Resources/lib/libRlapack.dylib

##

## locale:

## [1] de\_DE.UTF-8/de\_DE.UTF-8/de\_DE.UTF-8/C/de\_DE.UTF-8/de\_DE.UTF-8

##

## attached base packages:

## [1] parallel stats graphics grDevices utils datasets methods

## [8] base

##

## other attached packages:

## [1] ggrepel\_0.8.0 reshape2\_1.4.3 h2o\_3.20.0.2

## [4] corrplot\_0.84 caret\_6.0-80 doParallel\_1.0.11

## [7] iterators\_1.0.9 foreach\_1.4.4 ellipse\_0.4.1

## [10] igraph\_1.2.1 bindrcpp\_0.2.2 mice\_3.1.0

## [13] lattice\_0.20-35 forcats\_0.3.0 stringr\_1.3.1

## [16] dplyr\_0.7.5 purrr\_0.2.5 readr\_1.1.1

## [19] tidyr\_0.8.1 tibble\_1.4.2 ggplot2\_2.2.1

## [22] tidyverse\_1.2.1

##

## loaded via a namespace (and not attached):

## [1] minqa\_1.2.4 colorspace\_1.3-2 class\_7.3-14

## [4] rprojroot\_1.3-2 pls\_2.6-0 rstudioapi\_0.7

## [7] DRR\_0.0.3 prodlim\_2018.04.18 lubridate\_1.7.4

## [10] xml2\_1.2.0 codetools\_0.2-15 splines\_3.5.0

## [13] mnormt\_1.5-5 robustbase\_0.93-1 knitr\_1.20

## [16] RcppRoll\_0.3.0 jsonlite\_1.5 nloptr\_1.0.4

## [19] broom\_0.4.4 ddalpha\_1.3.4 kernlab\_0.9-26

## [22] sfsmisc\_1.1-2 compiler\_3.5.0 httr\_1.3.1

## [25] backports\_1.1.2 assertthat\_0.2.0 Matrix\_1.2-14

## [28] lazyeval\_0.2.1 cli\_1.0.0 htmltools\_0.3.6

## [31] tools\_3.5.0 gtable\_0.2.0 glue\_1.2.0

## [34] Rcpp\_0.12.17 cellranger\_1.1.0 nlme\_3.1-137

## [37] blogdown\_0.6 psych\_1.8.4 timeDate\_3043.102

## [40] xfun\_0.2 gower\_0.1.2 lme4\_1.1-17

## [43] rvest\_0.3.2 pan\_1.4 DEoptimR\_1.0-8

## [46] MASS\_7.3-50 scales\_0.5.0 ipred\_0.9-6

## [49] hms\_0.4.2 RColorBrewer\_1.1-2 yaml\_2.1.19

## [52] rpart\_4.1-13 stringi\_1.2.3 randomForest\_4.6-14

## [55] e1071\_1.6-8 lava\_1.6.1 geometry\_0.3-6

## [58] bitops\_1.0-6 rlang\_0.2.1 pkgconfig\_2.0.1

## [61] evaluate\_0.10.1 bindr\_0.1.1 recipes\_0.1.3

## [64] labeling\_0.3 CVST\_0.2-2 tidyselect\_0.2.4

## [67] plyr\_1.8.4 magrittr\_1.5 bookdown\_0.7

## [70] R6\_2.2.2 mitml\_0.3-5 dimRed\_0.1.0

## [73] pillar\_1.2.3 haven\_1.1.1 foreign\_0.8-70

## [76] withr\_2.1.2 RCurl\_1.95-4.10 survival\_2.42-3

## [79] abind\_1.4-5 nnet\_7.3-12 modelr\_0.1.2

## [82] crayon\_1.3.4 jomo\_2.6-2 xgboost\_0.71.2

## [85] utf8\_1.1.4 rmarkdown\_1.10 grid\_3.5.0

## [88] readxl\_1.1.0 data.table\_1.11.4 ModelMetrics\_1.1.0

## [91] digest\_0.6.15 stats4\_3.5.0 munsell\_0.5.0

## [94] magic\_1.5-8