# Vergleich von verschiedenen KI-Klassifizierungsmethoden am Beispiel des Fashion MNIST-Datensatzes in RStudio und Python (Teil 3)

Lars Mehnen

11/03/2022

In dieser Reihe werde wir verschiedene maschinelle und Deep-Learning-Methoden vergleichen, um Kleidungsklassen anhand von Bildern des Fashion MNIST-Datensatzes zu erstellen bzw. mit dem erstellten Modellen zu klassifizieren. Wir haben die Hauptkomponentenanalyse (PCA) verwendet, um die Datendimensionalität zu reduzieren, und eine Funktion geschrieben, um die Leistung der Modelle zu bewerten, die wir in diesem Beitrag schätzen werden, nämlich baumbasierte Methoden (Random Forests und Boosting).

```
knitr::opts_chunk$set(echo = TRUE)
```

```
library(devtools)
```

```
## Lade nötiges Paket: usethis
```

```
devtools::install_github("rstudio/keras")
```

```
## Skipping install of 'keras' from a github remote, the SHA1 (68ba8c45) has not changed since last ins
##   Use `force = TRUE` to force installation
```

```
library(keras)
#install_keras()
fashion_mnist = keras::dataset_fashion_mnist()
```

```
## Loaded Tensorflow version 2.7.0
```

```
library(magrittr)
c(train.images, train.labels) %<-% fashion_mnist$train
c(test.images, test.labels) %<-% fashion_mnist$test
train.images = data.frame(t(apply(train.images, 1, c))) / max(fashion_mnist$train$x)
test.images = data.frame(t(apply(test.images, 1, c))) / max(fashion_mnist$train$x)
pixs = ncol(fashion_mnist$train$x)
names(train.images) = names(test.images) = paste0('pixel', 1:(pixs^2))
train.labels = data.frame(label = factor(train.labels))
test.labels = data.frame(label = factor(test.labels))
train.data = cbind(train.labels, train.images)
test.data = cbind(test.labels, test.images)
cloth_cats = c('Top', 'Trouser', 'Pullover', 'Dress', 'Coat',
               'Sandal', 'Shirt', 'Sneaker', 'Bag', 'Boot')
train.classes = factor(cloth_cats[as.numeric(as.character(train.labels$label)) + 1])
test.classes = factor(cloth_cats[as.numeric(as.character(test.labels$label)) + 1])
```

```
library(stats)
cov.train = cov(train.images)
```

```
pca.train = prcomp(cov.train)
plotdf = data.frame(index = 1:(pixs^2),
                    cumvar = summary(pca.train)$importance["Cumulative Proportion", ])
pca.dims = which(plotdf$cumvar >= .995)[1]
pca.rot = pca.train$rotation[, 1:pca.dims]
train.images.pca = data.frame(as.matrix(train.images) %*% pca.rot)
test.images.pca  = data.frame(as.matrix(test.images) %*% pca.rot)
train.data.pca = cbind(train.images.pca, label = factor(train.data$label))
test.data.pca = cbind(test.images.pca, label = factor(test.data$label))
```

```
model_performance = function(fit, trainX, testX, trainY, testY, model_name){

  # Predictions on train and test data for different types of models
  if (any(class(fit) == "rpart")){

    library(rpart)
    pred_train = predict(fit, newdata = trainX, type = "class")
    pred_test = predict(fit, newdata = testX, type = "class")

  } else if (any(class(fit) == "train")){

    library(data.table)
    pred_dt = as.data.table(fit$pred[, names(fit$bestTune)])
    names(pred_dt) = names(fit$bestTune)
    index_list = lapply(1:ncol(fit$bestTune), function(x, DT, tune_opt){
      return(which(DT[, Reduce(`&`, lapply(.SD, `==`, tune_opt[, x])), .SDcols = names(tune_opt)[x]]))
    }, pred_dt, fit$bestTune)
    rows = Reduce(intersect, index_list)
    pred_train = fit$pred$pred[rows]
    pred_test = predict(fit, newdata = testX)
    trainY = fit$pred$obs[rows]

  } else {

    print(paste0("Error: Function evaluation unknown for object of type ", class(fit)))
    break

  }

  # Performance metrics on train and test data
  library(MLmetrics)
  df = data.frame(accuracy_train = Accuracy(trainY, pred_train),
                  precision_train = Precision(trainY, pred_train),
                  recall_train = Recall(trainY, pred_train),
                  F1_train = F1_Score(trainY, pred_train),
                  accuracy_test = Accuracy(testY, pred_test),
                  precision_test = Precision(testY, pred_test),
                  recall_test = Recall(testY, pred_test),
                  F1_test = F1_Score(testY, pred_test),
                  model = model_name)

  print(df)

  return(df)
```

```
}
```

# Entscheidungsbaum basierte Methoden

¸ In diesem ersten Unterabschnitt werden wir verschiedene baumbasierte Methoden vergleichen: `Random Forests` und `Gradient-Boosted Trees`. Baumbasierte Verfahren segmentieren den Prädiktorraum in eine Anzahl einfacherer Teile unter Verwendung einiger Entscheidungsregeln, die in einem Entscheidungsbaum zusammengefasst werden können. Der Fokus liegt hier auf `Klassifikationsbäumen`, da die Ergebnisvariable des Fashion MNIST Datensatzes kategorial ist und 10 Klassen beinhaltet. Leider haben Einzelbäume im Vergleich zu anderen Klassifikationsansätzen wie der `logistischen Regression` oder der `Diskriminanzanalyse` eine relativ geringe Vorhersagekraft. Um die Vorhersagegenauigkeit zu verbessern, aggregieren Ensemble-Methoden viele einzelne Entscheidungsbäume. Dadurch bieten sie eine einfache Möglichkeit, die Vorhersagegüte zu verbessern und gleichzeitig die Varianz zu verringern. Im Folgenden werden wir `Random Forests` und `Gradient-Boosted Trees` als Ensemble-Methoden anwenden. Erstere sind einfacher zu implementieren, da sie robuster gegenüber Überanpassung sind und weniger Abstimmung erfordern, während letztere im Allgemeinen andere baumbasierte Methoden in Bezug auf Vorhersagegenauigkeit übertreffen. Die Modelle werden hier im überwachten Modus (`supervides learning`) eingesetzt, da `gelabelte` Daten verfügbar sind und das Ziel darin besteht, Klassen vorherzusagen. Der Vorteil von Entscheidungsbäumen besteht darin, dass sie leicht zu interpretieren und zu visualisieren sind. Der Nachteil besteht darin, dass sie tendenziell unter einer hohen Varianz leiden. Das heißt, wenn wir einen Datensatz in zwei Hälften teilen und einen Entscheidungsbaum auf beide Hälften anwenden, können die Ergebnisse sehr unterschiedlich sein.

Eine Möglichkeit, die Varianz von Entscheidungsbäumen zu verringern, besteht in der Verwendung einer als Bagging bezeichneten Methode, die wie folgt funktioniert:

1. Nehmen Sie $b$ Bootstrap-Proben aus dem Originaldatensatz.
2. Erstellen Sie einen Entscheidungsbaum für jedes Bootstrap-Beispiel.
3. Berechnen Sie den Durchschnitt der Vorhersagen jedes Baums, um ein endgültiges Modell zu erhalten.

Der Vorteil dieses Ansatzes besteht darin, dass ein Bagged-Modell in der Regel eine Verbesserung der Testfehlerrate im Vergleich zu einem einzelnen Entscheidungsbaum bietet.

Der Nachteil ist, dass die Vorhersagen aus der Sammlung von Bagged-Trees stark korreliert sein können, wenn der Datensatz zufällig einen sehr starken Prädiktor enthält. In diesem Fall verwenden die meisten Bagged-Trees genau diesen Prädiktor für die erste Aufteilung, was zu Bäumen führt, die einander ähnlich sind und stark korrelierte Vorhersagen aufweisen.

Wenn wir also die Vorhersagen jedes Baums mitteln, um ein endgültiges Bagged-Modell zu erhalten, ist es möglich, dass dieses Modell die Varianz, im Vergleich zu einem einzelnen Entscheidungsbaum, nicht wesentlich reduziert.

Eine Möglichkeit, dieses Problem zu umgehen, ist die Verwendung einer Methode, die als `zufällige Gesamtstrukturen` bezeichnet wird.

## Random Forest

Einzelne nicht beschnittene und beschnittene (`pruned`) Klassifikationsbäume schneiden oft nicht sehr gut ab (64 % Genauigkeit auf dem Testsatz). `Random Forests` verwenden `Bootstrap-Aggregation`, um die Varianz der Ergebnisse zu reduzieren.

Ähnlich wie beim Bagging nehmen Random Forests auch $b$ Bootstrap-Proben aus einem Originaldatensatz.

Wenn jedoch ein Entscheidungsbaum für jede Bootstrap-Stichprobe erstellt wird, wird jedes Mal, wenn eine Teilung in einem Baum berücksichtigt wird, nur eine zufällige Stichprobe von $m$ Prädiktoren als Teilungskandidaten aus dem vollständigen Datensatz von $p$ Prädiktoren betrachtet.

Hier ist die vollständige Methode, mit der zufällige Gesamtstrukturen ein Modell erstellen:

1. Nehmen Sie $b$ Bootstrap-Proben aus dem Originaldatensatz.
2. Erstellen Sie einen Entscheidungsbaum für jedes Bootstrap-Beispiel. Beim Erstellen des Baums wird jedes Mal, wenn eine Teilung berücksichtigt wird, nur eine zufällige Stichprobe von $m$ Prädiktoren als Teilungskandidaten aus dem vollständigen Datensatz von $p$ Prädiktoren betrachtet.
3. Danach bilden des Durchschnitts der Vorhersagen jedes Baums, um ein endgültiges Modell zu erhalten.

Mit dieser Methode wird die Menge von Bäumen in einem Random Forest, im Vergleich zu den durch Bagging erzeugten Bäume, dekorreliert.

Wenn wir also die durchschnittlichen Prädikationen jedes Baums verwenden, um ein endgültiges Modell zu erhalten, weisen diese tendenziell eine geringere Variabilität auf und führen zu einer geringeren Testfehlerrate im Vergleich zu einem der vorhergendenden Modelle.

Bei der Verwendung zufälliger Gesamtstrukturen betrachten wir normalerweise m = sqrt(p)-Prädiktoren, jedes Mal als Kandidaten, wenn wir einen Entscheidungsbaum teilen.

Wenn wir beispielsweise p = 16 Gesamtprädiktoren in einem Datensatz haben, betrachten wir normalerweise nur m = sqrt(16) = 4 Prädiktoren als potenzielle Split-Kandidaten bei jedem Split.

```
Technischer Hinweis: Es ist interessant festzustellen, dass wenn wir m = p wählen
(d.h. alle Prädiktoren bei jedem Split als Split-Kandidaten betrachten),
dies der einfachen Verwendung von Bagging entspricht.
```

Out-of-Bag-Fehlerschätzung

Ähnlich wie beim Bagging können wir den Testfehler eines Random Forest-Modells mithilfe der Out-of-Bag-Schätzung berechnen.

Es kann gezeigt werden, dass jedes Bootstrap-Beispiel etwa 2/3 der Beobachtungen aus dem Originaldatensatz enthält. Das verbleibende Drittel der Beobachtungen, die nicht zur Anpassung an den Baum verwendet wurden, werden als OOB-Beobachtungen (Out-of-Bag) bezeichnet.

Wir können den Wert für die i-te Beobachtung im Originaldatensatz vorhersagen, indem wir die durchschnittliche Vorhersage von jedem der Bäume nehmen, in denen diese Beobachtung OOB war.

Mit diesem Ansatz können wir eine Vorhersage für alle n Beobachtungen im Originaldatensatz erstellen und so eine Fehlerrate berechnen, die eine gültige Schätzung des Testfehlers darstellt.

Der Vorteil dieses Ansatzes zur Schätzung des Testfehlers besteht darin, dass er viel schneller als die k-fache Kreuzvalidierung ist, insbesondere wenn der Datensatz groß ist.

Die Vor- und Nachteile von Random Forests

Vorteile:

```
In den meisten Fällen bieten Random Forests eine Verbesserung der Genauigkeit
im Vergleich zu Bagging-Modellen und insbesondere im Vergleich zu Einzelentscheidungsbäumen.
Random Forests sind robust gegenüber Ausreißern.
Für die Verwendung zufälliger Gesamtstrukturen ist keine Vorverarbeitung erforderlich.
```

Nachteile:

```
Sie sind schwer zu interpretieren.
Sie können rechenintensiv (d.h. langsam) sein, um auf großen Datenmengen aufzubauen.
In der Praxis verwenden Datenwissenschaftler normalerweise zufällige Gesamtstrukturen,
um die Vorhersagegenauigkeit zu maximieren, sodass die Tatsache, dass sie nicht leicht
zu interpretieren sind, normalerweise kein Problem darstellt.
```

Wir beginnen damit, die Anzahl der Variablen zu optimieren, die zufällig als Kandidaten bei jedem Split (`mtry`) ausgewählt werden, wofür wir das `caret`-Framework verwenden. Der Vorteil des `caret`-Frameworks

besteht darin, dass wir eine große Anzahl (zum Zeitpunkt des Schreibens 238) verschiedener Modelltypen mithilfe von Kreuzvalidierung mit ähnlichen Codezeilen und Daten-Strukturen leicht trainieren und evaluieren können. Für unsere `random-forest` lassen wir die Methode `repeatedcv` eine fünffache Kreuzvalidierung mit fünf Wiederholungen durchführen. Im Moment bauen wir einen `random-forest` mit 200 Bäumen auf, da frühere Analysen mit diesen Daten gezeigt haben, dass der Fehler nicht wesentlich abnimmt, wenn die Anzahl der Bäume größer als 200 ist, während eine größere Anzahl von Bäumen mehr Rechenleistung erfordert. Wir werden später sehen, dass 200 Bäume für diese Analyse tatsächlich ausreichen. Wir lassen den Algorithmus bestimmen, welches das beste Modell ist (basierend auf der Genauigkeitsmetrik), und lassen den Algorithmus, das Modell für pca.dims (=17) verschiedene Werte von `mtry` ausprobieren. Wir spezifizieren zunächst die Parameter in `rf_rand_control`: Wir führen eine 5-fache Kreuzvalidierung mit 5 Wiederholungen durch (`method = "cv"`, `number = 5` and `repeats = 5`), erlauben eine parallele Berechnung (`allowParallel = TRUE`) und speichern die vorhergesagten Werte (`savePredictions = TRUE`).

```
library(caret)
```

```
## Lade nötiges Paket: ggplot2
```

```
## Lade nötiges Paket: lattice
```

```
rf_rand_control = trainControl(method = "repeatedcv",
                               search = "random",
                               number = 5,
                               repeats = 5,
                               allowParallel = TRUE,
                               savePredictions = TRUE)
set.seed(815)
rf_rand = train(x = train.images.pca,
                y = train.data.pca$label,
                method = "rf",
                ntree = 200,
                metric = "Accuracy",
                trControl = rf_rand_control,
                tuneLength = pca.dims
               )
```

```
print(rf_rand)
```

```
## Random Forest
##
## 60000 samples
##    17 predictor
##    10 classes: '0', '1', '2', '3', '4', '5', '6', '7', '8', '9'
##
## No pre-processing
## Resampling: Cross-Validated (5 fold, repeated 5 times)
## Summary of sample sizes: 48000, 48000, 48000, 48000, 48000, 48000, ...
## Resampling results across tuning parameters:
##
##   mtry  Accuracy   Kappa
##    1    0.8473067  0.8303407
##    2    0.8518100  0.8353444
##    3    0.8528167  0.8364630
##    4    0.8535367  0.8372630
##    6    0.8538200  0.8375778
##    7    0.8537500  0.8375000
##    9    0.8534867  0.8372074
```

```
##   10    0.8529700  0.8366333
##   11    0.8526567  0.8362852
##   12    0.8519000  0.8354444
##   13    0.8512567  0.8347296
##   14    0.8505267  0.8339185
##   17    0.8462633  0.8291815
##
## Accuracy was used to select the optimal model using the largest value.
## The final value used for the model was mtry = 6.
```

```
mp.rf.rand = model_performance(rf_rand, train.images.pca, test.images.pca,
                               train.data.pca$label, test.data.pca$label, "random_forest_random")
```

```
##
## Attache Paket: 'MLmetrics'
```

```
## Die folgenden Objekte sind maskiert von 'package:caret':
##
##     MAE, RMSE
```

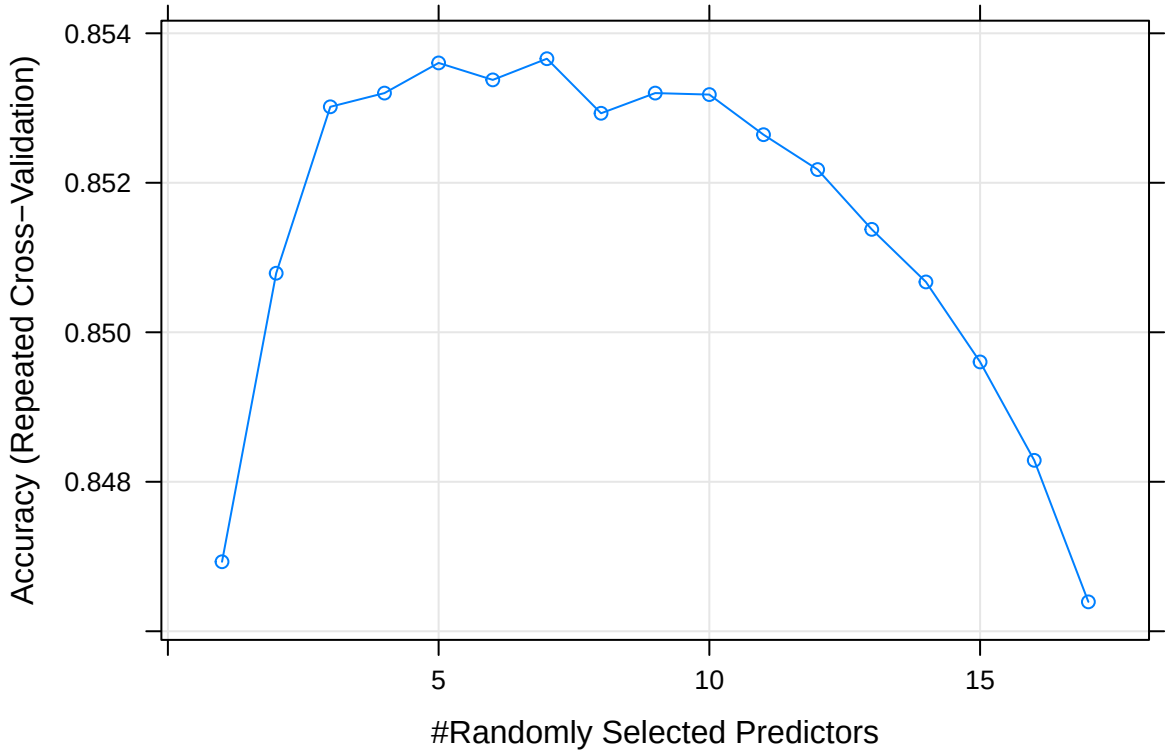
```
## Das folgende Objekt ist maskiert 'package:base':
##
##     Recall
```

```
##   accuracy_train precision_train recall_train  F1_train accuracy_test
## 1        0.85382       0.8024518    0.8400333 0.8208126        0.8488
##   precision_test recall_test   F1_test                model
## 1      0.8063892       0.833 0.8194786 random_forest_random
```

Wir können auch das `caret`-Framework verwenden, um eine Rastersuche mit vordefinierten Werten für `mtry` anstelle einer zufälligen Suche wie oben durchzuführen.

```
rf_grid_control = trainControl(method = "repeatedcv",
                               search = "grid",
                               number = 5,
                               repeats = 5,
                               allowParallel = TRUE,
                               savePredictions = TRUE)
set.seed(815)
rf_grid = train(x = train.images.pca,
                y = train.data.pca$label,
                method = "rf",
                ntree = 200,
                metric = "Accuracy",
                trControl = rf_grid_control,
                tuneGrid = expand.grid(.mtry = c(1:pca.dims)))
```

```
plot(rf_grid)
```




```
mp.rf.grid = model_performance(rf_grid, train.images.pca, test.images.pca,
                               train.data.pca$label, test.data.pca$label, "random_forest_grid")
```

```
##   accuracy_train precision_train recall_train  F1_train accuracy_test
## 1        0.85366        0.802326    0.8393667 0.8204284        0.8485
##   precision_test recall_test   F1_test              model
## 1      0.8056093       0.833 0.8190757 random_forest_grid
```

Wie die Ergebnisse zeigen, wählt die Zufallssuche "mtry=4" als optimalen Parameter aus, was zu 85 % Trainings- und Testsatzgenauigkeit führt. Die Grid-Suche wählt `mtry=5` aus und erreicht ähnliche Genauigkeiten für beide Werte von 4 und 5 für `mtry`. Wir können den Ergebnissen entnehmen, dass laut `rf_rand` `mtry`-Werte von 4 und 5 zu sehr ähnlichen Ergebnissen führen, was auch für `mtry`-Werte von 5 und 6 für `rf_grid` gilt. Obwohl die Ergebnisse von `rf_rand` und `rf_grid` sehr ähnlich sind, wählen wir anhand der Genauigkeit das beste Modell aus und speichern dieses in `rf_best`. Für dieses Modell betrachten wir die Beziehung zwischen dem Fehler und der `random forest` Größe sowie die Receiver Operating Characteristic (ROC)-Kurven für jede Klasse. Beginnen wir damit, das beste Modell von `rf_rand` und `rf_grid` zu subtrahieren.

```
rf_models = list(rf_rand$finalModel, rf_grid$finalModel)
rf_accs = unlist(lapply(rf_models, function(x){ sum(diag(x$confusion)) / sum(x$confusion) }))
rf_best = rf_models[[which.max(rf_accs)]]
```

Als nächstes zeigen wir die Beziehung zwischen der Größe des `random forest` und dem Fehler mit der Funktion `plot()` aus dem Paket `randomForest` auf.

```
library(randomForest)
```

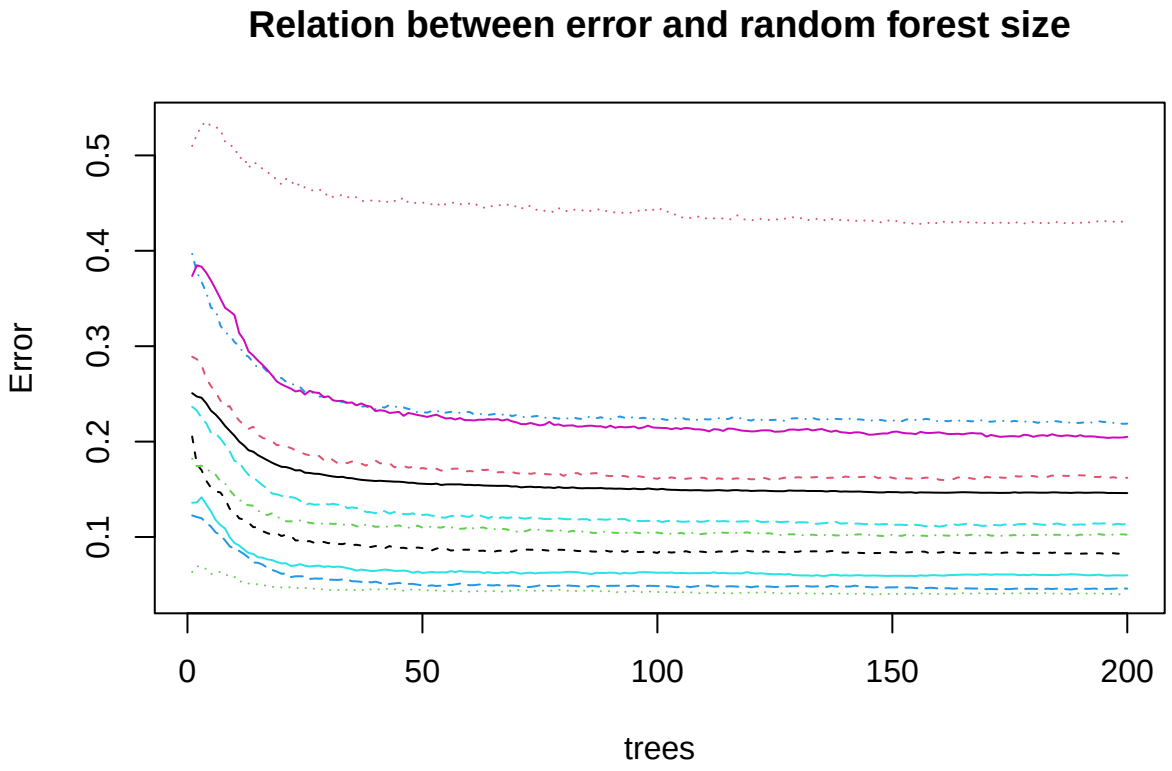
```
## randomForest 4.7-1
```

```
## Type rfNews() to see new features/changes/bug fixes.
```

```
##
## Attache Paket: 'randomForest'
```

```
## Das folgende Objekt ist maskiert 'package:ggplot2':
##
##     margin
```

```
plot(rf_best, main = "Relation between error and random forest size")
```



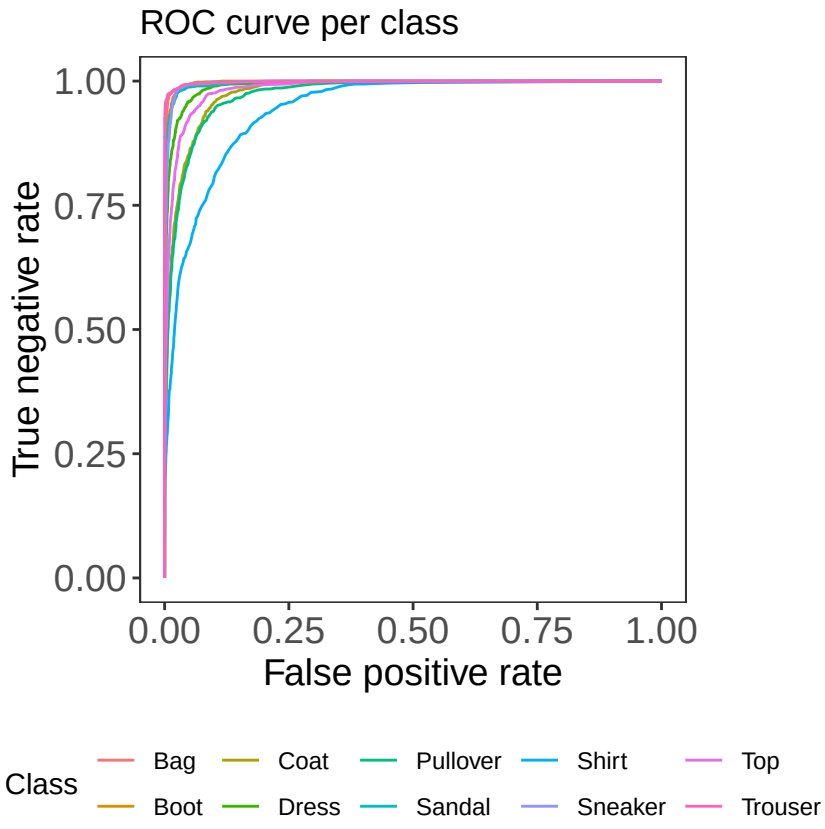

Wir beobachten in diesem Diagramm, dass der Fehler für keine der Klassen nach etwa 100 Bäumen mehr abnimmt. Wir können daraus schließen, dass unsere `randomForest`-Größe von 200 ausreichend ist. Wir können auch die `varImpPlot()`-Funktion aus dem `randomForest`-Paket verwenden, um die Relevanz für jede Variable darzustellen. Wir werden das hier nicht vorzeigen, da es nicht so aussagekräftig ist, da unsere Variablen ja Hauptkomponenten der tatsächlichen Pixel sind, es is aber immer gut daran zu denken, wenn diese Analysen auf andere Daten ausgedehnt werden. Schließlich zeichnen wir die ROC-Kurven für jede Klasse. Die Fläche unter dieser Kurve ist der Anteil der korrekten Klassifikationen für diese bestimmte Klasse, je weiter also die Kurve von der 45-Grad-Linie nach oben links "gezogen" wird, desto besser ist die Klassifikation für diese Klasse. Auf der x-Achse eines ROC-Diagramms haben wir normalerweise die Falsch-Positiv-Rate (false positive / (true negative + false positive)) und auf der y-Achse die Richtig-Positiv-Rate (true positive / (true positive + false negative)). Im Wesentlichen hilft uns das ROC-Diagramm, die Leistung unseres Modells in Bezug auf die Vorhersage verschiedener Klassen zu vergleichen. Wir müssen zuerst die Daten für die ROC-Kurve für jede Klasse (oder Kleidungskategorie) in unseren Daten erhalten, die wir alle zeilenweise zusammen führen, einschließlich der Label für die Klassen.

```
library(ROCR)
library(plyr)
pred_roc = predict(rf_best, test.images.pca, type = "prob")
classes = unique(test.data.pca$label)
classes = classes[order(classes)]
plot_list = list()
for (i in 1:length(classes)) {
  actual = ifelse(test.data.pca$label == classes[i], 1, 0)
  pred = prediction(pred_roc[, i], actual)
  perf = performance(pred, "tpr", "fpr")
```

```
  plot_list[[i]] = data.frame(matrix(NA, nrow = length(perf@x.values[[1]]), ncol = 2))
  plot_list[[i]]['x'] = perf@x.values[[1]]
  plot_list[[i]]['y'] = perf@y.values[[1]]
}
plotdf = rbind.fill(plot_list)
plotdf["Class"] = rep(cloth_cats, unlist(lapply(plot_list, nrow)))
```

Als nächstes zeichnen wir die ROC-Kurven für jede Klasse.

```
library(ggplot2)
my_theme = function () {
  theme_bw() +
    theme(axis.text = element_text(size = 14),
          axis.title = element_text(size = 14),
          strip.text = element_text(size = 14),
          panel.grid.major = element_blank(),
          panel.grid.minor = element_blank(),
          panel.background = element_blank(),
          legend.position = "bottom",
          strip.background = element_rect(fill = 'white', colour = 'white'))
}
ggplot() +
  geom_line(data = plotdf, aes(x = x, y = y, color = Class)) +
  labs(x = "False positive rate", y = "True negative rate", color = "Class") +
  ggtitle("ROC curve per class") +
  theme(legend.position = c(0.85, 0.35)) +
  coord_fixed() +
  my_theme()
```



Wir beobachten anhand der ROC-Kurven, dass Hemden und Pullover am häufigsten falsch klassifiziert werden (wie wir zuvor anhand der Konfusionsmatrix gesehen haben), während Hosen, Taschen, Stiefel und Turnschuhe am häufigsten korrekt klassifiziert werden. Dies entspricht auch dem, was wir zuvor anhand der aufgetragenen Konfusionsmatrix beobachtet haben. Eine mögliche Erklärung könnte sein, dass Hemden und Pullover in ihrer Form anderen Kategorien wie Oberteilen, Mänteln und Kleidern sehr ähnlich sehen können, wohingegen Taschen, Hosen, Stiefel und Turnschuhe sich von anderen Kategorien doch mehr unterscheiden.

## Gradient-Boosted Trees

Während in `random forest`'s jeder Baum mithilfe einer zufälligen Datenstichprobe eigenständig erstellt und unabhängig trainiert wird, enthält jeder neu erstellte Baum beim Boosten den Fehler des zuvor erstellten Baums. Das heißt, die Bäume werden sequentiell auf einer angepassten Version der Anfangsdaten erstellt, die kein Bootstrap-Sampling erfordert. Aus diesem Grund sind `boost`-Bäume normalerweise kleiner und flacher als die Bäume in `random forest`'s, wodurch der Baum dort verbessert wird, wo er noch nicht gut genug funktioniert. Boosting wird oft nachgesagt, dass es Random Forests übertrifft, was hauptsächlich daran liegt, dass der Ansatz langsam lernt. Dies kann noch weiter durch einen seiner Parameter (z. B. shrinkage) gesteuert werden, den wir später anpassen werden.

Beim Boosten ist es wichtig, die Parameter gut abzustimmen und mit verschiedenen Werten der Parameter herumzuspielen, was mit dem `caret`-Framework leicht erreicht werden kann. Diese Parameter umfassen die Lernrate (`eta`), die minimal erforderliche Verlustreduzierung zur weiteren Partitionierung auf einem Blattknoten des Baums (`gamma`), die maximale Tiefe eines Baums (`max_depth`), die Anzahl der Bäume (`nrounds`), die minimale Anzahl von Beobachtungen in den Knoten der Bäume (`min_child_weight`), der Anteil der Beobachtungen des Trainingssatzes, die zufällig ausgewählt wurden, um Bäume wachsen/erstellen zu lassen (`subsample`), und der Anteil der unabhängigen Variablen, die für jeden Baum verwendet werden soll (`colsample_bytree`). Eine Übersicht aller Parameter finden Sie hier. Auch hier verwenden wir das `caret`-Framework, um unser Boosting-Modell abzustimmen.

```
xgb_control = trainControl(
  method = "cv",
  number = 5,
  classProbs = TRUE,
  allowParallel = TRUE,
  savePredictions = TRUE
)
```

Als nächstes definieren wir die möglichen Kombinationen der Tuning-Parameter in Form eines Rasters namens „xgb_grid“.

```
xgb_grid = expand.grid(
  nrounds = c(50, 50),
  max_depth = seq(5, 15, 5),
  eta = c(0.002, 0.02, 0.2),
  gamma = c(0.1, 0.5, 1.0),
  colsample_bytree = 1,
  min_child_weight = c(1, 2, 3),
  subsample = c(0.5, 0.75, 1)
)
```

Wir setzen den Seed und trainieren dann das Modell auf die transformierten Hauptkomponenten der Trainingsdaten unter Verwendung von `xgb_control` and `xgb_grid`, wie vorher angegeben. Bitte beachten Sie, dass dies aufgrund der relativ großen Anzahl von Tuning-Parametern und damit größeren Anzahl möglicher Parameter-Kombinationen ziemlich lange dauern kann.

```
set.seed(815)
xgb_tune = train(x = train.images.pca,
                 y = train.classes,
```

```
                method = "xgbTree",
                trControl = xgb_control,
                tuneGrid = xgb_grid,
                verbosity = 0
)
xgb_tune
```

```
## eXtreme Gradient Boosting
##
## 60000 samples
##    17 predictor
##    10 classes: 'Bag', 'Boot', 'Coat', 'Dress', 'Pullover', 'Sandal', 'Shirt', 'Sneaker', 'Top', 'Trou
##
## No pre-processing
## Resampling: Cross-Validated (5 fold)
## Summary of sample sizes: 48000, 48000, 48000, 48000, 48000
## Resampling results across tuning parameters:
##
##   eta    max_depth  gamma  min_child_weight  subsample  Accuracy   Kappa
##   0.002   5         0.1    1                 0.50       0.7697333  0.7441481
##   0.002   5         0.1    1                 0.75       0.7650500  0.7389444
##   0.002   5         0.1    1                 1.00       0.7591667  0.7324074
##   0.002   5         0.1    2                 0.50       0.7702667  0.7447407
##   0.002   5         0.1    2                 0.75       0.7661000  0.7401111
##   0.002   5         0.1    2                 1.00       0.7589833  0.7322037
##   0.002   5         0.1    3                 0.50       0.7689667  0.7432963
##   0.002   5         0.1    3                 0.75       0.7648667  0.7387407
##   0.002   5         0.1    3                 1.00       0.7591000  0.7323333
##   0.002   5         0.5    1                 0.50       0.7705000  0.7450000
##   0.002   5         0.5    1                 0.75       0.7661667  0.7401852
##   0.002   5         0.5    1                 1.00       0.7591667  0.7324074
##   0.002   5         0.5    2                 0.50       0.7692167  0.7435741
##   0.002   5         0.5    2                 0.75       0.7652000  0.7391111
##   0.002   5         0.5    2                 1.00       0.7590000  0.7322222
##   0.002   5         0.5    3                 0.50       0.7691333  0.7434815
##   0.002   5         0.5    3                 0.75       0.7654667  0.7394074
##   0.002   5         0.5    3                 1.00       0.7591500  0.7323889
##   0.002   5         1.0    1                 0.50       0.7691333  0.7434815
##   0.002   5         1.0    1                 0.75       0.7652167  0.7391296
##   0.002   5         1.0    1                 1.00       0.7591333  0.7323704
##   0.002   5         1.0    2                 0.50       0.7699500  0.7443889
##   0.002   5         1.0    2                 0.75       0.7664500  0.7405000
##   0.002   5         1.0    2                 1.00       0.7590333  0.7322593
##   0.002   5         1.0    3                 0.50       0.7697333  0.7441481
##   0.002   5         1.0    3                 0.75       0.7654833  0.7394259
##   0.002   5         1.0    3                 1.00       0.7592000  0.7324444
##   0.002  10         0.1    1                 0.50       0.8267333  0.8074815
##   0.002  10         0.1    1                 0.75       0.8266500  0.8073889
##   0.002  10         0.1    1                 1.00       0.8139000  0.7932222
##   0.002  10         0.1    2                 0.50       0.8245000  0.8050000
##   0.002  10         0.1    2                 0.75       0.8254333  0.8060370
##   0.002  10         0.1    2                 1.00       0.8126500  0.7918333
##   0.002  10         0.1    3                 0.50       0.8225333  0.8028148
##   0.002  10         0.1    3                 0.75       0.8235333  0.8039259
```

```
##   0.002  10         0.1    3                  1.00       0.8119667  0.7910741
##   0.002  10         0.5    1                  0.50       0.8265833  0.8073148
##   0.002  10         0.5    1                  0.75       0.8255833  0.8062037
##   0.002  10         0.5    1                  1.00       0.8135667  0.7928519
##   0.002  10         0.5    2                  0.50       0.8249167  0.8054630
##   0.002  10         0.5    2                  0.75       0.8244833  0.8049815
##   0.002  10         0.5    2                  1.00       0.8123833  0.7915370
##   0.002  10         0.5    3                  0.50       0.8221000  0.8023333
##   0.002  10         0.5    3                  0.75       0.8234000  0.8037778
##   0.002  10         0.5    3                  1.00       0.8119000  0.7910000
##   0.002  10         1.0    1                  0.50       0.8264500  0.8071667
##   0.002  10         1.0    1                  0.75       0.8270167  0.8077963
##   0.002  10         1.0    1                  1.00       0.8132833  0.7925370
##   0.002  10         1.0    2                  0.50       0.8248500  0.8053889
##   0.002  10         1.0    2                  0.75       0.8253833  0.8059815
##   0.002  10         1.0    2                  1.00       0.8123000  0.7914444
##   0.002  10         1.0    3                  0.50       0.8220500  0.8022778
##   0.002  10         1.0    3                  0.75       0.8234667  0.8038519
##   0.002  10         1.0    3                  1.00       0.8115667  0.7906296
##   0.002  15         0.1    1                  0.50       0.8333333  0.8148148
##   0.002  15         0.1    1                  0.75       0.8356000  0.8173333
##   0.002  15         0.1    1                  1.00       0.8207000  0.8007778
##   0.002  15         0.1    2                  0.50       0.8305333  0.8117037
##   0.002  15         0.1    2                  0.75       0.8331000  0.8145556
##   0.002  15         0.1    2                  1.00       0.8189500  0.7988333
##   0.002  15         0.1    3                  0.50       0.8272833  0.8080926
##   0.002  15         0.1    3                  0.75       0.8305000  0.8116667
##   0.002  15         0.1    3                  1.00       0.8186000  0.7984444
##   0.002  15         0.5    1                  0.50       0.8331000  0.8145556
##   0.002  15         0.5    1                  0.75       0.8358667  0.8176296
##   0.002  15         0.5    1                  1.00       0.8206000  0.8006667
##   0.002  15         0.5    2                  0.50       0.8299000  0.8110000
##   0.002  15         0.5    2                  0.75       0.8332667  0.8147407
##   0.002  15         0.5    2                  1.00       0.8191500  0.7990556
##   0.002  15         0.5    3                  0.50       0.8282667  0.8091852
##   0.002  15         0.5    3                  0.75       0.8303667  0.8115185
##   0.002  15         0.5    3                  1.00       0.8185833  0.7984259
##   0.002  15         1.0    1                  0.50       0.8336333  0.8151481
##   0.002  15         1.0    1                  0.75       0.8361333  0.8179259
##   0.002  15         1.0    1                  1.00       0.8203833  0.8004259
##   0.002  15         1.0    2                  0.50       0.8303667  0.8115185
##   0.002  15         1.0    2                  0.75       0.8332833  0.8147593
##   0.002  15         1.0    2                  1.00       0.8191000  0.7990000
##   0.002  15         1.0    3                  0.50       0.8277000  0.8085556
##   0.002  15         1.0    3                  0.75       0.8299333  0.8110370
##   0.002  15         1.0    3                  1.00       0.8182333  0.7980370
##   0.020   5         0.1    1                  0.50       0.7862500  0.7625000
##   0.020   5         0.1    1                  0.75       0.7842000  0.7602222
##   0.020   5         0.1    1                  1.00       0.7811833  0.7568704
##   0.020   5         0.1    2                  0.50       0.7871833  0.7635370
##   0.020   5         0.1    2                  0.75       0.7842833  0.7603148
##   0.020   5         0.1    2                  1.00       0.7810167  0.7566852
##   0.020   5         0.1    3                  0.50       0.7857667  0.7619630
##   0.020   5         0.1    3                  0.75       0.7837667  0.7597407
```

```
##   0.020   5         0.1    3                 1.00       0.7809000  0.7565556
##   0.020   5         0.5    1                 0.50       0.7870167  0.7633519
##   0.020   5         0.5    1                 0.75       0.7843833  0.7604259
##   0.020   5         0.5    1                 1.00       0.7811333  0.7568148
##   0.020   5         0.5    2                 0.50       0.7863167  0.7625741
##   0.020   5         0.5    2                 0.75       0.7834000  0.7593333
##   0.020   5         0.5    2                 1.00       0.7807500  0.7563889
##   0.020   5         0.5    3                 0.50       0.7854333  0.7615926
##   0.020   5         0.5    3                 0.75       0.7835167  0.7594630
##   0.020   5         0.5    3                 1.00       0.7806000  0.7562222
##   0.020   5         1.0    1                 0.50       0.7863167  0.7625741
##   0.020   5         1.0    1                 0.75       0.7843167  0.7603519
##   0.020   5         1.0    1                 1.00       0.7809833  0.7566481
##   0.020   5         1.0    2                 0.50       0.7860667  0.7622963
##   0.020   5         1.0    2                 0.75       0.7840833  0.7600926
##   0.020   5         1.0    2                 1.00       0.7807500  0.7563889
##   0.020   5         1.0    3                 0.50       0.7857000  0.7618889
##   0.020   5         1.0    3                 0.75       0.7844333  0.7604815
##   0.020   5         1.0    3                 1.00       0.7807500  0.7563889
##   0.020  10         0.1    1                 0.50       0.8355667  0.8172963
##   0.020  10         0.1    1                 0.75       0.8366667  0.8185185
##   0.020  10         0.1    1                 1.00       0.8330167  0.8144630
##   0.020  10         0.1    2                 0.50       0.8350833  0.8167593
##   0.020  10         0.1    2                 0.75       0.8357000  0.8174444
##   0.020  10         0.1    2                 1.00       0.8321667  0.8135185
##   0.020  10         0.1    3                 0.50       0.8336167  0.8151296
##   0.020  10         0.1    3                 0.75       0.8345500  0.8161667
##   0.020  10         0.1    3                 1.00       0.8317000  0.8130000
##   0.020  10         0.5    1                 0.50       0.8357333  0.8174815
##   0.020  10         0.5    1                 0.75       0.8364667  0.8182963
##   0.020  10         0.5    1                 1.00       0.8323000  0.8136667
##   0.020  10         0.5    2                 0.50       0.8353500  0.8170556
##   0.020  10         0.5    2                 0.75       0.8350500  0.8167222
##   0.020  10         0.5    2                 1.00       0.8320167  0.8133519
##   0.020  10         0.5    3                 0.50       0.8326167  0.8140185
##   0.020  10         0.5    3                 0.75       0.8350667  0.8167407
##   0.020  10         0.5    3                 1.00       0.8315667  0.8128519
##   0.020  10         1.0    1                 0.50       0.8360000  0.8177778
##   0.020  10         1.0    1                 0.75       0.8364667  0.8182963
##   0.020  10         1.0    1                 1.00       0.8323333  0.8137037
##   0.020  10         1.0    2                 0.50       0.8344333  0.8160370
##   0.020  10         1.0    2                 0.75       0.8356000  0.8173333
##   0.020  10         1.0    2                 1.00       0.8319333  0.8132593
##   0.020  10         1.0    3                 0.50       0.8325833  0.8139815
##   0.020  10         1.0    3                 0.75       0.8341167  0.8156852
##   0.020  10         1.0    3                 1.00       0.8314167  0.8126852
##   0.020  15         0.1    1                 0.50       0.8431333  0.8257037
##   0.020  15         0.1    1                 0.75       0.8449167  0.8276852
##   0.020  15         0.1    1                 1.00       0.8416500  0.8240556
##   0.020  15         0.1    2                 0.50       0.8411000  0.8234444
##   0.020  15         0.1    2                 0.75       0.8437333  0.8263704
##   0.020  15         0.1    2                 1.00       0.8408000  0.8231111
##   0.020  15         0.1    3                 0.50       0.8400333  0.8222593
##   0.020  15         0.1    3                 0.75       0.8428667  0.8254074
```

```
##   0.020  15         0.1    3                  1.00       0.8403667  0.8226296
##   0.020  15         0.5    1                  0.50       0.8425667  0.8250741
##   0.020  15         0.5    1                  0.75       0.8448500  0.8276111
##   0.020  15         0.5    1                  1.00       0.8405000  0.8227778
##   0.020  15         0.5    2                  0.50       0.8414167  0.8237963
##   0.020  15         0.5    2                  0.75       0.8450500  0.8278333
##   0.020  15         0.5    2                  1.00       0.8404333  0.8227037
##   0.020  15         0.5    3                  0.50       0.8398333  0.8220370
##   0.020  15         0.5    3                  0.75       0.8424333  0.8249259
##   0.020  15         0.5    3                  1.00       0.8403167  0.8225741
##   0.020  15         1.0    1                  0.50       0.8425833  0.8250926
##   0.020  15         1.0    1                  0.75       0.8433500  0.8259444
##   0.020  15         1.0    1                  1.00       0.8409167  0.8232407
##   0.020  15         1.0    2                  0.50       0.8413000  0.8236667
##   0.020  15         1.0    2                  0.75       0.8430500  0.8256111
##   0.020  15         1.0    2                  1.00       0.8403000  0.8225556
##   0.020  15         1.0    3                  0.50       0.8398000  0.8220000
##   0.020  15         1.0    3                  0.75       0.8417500  0.8241667
##   0.020  15         1.0    3                  1.00       0.8398000  0.8220000
##   0.200   5         0.1    1                  0.50       0.8414333  0.8238148
##   0.200   5         0.1    1                  0.75       0.8385833  0.8206481
##   0.200   5         0.1    1                  1.00       0.8361333  0.8179259
##   0.200   5         0.1    2                  0.50       0.8405500  0.8228333
##   0.200   5         0.1    2                  0.75       0.8400000  0.8222222
##   0.200   5         0.1    2                  1.00       0.8368333  0.8187037
##   0.200   5         0.1    3                  0.50       0.8400000  0.8222222
##   0.200   5         0.1    3                  0.75       0.8398833  0.8220926
##   0.200   5         0.1    3                  1.00       0.8363167  0.8181296
##   0.200   5         0.5    1                  0.50       0.8402667  0.8225185
##   0.200   5         0.5    1                  0.75       0.8385667  0.8206296
##   0.200   5         0.5    1                  1.00       0.8367667  0.8186296
##   0.200   5         0.5    2                  0.50       0.8407333  0.8230370
##   0.200   5         0.5    2                  0.75       0.8394167  0.8215741
##   0.200   5         0.5    2                  1.00       0.8363667  0.8181852
##   0.200   5         0.5    3                  0.50       0.8397833  0.8219815
##   0.200   5         0.5    3                  0.75       0.8393500  0.8215000
##   0.200   5         0.5    3                  1.00       0.8369167  0.8187963
##   0.200   5         1.0    1                  0.50       0.8398333  0.8220370
##   0.200   5         1.0    1                  0.75       0.8393833  0.8215370
##   0.200   5         1.0    1                  1.00       0.8362667  0.8180741
##   0.200   5         1.0    2                  0.50       0.8398500  0.8220556
##   0.200   5         1.0    2                  0.75       0.8393833  0.8215370
##   0.200   5         1.0    2                  1.00       0.8362667  0.8180741
##   0.200   5         1.0    3                  0.50       0.8402833  0.8225370
##   0.200   5         1.0    3                  0.75       0.8387833  0.8208704
##   0.200   5         1.0    3                  1.00       0.8364000  0.8182222
##   0.200  10         0.1    1                  0.50       0.8556000  0.8395556
##   0.200  10         0.1    1                  0.75       0.8551500  0.8390556
##   0.200  10         0.1    1                  1.00       0.8528500  0.8365000
##   0.200  10         0.1    2                  0.50       0.8540500  0.8378333
##   0.200  10         0.1    2                  0.75       0.8551333  0.8390370
##   0.200  10         0.1    2                  1.00       0.8539833  0.8377593
##   0.200  10         0.1    3                  0.50       0.8545667  0.8384074
##   0.200  10         0.1    3                  0.75       0.8558833  0.8398704
```

```
##   0.200  10         0.1    3                1.00       0.8542667  0.8380741
##   0.200  10         0.5    1                0.50       0.8554833  0.8394259
##   0.200  10         0.5    1                0.75       0.8560500  0.8400556
##   0.200  10         0.5    1                1.00       0.8522833  0.8358704
##   0.200  10         0.5    2                0.50       0.8545500  0.8383889
##   0.200  10         0.5    2                0.75       0.8566000  0.8406667
##   0.200  10         0.5    2                1.00       0.8536667  0.8374074
##   0.200  10         0.5    3                0.50       0.8545000  0.8383333
##   0.200  10         0.5    3                0.75       0.8552500  0.8391667
##   0.200  10         0.5    3                1.00       0.8545667  0.8384074
##   0.200  10         1.0    1                0.50       0.8532000  0.8368889
##   0.200  10         1.0    1                0.75       0.8553833  0.8393148
##   0.200  10         1.0    1                1.00       0.8534000  0.8371111
##   0.200  10         1.0    2                0.50       0.8547500  0.8386111
##   0.200  10         1.0    2                0.75       0.8552833  0.8392037
##   0.200  10         1.0    2                1.00       0.8518833  0.8354259
##   0.200  10         1.0    3                0.50       0.8535000  0.8372222
##   0.200  10         1.0    3                0.75       0.8541000  0.8378889
##   0.200  10         1.0    3                1.00       0.8536000  0.8373333
##   0.200  15         0.1    1                0.50       0.8568500  0.8409444
##   0.200  15         0.1    1                0.75       0.8574000  0.8415556
##   0.200  15         0.1    1                1.00       0.8563667  0.8404074
##   0.200  15         0.1    2                0.50       0.8566333  0.8407037
##   0.200  15         0.1    2                0.75       0.8581667  0.8424074
##   0.200  15         0.1    2                1.00       0.8564333  0.8404815
##   0.200  15         0.1    3                0.50       0.8567167  0.8407963
##   0.200  15         0.1    3                0.75       0.8571500  0.8412778
##   0.200  15         0.1    3                1.00       0.8564333  0.8404815
##   0.200  15         0.5    1                0.50       0.8562667  0.8402963
##   0.200  15         0.5    1                0.75       0.8569000  0.8410000
##   0.200  15         0.5    1                1.00       0.8548000  0.8386667
##   0.200  15         0.5    2                0.50       0.8558167  0.8397963
##   0.200  15         0.5    2                0.75       0.8573667  0.8415185
##   0.200  15         0.5    2                1.00       0.8562500  0.8402778
##   0.200  15         0.5    3                0.50       0.8565833  0.8406481
##   0.200  15         0.5    3                0.75       0.8575167  0.8416852
##   0.200  15         0.5    3                1.00       0.8567333  0.8408148
##   0.200  15         1.0    1                0.50       0.8550167  0.8389074
##   0.200  15         1.0    1                0.75       0.8544500  0.8382778
##   0.200  15         1.0    1                1.00       0.8541500  0.8379444
##   0.200  15         1.0    2                0.50       0.8554667  0.8394074
##   0.200  15         1.0    2                0.75       0.8557167  0.8396852
##   0.200  15         1.0    2                1.00       0.8547167  0.8385741
##   0.200  15         1.0    3                0.50       0.8553667  0.8392963
##   0.200  15         1.0    3                0.75       0.8560667  0.8400741
##   0.200  15         1.0    3                1.00       0.8552667  0.8391852
##
## Tuning parameter 'nrounds' was held constant at a value of 50
## Tuning
##  parameter 'colsample_bytree' 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 = 15, eta
##  = 0.2, gamma = 0.1, colsample_bytree = 1, min_child_weight = 2 and subsample
##  = 0.75.
```

Werfen wir mal einen Blick auf die Tunig-Parameter, die zur höchsten Genauigkeit führen, und auf die Modellleistung insgesamt.

```
xgb_tune$results[which.max(xgb_tune$results$Accuracy), ]
```

```
##     eta max_depth gamma colsample_bytree min_child_weight subsample nrounds
## 221 0.2        15   0.1                1                2      0.75      50
##      Accuracy     Kappa  AccuracySD     KappaSD
## 221 0.8581667 0.8424074 0.002486072 0.002762303
```

```
mp.xgb = model_performance(xgb_tune, train.images.pca, test.images.pca,
                           train.classes, test.classes, "xgboost")
```

```
##   accuracy_train precision_train recall_train  F1_train accuracy_test
## 1      0.8581667       0.9512599    0.9563333 0.9537899         0.852
##   precision_test recall_test   F1_test   model
## 1       0.950495        0.96 0.9552239 xgboost
```

Die optimale Kombination von Tuning-Parameterwerten führte zu 85,8 % Trainings- und 85,2 % Testgenauigkeit. Obwohl es zu einer leichten Überanpassung kommen kann, schneidet das Modell erwartungsgemäß etwas besser ab als der Random Forest. Werfen wir einen Blick auf die Konfusionsmatrix für die Vorhersagen des Testsets, um zu sehen, welche Kleidungskategorien am häufigsten richtig oder falsch klassifiziert werden.

```
table(pred = predict(xgb_tune, test.images.pca),
      true = test.classes)
```

```
##           true
## pred       Bag Boot Coat Dress Pullover Sandal Shirt Sneaker Top Trouser
##   Bag      960    1    5     4        6      5    17       2   8       2
##   Boot       2  942    0     0        0     30     0      48   0       0
##   Coat       3    0  772    29      112      0   102       0   5       6
##   Dress      7    0   32   884        9      2    29       0  34      22
##   Pullover   8    0  121     8      767      0   109       0  16       3
##   Sandal     7   17    0     0        0    913     2      39   3       0
##   Shirt      5    0   67    35       88      0   583       0  98       3
##   Sneaker    4   40    0     0        0     50     0     911   0       0
##   Top        4    0    2    28       17      0   157       0 833       9
##   Trouser    0    0    1    12        1      0     1       0   3     955
```

Wie wir bei den Random Forest's gesehen haben, werden Pullover, Hemden und Mäntel am häufigsten verwechselt, während Hosen, Stiefel, Taschen und Turnschuhe am häufigsten richtig klassifiziert werden.