HarvardX - PH125.9x - DataScience: Capstone - ImageProcessing

Jan Brettschneider

15th April 2020

1. Introduction

This document summarizes the chosen project "Image Processing" for the Capstone Exam in course of the HardvardX Data Science Certificate.

The dataset used is the **Fashion-MNIST** dataset published by the company Zalando. It contains 28x28 pixel grayscale images of the articles which are associated with 10 classes. It is commonly used for benchmarking machine learning algorithms. Due to the fact that we already used the classical MNIST dataset with numbers, I'm really interested in the processing of pictures and this dataset is already tidy and proven, I decided to give it a go.

This project again will have the classical approach of machine learning projects, although I will put the focus on the machine learning part of things. The steps performed are as follows:

1. Download the raw data and import the files into R Studio
2. Process and explore the data, to find out more how to proceed during model creation
3. Build and optimize different models
4. Use the best model on the test set and evaluate the results
5. Conclusion, limitations and comments

The ambition is to get a better understanding of which parameters and methods can be used for image processing problems.

Loading the data

The original dataset can be found in different sources. I downloaded the data from **Kaggle** and reuploaded it into my dropbox so it can be automatically downloaded without having to register on the website.

In addition to the packages used during the MovieLens project I utilized the **downloader**, **matrixStats**, **Matrix**, **randomForest**, **rpart**, **tibble** and **knitr** packages. The code for downloading and extracting the model looks as follows:

```r
#################################
# Create train set, test set
#################################

# Load required packages if required
if(!require(tidyverse)) install.packages("tidyverse", repos = "http://cran.us.r-project.org")
if(!require(caret)) install.packages("caret", repos = "http://cran.us.r-project.org")
if(!require(data.table)) install.packages("data.table", repos = "http://cran.us.r-project.org")
if(!require(downloader)) install.packages("downloader", repos = "http://cran.us.r-project.org")
if(!require(matrixStats)) install.packages("matrixStats", repos = "http://cran.us.r-project.org")
if(!require(Matrix)) install.packages("Matrix", repos = "http://cran.us.r-project.org")
if(!require(randomForest)) install.packages("randomForest", repos = "http://cran.us.r-project.org")
if(!require(rpart)) install.packages("rpart", repos = "http://cran.us.r-project.org")
if(!require(tibble)) install.packages("tibble", repos = "http://cran.us.r-project.org")
if(!require(knitr)) install.packages("knitr", repos = "http://cran.us.r-project.org")

# Download the dataset from my Dropbox and unzip it
url <- "https://www.dropbox.com/s/7yzdmv2im26hauh/fashionmnist.zip?raw=1"
download(url, dest="fashionmnist.zip", mode="wb")
unzip ("fashionmnist.zip", exdir = ".")
```

Afterwards we are loading the .csv files into R and storing them into our variables:

```r
# Create the data set for model building and verification
verification = read.csv("fashion-mnist_test.csv", header = TRUE)
modelling = read.csv("fashion-mnist_train.csv", header = TRUE)
```

Now two datasets have been created one for modelling with 60,000 samples and one for the testing/verification of our final model with 10,000 samples. Let's take a look at the structure of the data:

```r
modelling[1:5,1:10]
```

```
##   label pixel1 pixel2 pixel3 pixel4 pixel5 pixel6 pixel7 pixel8 pixel9
## 1     2      0      0      0      0      0      0      0      0      0
## 2     9      0      0      0      0      0      0      0      0      0
## 3     6      0      0      0      0      0      0      0      5      0
## 4     0      0      0      0      1      2      0      0      0      0
## 5     3      0      0      0      0      0      0      0      0      0
```

Each row contains the label in the first column and greyvalue for the 28 by 28 pixels of the image stored in the columns 2 to 785. Formulated differently, we have one picture plus label stored in each row of the matrix.

Preparing the data

For easier processing we are also seperating the labels from the images:

```r
# Seperate labels and images
x <- modelling[,2:ncol(modelling)]
y <- factor(modelling[,1])
x_verification <- verification[,2:ncol(verification)]
y_verification <- factor(verification[,1])
```

The images are currently stored in a matrix as you can see here:

```r
dim(x)
```

```
## [1] 60000   784
```

```r
dim(x_verification)
```

```
## [1] 10000   784
```

For some kind of analysis it is easier to have arrays, so we create an additional dataset with reshaped data:

```r
x_images <- array(as.numeric(unlist(x)), dim=c(60000, 28, 28))
x_verification_images <- array(as.numeric(unlist(x_verification)),
                               dim=c(10000, 28, 28))
```

Now we have the data in the following format:

```r
dim(x_images)
```

```
## [1] 60000    28    28
```

```r
dim(x_verification_images)
```

```
## [1] 10000    28    28
```

We have to split the dataset into training-set and test-set to train our models. We are building several sets (arrays and matrizes) because the methods use different input, but basically we are conducting the same split:

```r
# Splitting the modelling set
mod <- modelling[1:50000,]
test <- modelling[50001:60000,]

# Seperating pictures and labels - matrix
x_mod <- mod[,2:785]
y_mod <- as.factor(mod[,1])
x_test <- test[,2:785]
y_test <- as.factor(test[,1])

# Seperating pictures and labels - array
x_sub <- x_images[1:50000,,]
y_sub <- y[1:50000]
x_test_sub <- x_images[50001:60000,,]
y_test_sub <- y[50001:60000]
```

And as a last step in this section we will create the fashion categories according to the numbers which indicate the categories:

```r
fashion_cat = c('T-shirt/top',
                'Trouser',
                'Pullover',
                'Dress',
                'Coat',
                'Sandal',
                'Shirt',
                'Sneaker',
                'Bag',
                'Ankle boot')
```

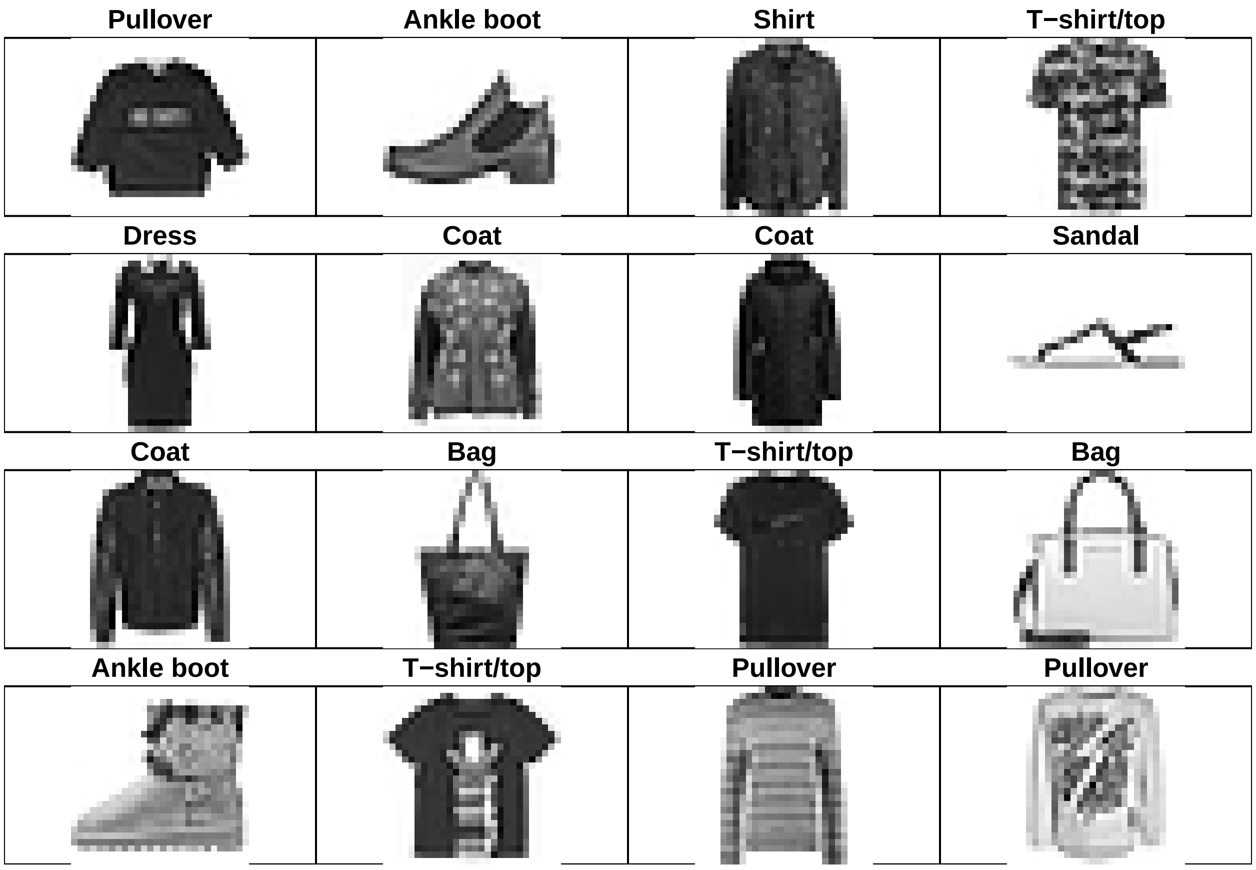
2. Analysing the data

In this section we will explore the data more in depth. How do the images acutally look like, do we see patterns between the categories or do we have some promising points for leveraging the most out of our method toolbox?

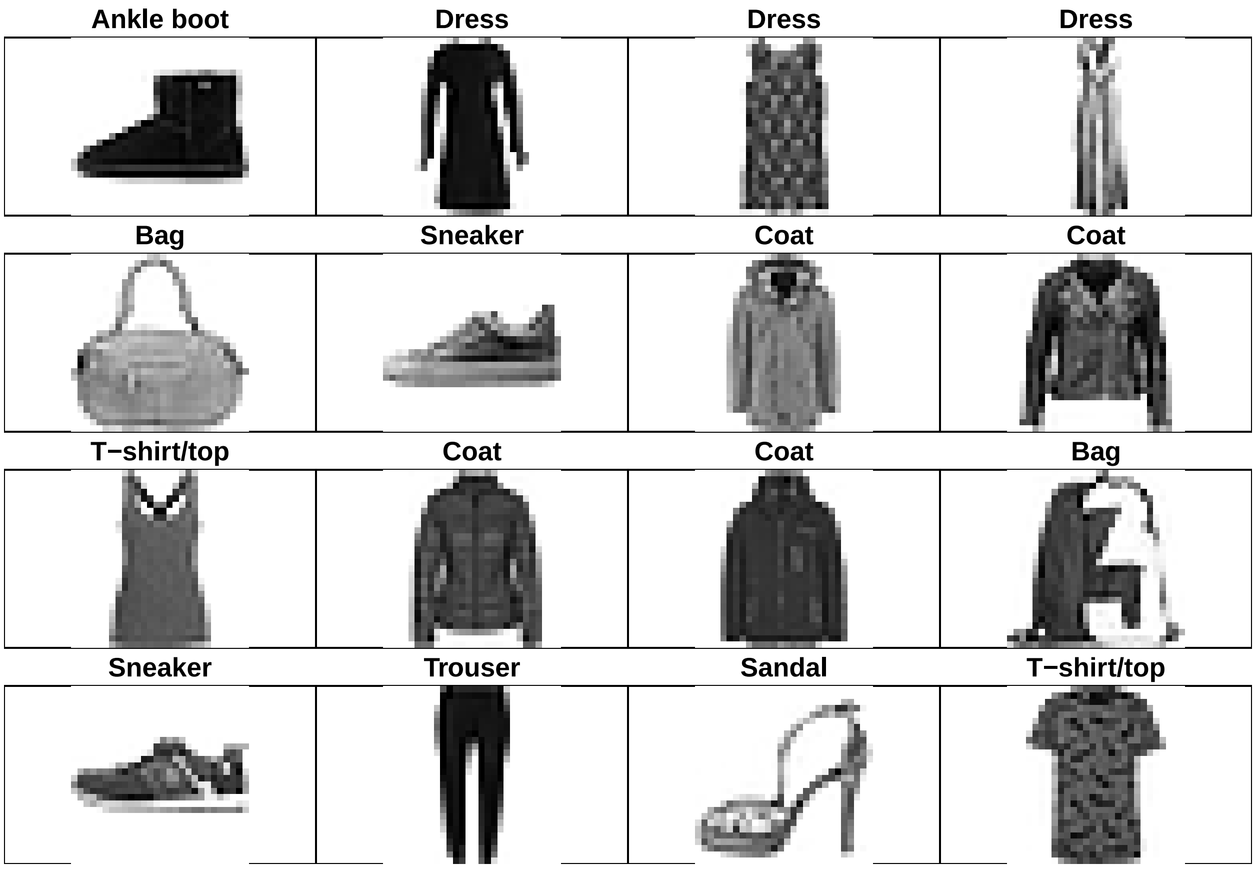
The pictures

Let's take a look at the first 32 pictures of the dataset:

```r
# Plot the first 16 images with labels to get an impression how they look
# Define a four by four grid for the plot
par(mfrow=c(4,4))
# Setting the margins around the pictures with "mar"
par(mar=c(0, 0, 1.5, 0))

# Plot every image (remember to reverse) on a gray scale with label
# Reverse the color scheme to get a white background
for (i in 1:16) {
  image(1:28, 1:28, as.matrix(x_images[i,,])[,28:1], col = gray((255:0)/255), xaxt = 'n',
        yaxt = 'n', xlab = "", ylab = "", asp = 1, main = paste(fashion_cat[y[i]]))
}
```



```r
# Plot the next 16 images with the same logic
for (i in 17:32) {
  image(1:28, 1:28, as.matrix(x_images[i,,])[,28:1], col = gray((255:0)/255), xaxt = 'n',
        yaxt = 'n', xlab = "", ylab = "", asp = 1, main = paste(fashion_cat[y[i]]))
}
```

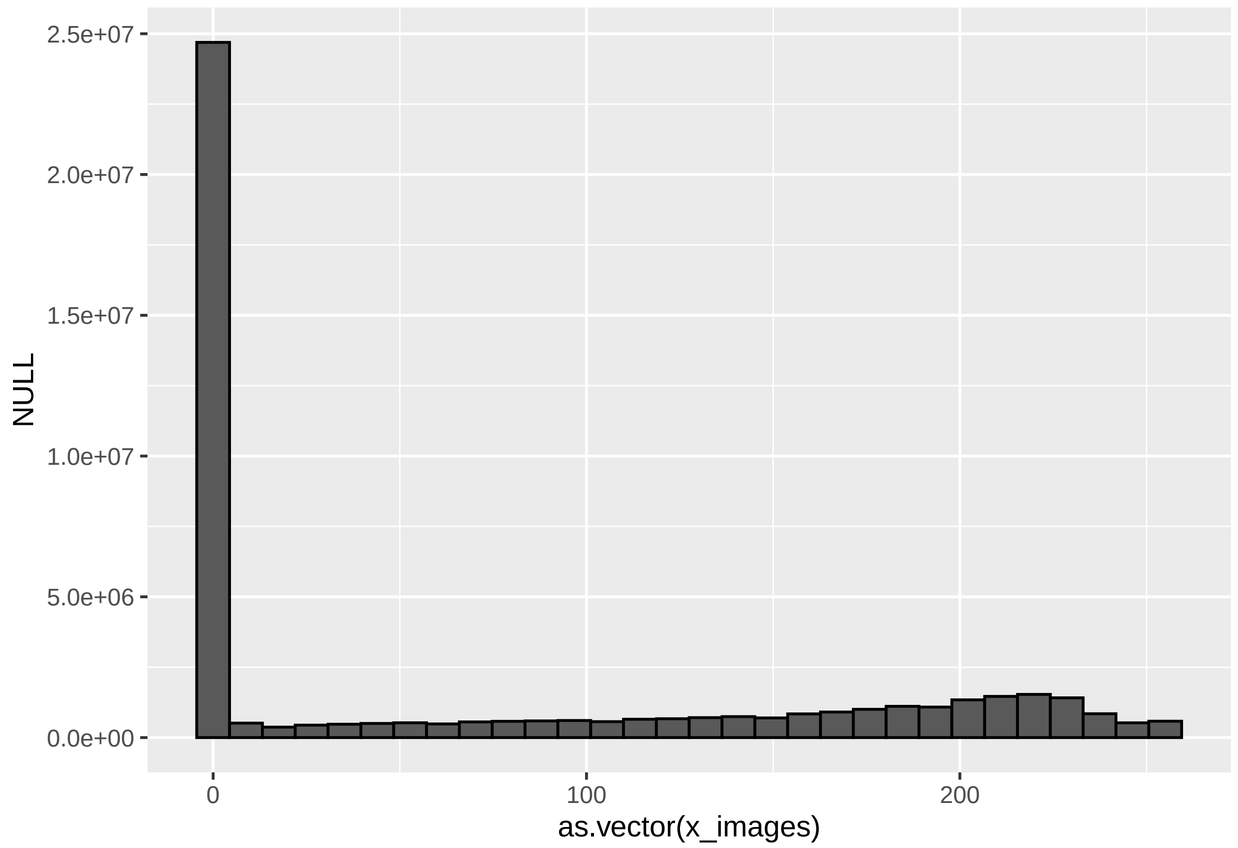


As we can see, we have some "low quality" pictures of several types of fashion, which can be more or less easily identified manually. But how can we do that with help of machine learning?

Color analysis

Let us start with the basics. First let us check whether we can set a threshold for making the pictures even simpler:

```
qplot(as.vector(x_images), bins = 30, color = I("black"))
```

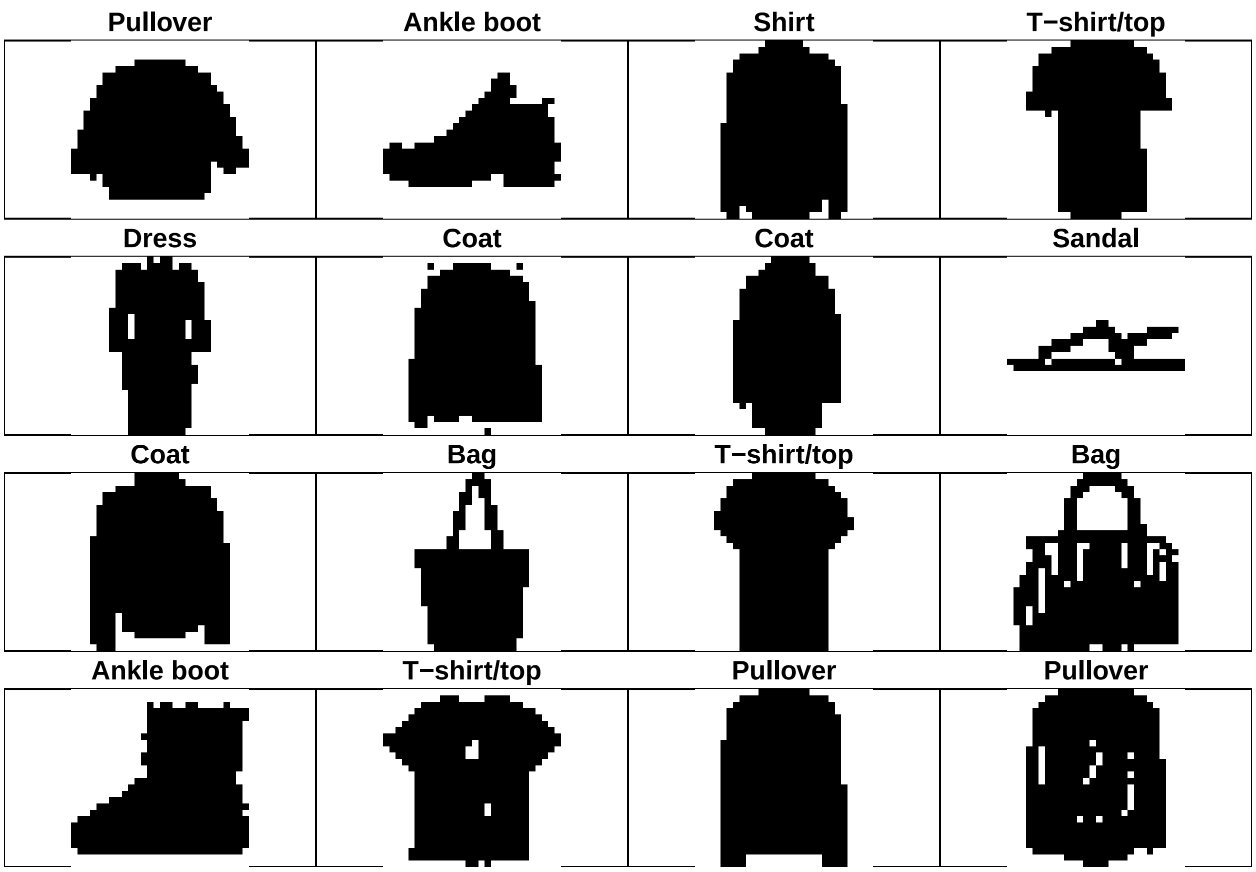


As we can see there is no clear cutoff, but we can try to set a low cutoff threshold, so we only have black and white left without anything in between. Here is the code we use on the subset for a cutoff below 10:

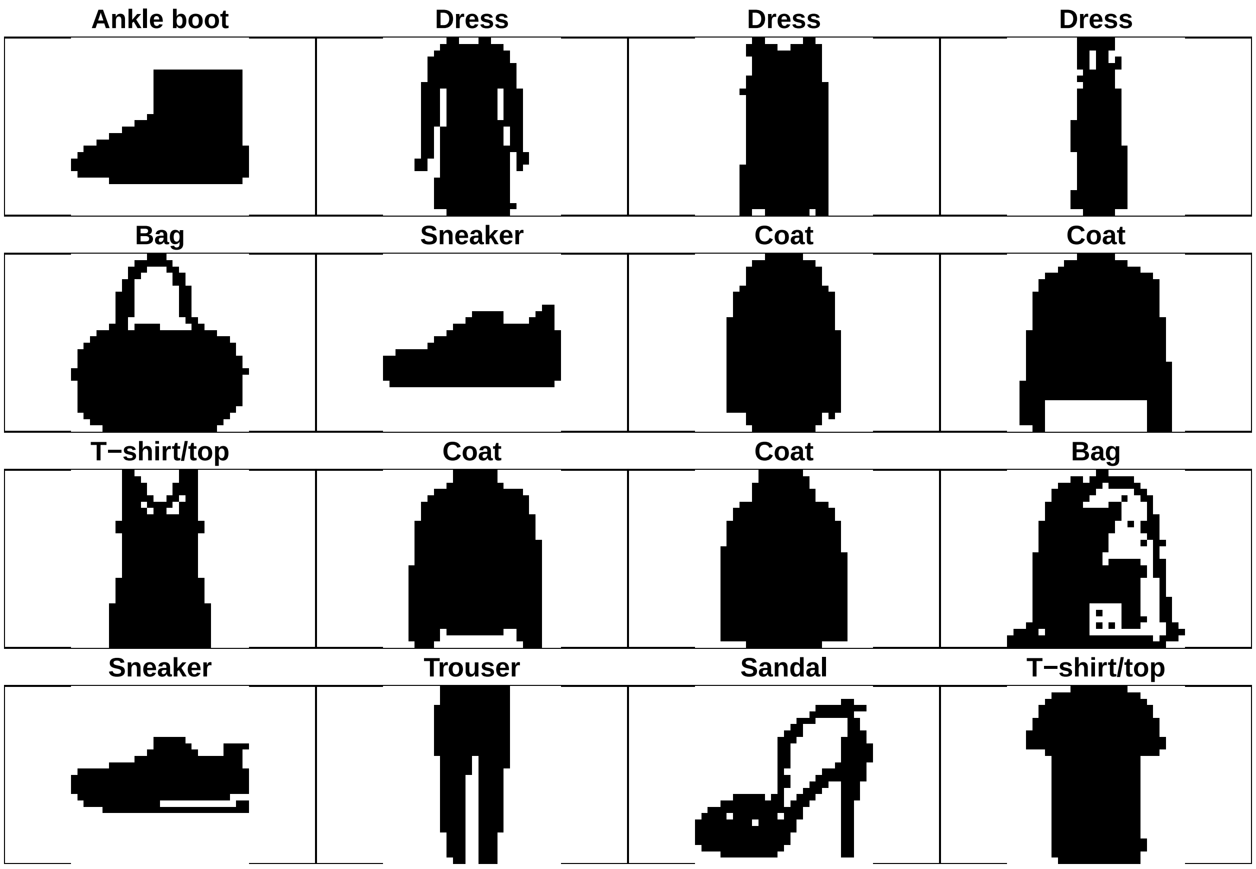
```r
x_sub_bw <- ifelse(x_sub < 10, 0 , 1)
```

Here are the same 32 pictures again

```r
# Plot the first 16 images with labels to get an impression how they look
# Define a four by four grid for the plot
par(mfrow=c(4,4))
# Setting the margins around the pictures with "mar"
par(mar=c(0, 0, 1.5, 0))

# Plot every image (remember to reverse) on a gray scale with label
# Reverse the color scheme to get a white background
for (i in 1:16) {
  image(1:28, 1:28, as.matrix(x_sub_bw[i,,])[,28:1], col = gray(1:0), xaxt = 'n',
        yaxt = 'n', xlab = "", ylab = "", asp = 1, main = paste(fashion_cat[y[i]]))
}
```



```r
# Plot the next 16 images with the same logic
for (i in 17:32) {
  image(1:28, 1:28, as.matrix(x_sub_bw[i,,])[,28:1], col = gray(1:0), xaxt = 'n',
        yaxt = 'n', xlab = "", ylab = "", asp = 1, main = paste(fashion_cat[y[i]]))
}
```



As we can see, this does not help us. for example picture 3, 9 and 15 (shirt, coat and pullover) now look even more similar. So we will not pursue this approach any further.

Next, we can take a look at the intensity of the pictures, so the mean graycolor value for each picture. We exclude the first column, which contains the label, group by the label and add the fashion category vector so we have a better understanding:

```r
arrange(cbind(modelling %>% mutate(row_mean = rowMeans(modelling[,-1])) %>%
        group_by(label) %>% summarize(avg_pic_int = mean(row_mean)),fashion_cat),
        desc(avg_pic_int))
```

```
##    label avg_pic_int fashion_cat
## 1      4    98.15566        Coat
## 2      2    95.71933    Pullover
## 3      8    90.03576         Bag
## 4      6    85.11510       Shirt
## 5      0    82.87926 T-shirt/top
## 6      9    77.00715  Ankle boot
## 7      3    66.20149       Dress
## 8      1    56.74837     Trouser
## 9      7    42.83567     Sneaker
## 10     5    34.87052      Sandal
```

Here is the standard deviation for the intensity by category:

```r
arrange(cbind(modelling %>% mutate(row_sds = rowSds(as.matrix(modelling[,-1]))) %>%
        group_by(label) %>% summarize(avg_sds = mean(row_sds)),fashion_cat),
        desc(avg_sds))
```

```
##    label  avg_sds fashion_cat
## 1      9 91.94726  Ankle boot
## 2      4 90.36731        Coat
## 3      8 87.32423         Bag
## 4      3 85.63856       Dress
## 5      1 85.02875     Trouser
## 6      2 82.15846    Pullover
## 7      0 81.42681 T-shirt/top
## 8      6 76.85188       Shirt
## 9      7 72.85919     Sneaker
## 10     5 63.28603      Sandal
```

Furthermore we look how many pixels are unequal to zero, and therefore used for the actual image excluding the background:

```r
arrange(cbind(modelling %>% mutate(pixels = rowSums(modelling[,-1]>0)) %>%
        group_by(label) %>% summarize(avg_pixels_used = mean(pixels)),fashion_cat),
        desc(avg_pixels_used))
```

```
##    label avg_pixels_used fashion_cat
## 1      2        508.4588    Pullover
## 2      6        493.3368       Shirt
## 3      4        471.9568        Coat
## 4      0        466.3948 T-shirt/top
## 5      8        458.3810         Bag
## 6      9        380.6383  Ankle boot
## 7      3        336.6065       Dress
## 8      1        273.8315     Trouser
## 9      7        264.6172     Sneaker
## 10     5        251.6750      Sandal
```

We can also calculate the average pixel intensity by dividing the sum of the grayscale values divided by the number of pixels unequal to zero:

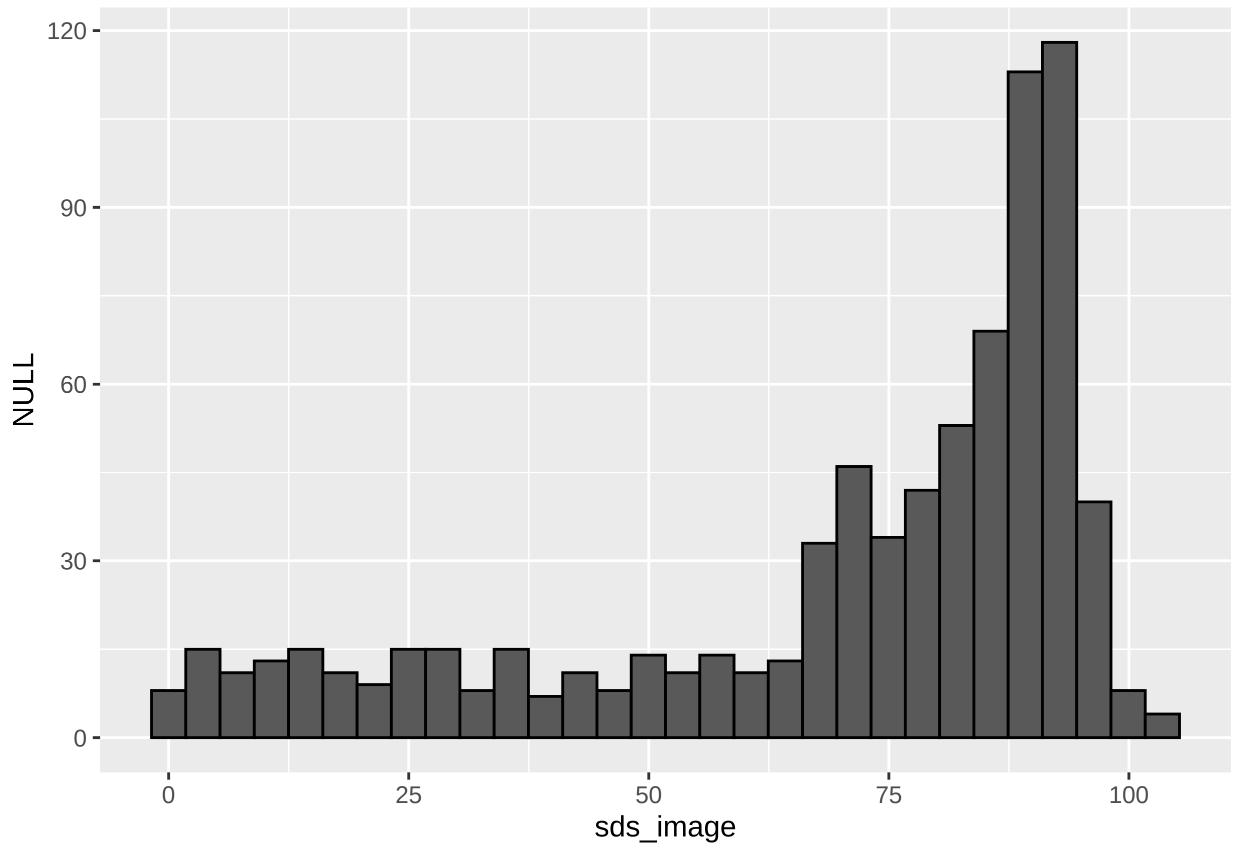
```r
arrange(cbind(modelling %>% mutate(pixel_int = rowSums(modelling[,-1])/
        rowSums(modelling[,-1]>0)) %>% group_by(label) %>%
        summarize(avg_pixel_int = mean(pixel_int)),fashion_cat),
        desc(avg_pixel_int))
```

```
##    label avg_pixel_int fashion_cat
## 1      1      163.6052     Trouser
## 2      4      163.0429        Coat
## 3      9      158.2658  Ankle boot
## 4      3      155.0386       Dress
## 5      8      152.7075         Bag
## 6      2      147.7495    Pullover
## 7      0      139.5437 T-shirt/top
## 8      6      135.2090       Shirt
## 9      7      126.4115     Sneaker
## 10     5      107.2801      Sandal
```

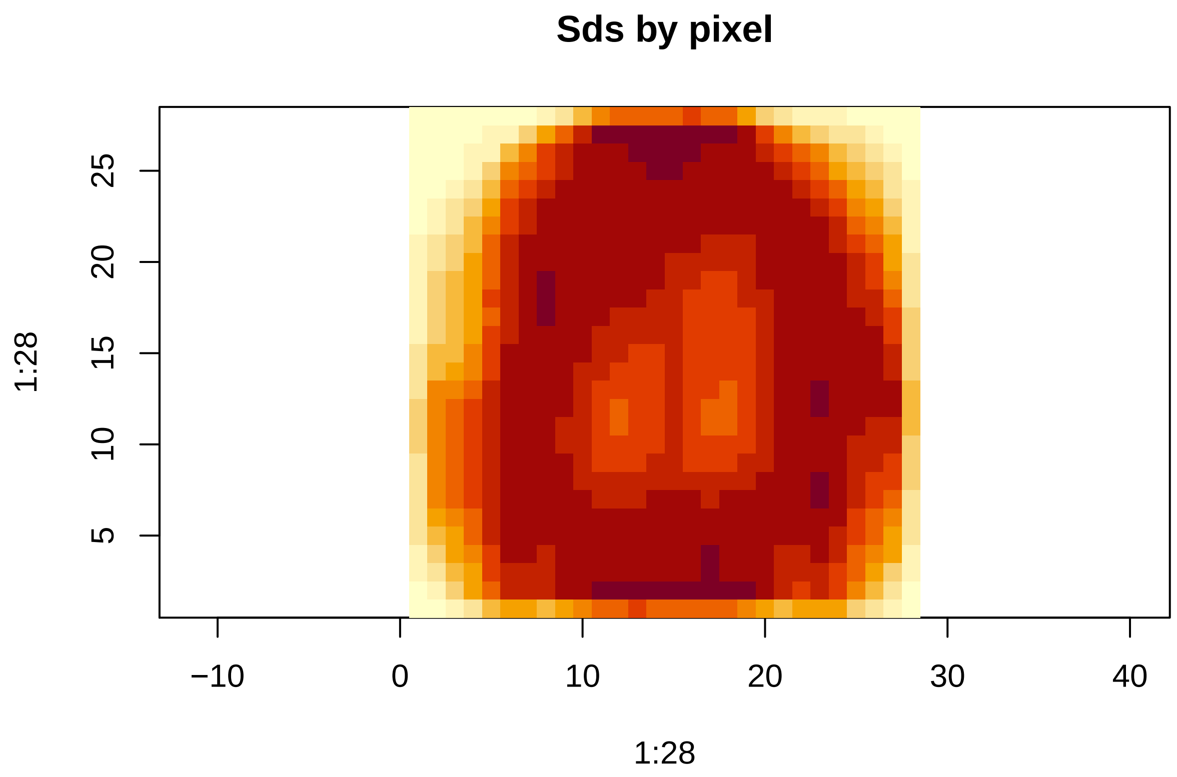
Variability by pixel

Now we take a look at the variability by pixel on overall level. Herefore we calculate and plot the standard deviation by pixel and make it even more understandable by visualizing the results. The code looks as follows:

```r
sds_image <- colSds(as.matrix(x))
qplot(sds_image, bins = "30", color = I("black"))
```

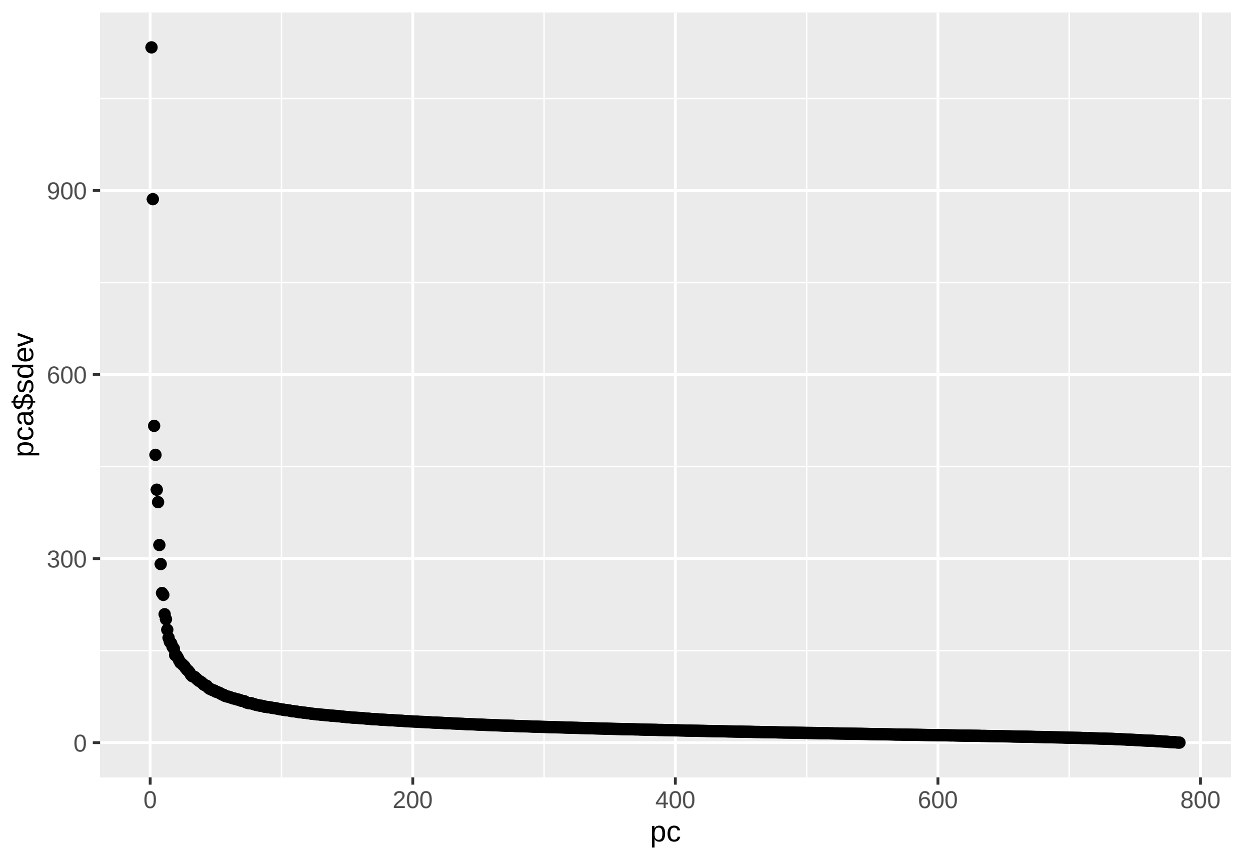


```r
image(1:28, 1:28, matrix(sds_image, 28, 28)[,28:1], asp = 1, main = "Sds by pixel")
```



Which are the most important components? With help of a principal component analysis we can find out. We perform the analysis and look at the results as a graph and by numbers:

```
# Principial Component Analysis - to reduce the model
pca <- prcomp(x)

# Plot
pc <- 1:ncol(x)
qplot(pc, pca$sdev)
```

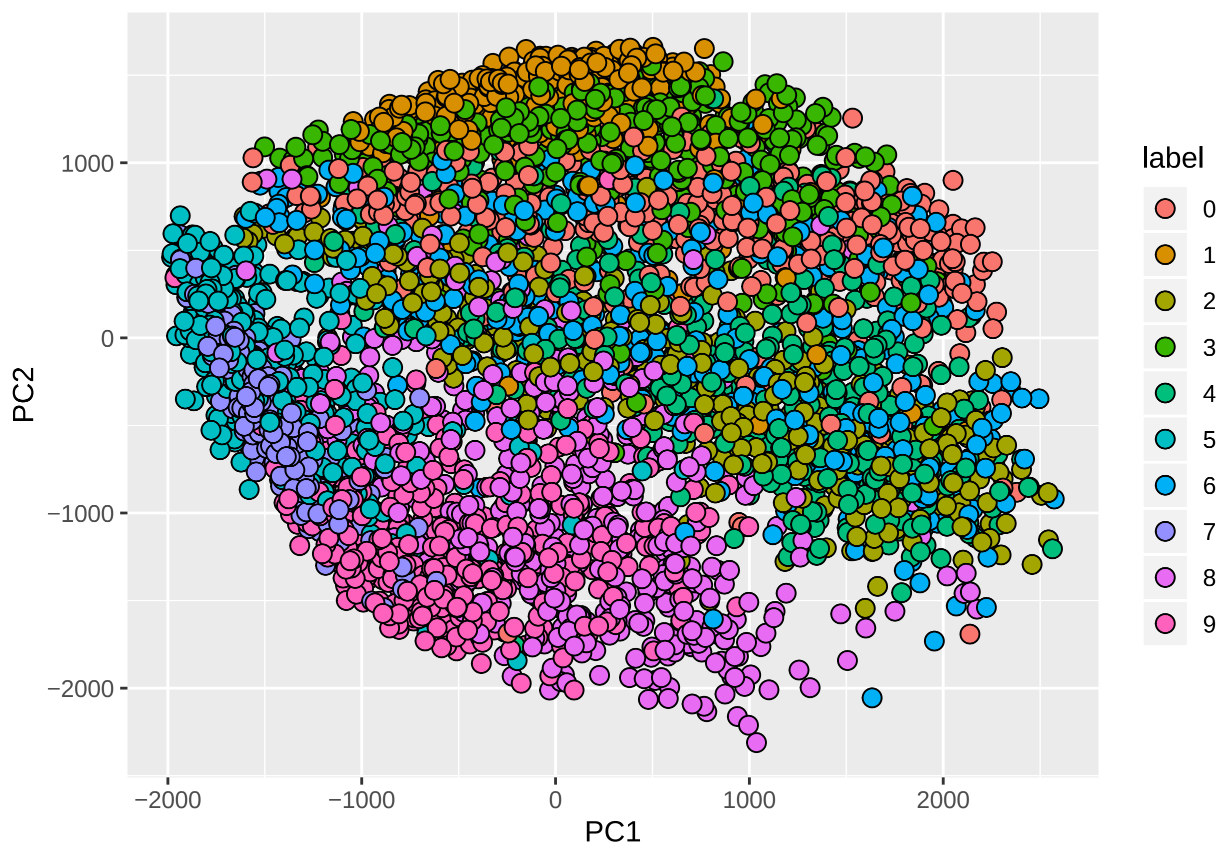


```
# Summary
summary(pca)$importance[,1:5]
```

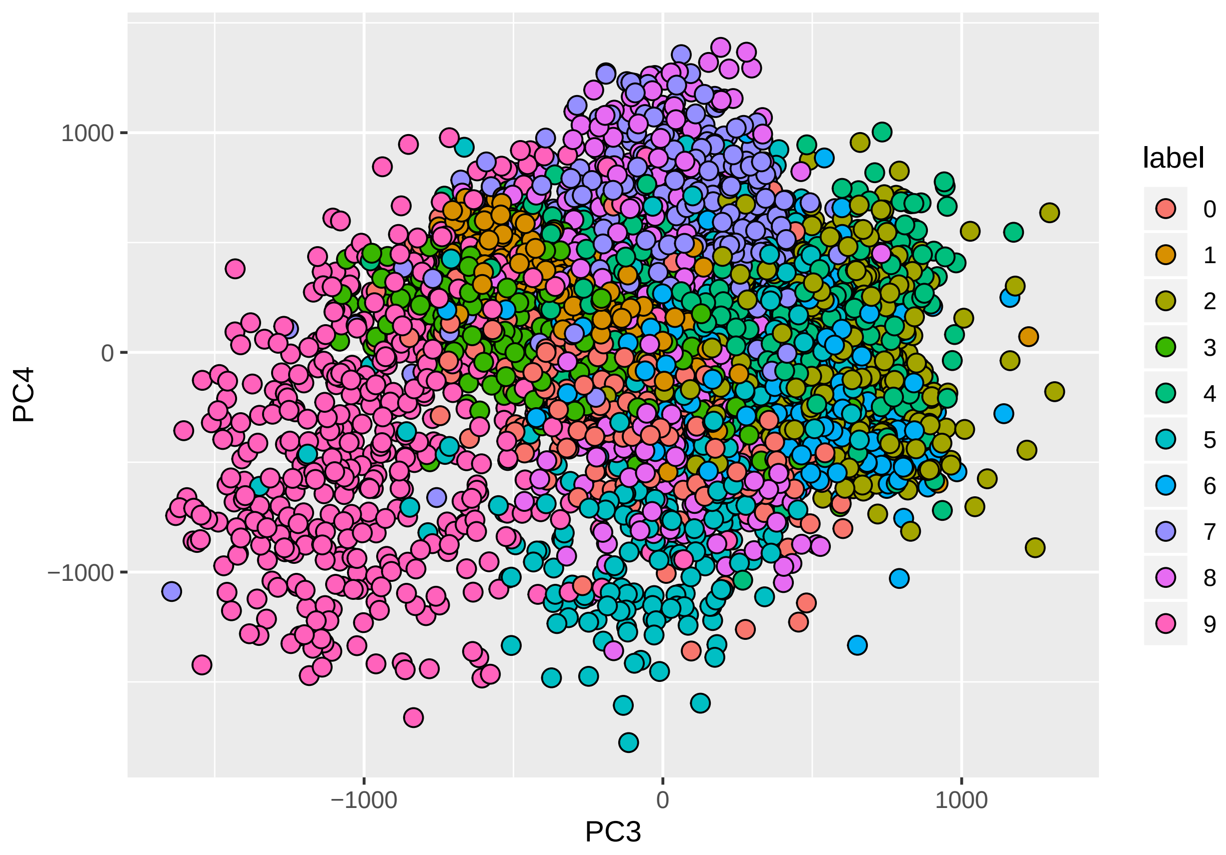
```
##                               PC1       PC2       PC3       PC4       PC5
## Standard deviation     1133.44508 886.01749 516.43074 469.14442 412.32709
## Proportion of Variance    0.29011   0.17728   0.06023   0.04970   0.03839
## Cumulative Proportion     0.29011   0.46739   0.52762   0.57732   0.61571
```

The following two plots make the results more visual. We plot the first two and the two components after seperatly to visually assess the grouping of the different classes by component.

```
# Plot the first two components, sample size 5000
data.frame(PC1 = pca$x[,1], PC2 = pca$x[,2],
        label=factor(modelling$label)) %>%
        sample_n(5000) %>%
        ggplot(aes(PC1, PC2, fill=label))+
        geom_point(cex=3, pch=21)
```



```
# Plot the next two components, sample size 5000
data.frame(PC3 = pca$x[,3], PC4 = pca$x[,4],
           label=factor(modelling$label)) %>%
  sample_n(5000) %>%
  ggplot(aes(PC3, PC4, fill=label))+
  geom_point(cex=3, pch=21)
```



So what can we conclude from the analysis? We can see that we can't set a threshold for the grayscale value to enhance the possibility for recognizing images, because the shapes in some categories are to similar.

What we can see is that we have a significant difference when looking at the color intensity for the images. Coats, pullovers and bags have much more "average grey-intensity" than trousers, sneakers and sandals. Ankle boats and coats have a high standard deviations, whereas shirts, sneakers and sandals significantly lower ones.

These metrics do not take the actual "pixels used" into consideration. Pullovers have the highest average non-black pixels, double as much as sandals. The intensity for the non-black pixels could also be an interesting aspect, because they create seperation in another area.

The principal component analysis gave us further insight. With just 14 of the 784 factors (or pixels) we can explain over 75 percent of the data. There seems to be room for reducing the model while using the more generic fitting methods.

We can use this "parameter based" approach first for build models and compare the results to other methodes which seem to be more suited for the given task like knn, experimenting with the number of factors used.

3. Modelling

As we discussed in the earlier section, we will test different methods starting with the very basic approach of taking the caluclated key metrics for setting up a model.

Model based on key metrics

We set up the model on key metrics:

```
# Calculating the key metrics for the training/modelling set
x_mod_key_metrics <- mod %>% mutate(pic_int = rowMeans(mod[,-1]),
                                    pic_sds = rowSds(as.matrix(mod[,-1])),
                                    pixels = rowSums(mod[,-1]>0),
                                    pixel_int = rowSums(mod[,-1])/
                                      rowSums(mod[,-1]>0)) %>%
  select(label, pic_int, pic_sds, pixels, pixel_int)

# Calculating the key metrics for the test set
x_test_key_metrics <- test %>% mutate(pic_int = rowMeans(test[,-1]),
                                    pic_sds = rowSds(as.matrix(test[,-1])),
                                    pixels = rowSums(test[,-1]>0),
                                    pixel_int = rowSums(test[,-1])/
                                      rowSums(test[,-1]>0)) %>%
  select(label, pic_int, pic_sds, pixels, pixel_int)

# Factorizing the labels
x_mod_key_metrics$label <- as.factor(x_mod_key_metrics$label)
x_test_key_metrics$label <- as.factor(x_test_key_metrics$label)

# Training the model
fit_key_met <- knn3(x_mod_key_metrics[,2:5], x_mod_key_metrics[,1])
# Predicting for the test set
y_hat_key_met <- predict(fit_key_met, x_test_key_metrics[,2:5], type = "class")
# Checking the accuracy
acc_key <- confusionMatrix(y_hat_key_met, factor(x_test_key_metrics[,1]))$overall["Accuracy"]
acc_key
```

```
## Accuracy
##   0.3854
```

With this very basic approach we get a very low accuracy, even lower than expected. It is even so low, that I will not try to further use these key metrics, but focus on the results of the principal component analysis to get better results.

K-nearest-neighbors

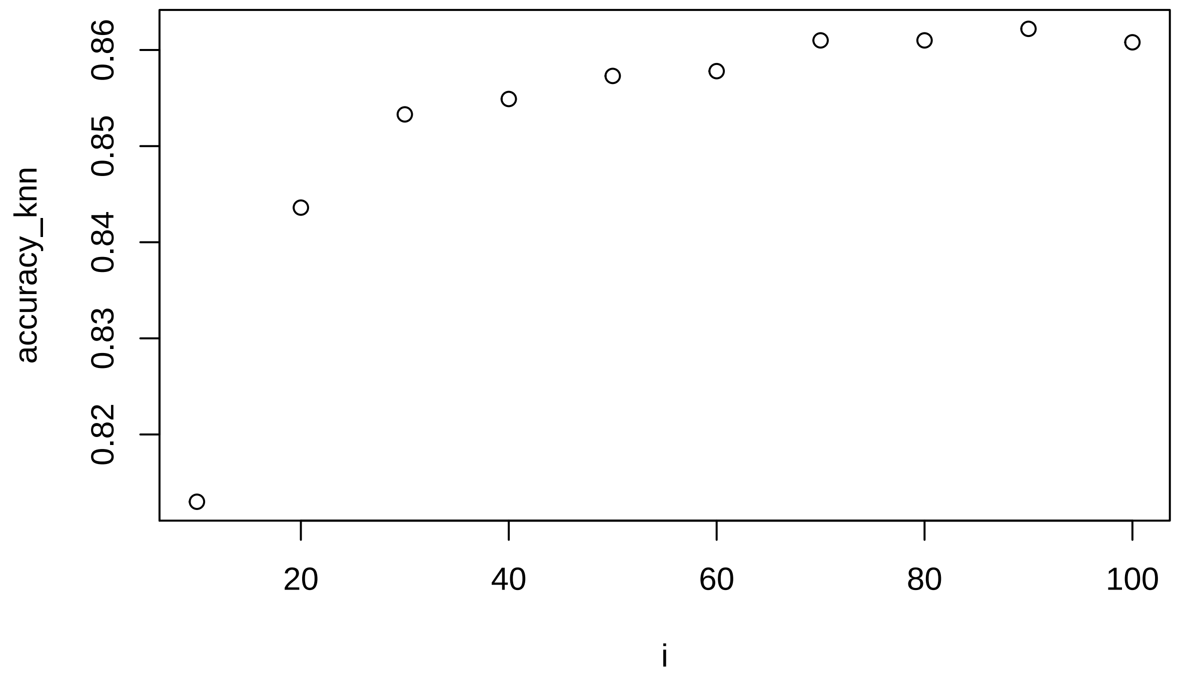
We will start with an approach to understand the relationship between number of components used and the accuracy we get. We will use a number of components in the range from 10 to 100 in steps of 10:

```
# PCA for the training model (before we used the whole data set)
pca_mod <- prcomp(x_mod)
col_means <- colMeans(x_test)

# Prepare the test set to conduct the same components as the training set
x_test_knn3 <- sweep(as.matrix(x_test), 2, col_means) %*% pca_mod$rotation


# Vector with components used
i <- seq(10, 100, 10)

# Sapply for all the values in the vector i
```

```
accuracy_knn <- sapply(i, function(k){

  # Reduce the training and test set
  x_train <- pca_mod$x[,1:k]
  x_test_knn3 <- x_test_knn3[,1:k]

  # Train the model with knn3
  fit_knn3 <- knn3(x_train, y_mod)

  # Predict for the test set
  y_hat_knn3 <- predict(fit_knn3, x_test_knn3, type = "class")

  # Print the overall accuracy and the differences by class
  return(confusionMatrix(y_hat_knn3, factor(y_test))$overall["Accuracy"])
}
)

# Table of results
cbind(i, accuracy_knn)
```

```
##            i accuracy_knn
## Accuracy  10       0.8130
## Accuracy  20       0.8436
## Accuracy  30       0.8533
## Accuracy  40       0.8549
## Accuracy  50       0.8573
## Accuracy  60       0.8578
## Accuracy  70       0.8610
## Accuracy  80       0.8610
## Accuracy  90       0.8622
## Accuracy 100       0.8608
```

```
# Plot accuracy vs components used
plot(i, accuracy_knn)
```



As we can see, the accuracy does not increase significantly after 30 components, we even can see some dips in the curve. Lets' take a how the accuracy is spread accross the different categories. Were are we accurate and where do we have problems? Herefore we use the best components number from the previous calculation and take a deeper look at the confusion matrix:

```
k <- i[which.max(accuracy_knn)]

x_train <- pca_mod$x[,1:k]

# Prepare the test set to conduct the same components as the training set
x_test_knn3 <- sweep(as.matrix(x_test), 2, col_means) %*% pca_mod$rotation
x_test_knn3 <- x_test_knn3[,1:k]

# Train the model with knn3
fit_knn3 <- knn3(x_train, y_mod)

# Predict for the test set
y_hat_knn3 <- predict(fit_knn3, x_test_knn3, type = "class")

# Print the overall accuracy and the differences by class
acc_knn3 <- confusionMatrix(y_hat_knn3, factor(y_test))$overall["Accuracy"]
acc_knn3
```

```
## Accuracy
##   0.8615
```

```r
cbind(fashion_cat, confusionMatrix(y_hat_knn3, factor(y_test))$byClass[,11])
```

```
##          fashion_cat
## Class: 0 "T-shirt/top" "0.913571426906242"
## Class: 1 "Trouser"     "0.98739649847725"
## Class: 2 "Pullover"    "0.868301582465159"
## Class: 3 "Dress"       "0.937366396550854"
## Class: 4 "Coat"        "0.863526489721302"
## Class: 5 "Sandal"      "0.947565283120792"
## Class: 6 "Shirt"       "0.78915590397808"
## Class: 7 "Sneaker"     "0.965972262162052"
## Class: 8 "Bag"         "0.980900063498068"
## Class: 9 "Ankle boot"  "0.974273428232503"
```

As it seems, we are having the most problems for the categories pullover, coat and shirt.

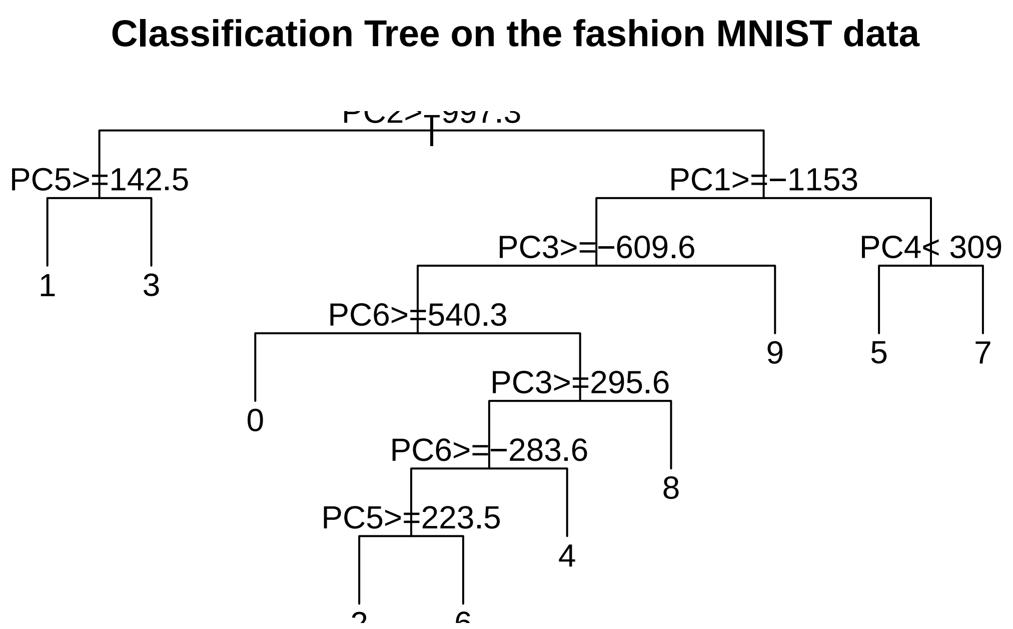
Classification Trees

Does a classification tree perform better? Let's try it using the "rpart" function:

```r
# Take the most important 9 predictors for the 10 categories
k <- 9

# Create the training set
x_train <- pca_mod$x[,1:k]
x_train <- data.frame(y_mod, x_train)

# Create the test set
x_test_rpart <- sweep(as.matrix(x_test), 2, col_means) %*% pca_mod$rotation
x_test_rpart <- data.frame(x_test_rpart[,1:k])

# Fit the model
r_fit = rpart(y_mod ~ ., data = data.frame(x_train), method = "class")

# Predict
y_hat_rpart <- predict(r_fit, data.frame(x_test_rpart), type = "class")

# Accuracy
acc_rpart <- confusionMatrix(y_hat_rpart, y_test)$overall["Accuracy"]
acc_rpart
```

```
## Accuracy
##   0.6014
```

```r
# Plot the tree
plot(r_fit, uniform = TRUE, main = "Classification Tree on the fashion MNIST data")
text(r_fit)
```



This method does not help improve accuracy, we are way to basic here. Nevertheless, clustering seems to be a good approach, so let's try random forests next.

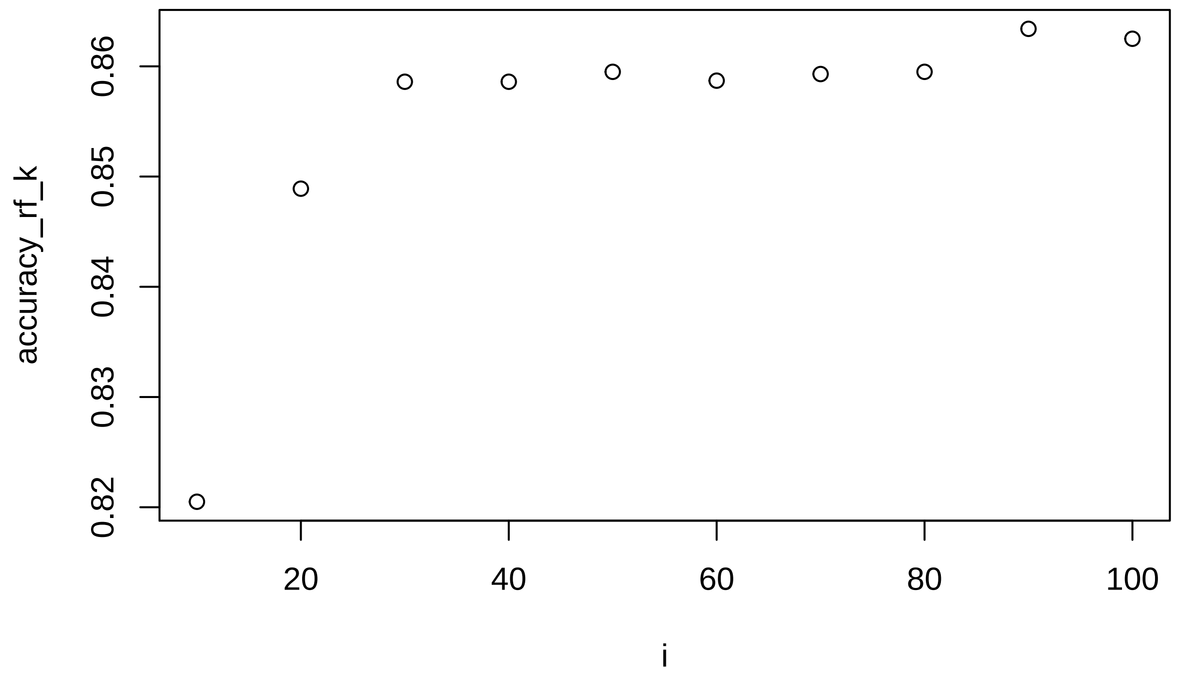
Random Forests

First we will take a look into how the dependency between accuracy and components used unfolds. Herefore we add more and more components in order of the importance given by the prinicpal component analysis:

```r
# From 10 to 100 in steps of 10
i <- seq(10, 100, 10)

# Sapply for the vector i
accuracy_rf_k <- sapply(i, function(k){

  # Create the training set
  x_train <- pca_mod$x[,1:k]
  x_train <- data.frame(y_mod, x_train)

  # Create the test set
  x_test_rpart <- sweep(as.matrix(x_test), 2, col_means) %*% pca_mod$rotation
  x_test_rpart <- data.frame(x_test_rpart[,1:k])

  # Model building
  rf <- randomForest(
       y_mod ~ .,
       data = x_train,
       ntree = 50
  )

  # Prediction on the test set
  y_hat_rf <- predict(rf, data.frame(x_test_rpart), type = "class")

  # Return the accuracy
  return(confusionMatrix(y_hat_rf, y_test)$overall["Accuracy"])
}
)

# Get the accuracy numbers
cbind(i, accuracy_rf_k)
```

```
##            i accuracy_rf_k
## Accuracy  10        0.8205
## Accuracy  20        0.8489
## Accuracy  30        0.8586
## Accuracy  40        0.8586
## Accuracy  50        0.8595
## Accuracy  60        0.8587
## Accuracy  70        0.8593
## Accuracy  80        0.8595
## Accuracy  90        0.8634
## Accuracy 100        0.8625
```

```
# Plot the diagram
plot(i, accuracy_rf_k)
```

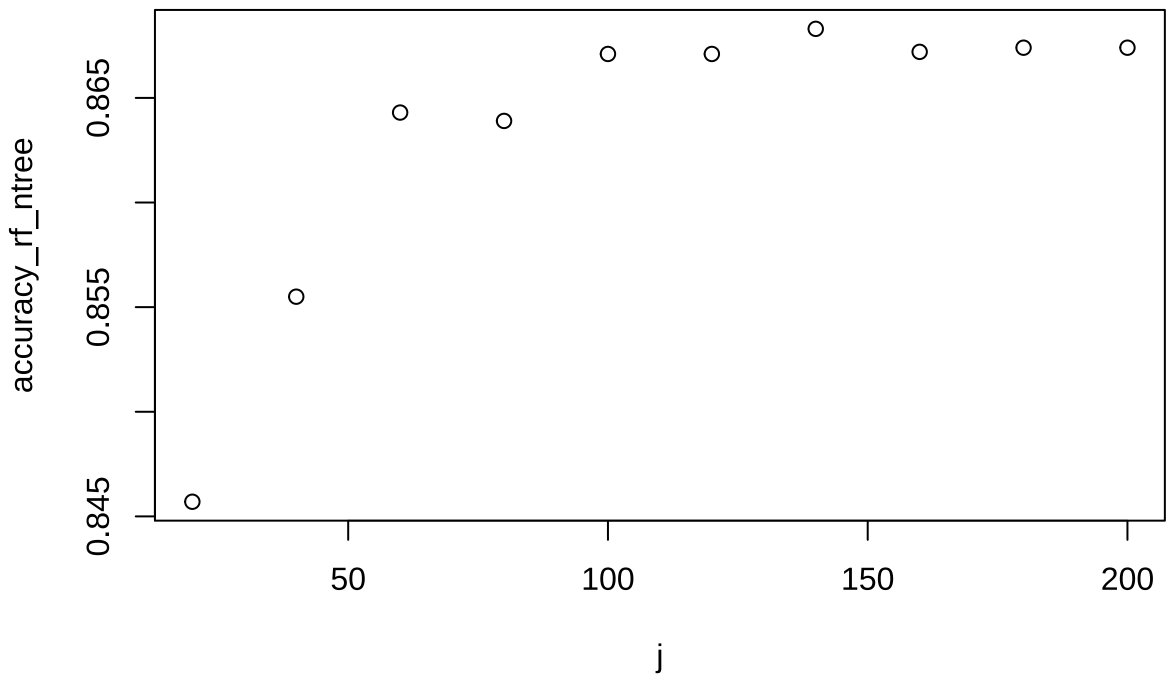


In a second step we vary the number of trees for the model:

```
# Vector with optimization values
j <- seq(20, 200, 20)

# Take the best number of components from the previous optimization
k <- i[which.max(accuracy_rf_k)]

# Sapply for the vector j
accuracy_rf_ntree <- sapply(j, function(n){

  # Create the training set
  x_train <- pca_mod$x[,1:k]
  x_train <- data.frame(y_mod, x_train)

  # Create the test set
  x_test_rpart <- sweep(as.matrix(x_test), 2, col_means) %*% pca_mod$rotation
  x_test_rpart <- data.frame(x_test_rpart[,1:k])

  # Model building
  rf <- randomForest(
    y_mod ~ .,
    data = x_train,
    ntree = n
  )
  # Prediction on the test set
  y_hat_rf <- predict(rf, data.frame(x_test_rpart), type = "class")

  # Return the accuracy
  return(confusionMatrix(y_hat_rf, y_test)$overall["Accuracy"])
}
)

# Get the accuracy numbers
cbind(j, accuracy_rf_ntree)
```

```
##            j accuracy_rf_ntree
## Accuracy  20            0.8457
## Accuracy  40            0.8555
## Accuracy  60            0.8643
## Accuracy  80            0.8639
## Accuracy 100            0.8671
## Accuracy 120            0.8671
## Accuracy 140            0.8683
## Accuracy 160            0.8672
## Accuracy 180            0.8674
## Accuracy 200            0.8674
```

```
acc_rf <- max(accuracy_rf_ntree)

# Plot the diagram
plot(j, accuracy_rf_ntree)
```



Random forests look good, especially when tuning the parameters we get a good accuracy. Let's dive into the results.

4. Results

Here we have a summary of the results:

Table 1: Accuracy of the models

Method	Accuracy
Key Metrics	0.3854
Knn3	0.8615
RPart	0.6014
RandomForest	0.8683

As we can see random forests and k-nearest neighbors provide the highest accuracy, with random forests in our case being slightly better. The models are build on very different approaches.

The k-nearest-neighbors method works by using "k" of the nearest neighbors for classification. Small k values lead to noise which is negatively impacting the results. If the k values get too big, the problem is that datapoint with a too big distance are taken into consideration which might not be related to the point predicted. We can see for our model, that the largest k-value of 100 does not produce the highest accuracy.

As we could see for the rpart model, simple classification does not lead to sufficient results. If we however combine different descision trees within the random forest models we get a much better accuracy with a relatively moderate computation time. Here we also saw that more components and more trees do not necessarily leed to better results.

The main problem of all methods is to distinguish between pullovers, coats and shirts. These images as we saw in the data analysis are even hard to categorize as a human. Therefore an accuracy of over 86 percent for two of the models is a satisfactory result in my opinion.

As a last step we run the full models through the analysis and use the validation model for our final results. First the final accuracy of the k-nearest-neighbor method:

```r
# Principal component analysis on the full set
pca <- prcomp(x)
# Mean values for the columns of the verification set
col_means <- colMeans(x_verification)

# Take the best k value from the previous excerise
k <- i[which.max(accuracy_knn)]

# Build the train set
x_train <- pca$x[,1:k]

# Prepare the test set to conduct the same components as the training set
x_ver_knn3 <- sweep(as.matrix(x_verification), 2, col_means) %*% pca$rotation
x_ver_knn3 <- x_ver_knn3[,1:k]

# Train the model with knn3
fit_knn3 <- knn3(x_train, y)

# Predict for the test set
y_hat_knn3 <- predict(fit_knn3, x_ver_knn3, type = "class")

# Print the overall accuracy
acc_ver_knn <- confusionMatrix(y_hat_knn3, factor(y_verification))$overall["Accuracy"]
acc_ver_knn
```

```
## Accuracy
##   0.8641
```

And afterwards the random forest method:

```r
# Take the best parameters from the model building chapter
k <- i[which.max(accuracy_rf_k)]
n <- j[which.max(accuracy_rf_ntree)]

# Create the training set
x_train <- pca$x[,1:k]
x_train <- data.frame(y, x_train)

# Create the test set
x_ver_rpart <- sweep(as.matrix(x_verification), 2, col_means) %*% pca$rotation
x_ver_rpart <- data.frame(x_ver_rpart[,1:k])

# Model building
rf <- randomForest(
  y ~ .,
  data = x_train,
  ntree = n
)

# Prediction on the test set
y_hat_rf <- predict(rf, data.frame(x_ver_rpart), type = "class")

# Print the accuracy
acc_ver_rf <- confusionMatrix(y_hat_rf, y_verification)$overall["Accuracy"]
acc_ver_rf
```

```
## Accuracy
##   0.8676
```

The accuracies on the verification model are with 0.8641 for knn and 0.8676 for random forest very similar to what we experienced during the model building exercise.

5. Conclusion, limitations and comments

During this project we have taken a deeper look into the classification of images with help of different methods. We saw that two very different approaches led to very similar accuracies and that we had reoccuring problems with distigushing between certain fashion categories.

In my opinion the accuracy of the prediction can be improved by two different ways. First it can be tried to pre-process the images in order to get a better distinction between the aforementioned fashion categories. For example one idea would be to just take the borders and try to predict based on the pure shape of the objects. Deep learning methods like neural networks would be the other part I could think of. This is something I would like to have tried, but to be honest therefore I frist need to understand them, which I will try to do by taking additional courses.

Overall I have learned a lot during all the courses in this series and in particular during this project. I am really amazed how far we have traveled, through the basics of R, visualization and probability theory. During the later courses we have got to know some tools and methods which we all combined during the machine learning course which was one of my personal highlights. The two part capstone exam is great way to really dive into all what has been taught on your own and apply structure and methods to complete projects.

Hopefully you enjoyed this report and the whole program as much as I did and we will see each other in other courses, further exploring the world of data science. All the data you can find **here**.