ML - Project

“Android Gesture Recognition”

MC 520 Machine Learning

Rainhard Findling MSc

1710455004 - Christopher Ebner

1710455011 - Daniel Öttl

1710455014 - David Röbl

Mobile Computing Master

January 8, 2018

Goal

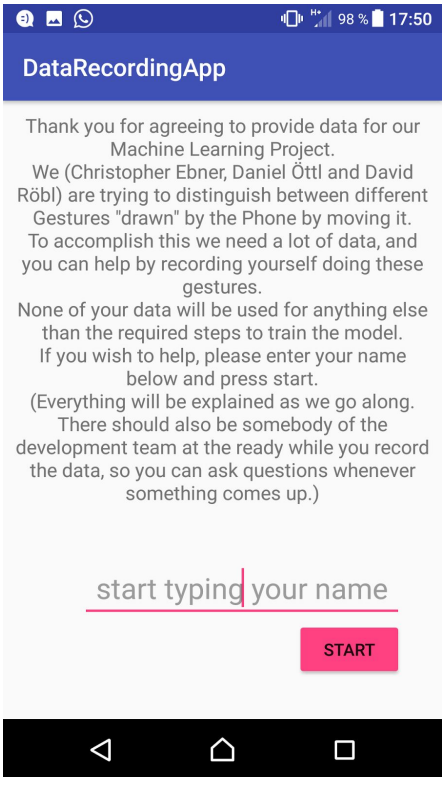
The main idea is to recognise moving gestures from smartphone users. More detailed, we want to classify gestures like shaking or drawing a circle with the phone. Based on those gestures an app could theoretically react to them, like swiping to the right plays the next song for example.

The App

We developed an Android App to record the training data. The app was designed to be as easy to use as possible (but still has many areas that could be improved as it is only a working prototype). It tries to explain to the user everything they have to do and make the recording experience as fault-proof as possible. Every step is explained along the way and every gesture is first described and a pictogram depicting the gesture is shown.
For each sensor value a dedicated .csv file is created. The same lines in all .csv files belong together, compiling a full dataset. To make this more fault-tolerant every .csv file also stores the username, device-id, recorded gesture and the number of the try, along with all values received from the sensor. This results in a different number of features for every recording.

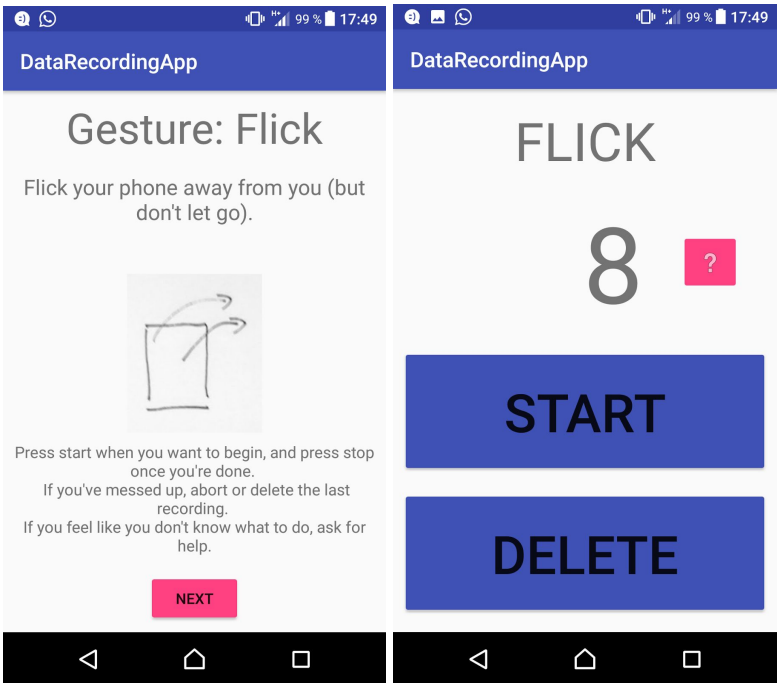
Entry

On startup of the application, the user is shown a text thanking them for participating in our data recording and then asked to enter their name. Thus we can attribute the recordings to single users and eventually train a population-independent model.



Explanation, Guidance

The user is guided through the app via explanation screens, with tell them what they have to do next. E.g. a gesture is explained with text and image before the user can first record the gesture. If they are unsure which gesture should be recorded, the can re-access the explanation screen via a help button.



Data Recording

Every gesture is recorded by the user. The user is first told how to record the gesture. On the recording screen they can then press [start] to start the recording, and after transcribing the gesture with the phone press [stop]. The advantage of this setup is that we do not have to trim the recording before we can use it. If the user messes up a recording they can either abort the recording (if it is still in progress) or delete the last made recording.

Data

For the dataset we made an own app which guides the user through a process where the user should make the gestures we want to classify. We recorded multiple sensor values, but only used the accelerometer recordings, since we thought that other sensor values would not add more information and the accelerometer data should be enough.

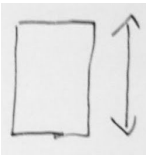
Gestures

In total the model should be able to classify six different gestures. Therefore the recording app asks the users to do the following gestures.

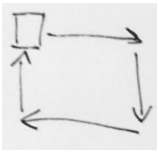
- **Circle**

- **Flick**

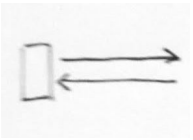
- **Shake**



- **Square**



- **Swipe**



- **Triangle**

Characteristics

The table below states some facts about the dataset:

Gestures	6 (15 times recorded per person)
Recorded persons	9
Samples	809 (should be 810, maybe a recording error) 808 after basic preprocessing (one too short sample was filtered)
Balanced	“Yes”, it is **nearly balanced** due to two missing/filtered samples 135 x CIRCLE 135 x FLICK **133** x SHAKE 135 x SQUARE 135 x SWIPE 135 x TRIANGLE
Recording frequency	20Hz (requested recording frequency in the recording app)

Note

We experienced that some gestures were different made by the different persons. Two examples are “shake” and “swipe”. Some did the shake gesture quite slow and just two times up and down while others were really shaking the phone.
Furthermore a misinterpretation of the swipe gesture within the project team resulted actually in two different gestures. Some did like “start” -> swipe to right -> “stop” while others made it like “start” -> swipe to right -> swipe back to start position -> “stop”.

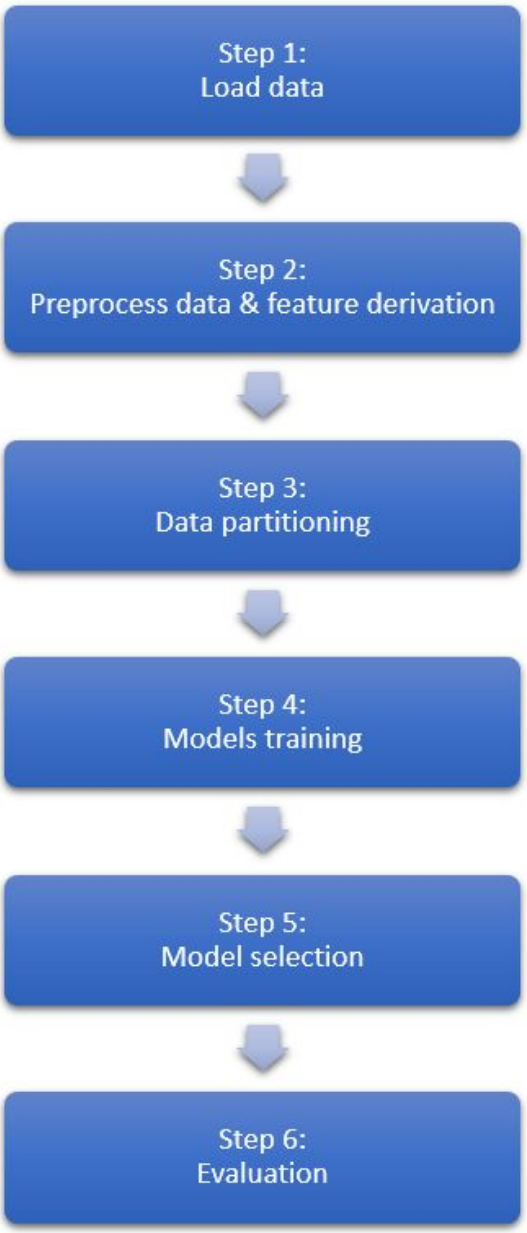
In addition, different directions (left, right, clockwise, anticlockwise, …) were used to make the gestures.

The recording app is also included in the zip archive.

Approach

Since we want to recognize gestures independent how the mobile device is held we decided to focus on calculated magnitude values based on the three axes. Furthermore we wanted to recognize gestures no matter which direction (left - right, clockwise - anticlockwise). Thus we decided to not take the recorded sensor values as features directly, but to derive features from the recordings which should be robust against that.

Our general approach step by step looks like following:



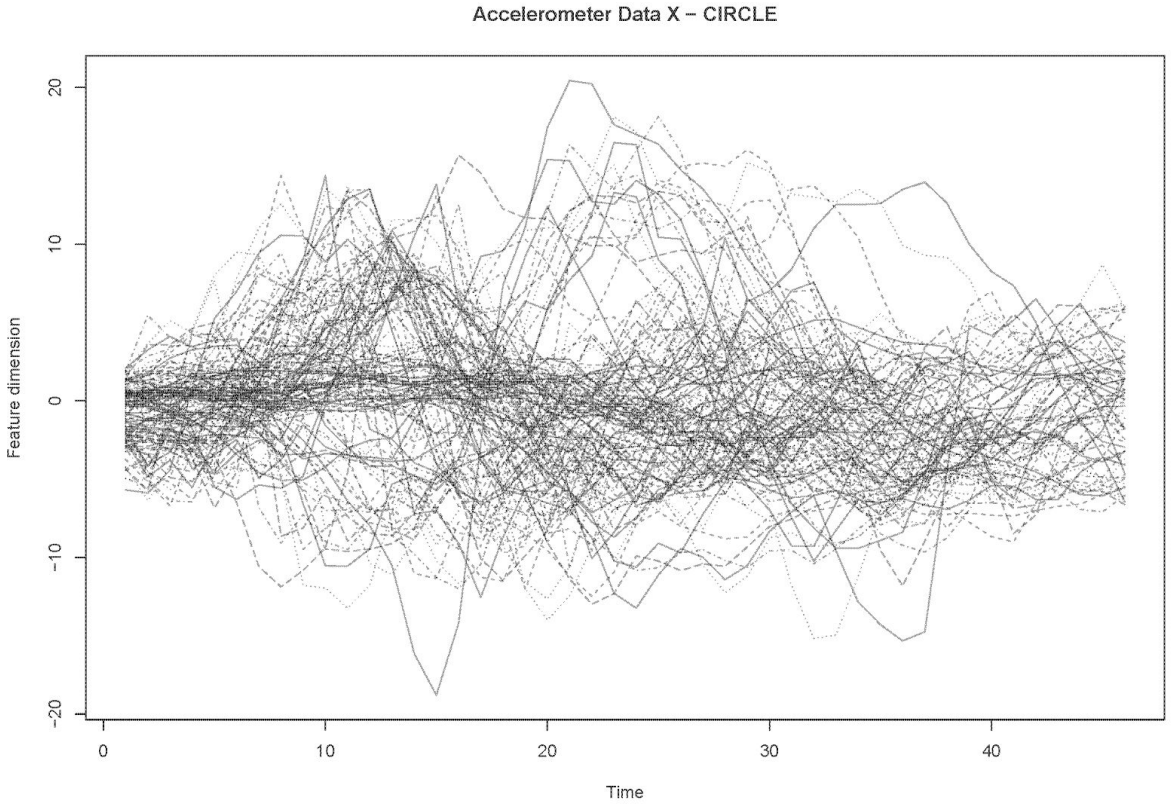
Step 1: Loading Data

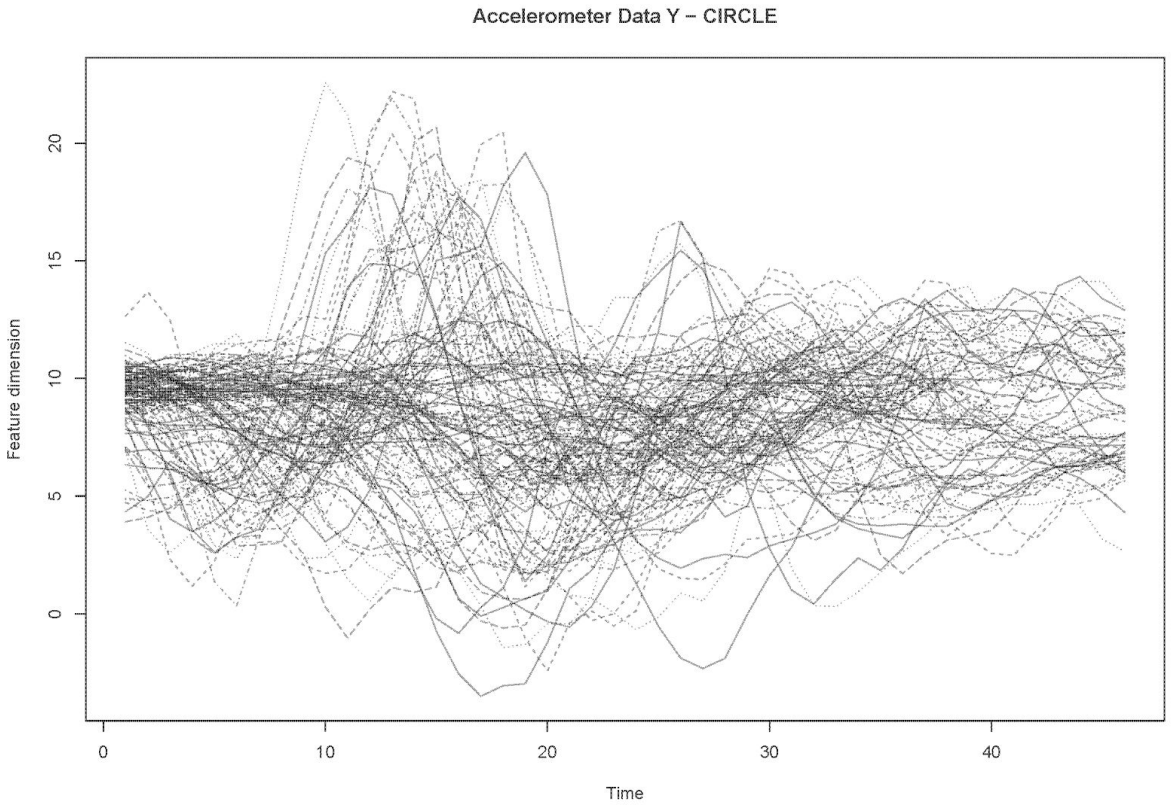
The data is stored in simple csv files (one file per axis), which contains the gesture, a person ID, sample nr of gesture for the person, android device ID and the sensor values as series. Without further preprocessing each sensor recording value would be a feature.

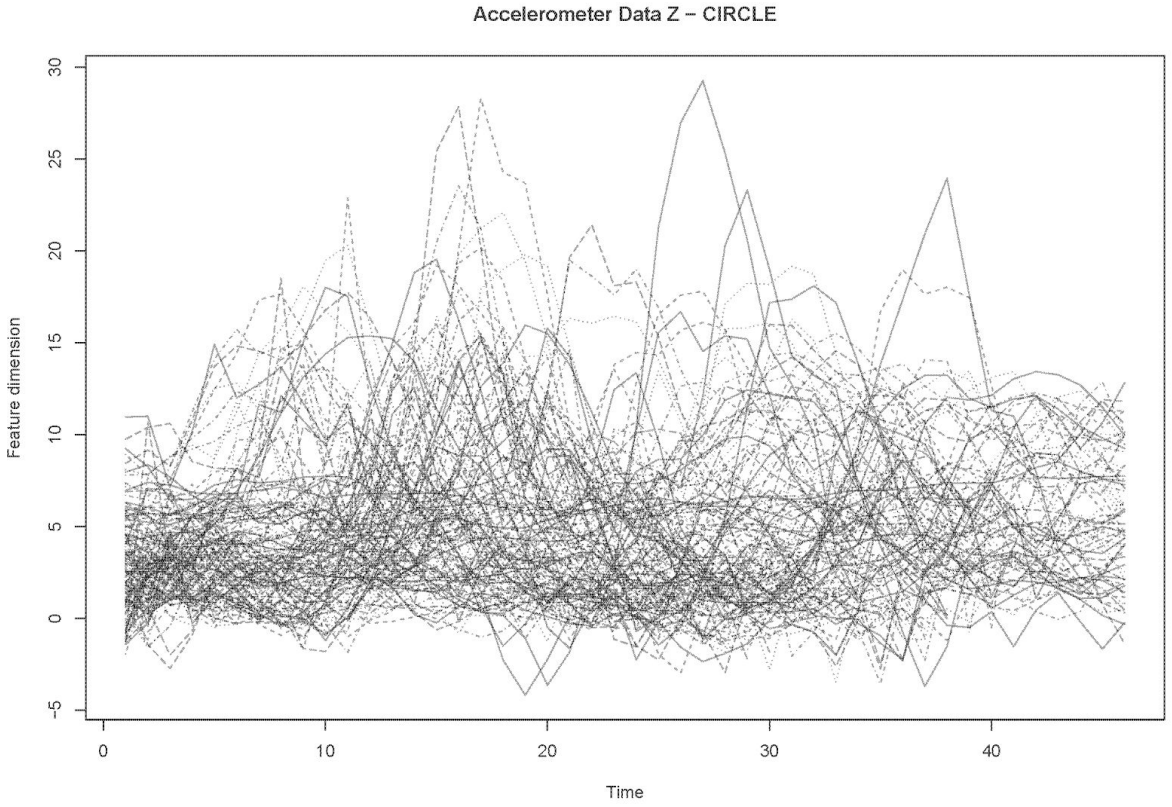
The raw data for the gestures “circle” and “flick” is visualized below. Since we have recordings for each axle, three plots per gesture are shown in this step.

Each plot stated in this report can be found in the “`plots`” directory as well.

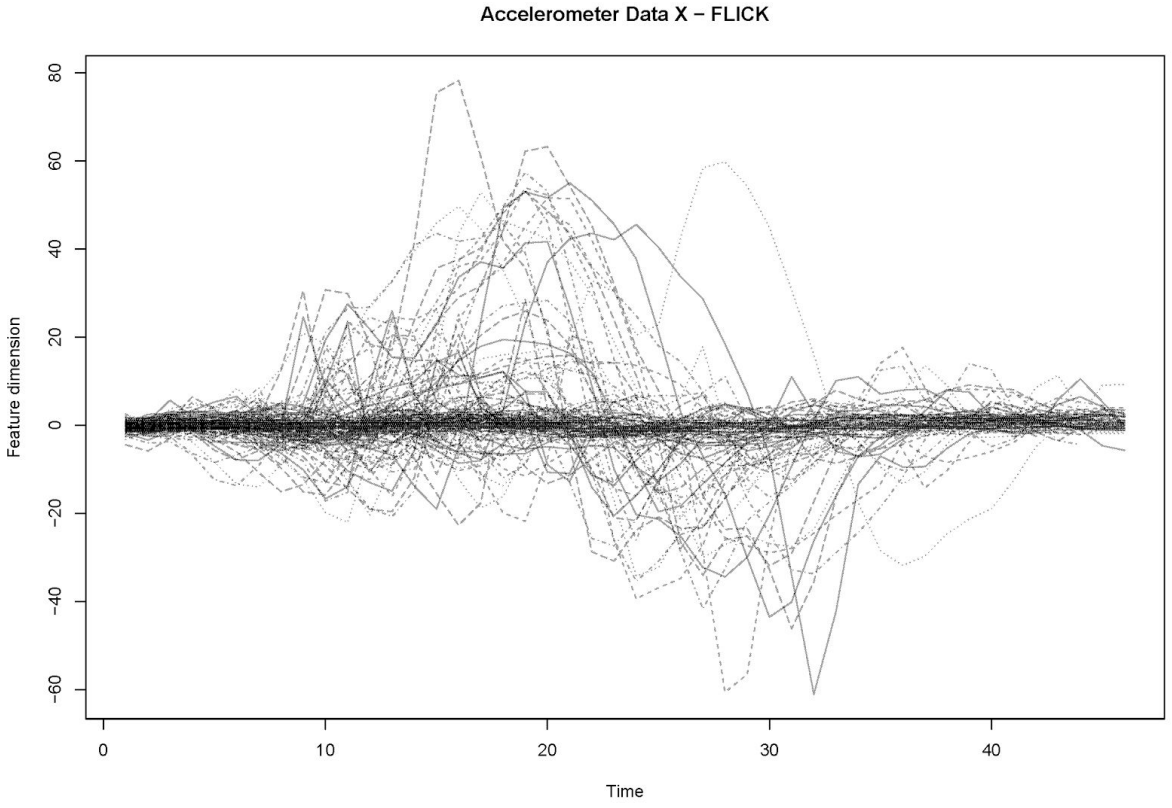
Raw “circle”

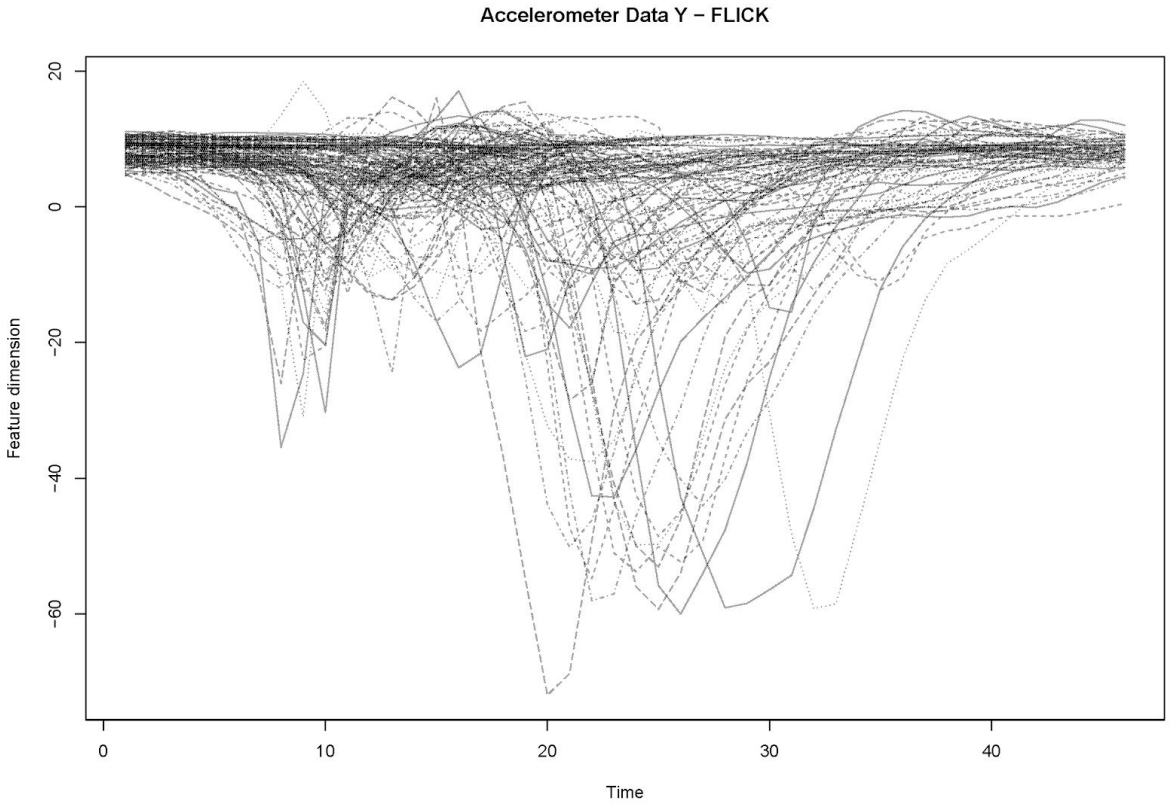


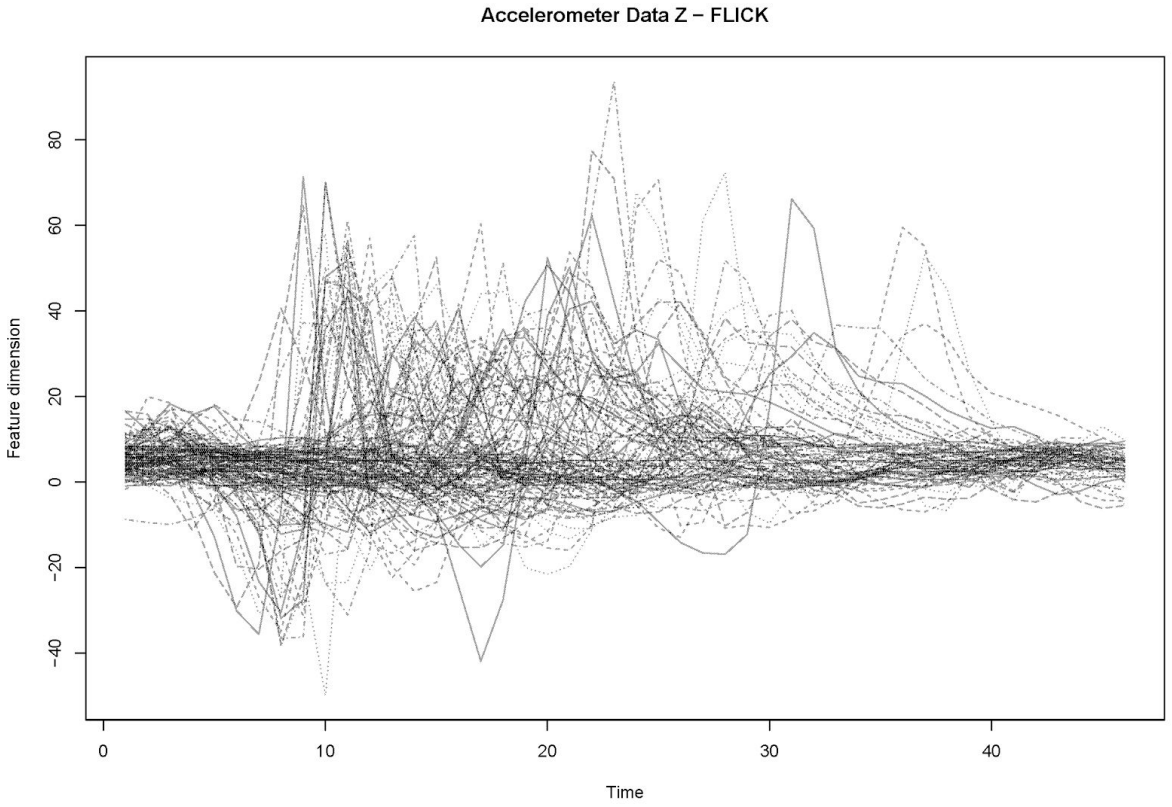

Accelerometer Data Y – CIRCLE
Feature dimension
Time
0
5
10
15
20
0
10
20
30
40


Accelerometer Data Z – CIRCLE
Feature dimension
Time
-5
0
5
10
15
20
25
30
0
10
20
30
40

Raw “flick”


Accelerometer Data X − FLICK
Feature dimension
80
60
40
20
0
−20
−40
−60
0
10
20
30
40
Time


Accelerometer Data Y − FLICK
Feature dimension
20
0
−20
−40
−60
0
10
20
30
40
Time



Step 2: Preprocess data & feature derivation

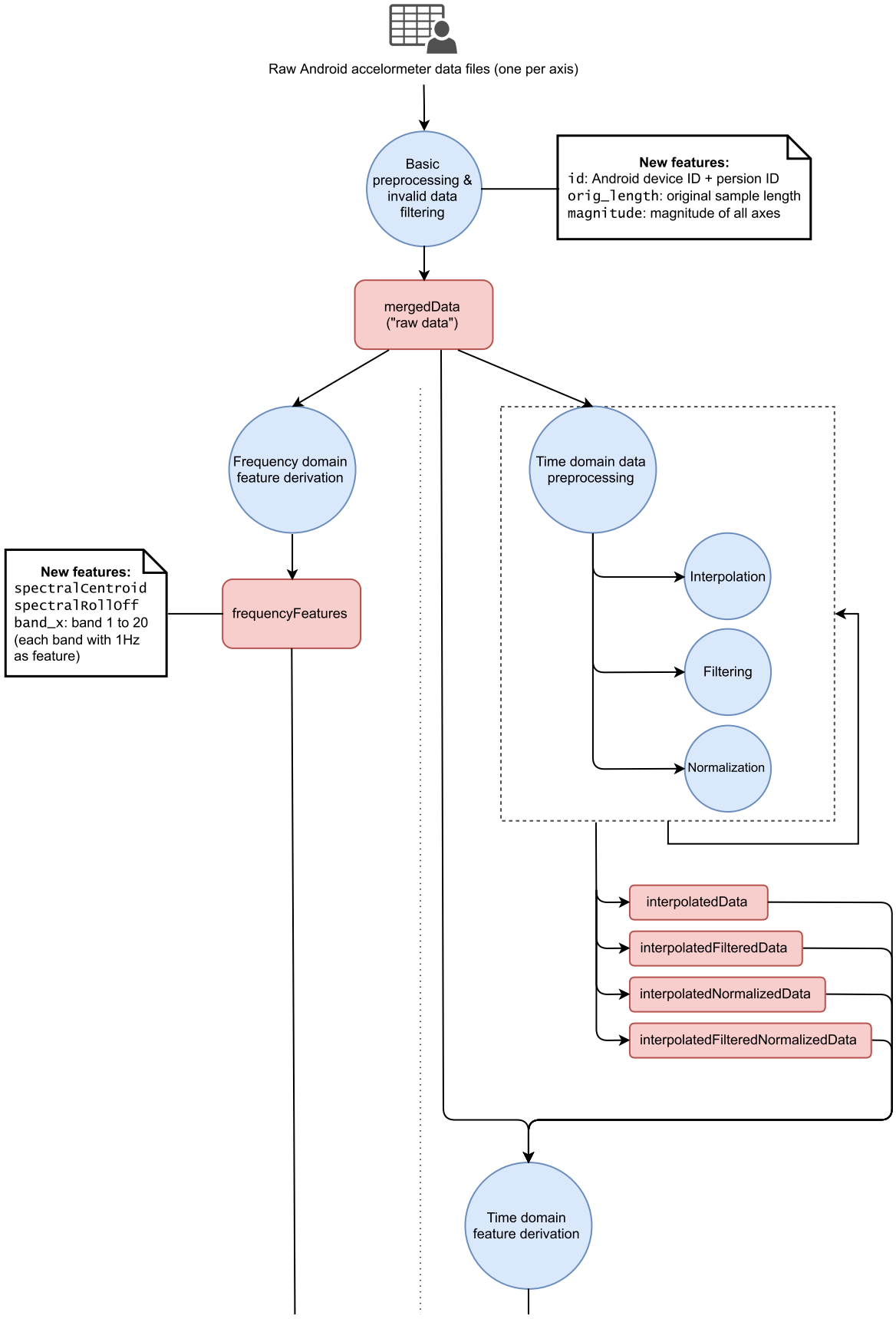
2.0 Approach

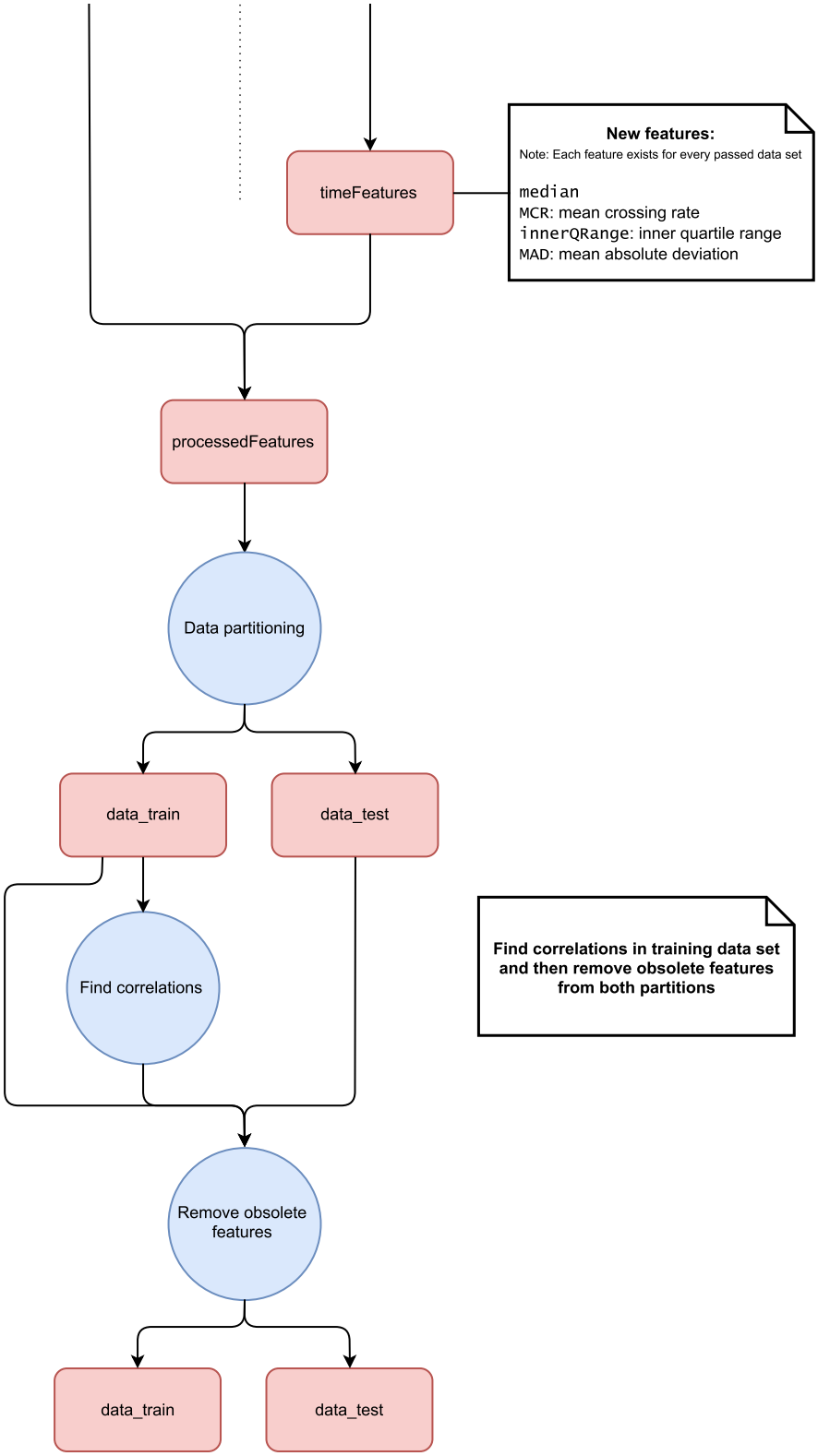
Since the data preprocessing and feature derivation took the most time, an own approach explanation for this step follows. The basic idea to do some basic data preparation and then derivate features from the time as well from the frequency domain.

The next two pages shows the detailed workflow for this step.

At first, basic preprocessing and feature derivation merges the three files to one single data frame and already creates some features. After that, the data is passed to two pipes, one for frequency domain and the other one for time domain feature derivation.
In the time domain multiple datasets are prepared in order to check which preprocessing steps make sense in our case, and which do not.

Finally all features are merged again to one dataset and obsolete features or features which do not add information are removed. Since the feature selection should be based on the training data, and not on all data, the data partitioning is already done here as well.


Raw Android accelormeter data files (one per axis)
Basic preprocessing & invalid data filtering
New features:
id: Android device ID + persion ID
orig_length: original sample length
magnitude: magnitude of all axes
mergedData ("raw data")
Frequency domain feature derivation
Time domain data preprocessing
Interpolation
Filtering
Normalization
New features:
spectralCentroid
spectralRollOff
band_x: band 1 to 20 (each band with 1Hz as feature)
frequencyFeatures
interpolatedData
interpolatedFilteredData
interpolatedNormalizedData
interpolatedFilteredNormalizedData
Time domain feature derivation


timeFeatures
New features:
Note: Each feature exists for every passed data set
median
MCR: mean crossing rate
innerQRange: inner quartile range
MAD: mean absolute deviation
processedFeatures
Data partitioning
data_train
data_test
Find correlations
Find correlations in training data set and then remove obsolete features from both partitions
Remove obsolete features
data_train
data_test

2.1 Basic preprocessing

Since the model should work with the magnitude, and not all three axis values as own feature set, the three files can be “merged” into one. In order to accomplish that, the magnitude values are calculated based on the three axles.

The magnitude calculation function just simply squares each axle value, sums them and takes the square root, like shown below.

Value: Magnitude
Params: Sensor value for each axle
Code:

```
01:    calcMagnitude <- function(x,y,z)
02:    {
03:           sqrt(x**2 + y**2 + z**2)
04:    }
```

As next step the device ID and the person ID is merged to a single column, since the combination is used for identifying a person, which will be necessary for a population independent data partitioning. The device ID was also recorded so that the samples can be distinguished later if two persons with the same name (or what was used for the ID) have been recorded on two different devices (Example: Two Thomas have been recorded, one on device x and the other on device y).

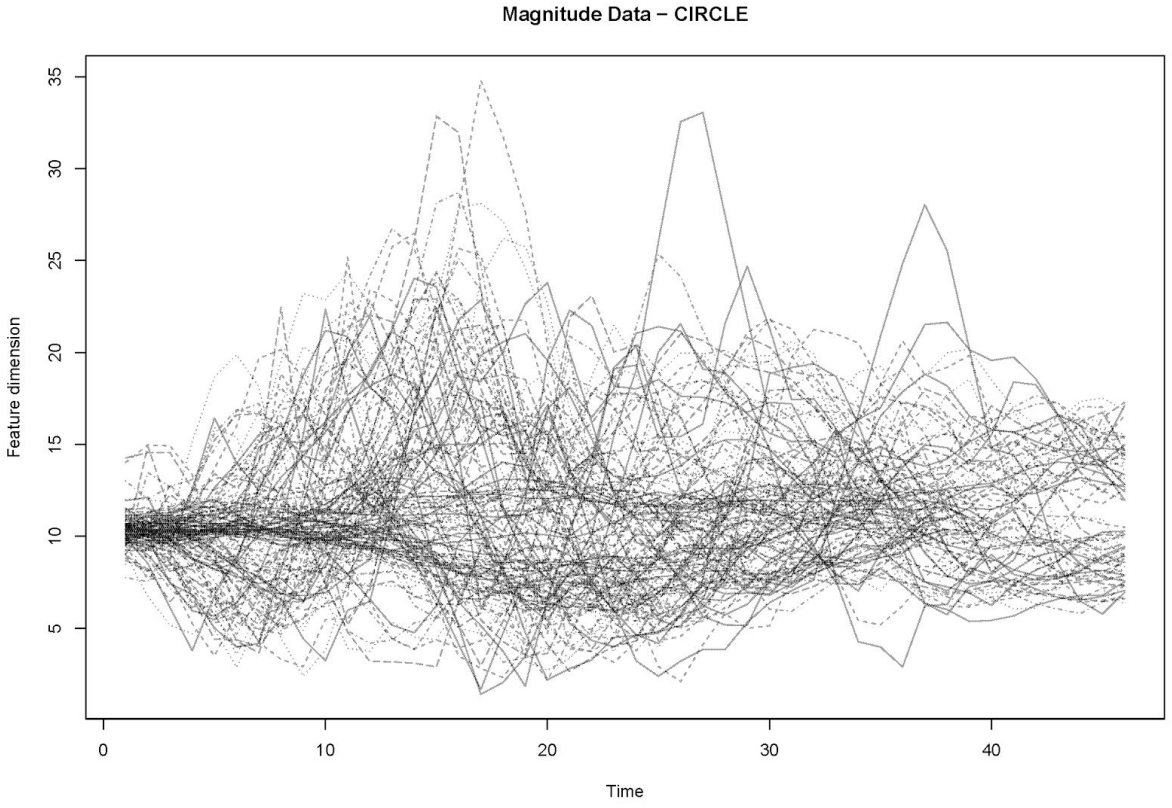
Furthermore the original sensor series length is extracted as own feature, since this information will be lost during the interpolation process.

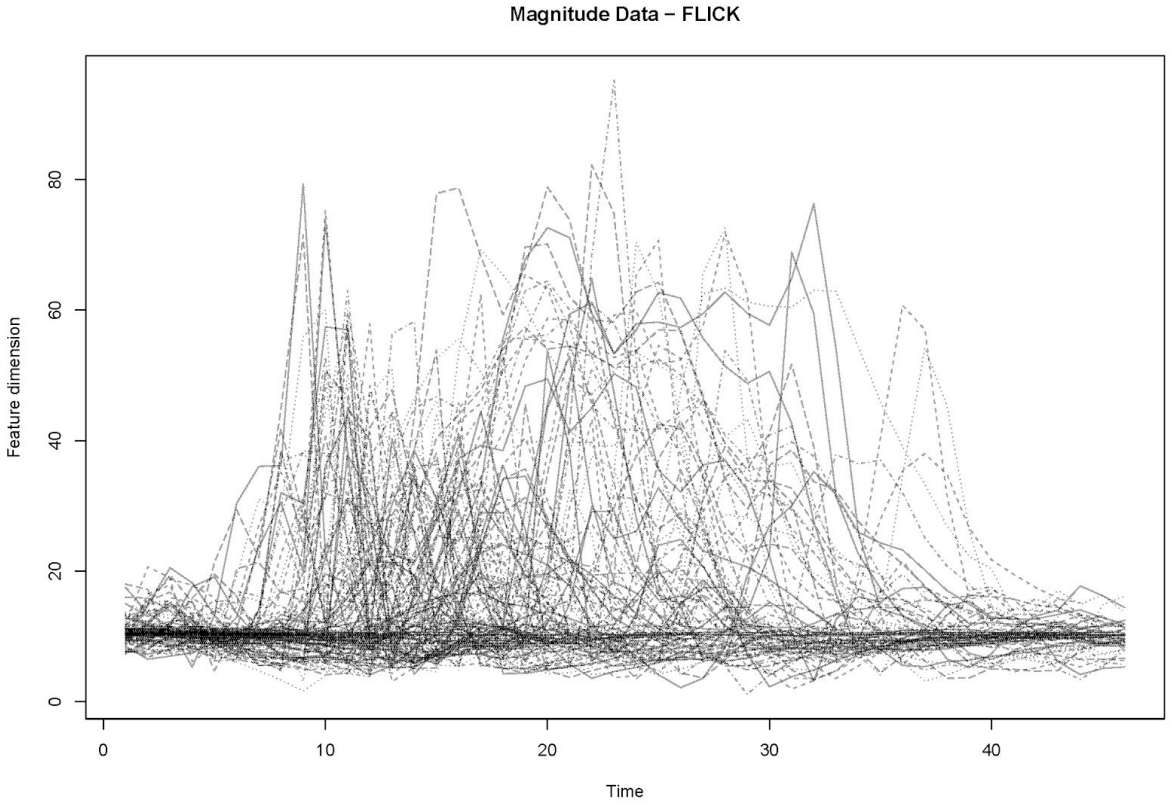
Additionally samples where the

So after the basic processing a single data frame exists with the following structure:

```
'data.frame':808 obs. of  161 variables:
 $ gesture       : Factor w/ 6 levels "CIRCLE","FLICK",..: 3 3 3 3 3 3 3 3 3 3 ...
 $ id            : Factor w/ 9 levels "Lucas Degner9d00a01b9f955546",..: 1 1 ...
 $ sample        : int  0 1 2 3 4 5 6 7 8 9 ...
 $ orig_length   : num  104 108 122 96 108 112 124 130 108 134 ...
 $ magnitude.t1  : num  10.01 9.5 8.86 9.54 9.77 ...
 $ magnitude.t2  : num  7.06 9.89 9.97 10.93 10.55 ...
 $ magnitude.t3  : num  16.4 11.2 11 15 10.6 ...
 $ magnitude.t4  : num  9.56 11 11.4 16.84 9.13 ...
 $ magnitude.t5  : num  9.4 10.66 10.9 14.4 8.31 …
 ...
 $ magnitude.tn
```

The magnitude values for the gestures “circle” and “flick” look now as following when being visualized. The plots show that the samples of the same gesture are not very similar. Just taking the magnitude values as features would result in bad model.





2.2 Frequency domain features

The frequency domain seems may could deliver us good features, since it should not make a difference if the samples are recorded slowly or fast. Therefore an function is implemented which process the features on one sample. The function is applied on each sample. Since the fast fourier transformation is done inside the function, it is only necessary to pass the "raw" time data.

Value: Frequency features
Params: One sample
Code:

```
01: calcFreqFeatures <- function(timeData) {
02:    frequencyFeature <- vector(mode = "double", length = 22)
03:    names(frequencyFeature) = c("spectralCentroid",
                                   "spectralRollOff",
                                   paste0('band_', 1:20) )
04:    frequencyData <- Mod(fft(as.vector(t(timeData))))
05:
06:    indices <- c(1:as.integer(length(frequencyData)/2-0.5))
07:    frequencyDataHalf <- frequencyData[indices+1]
08:
09:    # calculating spectral centroid
10:    frequencyFeature[1] <- sum(frequencyDataHalf*indices)
                              / sum(frequencyDataHalf)
11:
12:    # calculating spectral rolloff
13:    energy <- sum(frequencyDataHalf)
14:    currEnergy <- 0
15:    for (idx in indices) {
16:          currEnergy <- currEnergy + frequencyDataHalf[idx]
17:
18:          if (currEnergy >= energy * 0.8) {
19:                frequencyFeature[2] <- idx
20:                break
21:          }
22:    }
23:
24:    # calculating spectral band energies
25:    bandEnergies <- double(length = 20)
26:    bandEnergies[1:min(20,length(frequencyDataHalf))]
                  <- frequencyDataHalf[1:min(20,length(frequencyDataHalf))]
27:
28:    frequencyFeature[3:22] <- bandEnergies
29:
30:    # return frequency feature list
31:    frequencyFeature
32: }
```

Summed up this function processes 22 features. Spectral centroid, spectral roll-of and 20 bands where each band has a band with of one hertz. Therefore our frequency feature dataset looks like following:

```
'data.frame':808 obs. of  22 variables:
 $ spectralCentroid: num  23.3 18.3 18.5 14.4 16.1 ...
 $ spectralRollOff : num  41 27 27 20 22 22 26 31 25 31 ...
 $ band_1          : num  1798 1739 2266 1569 1504 ...
 $ band_2          : num  86.2 142.8 233 193.3 517.5 ...
 $ band_3          : num  455 343 536 324 125 ...
 $ band_4          : num  56.5 44.5 211.1 25.5 244.5 ...
 $ band_5          : num  139 84.4 231.5 102.9 142.9 ...
 $ band_6          : num  76.8 163.5 108.5 70.2 104.4 ...
 $ band_7          : num  135.1 120 90.7 150.8 101 ...
 $ band_8          : num  92 45.3 70.5 30.8 83.3 ...
 $ band_9          : num  131.5 30.7 68 228.7 160 ...
 $ band_10         : num  57.8 132.2 38.1 196.4 385.1 ...
 $ band_11         : num  107 185 123 172 129 ...
 $ band_12         : num  60.4 192.9 126.1 39.1 40.8 ...
 $ band_13         : num  78.6 107.6 225.9 92.1 55 ...
 $ band_14         : num  55.6 45.5 89 95.8 93.6 ...
 $ band_15         : num  39.4 25.1 91.6 246.3 91.2 ...
 $ band_16         : num  33.8 67.1 89.6 233.4 189 ...
 $ band_17         : num  7.97 101.82 154.11 316.42 144.24 ...
 $ band_18         : num  34.3 56.7 115.7 284.3 358.9 ...
 $ band_19         : num  47.2 170 188 818 358.8 ...
 $ band_20         : num  63.5 178.8 135.1 717.5 925.3 ...
```

After that, the feature domain preprocessing pipe is actually done. The next step is to process the time domain pipe and then merge all the features.

Step 2.3 Time domain features

For the processing of time domain features multiple datasets are prepared. Interpolation, filtering and normalization are used to accomplish that. Not every step is applied to each dataset.

The basic idea is to prepare four datasets and process all the feature on each dataset. In order to do that, we prepare these sets:

- Interpolated data
- Interpolated & filtered data
- Interpolated & normalized data
- Interpolated & filtered & normalized data

2.2.1 Interpolation

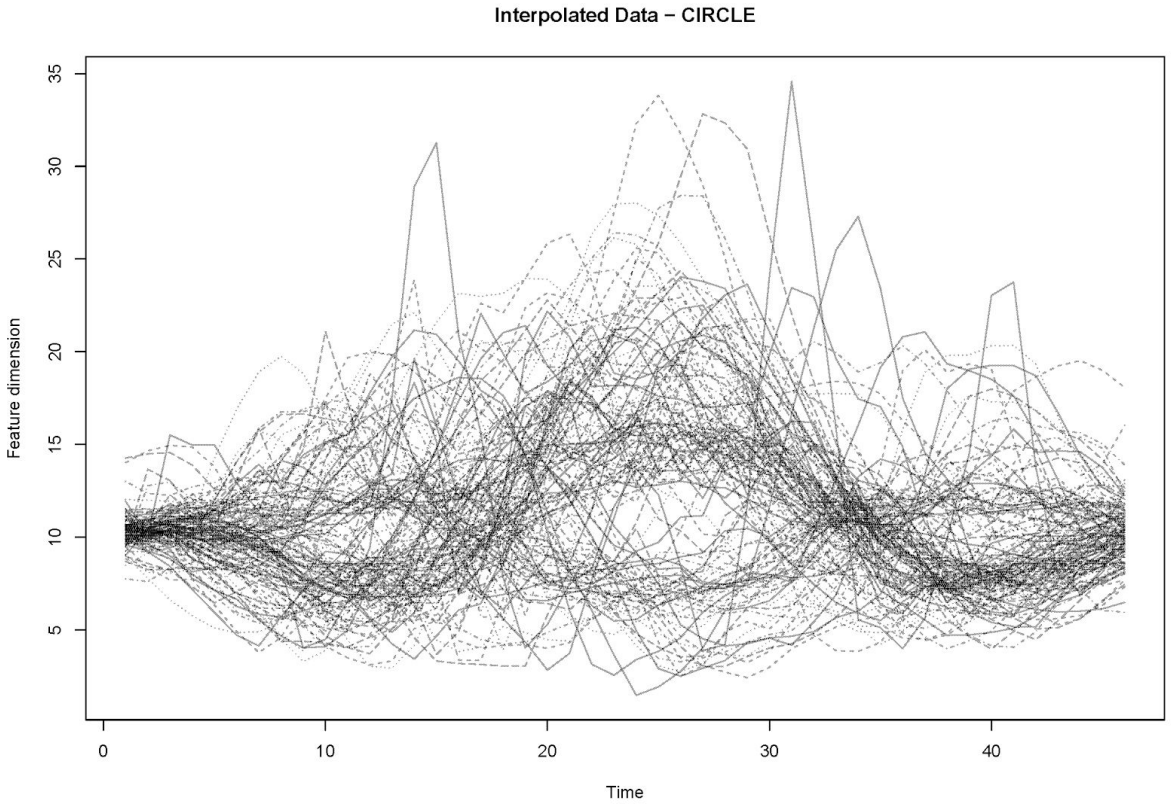
At the basic preprocessed the original sample length was already extracted as own feature, therefore it is not necessary to extract it here and furthermore the information of the sample length is not lost after all.

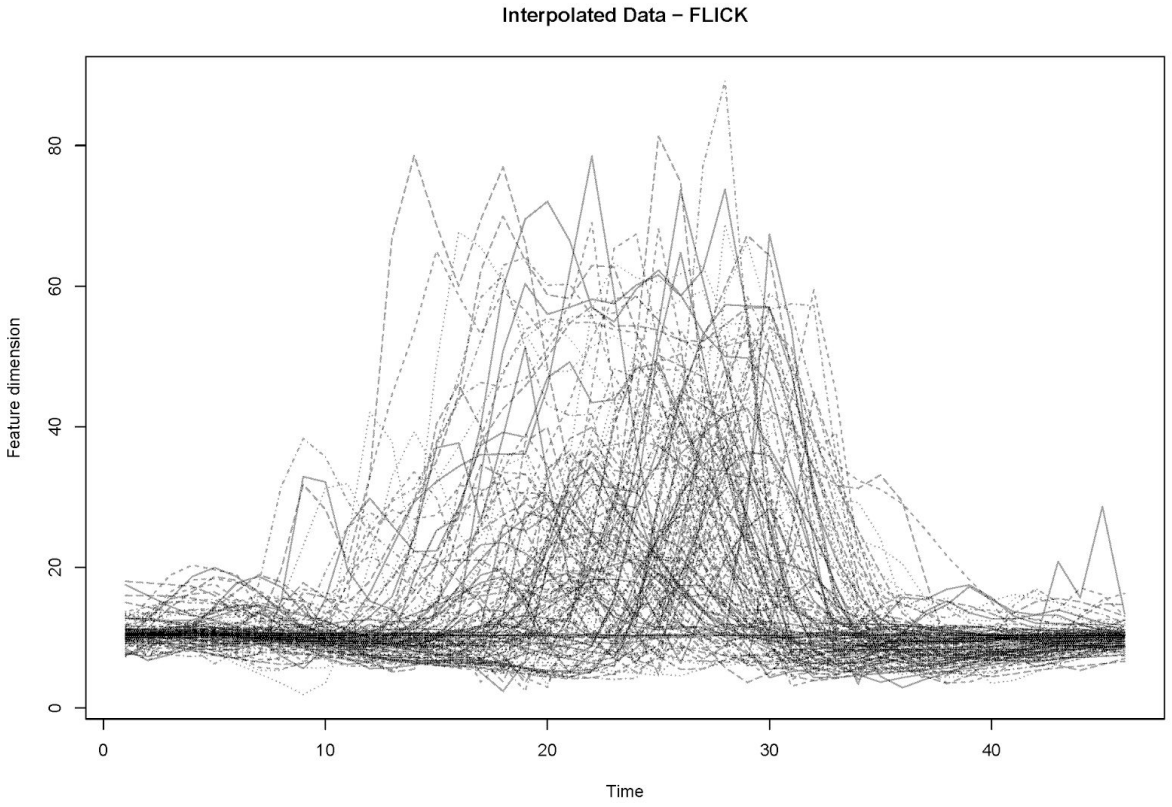
The interpolation process results in a sample length of 50 values. NA values are omitted and the `rule = 2` parameter makes sure that if an NA value is calculated, it will be replaced by the nearest not NA value. That happened at some examples in the assignment 5 and caused problems afterwards, so we applied the rule here again, so we do not have any NA values in our feature set.

Value: Interpolation
Parameter: One sensor value series
Code:

```
01: stepwidth = 0.02 # 50 values
02: interploated <- aaply(.data = as.matrix(mergedData[,-1:-4]),
                          .margins = 1, .fun = function(r) {
03:      t <- na.omit(as.numeric(r))
04:      acc <- approx(x = seq(0,1,1/(length(t)-1)),
                       y = t, xout = seq(0,1,stepwidth), rule = 2)$y
05: })
```

As shown below, the samples have now an equal length. Again the two gestures “circle” and “flick” are visualized.





2.3.2 Filtering

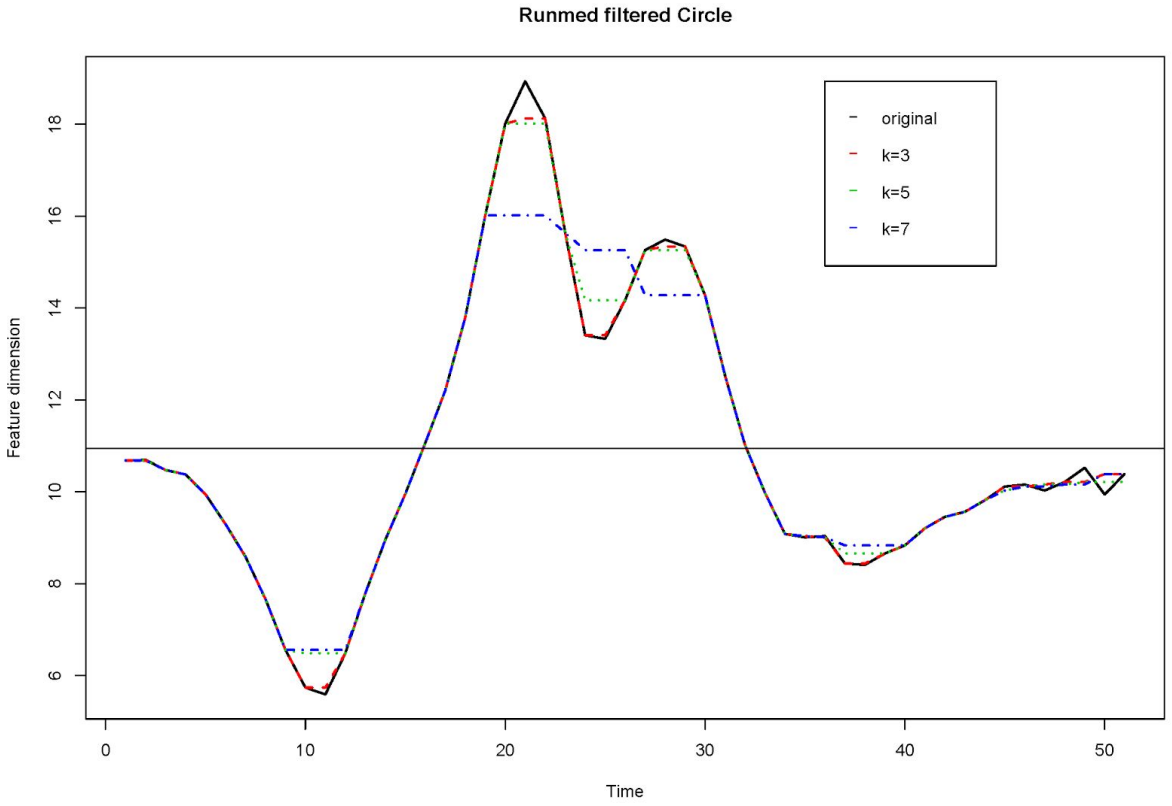
Normally filtering is done before interpolation, but here it is done vice-versa. The reason for that decision is that the different gestures have the same recording frequency but have different frequencies at how they are done.

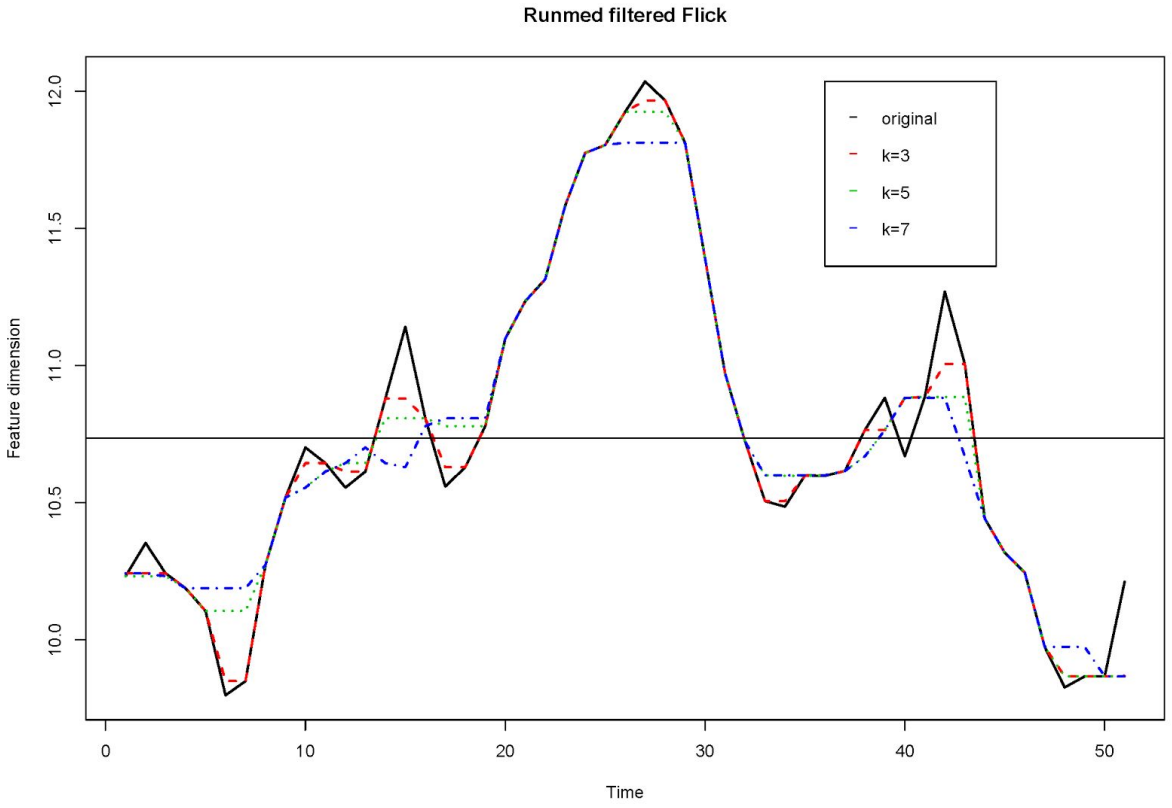
For example a circle is mostly done slow and results in a long recording sample, while the flick gesture is mostly done quickly and results in a short sample. If the same filter is applied to those gestures, which is necessary, this could and will lead to some problems. A short window for the filter may works better for a flick, but does not really affect a circle, while a wide window would work better on a circle, but maybe filters too much at a flick, since this gesture mostly has a more quickly changing amplitude.

Furthermore we cannot apply different filter for different gestures, since we would need to do the same for prediction after the model training. In a real case scenario it would be required to know gesture in before in order to apply the correct filter.
That is why we decided to apply the interpolation first and do the filtering afterwards. Then the samples are of equal length and quick changing amplitudes result in a slower changing amplitude and vice versa.

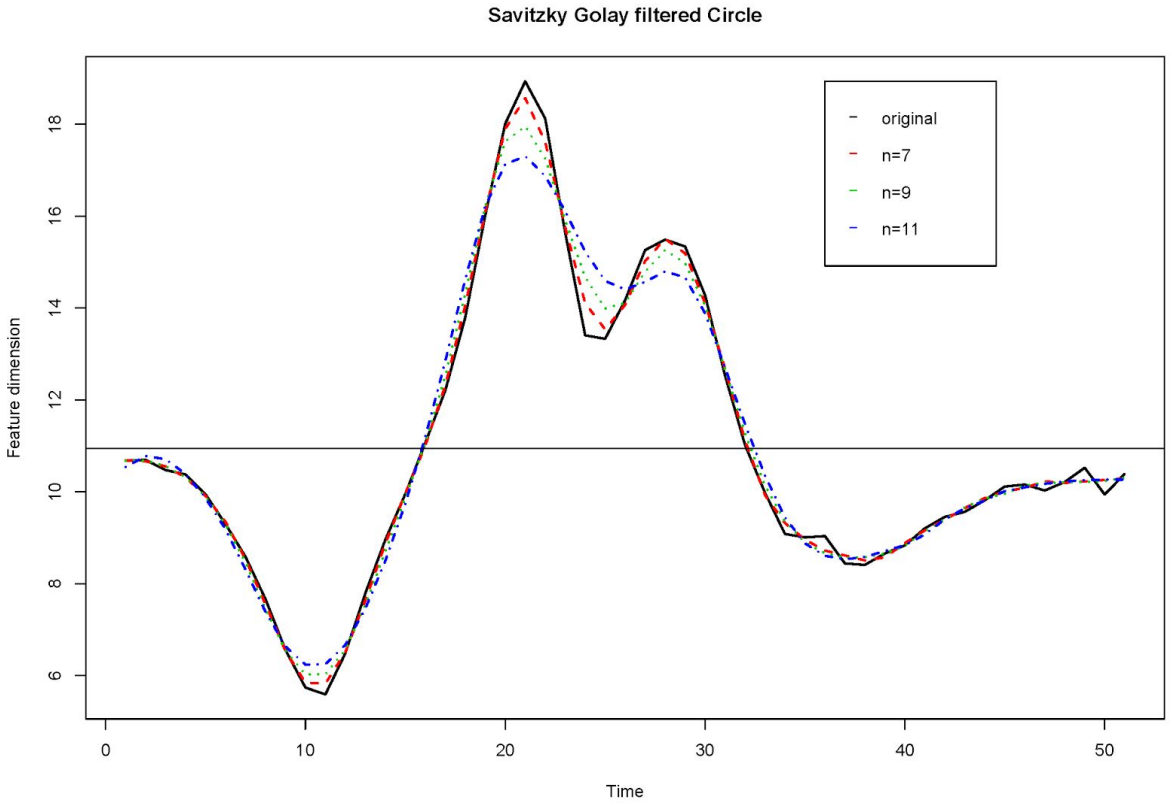
Two different filters, a running mean filter and a savitzky golay filter, with different parameters are applied to some samples in order to find the “best one”, which is then applied to all samples.

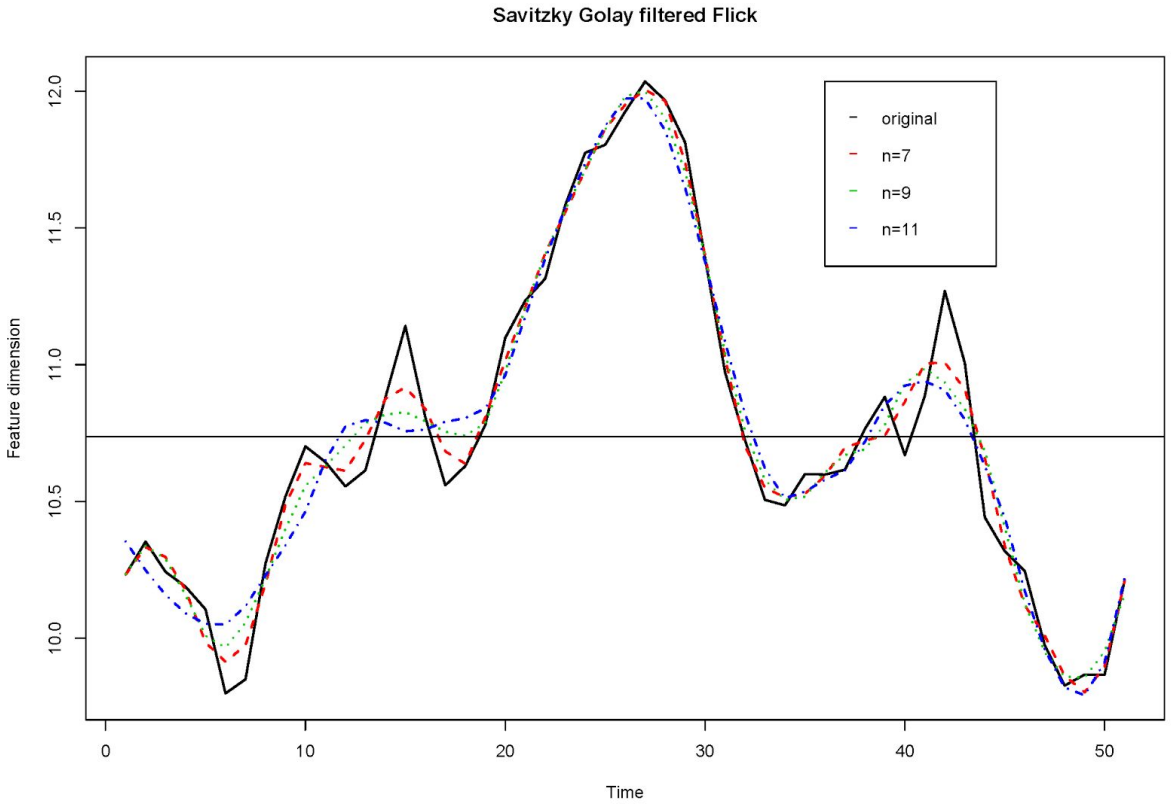
The following plots show some running mean filters applied to a circle and flick sample.





The same samples were filtered with some savitzky golay filters. All filters had an polynomial order of `p = 3`.



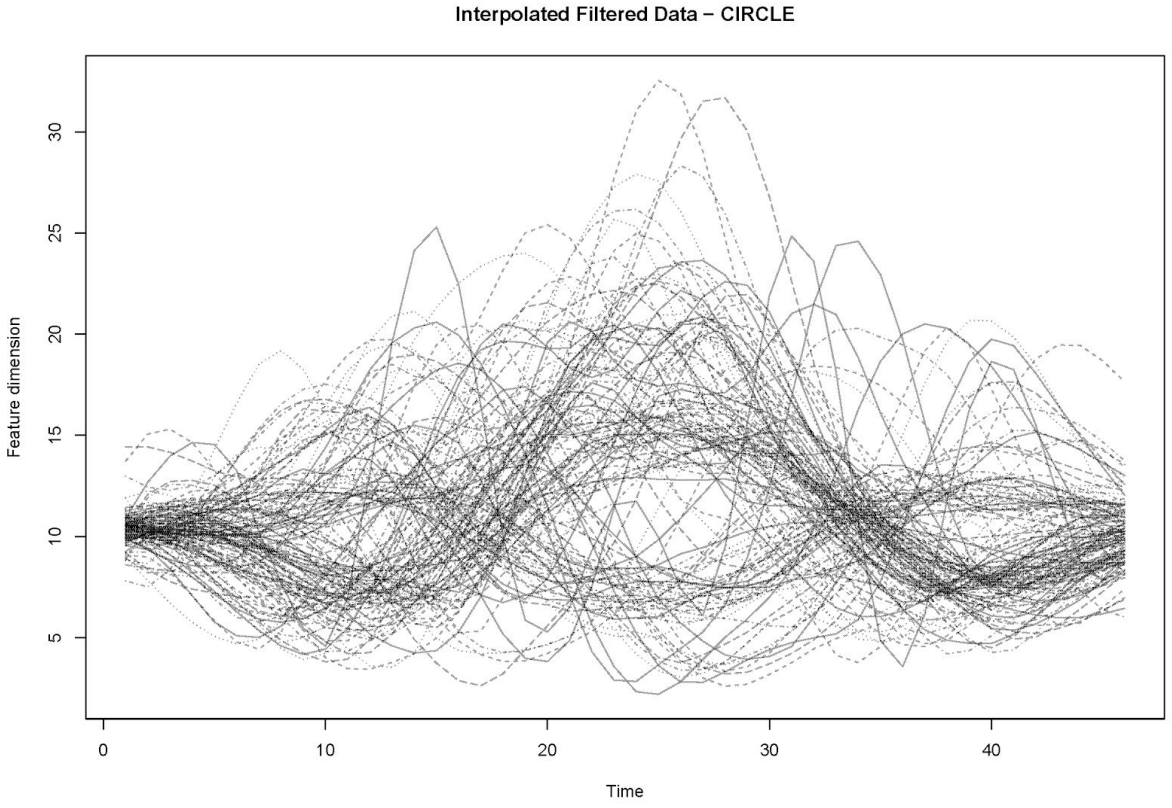


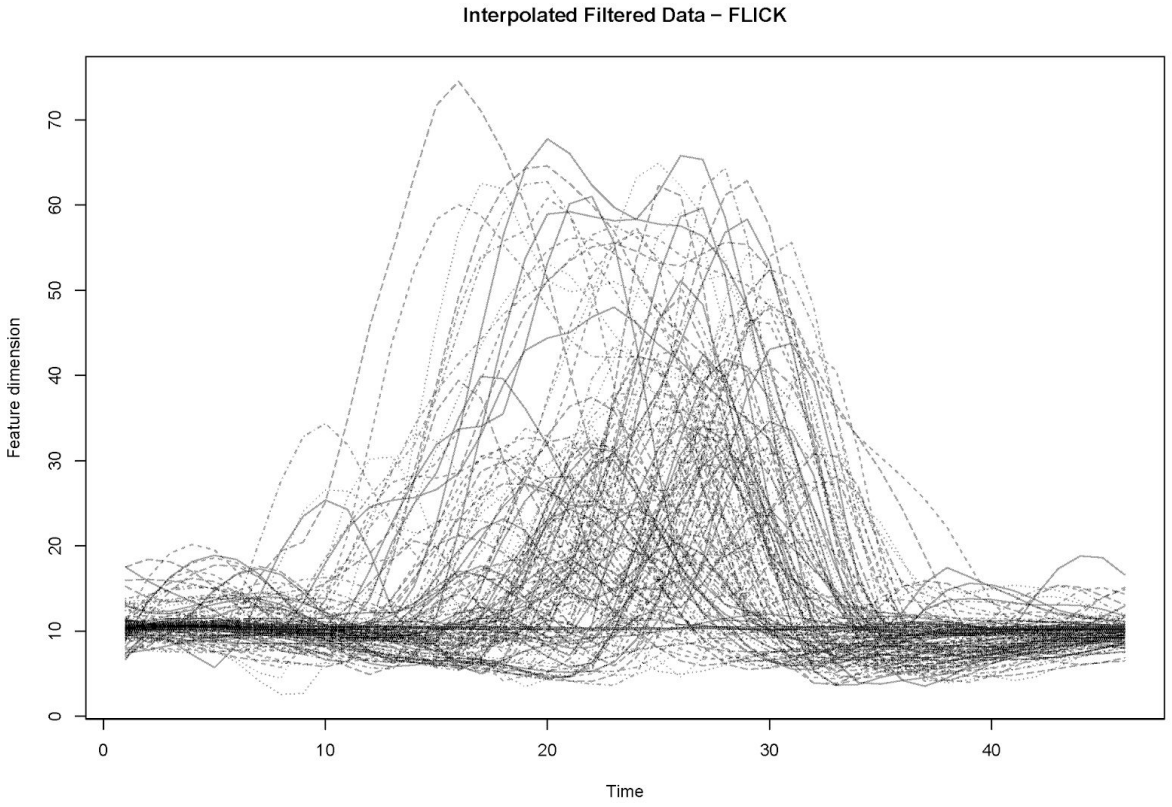
The horizontal line in the plots above visualizes the mean value of that sample which should help to make decisions, because for features like mean crossing rate this is essential.

After comparing multiple samples with those filters, the first diagnosis was that the running mean filter cuts off the peaks too much and a running mean filter with a small window size is also not optimal since it only cuts off short peaks.

Therefore the focus fell on the savitzky golay filter. After comparing again some samples, the final decision fell to the golden mean parameter setting with a window size of 9 (the green lines). A smaller filter sometimes smoothes to little, while a wider one sometimes removes peaks at all. This is always very different from gesture to gesture as well from sample to sample. Overall this was the best setting in our opinion.

After applying the final savitzky golay filter (`p = 3, k = 9`) an additional interpolated and filtered dataset exists which will be later used for feature derivation. Again all filtered samples for the circle and flick gesture are visualized below.





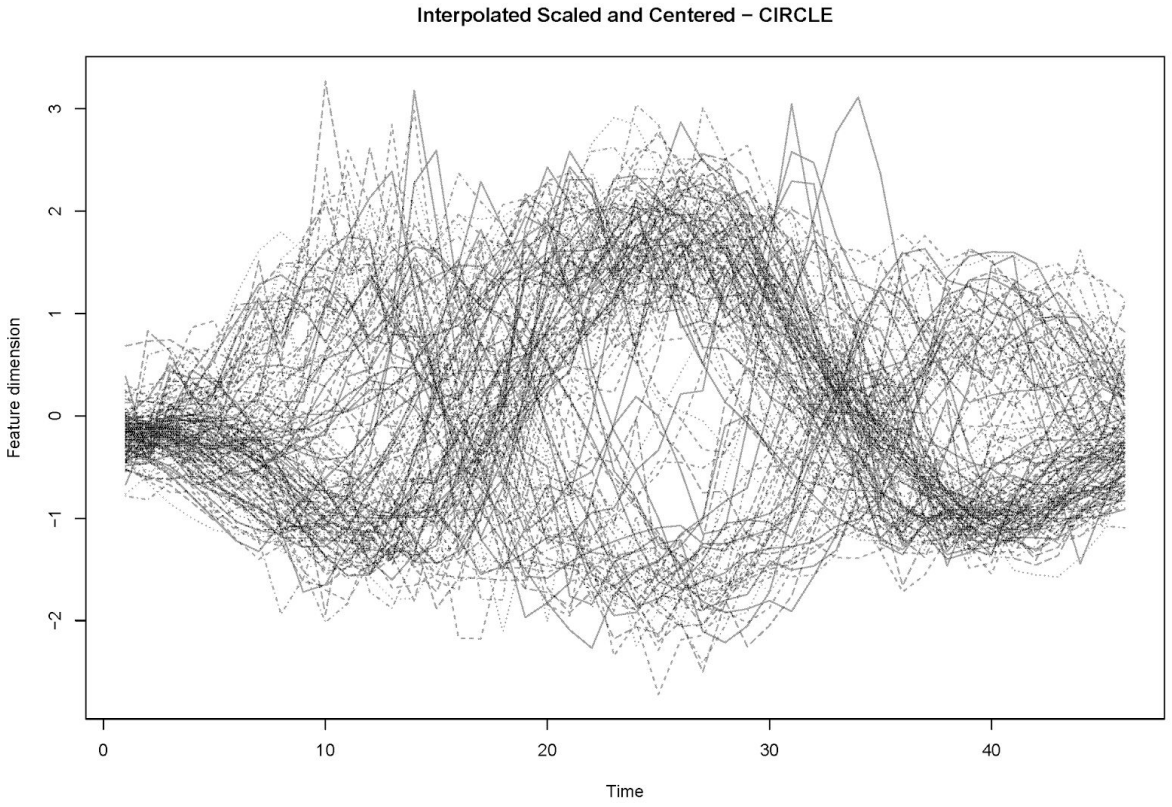
2.3.3 Normalization

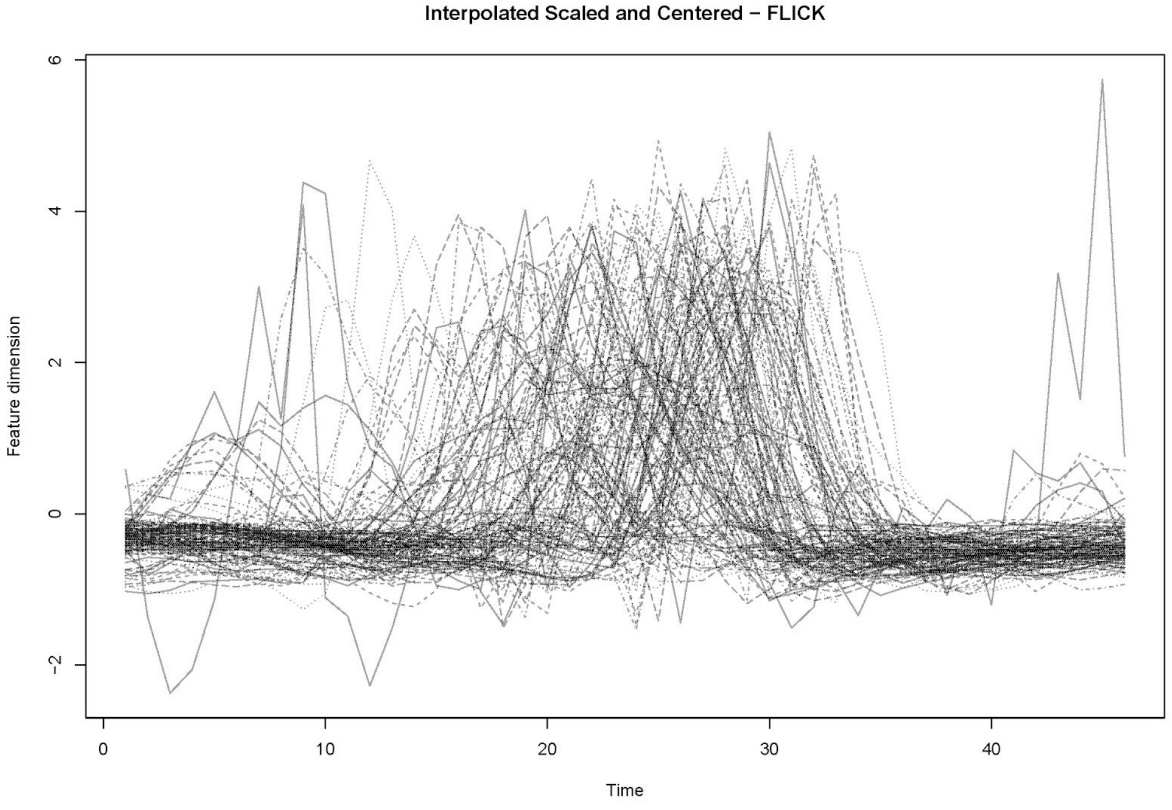
At this point two of the four (five with the raw data) datasets for the time feature derivation are prepared. These two are the interpolated and the interpolated filtered datasets. In order to prepare the other two datasets, the existing two sets are normalized.

The normalization is done through center the data and then scaling it. At the previous plots, one is able to see that scaling could affect the data a lot. Centering enables us to calculate the zero crossing rate (which equals the mean crossing rate when centered). The mean crossing rate should still be the zero crossing rate after scaling, because we apply the scaling on the already centered data.

After normalization the samples look more similar. The plots below (next page) show the interpolated and normalized dataset. Especially the circle samples support this statement. One is able to see two main waves, which are actually mirrored.

At this point the four (five with the raw data) datasets are prepared for feature derivation from the time domain, which will be the next step.


Interpolated Scaled and Centered − CIRCLE
Feature dimension
Time
3
2
1
0
−1
−2
0
10
20
30
40


Interpolated Scaled and Centered − FLICK
Feature dimension
Time
6
4
2
0
−2
0
10
20
30
40

2.3.4 Time domain feature derivation

Again, an own function for the processing of the features is applied to each sample. This is done for all four (five) datasets, so after that step there are a lot of time domain features.

Value: Time domain features
Params: `timeData` - one sample of data
`featurePrefix` - feature names prefix (since it is applied to multiple datasets)
`mcrThreshold` - threshold for the mean crossing rate (values below that threshold do not count as cross)

Code:

```
01: calcTimeFeatures <- function(timeData, featurePrefix, mcrThreshold) {
02:    timeFeature <- vector(mode = "double", length = 4)
03:    names(timeFeature) = c(paste0(featurePrefix,"median"),
                              paste0(featurePrefix,"MCR"),
                              paste0(featurePrefix,"innerQRange"),
                              paste0(featurePrefix,"MAD"))
04:
05:    # median
06:    timeFeature[1] <- median(timeData, na.rm = T)
07:
08:    # MCR
09:    mcCount <- 0
10:    dataMean <- mean(timeData, na.rm = T)
11:    mcFlag <- timeData[1] >= dataMean
12:
13:    for (i in 2:length(timeData)) {
14:       if(!is.na(timeData[i]) & mcFlag != (timeData[i] >= dataMean)) {
15:          if(abs(timeData[i] - dataMean) > mcrThreshold) {
16:             mcCount <- mcCount + 1
17:             mcFlag <- !mcFlag
18:          }
19:       }
20:    }
21:
22:   timeFeature[2] <- mcCount
23:
24:    # innerQRange
25:    timeFeature[3] <- IQR(timeData, na.rm = T)
26:
27:    # MAD
28:    timeFeature[4] <- mad(timeData, na.rm = T)
29:
30:    # return
31:    timeFeature
32: }
```

After applying this function to all preprocessed and the “raw” preprocessed dataset and merging them to a single data frame, the following time domain features exists.

```
'data.frame':808 obs. of  20 variables:
 $ raw_median                                 : num  27.2 27.3 30.4 30.1 34 ...
 $ raw_MCR                                    : num  24 24 24 16 22 26 22 20 ...
 $ raw_innerQRange                            : num  53.5 45.2 52.7 46.3 49.8 ...
 $ raw_MAD                                    : num  25.9 26.1 29.8 27.7 34.6 ...
 $ interpolated_median                        : num  27.3 28.1 33.9 30.4 34.4 ...
 $ interpolated_MCR                           : num  8 16 8 14 18 18 12 18 16 ...
 $ interpolated_innerQRange                   : num  40.3 39 50.4 46.8 43.3 ...
 $ interpolated_MAD                           : num  25.6 26.1 35.4 29.7 33.8 ...
 $ interpolated_filtered_median               : num  41.6 36.2 41.9 43.2 51.5 ...
 $ interpolated_filtered_MCR                  : num  2 2 2 2 2 2 2 2 2 ...
 $ interpolated_filtered_innerQRange          : num  53.7 47.5 56.5 49.7 44.2 ...
 $ interpolated_filtered_MAD                  : num  40.4 36 43.1 36.3 17.5 ...
 $ interpolated_normalised_median             : num  -0.375 -0.295 -0.247 ...
 $ interpolated_normalised_MCR                : num  1 1 2 2 2 2 2 1 4 2 ...
 $ interpolated_normalised_innerQRange        : num  1.28 1.28 1.59 1.6 1.5 ...
 $ interpolated_normalised_MAD                : num  0.812 0.857 1.118 1.013 ...
 $ interpolated_filtered_normalised_median    : num  0.0993 -0.03639 0.00651 ...
 $ interpolated_filtered_normalised_MCR       : num  2 2 2 2 2 2 2 2 2 2 ...
 $ interpolated_filtered_normalised_innerQRange: num  2.07 2 2.04 2.07 2 ...
 $ interpolated_filtered_normalised_MAD       : num  1.56 1.52 1.56 1.51 0.79 …
```

2.3.5 Choose features

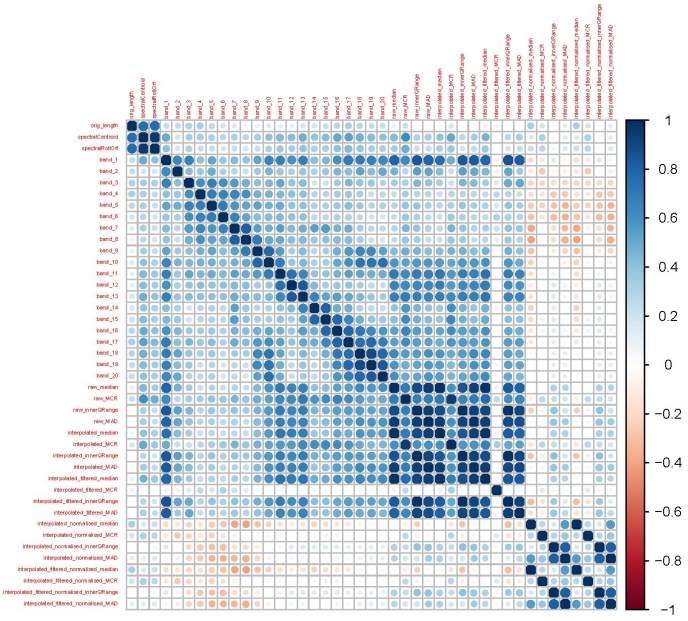
After merging all features (frequency domain features, time domain features and basic info like gesture, id and so on) there is a total amount of 43 features (without gesture, ID and sample nr). There are surely some features which do not help to predict at all or do not add information to the training process.
Additionally the feature “cleaning” will speed up the training process, if some features are removed.

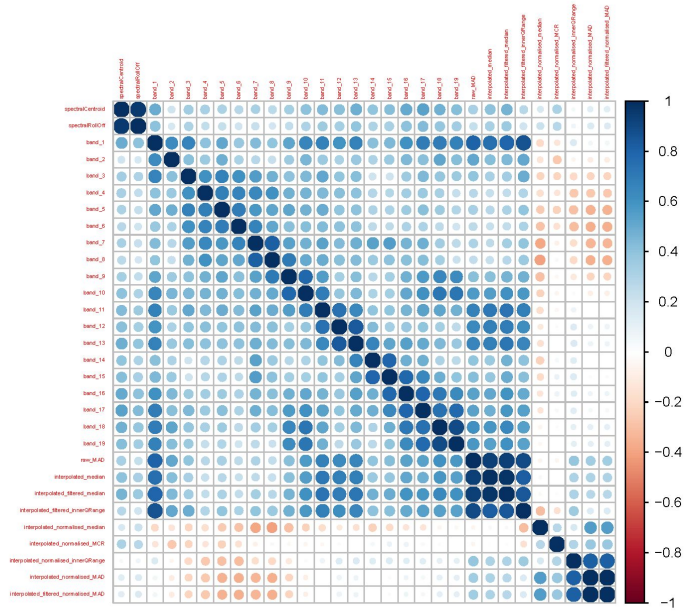
For this step the data partitioning is moved forward, because the feature correlation should be calculated only on the training data, as it is shown in the approach diagram.

Furthermore two different ways of data partitioning are done. Two partitions for population dependent and two partitions for population independent training and testing are created. For the population dependent partitions a 80:20 size is chosen which results in 647 training samples and 161 testing samples.
The population independent partition size is done different. One person is omitted in the training partition, which represents the test partition.

Due to the fact, that there are four data partitions, each model has to be trained twice.

This plots show the feature correlations for **dependent** data partitioning. The first one is the full data set and the second one is the reduced set, which is later taken for the training and validation process.





Since there are different partitions for **independent** population prediction this step is done twice. The plot does not show much difference (especially at this small resolution in the report; PDF is in the plots folder).

Below the reduced data set for dependent reduced data partitions are stated. Both, independent and dependent partitions, have the same features except one. The red highlighted interquartile range of the interpolated normalized dataset is an additional feature in the population dependent dataset.

An interesting fact is that the frequency bands themselve are good features.

Reduced data training partition:

```
'data.frame':   647 obs. of  37 variables:
 $ gesture                                    : Factor w/ 6 levels "CIRCLE","FLICK",..: ...
 $ id                                         : Factor w/ 9 levels "Lucas ...
 $ sample                                     : int  0 1 2 3 4 6 7 8 9 10 ...
 $ orig_length                                : num  104 108 122 96 108 124 130 108 134 ...
 $ spectralRollOff                            : num  41 27 27 20 22 26 31 25 31 27 ...
 $ band_1                                     : num  1798 1739 2266 1569 1504 ...
 $ band_2                                     : num  86.2 142.8 233 193.3 517.5 ...
 $ band_3                                     : num  455 343 536 324 125 ...
 $ band_4                                     : num  56.5 44.5 211.1 25.5 244.5 ...
 $ band_5                                     : num  139 84.4 231.5 102.9 142.9 ...
 $ band_6                                     : num  76.8 163.5 108.5 70.2 104.4 ...
 $ band_7                                     : num  135.1 120 90.7 150.8 101 ...
 $ band_8                                     : num  92 45.3 70.5 30.8 83.3 ...
 $ band_9                                     : num  131.5 30.7 68 228.7 160 ...
 $ band_10                                    : num  57.8 132.2 38.1 196.4 385.1 ...
 $ band_11                                    : num  107 185 123 172 129 ...
 $ band_12                                    : num  60.4 192.9 126.1 39.1 40.8 ...
 $ band_13                                    : num  78.6 107.6 225.9 92.1 55 ...
 $ band_14                                    : num  55.6 45.5 89 95.8 93.6 ...
 $ band_15                                    : num  39.4 25.1 91.6 246.3 91.2 ...
 $ band_16                                    : num  33.8 67.1 89.6 233.4 189 ...
 $ band_17                                    : num  7.97 101.82 154.11 316.42 144.24 ...
 $ band_18                                    : num  34.3 56.7 115.7 284.3 358.9 ...
 $ band_19                                    : num  47.2 170 188 818 358.8 ...
 $ band_20                                    : num  63.5 178.8 135.1 717.5 925.3 ...
 $ interpolated_median                        : num  27.3 28.1 33.9 30.4 34.4 ...
 $ interpolated_MCR                           : num  8 16 8 14 18 12 18 16 8 20 ...
 $ interpolated_filtered_MCR                  : num  2 2 2 2 2 2 2 2 2 2 ...
 $ interpolated_filtered_MAD                  : num  40.4 36 43.1 36.3 17.5 ...
 $ interpolated_normalised_median             : num  -0.375 -0.295 -0.247 -0.335 -0.228 ...
 $ interpolated_normalised_MCR                : num  1 1 2 2 2 1 4 2 1 ...
 $ interpolated_normalised_innerQRange        : num  1.28 1.28 1.59 1.6 1.5 ...
 $ interpolated_normalised_MAD                : num  0.812 0.857 1.118 1.013 1.174 ...
 $ interpolated_filtered_normalised_median    : num  0.0993 -0.03639 0.00651 0.11832 ...
 $ interpolated_filtered_normalised_MCR       : num  2 2 2 2 2 2 2 2 2 2 ...
 $ interpolated_filtered_normalised_innerQRange: num  2.07 2 2.04 2.07 2 ...
 $ interpolated_filtered_normalised_MAD       : num  1.56 1.52 1.56 1.51 0.79 ...
```

Step 3: Data partitioning

The data partitioning was already done before, since it was necessary for feature reduction. For more details see **2.3.5 Choose features**.

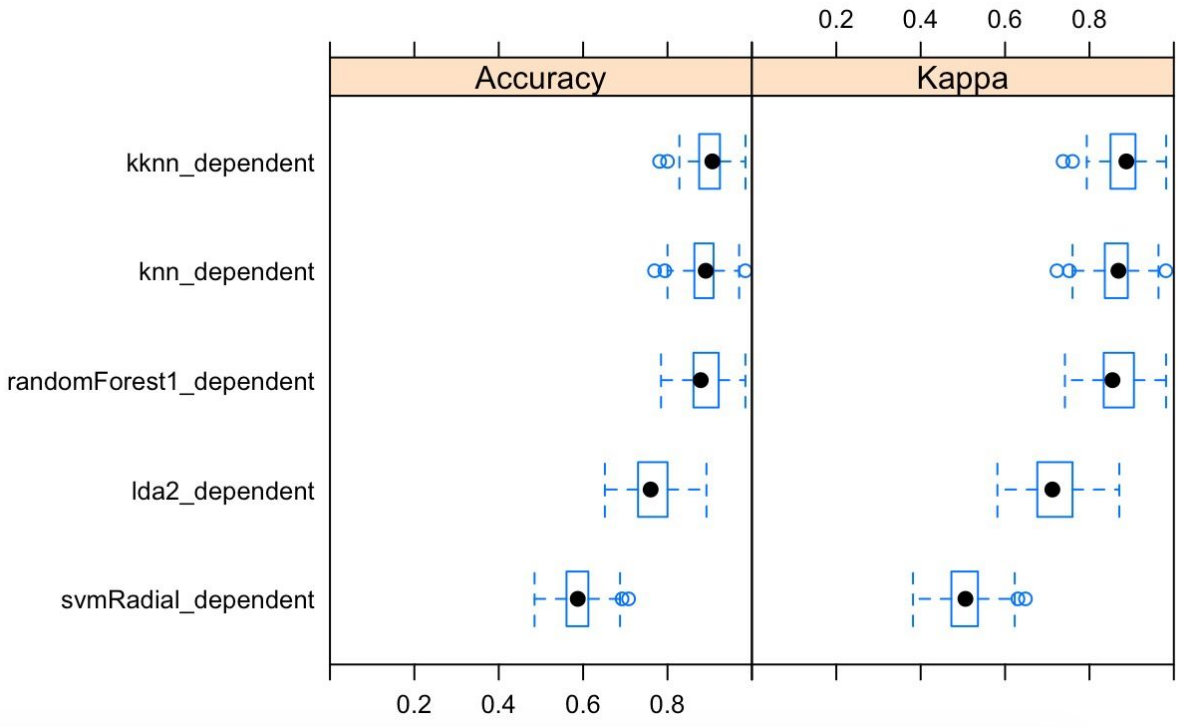
Step 4: Models training

As already mentioned each model has to be trained twice, since we have a population dependent and independent dataset.

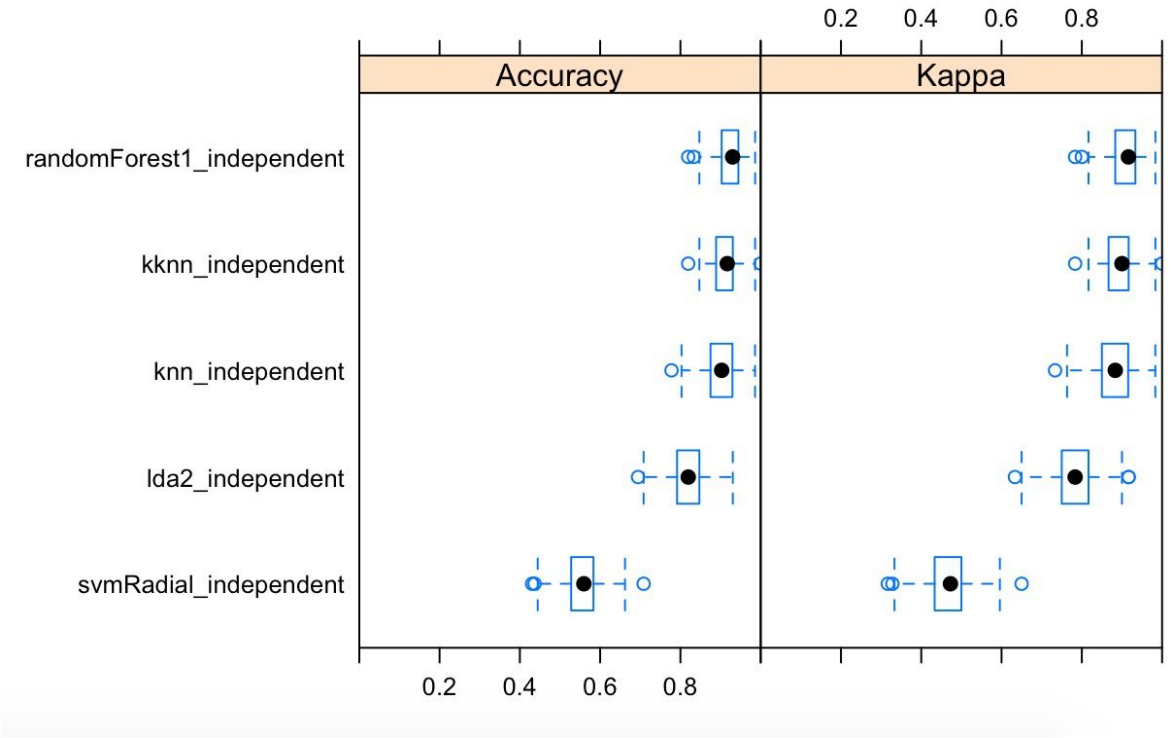
Overall these models have been trained:

- SVM Radial
- KNN
- KKNN
- LDA2
- Random Forest

Model performances dependent



Model performances independent



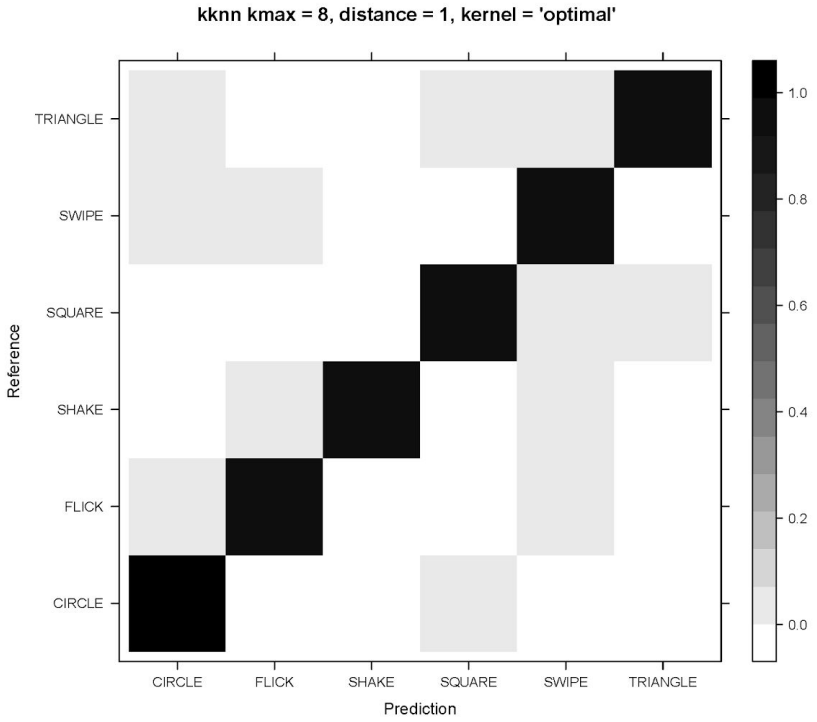
Step 5: Model selection

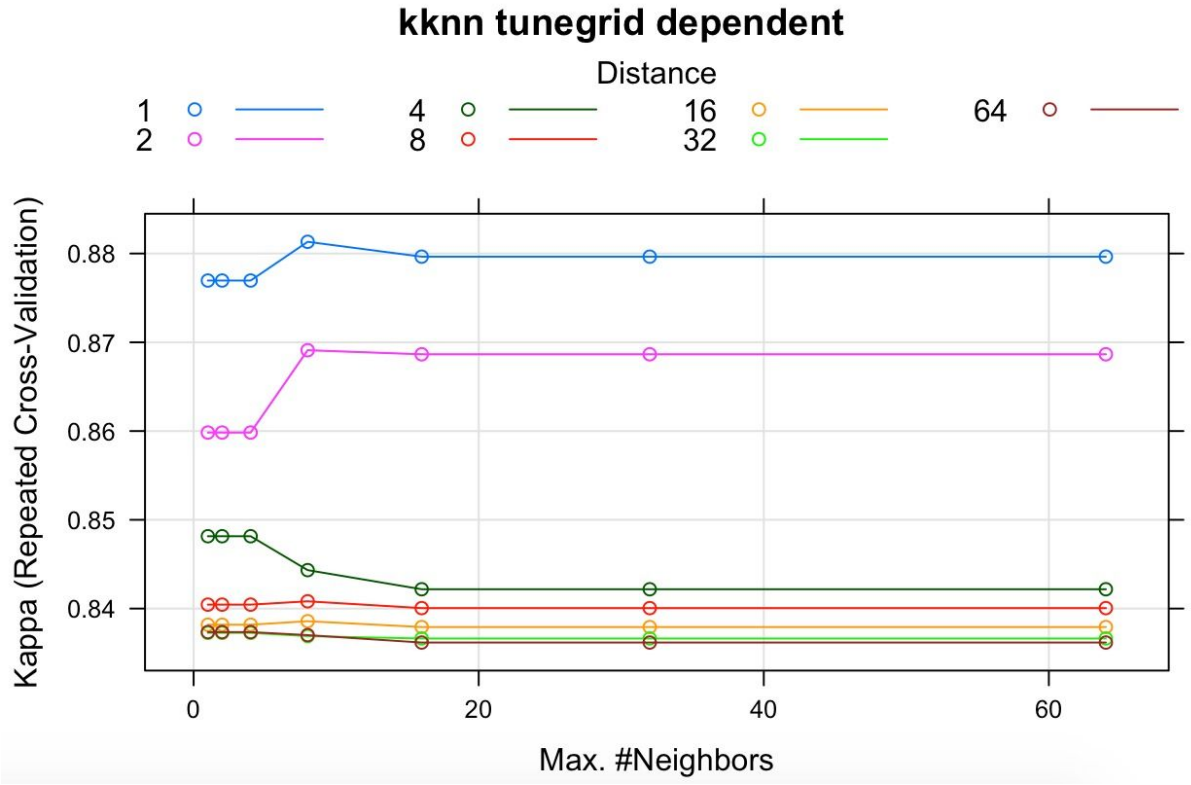
At both, dependent as well as independent the SVM radial and LDA2 models do not perform that good. Therefore we will not choose them as our final models.

Further the random forest for independent and the KKNN model for dependent population seem to perform the best. Although some more analysis should be done before choosing them as the best models.

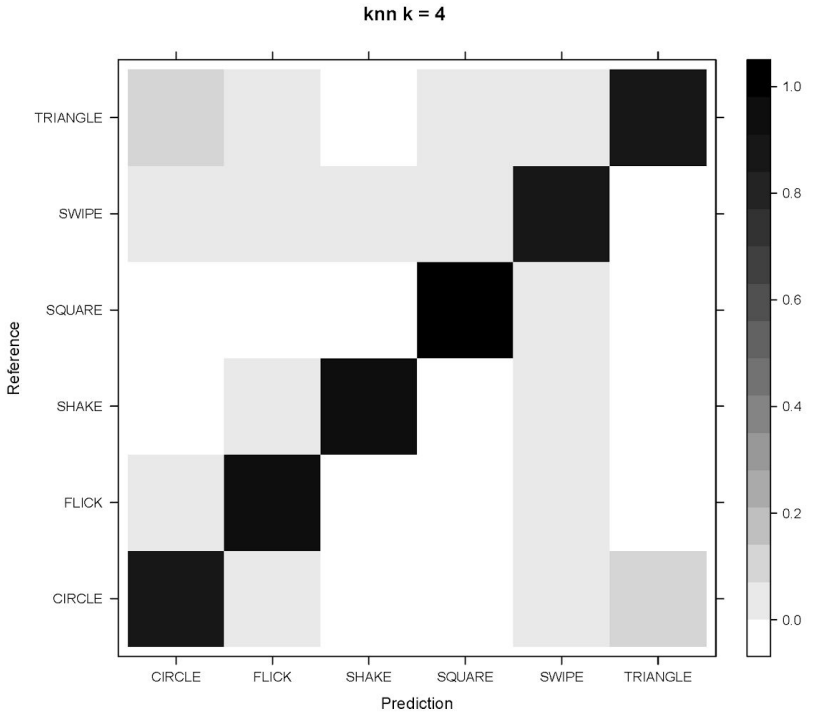
For both, dependent and independent, the best three models are more analyzed. Then for each the best model is then chosen. The following pages show all the models, where the dependent

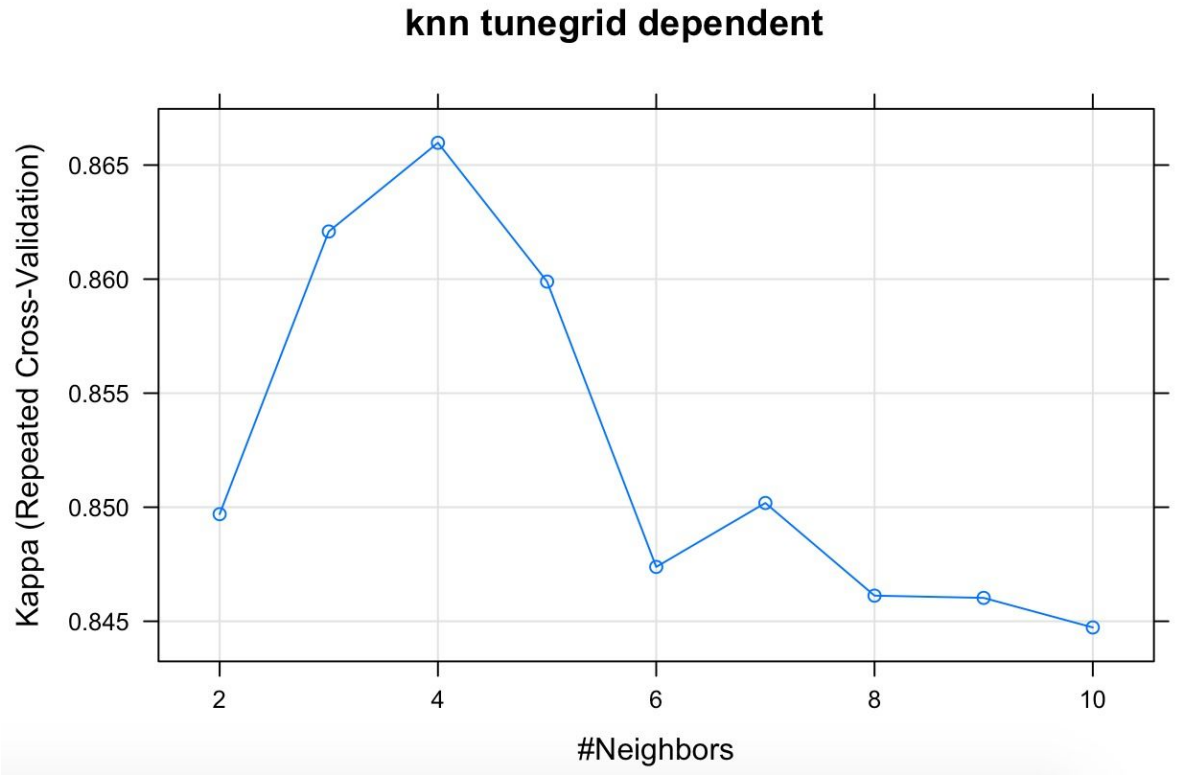
Dependent KKNN



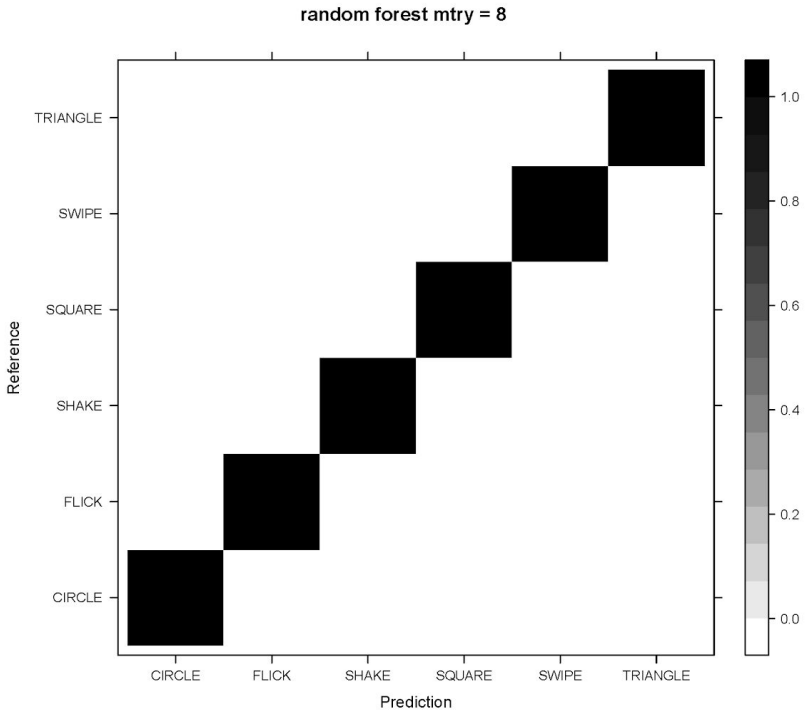


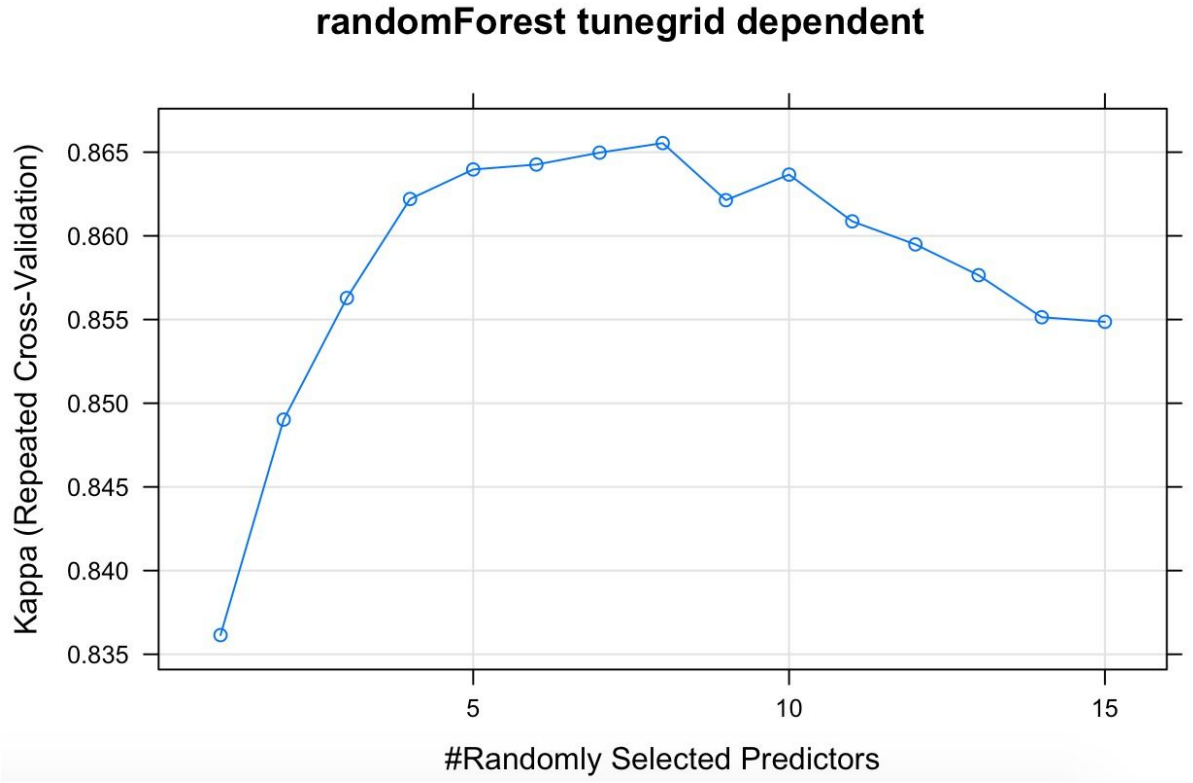
Dependent KNN



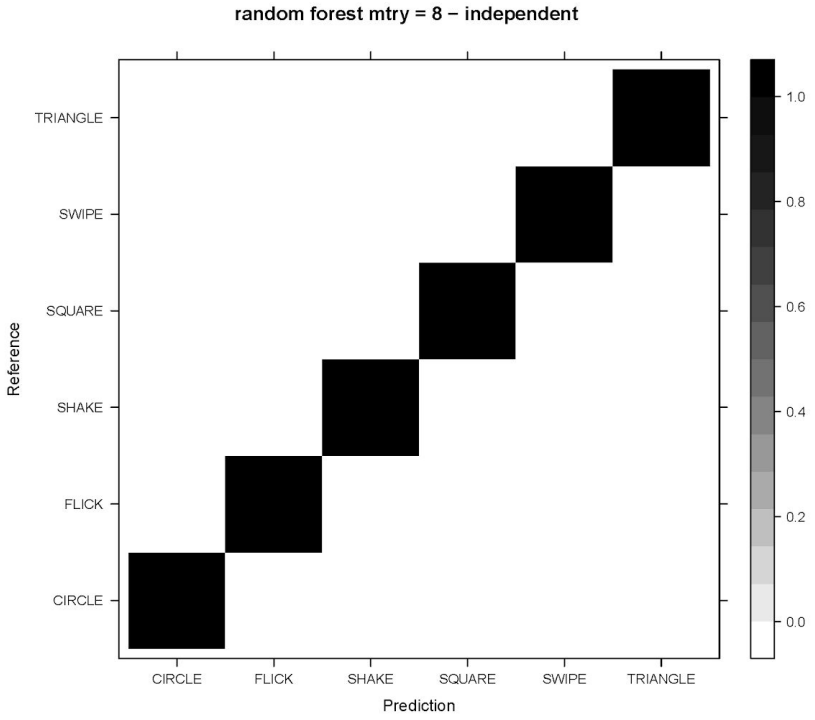


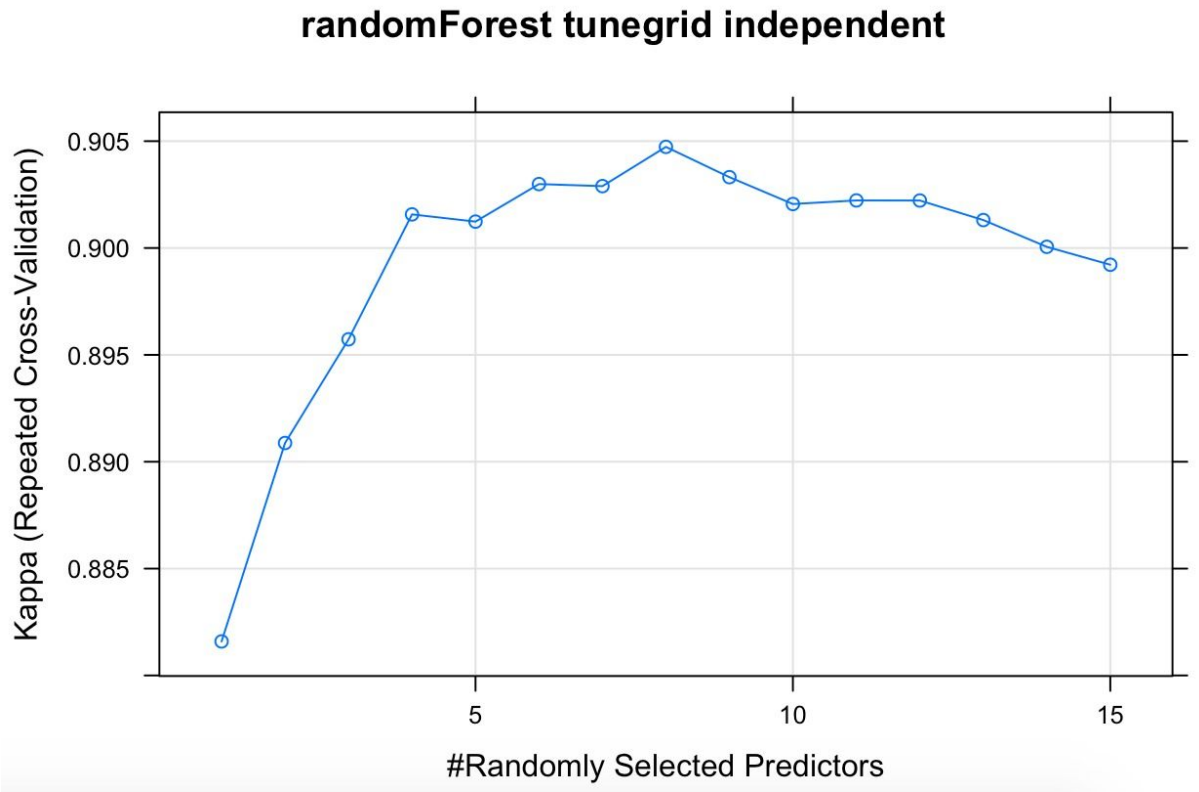
Dependent Random Forest



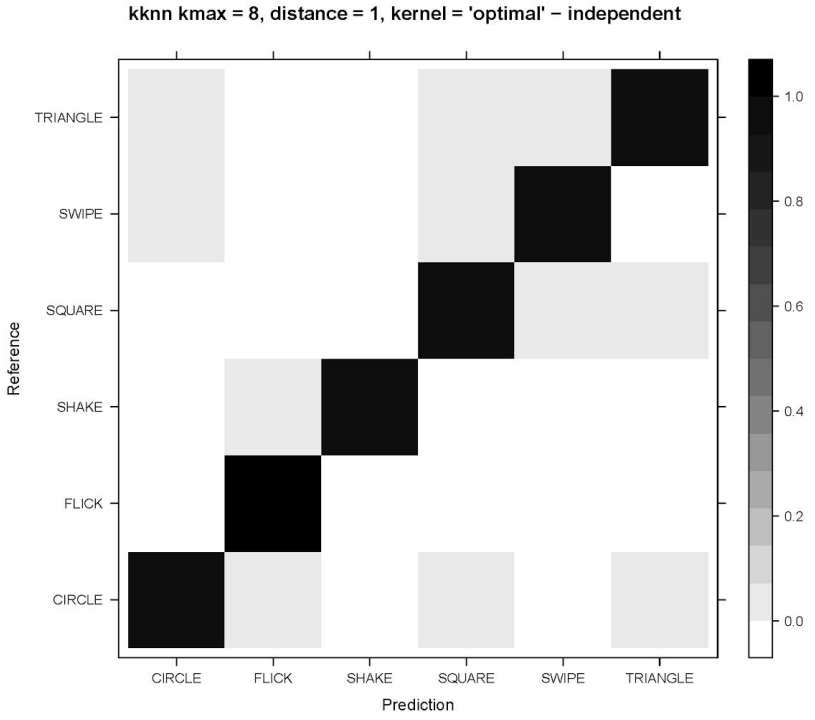


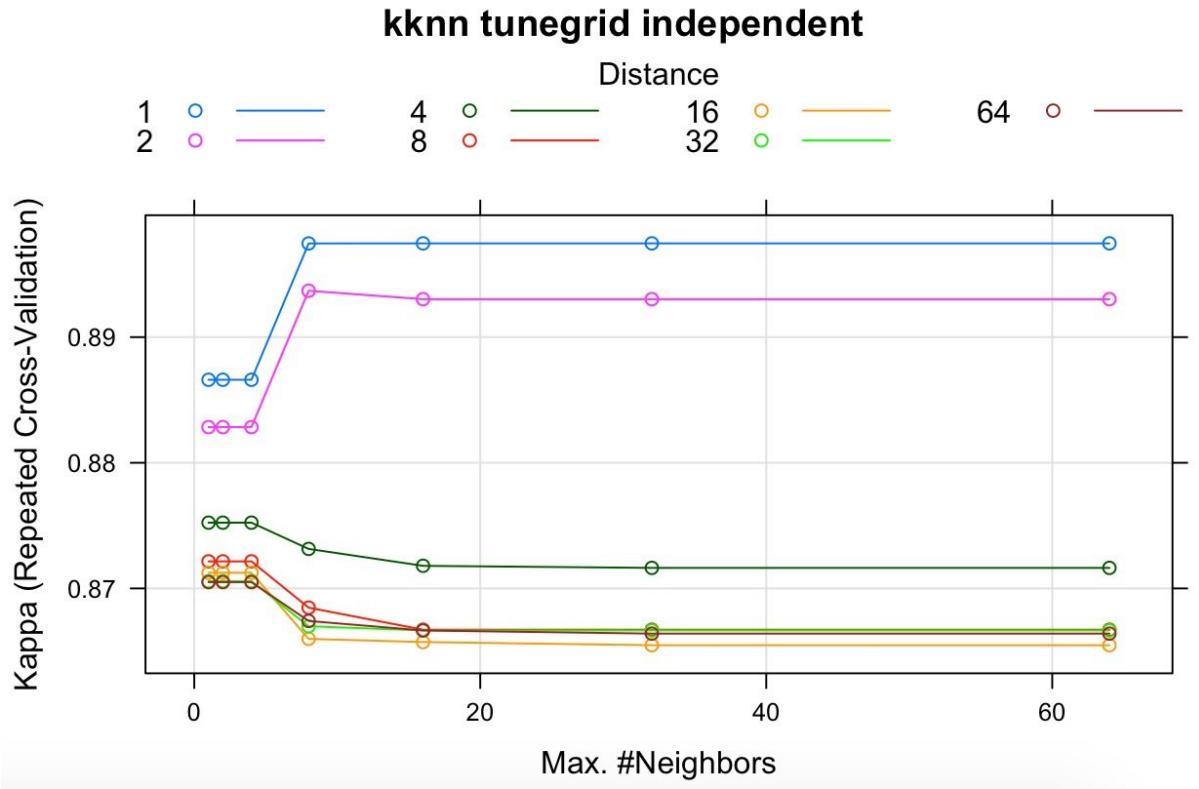
Independent Random Forest



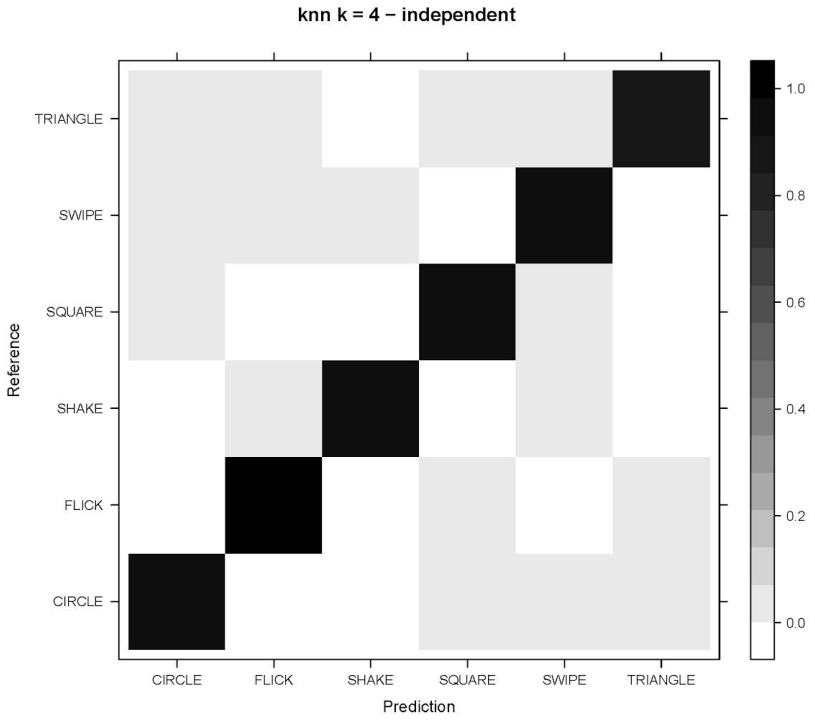


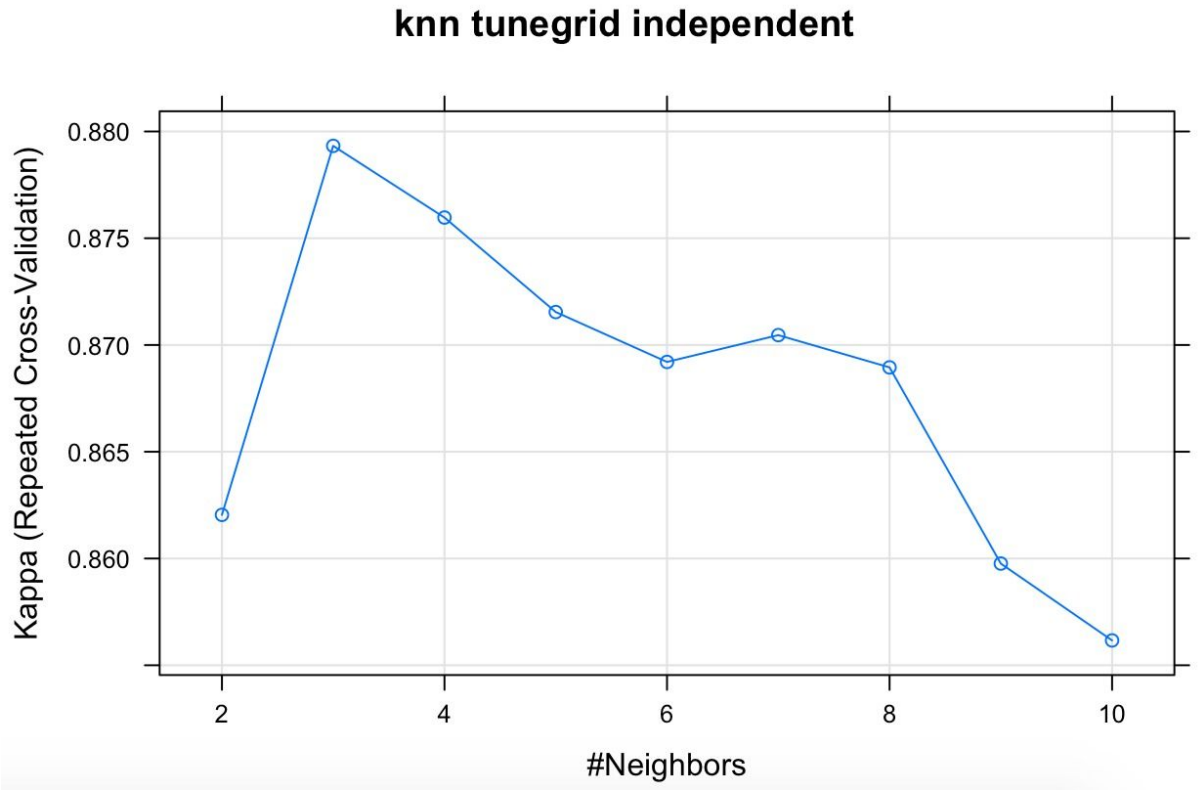
Independent KKNN





Independent KNN





Dependent model selection:

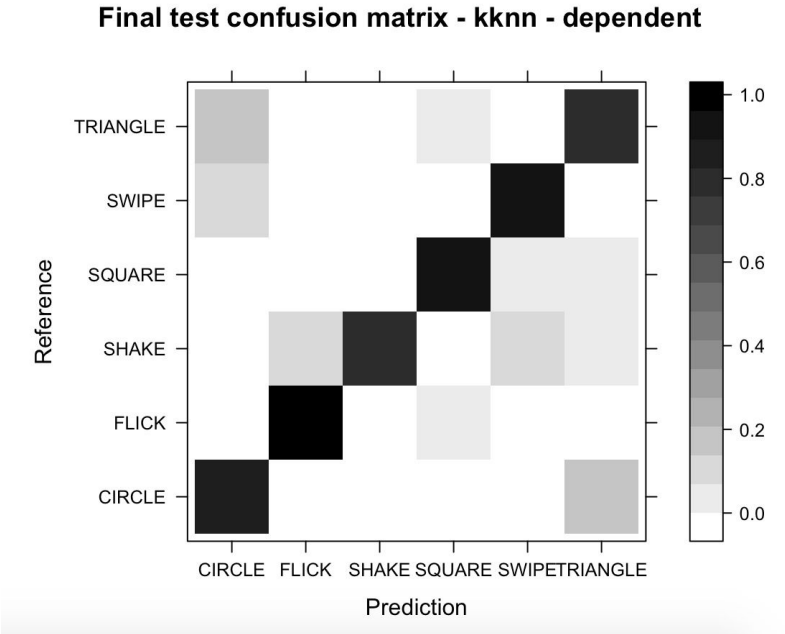
For the dependent data partition we chose the KKNN model, since the random forest model is obviously overtrained. Another option would have been to limit the random forest with its parameters.

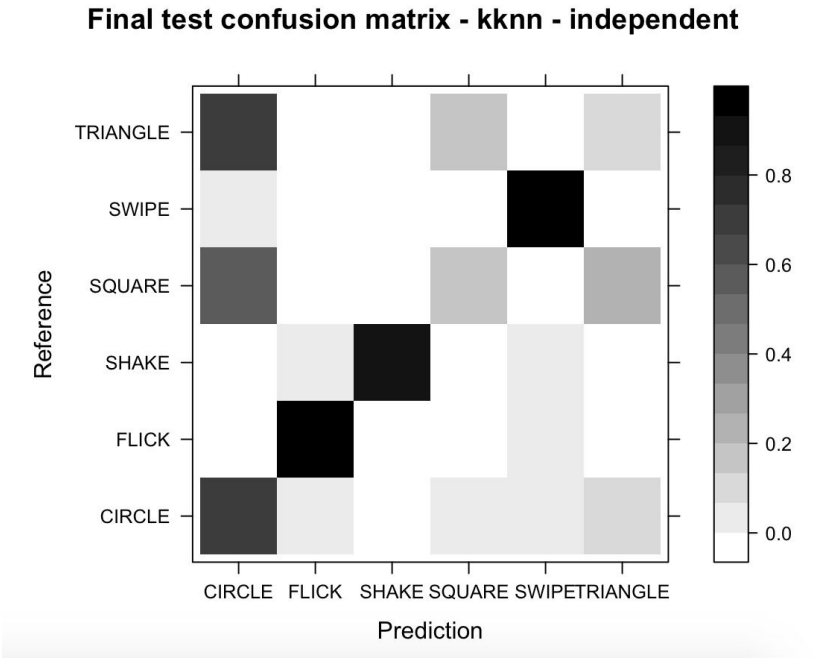
The KNN model also did not perform so bad, but the KKNN model seemed to perform good and should not overfit.

Independent model selection:

Here it is the same situation as for the dependent model selection. The random forest model is again obviously overfitted. Therefore the choice fell on the KKNN model.

Step 6: Evaluation





Conclusion

The population dependent model seems to work fine while the population independent model performance is not so good after all. Actually we expected already such a result.

The problem with the population independent try here is that we should have more samples here. That would already help a lot. The second big problem is that the excluded samples look already a bit different to the samples from others, since most persons did the gestures a bit different.

This means that the model learned the other persons which did the gestures more lookalike. And then the evaluation is done with a person, whose gestures are somehow different, that is why it does not perform so well.

The third problem occurred with the two gestures “triangle” and “square”. These two gestures tends to look similar to "circle" when the direction information is lost, since the have a similar peak frequency and length. Especially when these two are done a bit lazy, meaning that the person does not perform real stops at the edges, more like slowing down and speeding up again at the edges.

Afterall we think that the models are not that bad, but still there is space for improvement. The first big step would be to record way more data.