# MC520 Machine Learning – WS 2018/19
# Lab Report 2

Oliver Adam, Robert Gstöttner, Anastasiia Mishchenko, Roland Spindelbalker

25th November, 2018

## Toolchain

The toolchain we used consists of two main parts, the feature derivation and the model training. We tried several different approaches in both parts to get the best results for the image recognition task. Figure 1 summarizes the methods we used in each part.

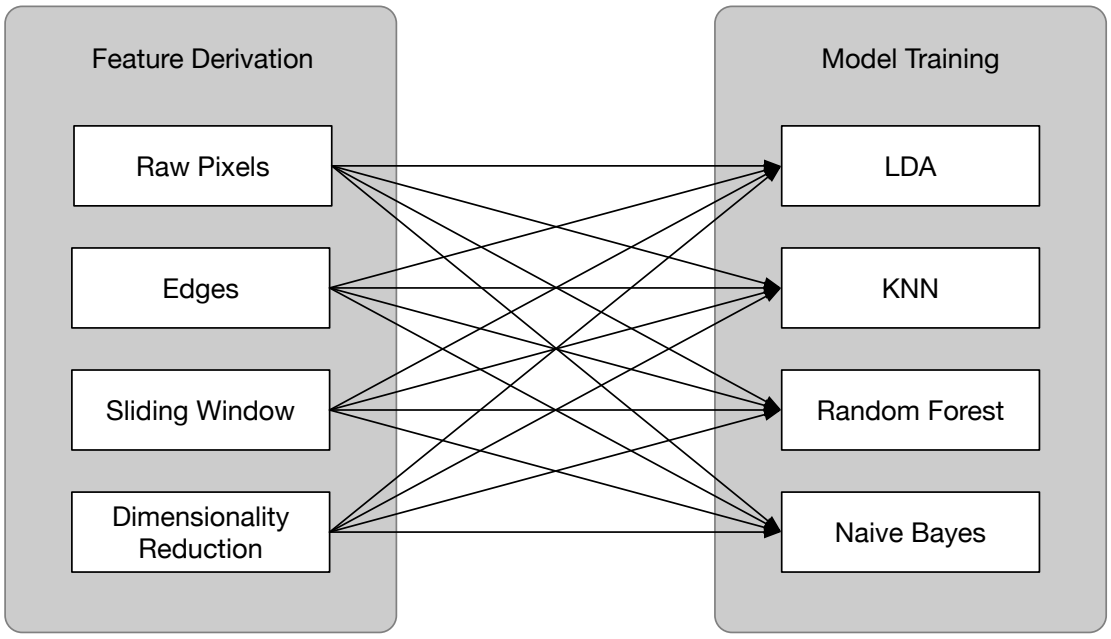



**Figure 1:** Toolchain

Each feature derivation process was used to transform the data into a new dataset which was then used to train the different models. So each model was trained with each feature derivation process which results in 16 different approaches. The dimensionality reduction process was performed with different thresholds which increases the number of combinations. However the best one was used in the end which also applies to the sliding window approach where multiple linear functions were applied to the windows.

## Data Analysis

The data consists of 574 preprocessed pictures in grayscale. These pictures are from 30 participants which are used to train models and derive features. To process these images they are collected and transformed into a data frame where a row is one picture and a column is a pixel. The first column indicates the person. Before using the data for models its also necessary to convert the person number into a "normalized" string by adding P and ensuring it is always (at least) 3 characters long. Then convert it to a factor (a class label).

To be able to plot the picture it is necessary to convert the picture raw data into a 50x50 matrix (pictures are 50x50) and convert it into a cimg, which can then be easly plotted (see image 2).

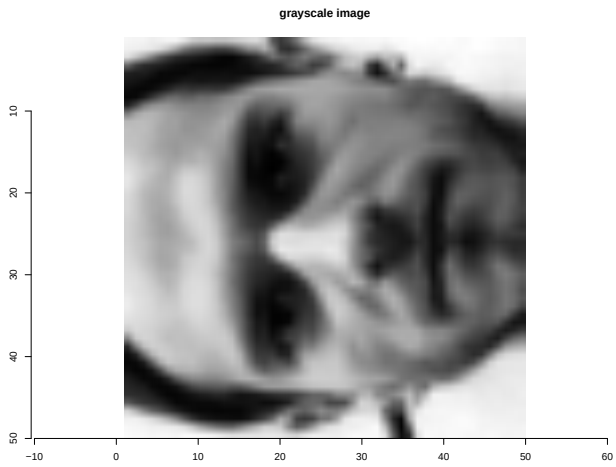



**Figure 2:** Grayscale image

Take some of the original pixel features of your data that lie new to each other (e.g. first 100) and do a correlation plot for them. What do you see?

The figure 3 shows the correlation between 100 pixels in the range from 1000 to 1100 pixels. These pixels are from the first image of the first person. The correlation can tell you that every 50 entries the image has a line break. It also shows that the correlation of the pixels is higher to its neighbours than others, especially in the top left corner. That means that neighbouring pixels have mostly the same grayscale and could be combined to reduce dimension and therefore features.

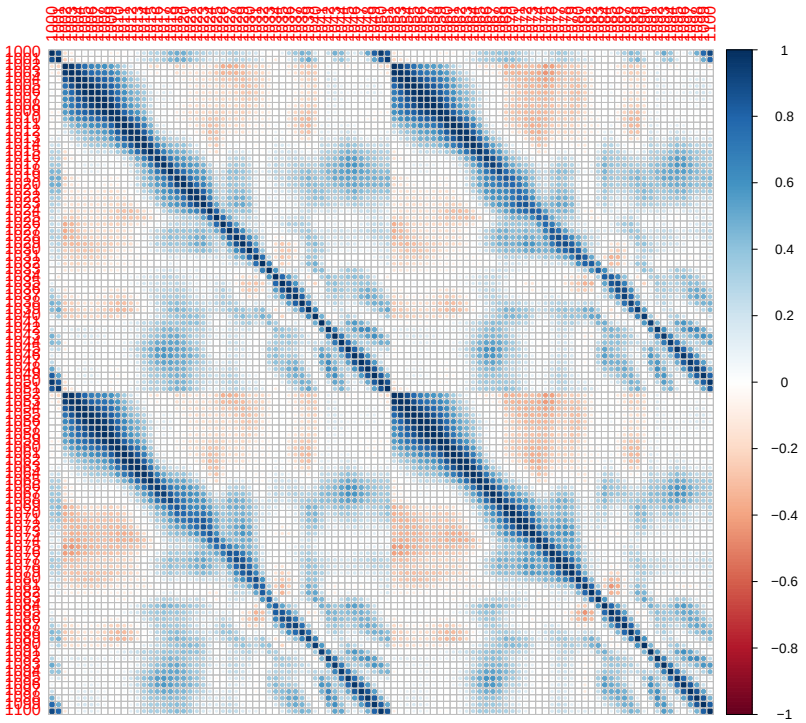



**Figure 3:** Correlation plot of nearby pixels (1000 - 1100 pixel in first picture)

Figure 4 shows the correlation of all the first 60 samples. The diagonal line from the top left corner to the bottom right corner represents the auto–correlation and therefore a perfect match indicated by the dark blue color. The small clusters of 16 correlation values indicate that 4 consecutive images are always very similar to each other and that the dataset is balanced. The correlation also shows that the raw pixel data could be used to distinguish between people. However due to the small size of the dataset used for the correlation plot (3 different people), this assumption could be wrong.

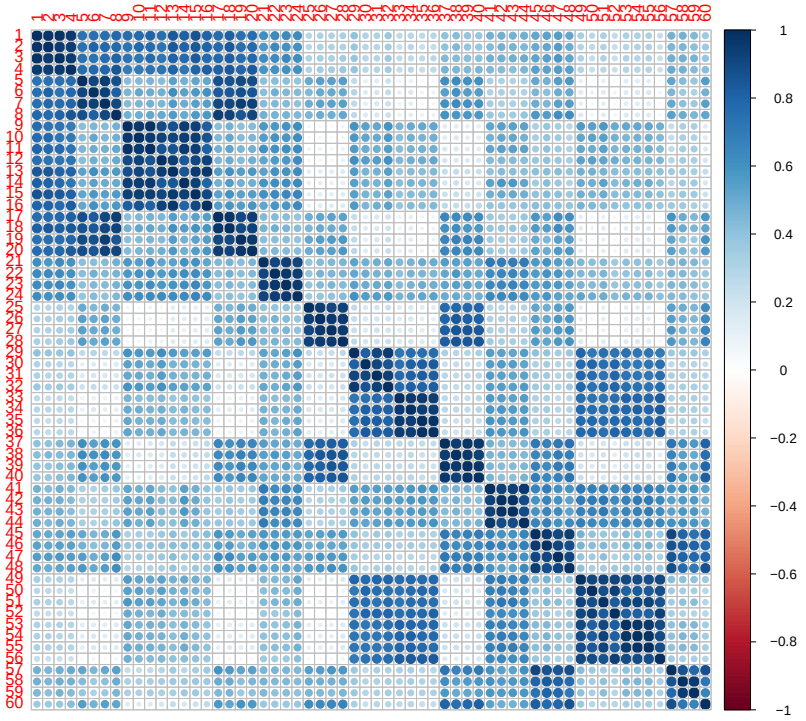



**Figure 4:** Correlation plot of all pixels for the first 60 samples

## Data Partitioning

75% of data is used for training and 25% for testing. This ratio was chosen because PS14 has only 5 pics (figure 5). With our distribution, at least 1 picture of each person will be used in model testing.

```
> summary(imgData$`1`)
P00 P01 P02 P03 P04 P05 P06 P07 P08 P09 P10 P11 P12 P13 P14 P15 P16 P17 P18 P19 P20 P21 P22 P23 P24 P25 P26
 20  20  20  20  20  20  20  20  20  20  20  20  20  20   5  20  20  20  16  20  20  20  20  17  16  20  20
P27 P28 P29
 20  20  20
```

**Figure 5:** Picture distribution

We also used K-Folded Cross Validation to ensure that the best model from the CV performance estimation is selected. We use 20 complete sets of folds to compute and 10 folds per set.

## Feature derivation

The first feature we used is of course the pixels itself. Because the pictures are nearly the same for each person this works quite well (better explained in models section). From that we derived two features. One is the edge detection and the other is a sliding window approach with various features.

### Edge detection

The edge detection is done via the magnitude of the gradient. First it is necessary to convert the raw image data to a 50x50 matrix to be able to use plyr without variable grouping. Then we can calculate the gradient along x and the gradient along the y axis of an image which can then be used to compute the Euclidean norm pixel-wise (the magnitude of the gradients). The magnitude of the gradients thus tell us how fast the image changes around a certain point. Image edges correspond to abrupt changes in the image, and so it's reasonable to estimate their location based on the norm of the gradient. The picture 6 shows the magnitude of the gradient of one picture.

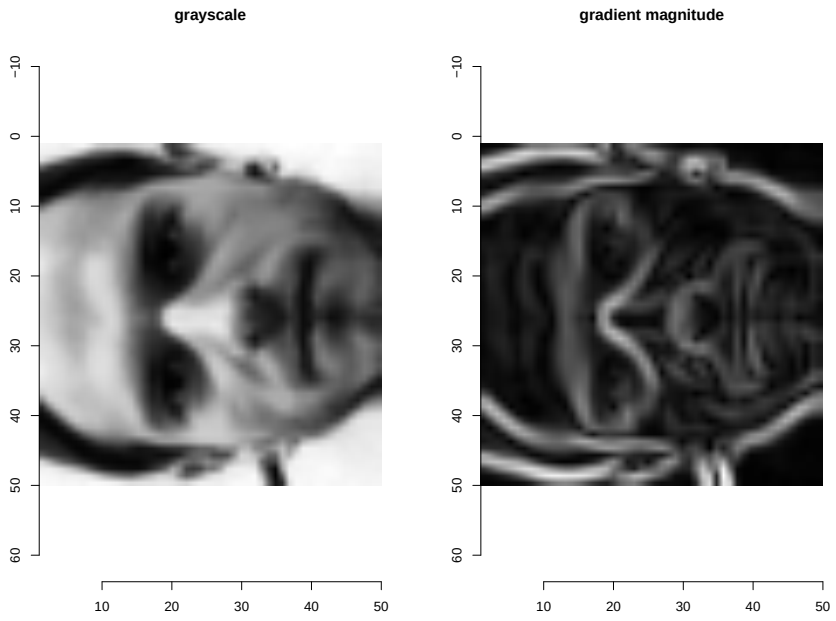



**Figure 6:** Gradient magnitude compared to preprocessed grayscale image

We then thought we could reduce the not so seeable pixels in the gradient magnitude via a threshold of 60% which takes only the 40% highest values in the picture (can be

seen in figure 7), but it only reduces the accuracy and Kappa, which is not what we want, so we didn't include threshold data in our training.

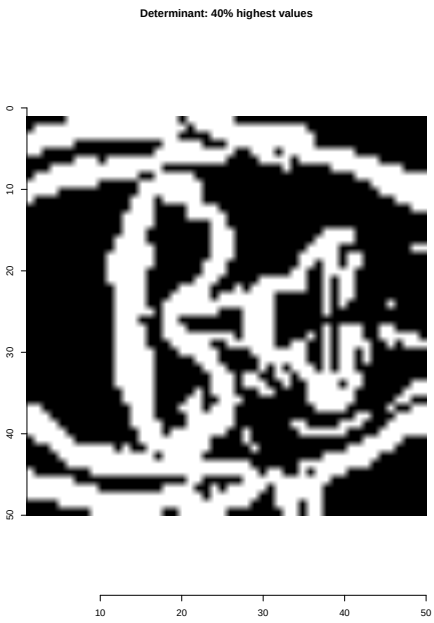



**Figure 7:** Gradient magnitude with threshold of 60%

## Sliding Window Approach

Because the correlation plot tells us that neighbouring pixels are holding the same information, we thought about going over the data with a sliding window and create new features out of it. To achieve this the data has to be converted into a 50x50 matrix which can then be used to apply a feature along the rows and columns with a sliding window. Later on the matrix is converted back to a data frame to be used for training. The following figures show the result of applying a sliding window approach with window size 4 and features like, median (figure 8), mean (figure 9), IQR (figure 10) and mad (figure 11).

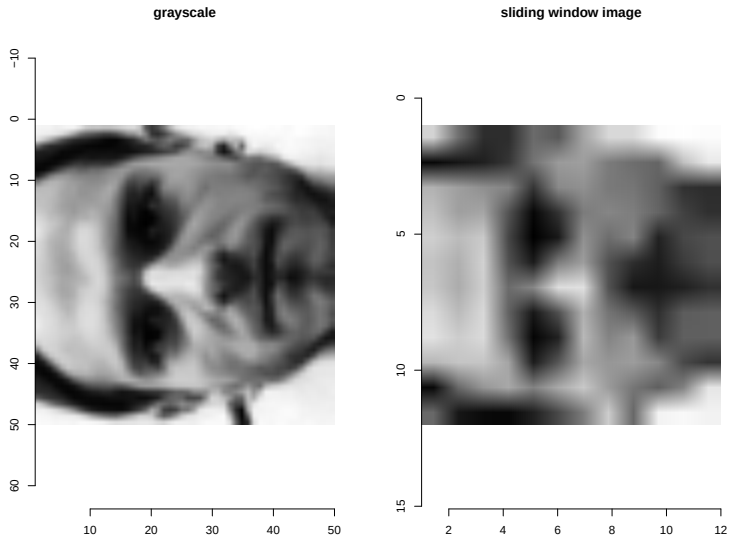



**Figure 8:** Sliding window approach with size of 4 and feature median

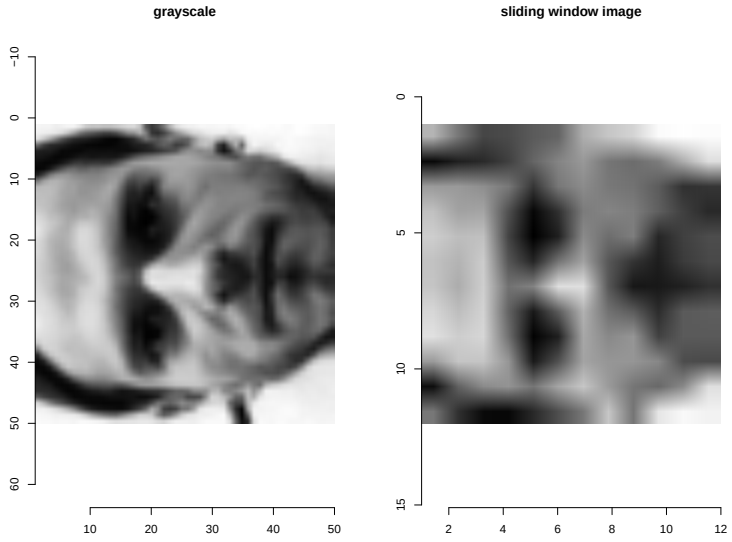



**Figure 9:** Sliding window approach with size of 4 and feature mean

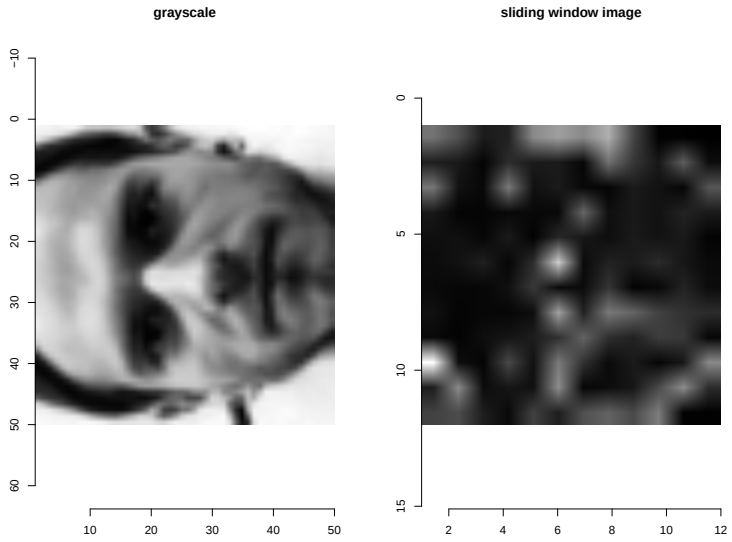



**Figure 10:** Sliding window approach with size of 4 and feature IQR

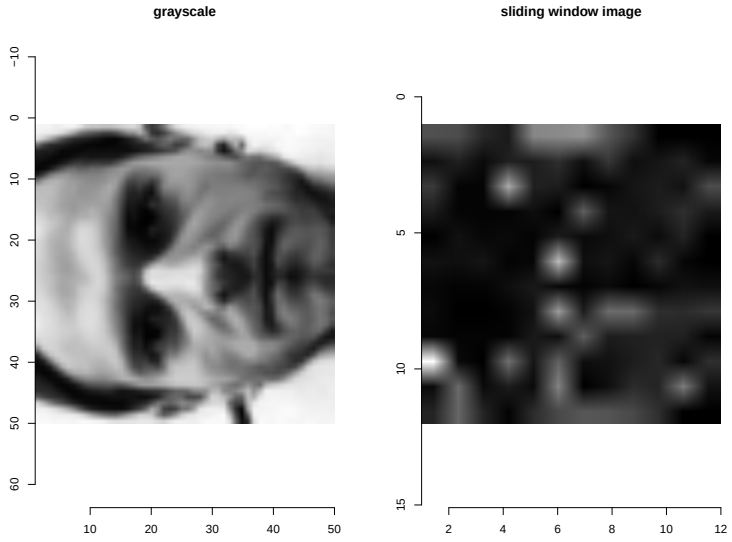



**Figure 11:** Sliding window approach with size of 4 and feature mad

The data, as you can see in the figures, is downsized by the number of 4 which gives us a image of size 12x12. IQR and mad gave us a very different image and tells us how different the pixels are to each other. The differences in model results can be seen in the model section.

## Dimensionality reduction

As the correlation plot in figure 3 shows, pixels which are near to each other are strongly correlated. This means that they don't carry much information and can therefore be removed.

To do so, we decided to use PCA to reduce the amount of features (dimensions), to make the problem easier. PCA allows you to configure the threshold up to which components are removed. To determine the threshold, the standard deviation of a component is used. If it is less or equal to the standard deviation of the first component times specified threshold value it is removed. We tried threshold values from 0.1 to 0.9 to test how much information we could omit without impairing the performance of the model.

In figure 13 you can see the downscaled images. It is clearly visible that the quality of the image decreases as the threshold value is increased. This of course does also affect the performance of the models trained with the downsampled images. The detailed results can be found in the Models training chapter.

We also tried out **Recursive Feature Elimination** to find the most important features. But the process took a very long time and the results were not really satisfying, so we discarded the idea again. Figure 12 shows the model performance when using the top 400 selected features.

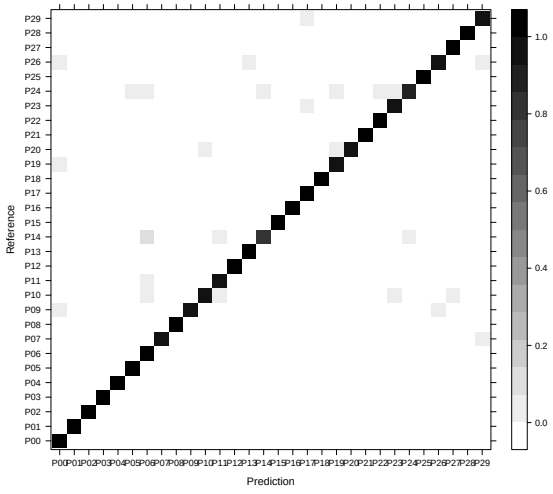



**Figure 12:** Confusion matrix for the top 400 selected features

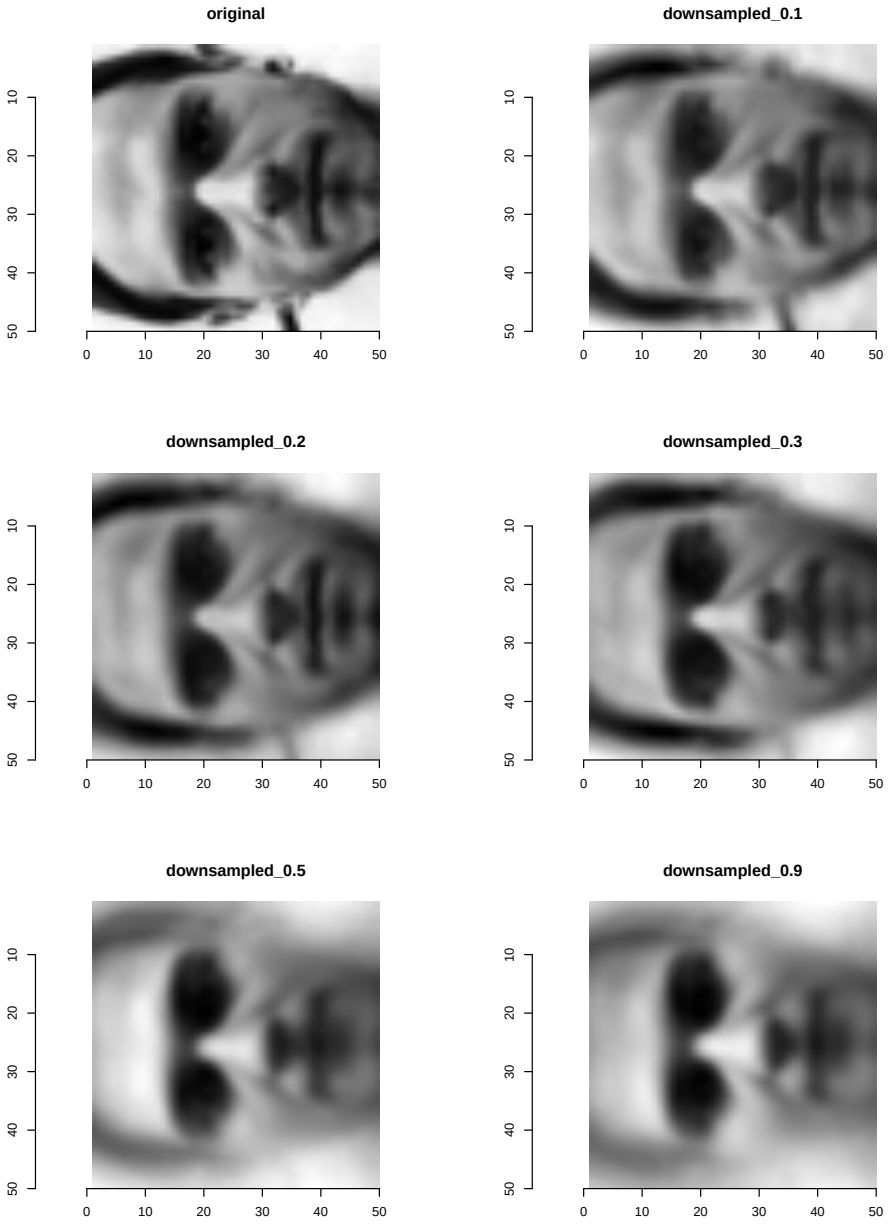



**Figure 13:** Comparison of downsampled images

## Models training

Model testing was made based on three data sets:

1. Raw Data - pixel data without any modification.
2. gMag Data - edge detection.
3. SW - sliding window.

After evaluating the training result, a model for the testing would be chosen and tested.

### LDA

LDA consist of properties of your data, calculated for each class. The mean and the variance are calculated for each class in case of a single input.For the multiple variables, means and covariance matrix are calculated, using multivariate Gaussian.

LDA simplify an input data, by assuming that:

1. Fed in data is Gaussian (each variable is bell-shaped when plotted).
2. Values of each variable vary around the mean by the same amount on average.

Based on the assumption, provided above, LDA estimates the mean and variance of input data for each class. It makes a prediction, based on an estimation that a new set of input belongs to each class. The class with higher probability is an output class (prediction is made).

Number of models computed per each dataset: 86 200

#### Raw Data

The result of the training is perfect (figure 14). The possible explanation is that the images are prepared very well. Such perfectness should be doubtful. Such accuracy generally means that the model is not over fitting, but it should be kept in mind, that can mean that something went wrong such as:

1. The data splitting was done wrong and validation data appeared in training data.
2. Some target leaking occurred and rows using information from the current target.

Another possible explanation is that the task itself is not so difficult for the model.

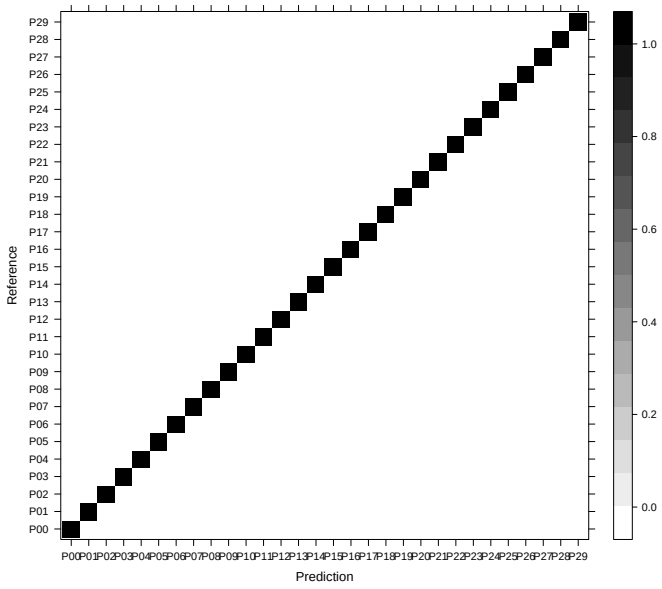



**Figure 14:** rawData LDA model training

### gMag Data

We can see that some of the samples were mispredicted in figure 15. As for P14, the reason might be that it has the lowest number of pictures provided. As it was stayed in "Guidelines for Best Practices in Biometrics Research" it was mentioned that the number of subjects in the dataset is known to impact the recognition accuracy. The accuracy (0.9980329) is very high, as well as kappa (0.9979564).

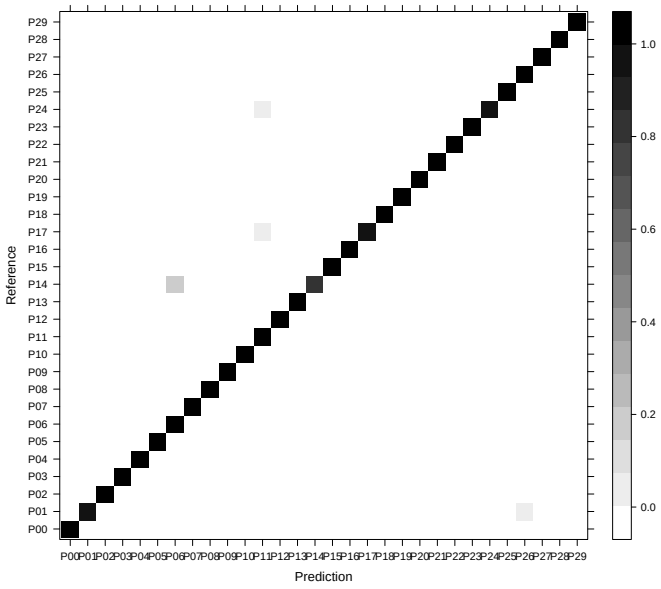



**Figure 15:** gMagData LDA model training

## SW

The accuracy (0.9987096) and kappa (0.9986588) was reduced compare to LDA full dataset. As it is presented in figure 16, we can see that some samples were already mismatch. The reduction of data influence the accuracy badly.

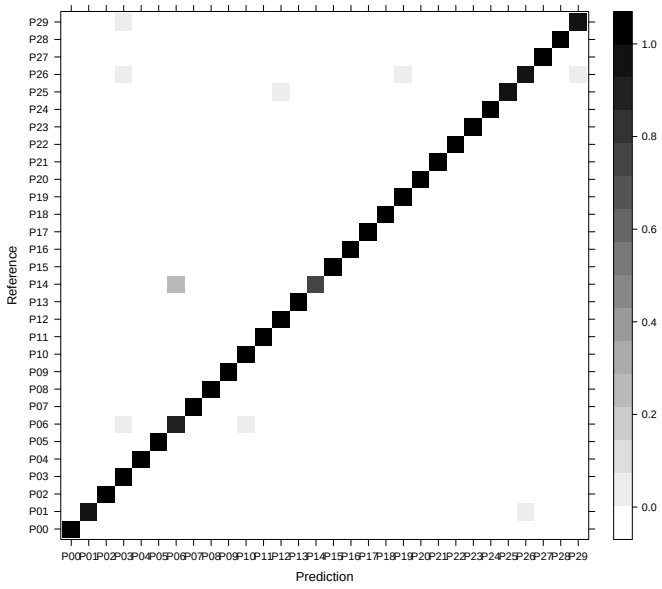



**Figure 16:** SW LDA model training

## PCA

The first thing we noticed is the notable performance improvement of the training process. When doing PCA internally it took much longer to train **one** model than it took to do the PCA manually and then train all of the models for the different threshold values.

Figure 18 shows the model performances after the downsampling. When a high threshold is used, the model is barely able to distinguish the people in the images, but when the threshold is decreased, the model performance increases. With a threshold value of 0.1, the model performed perfectly.

Figure 17 shows the performances for the different threshold values in one plot. Here, the rising performance with the decreasing threshold value can also be seen clearly. Also, notable performance jumps between 0.9 and 0.8, as well as 0.6, 0.5 and 0.4 can be spotted.

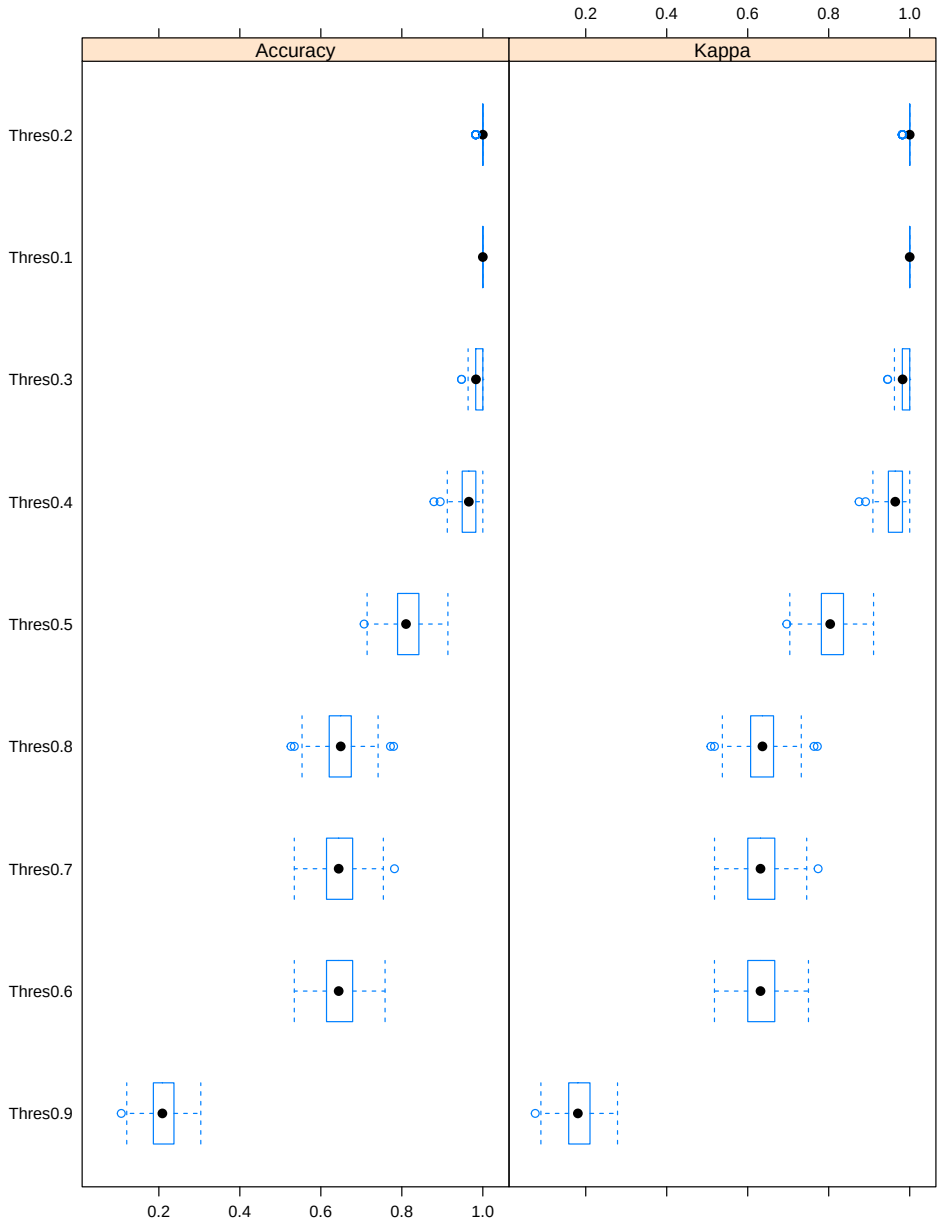
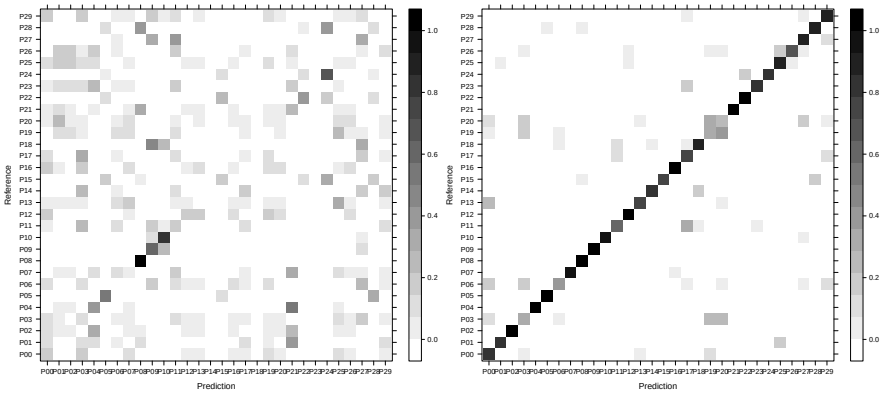
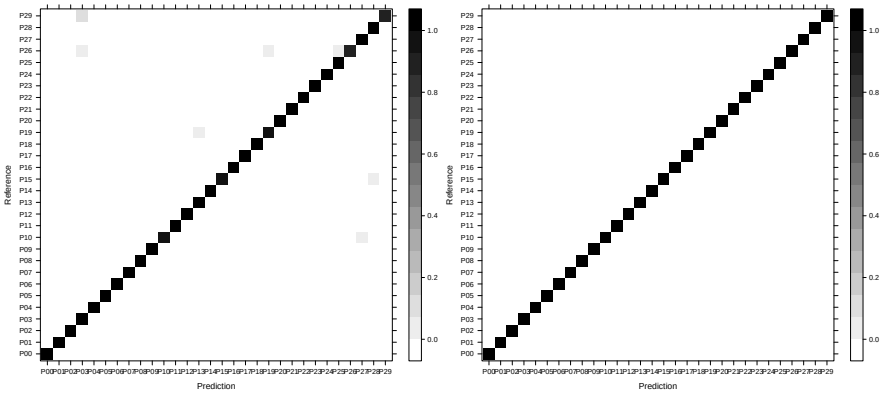



**Figure 17:** Comparison of training results with PCA




**(a)** Confusion Matrix for PCA downsampling with tol 0.9

**(b)** Confusion Matrix for PCA downsampling with tol 0.5




**(c)** Confusion Matrix for PCA downsampling with tol 0.3

**(d)** Confusion Matrix for PCA downsampling with tol 0.1

**Figure 18:** Confusion Matrices for LDA trained with PCA downsampled data

## KNN

KNN does not use training data to do any generalization. This lead to a pretty fast training phase in comparison with other algorithms. KNN stores all the training data because it needed during the testing phase.

It selects k entries (specified as hyperparameter) which are closest to the new sample and find the most common classification of these entries, after pass it to the new sample.

As a conclusion, KNN does not learn any model, it makes prediction just-in-time by calculating the similarity between an input sample and training entry.

Number of models computed per each dataset: 862 000

### Raw Data

The number of neighbours was set to 10, which extend the training time a lot. The final value used for the model was k = 1 (accuracy 0.9881389, kappa 0.9876733).

As it can be seen from the figure 19. The accuracy of the method run on the same dataset drop, compare to LDA results. There were more mismatches. P14 has the lowest accuracy in the diagonal, possibly due to the number of samples.

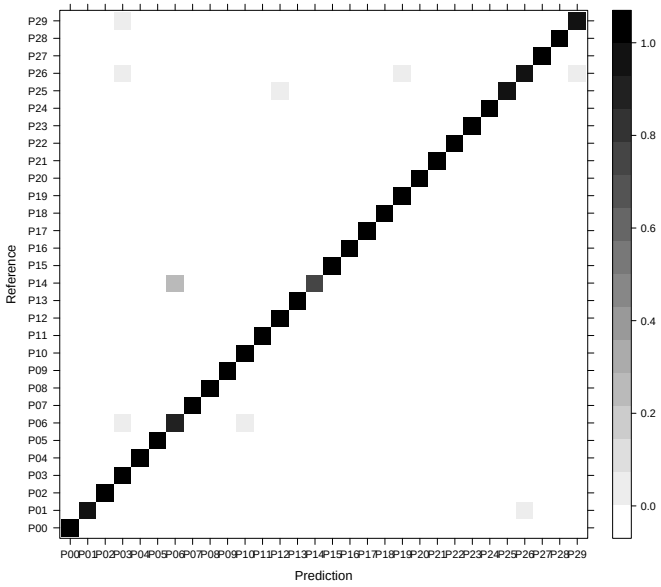



**Figure 19:** rawData KNN model training

### gMag Data

The accuracy and kappa of the gMagData KNN method also drop (0.9887185, 0.9882775 correspondingly). P26 has 4 mismatches.

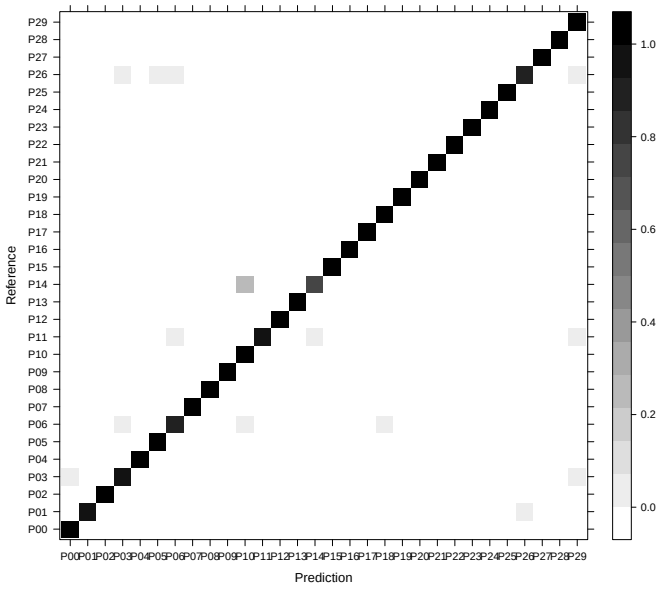



**Figure 20:** gMagData KNN model training

### SW

The final values used for the model is accuracy 0.9884187 and kappa 0.9879671. From the all data sets trained with KNN method, SW provide the biggest number of mismatches. The limitation of data influence good the performance and run time but the accuracy and reliability is dropping (figure 21).

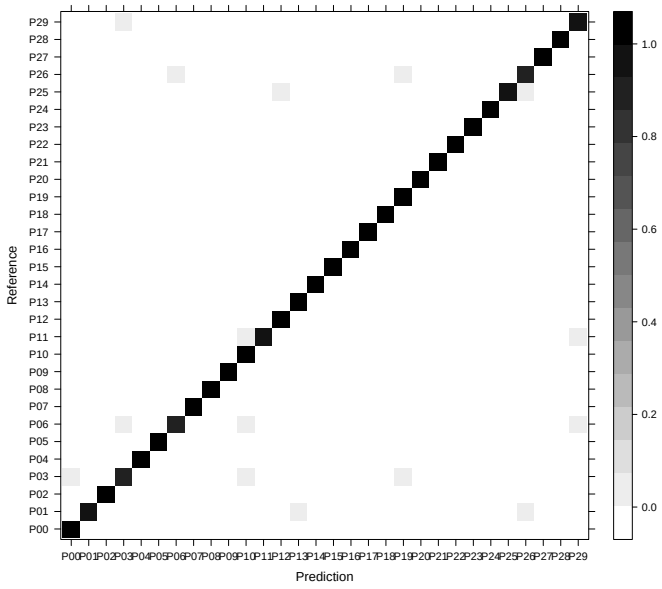



**Figure 21:** SW KNN model training

### Random Forest

While splitting the node, Random Forest search for the best feature among the number of features. This produces a wide diversity which leads to a better model. Only a random subset of the features is taken into consideration by the algorithm for splitting a node. It is one of the most used algorithms due to its ability to be used as in classification, as in regression tasks. If RF has enough trees, the classifier won't overfit the model which is true in most of the times. Another observation that it takes a lot of time to train but the output accuracy is very high.

Number of models computed per each dataset: 344 800

### Raw Data

As we can see from the figure 22 the accuracy over the 4 trees was different but still higher that KNN. The final value, used for the final model was mtry = 4.

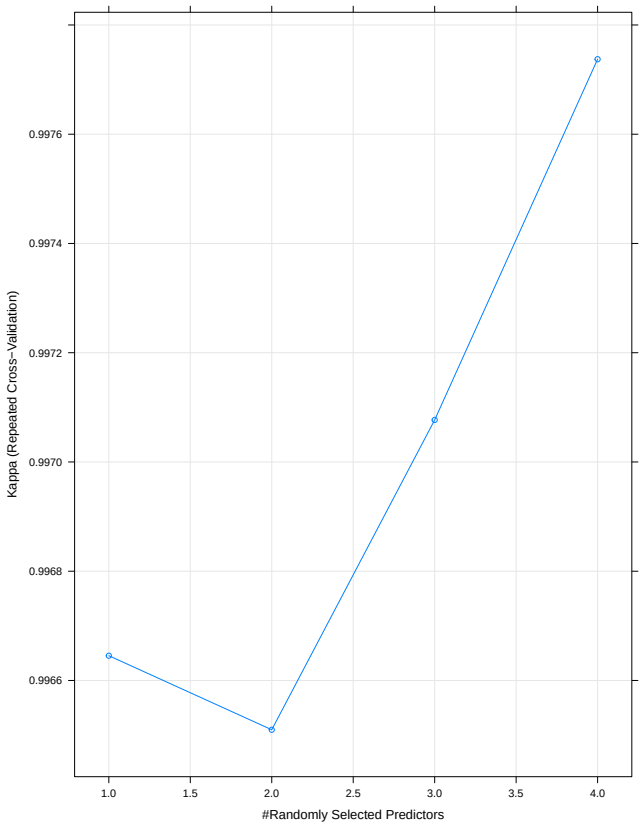



**Figure 22:** rawData RF model training

In the figure 23 we see that the mismatch prediction is minimal. It is still present but the accuracy is pretty impressive.

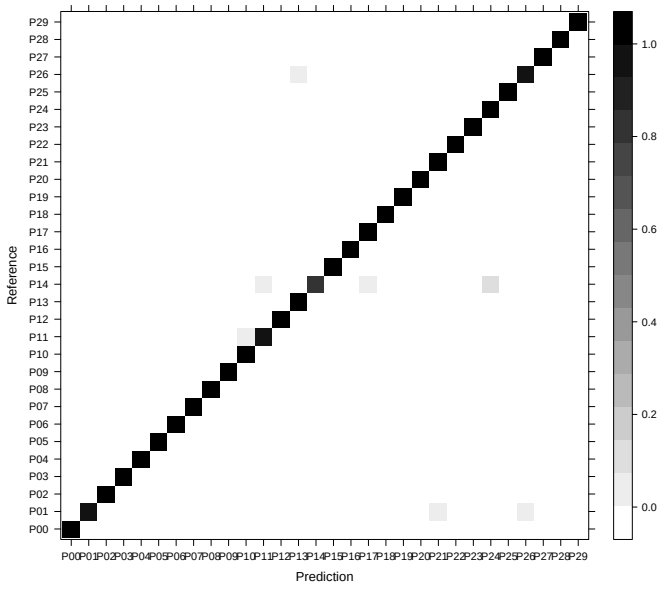



**Figure 23:** rawData RF model training levelplot

### gMag Data

The same situation we can observe with gMagData dataset. In the figure 24 the average accuracy is higher in comparison with KNN and LDA. The final value, used for the final model was mtry = 4.

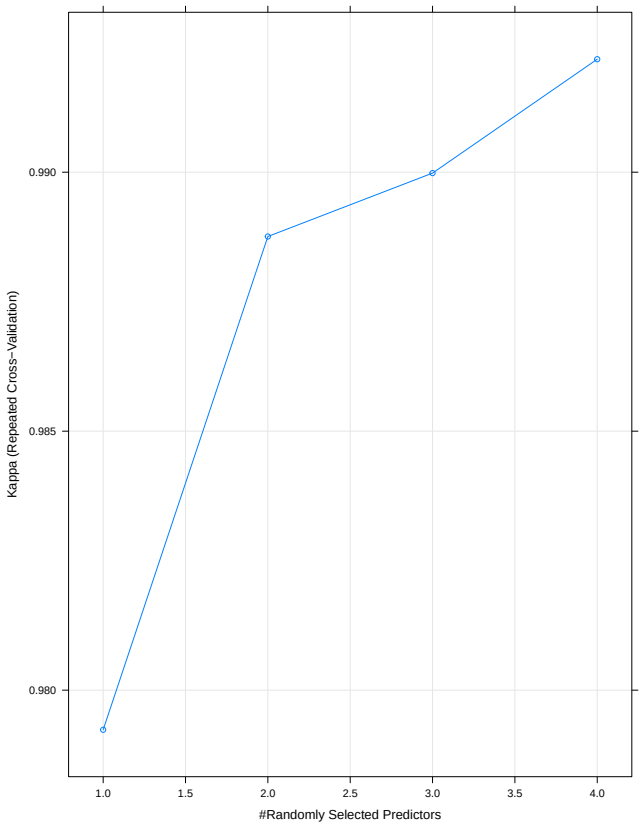



**Figure 24:** gMagData RF model training

In the figure 25 we can see that the model has more mismatches than KNN for the edge detection. That means that accuracy itself is not enough to choose the model.

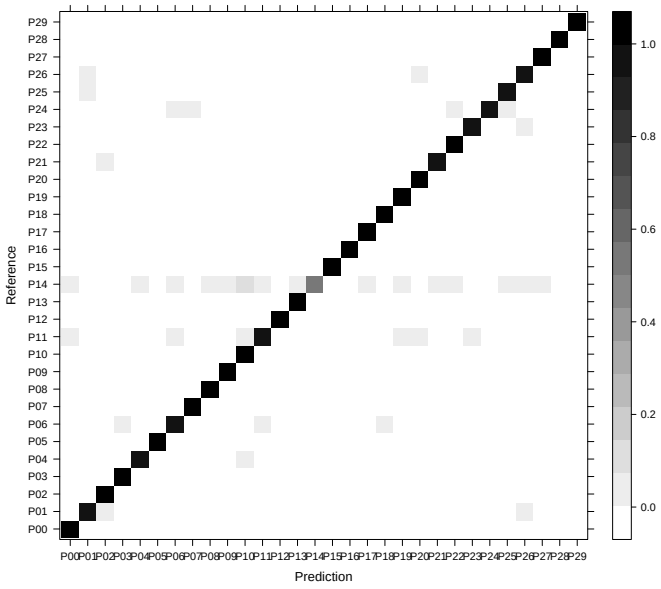



**Figure 25:** gMagData RF model training levelplot

## SW

Form the 4 randomly selected predictors, the second one was chosen for the final model because it provides the highest accuracy (figure 26). As for the full rawData set, the final value, used for the final model was 4. The accuracy is different as well (SW 0.9965019 vs rawData 0.9978214).

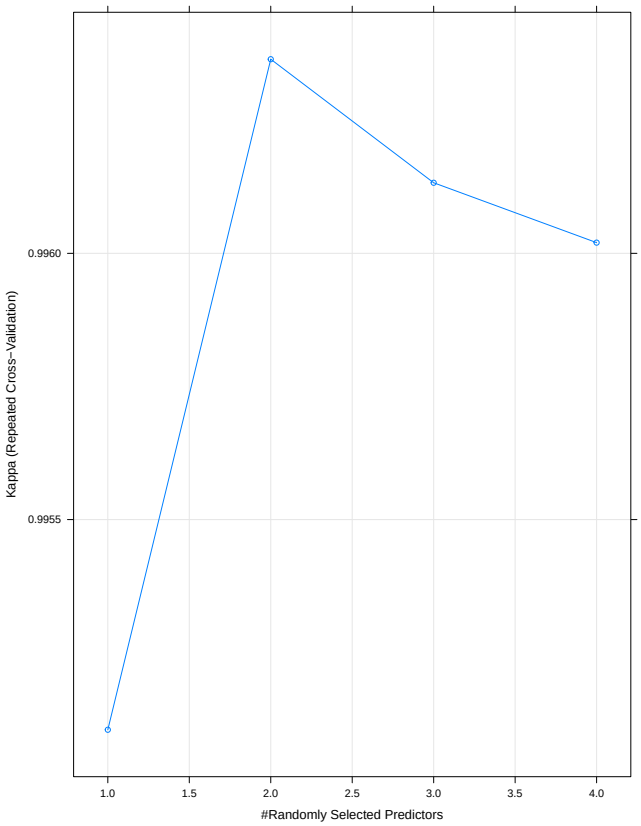



**Figure 26:** SW RF model training

As it seen from the figure 27 the model provide less mismatches compare to gMagData set but more compare to rawData.

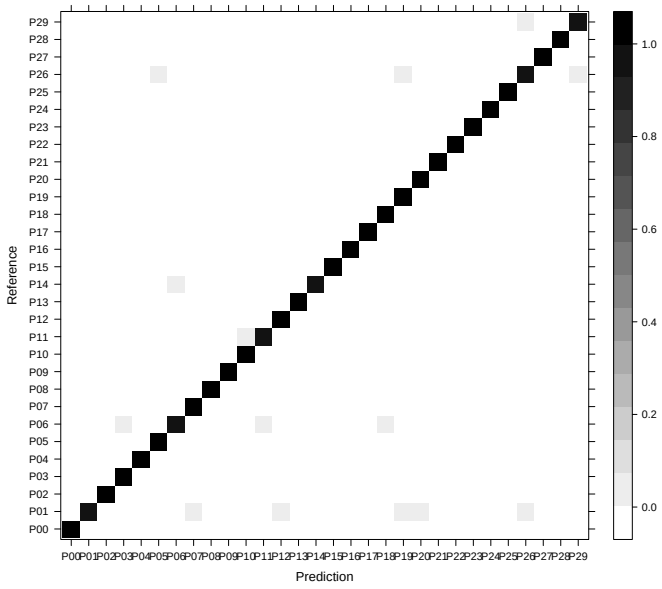



**Figure 27:** SW RF model training levelplot

### Naive Bayes

Naive Bayes classifier assumes that the presence of a particular feature in a class is unrelated to the presence of any other feature. It performs good in a multi class prediction. Rather than attempting to calculate the values of each attribute value, they are assumed to be conditionally independent given the target value.

Number of models computed per each dataset: 86 200

### Raw Data

As we can see in the figure 28, the output of the model, using NB method produce a lot of mismathces. LDA, KNN, RF provided more accurate and "clear" results.

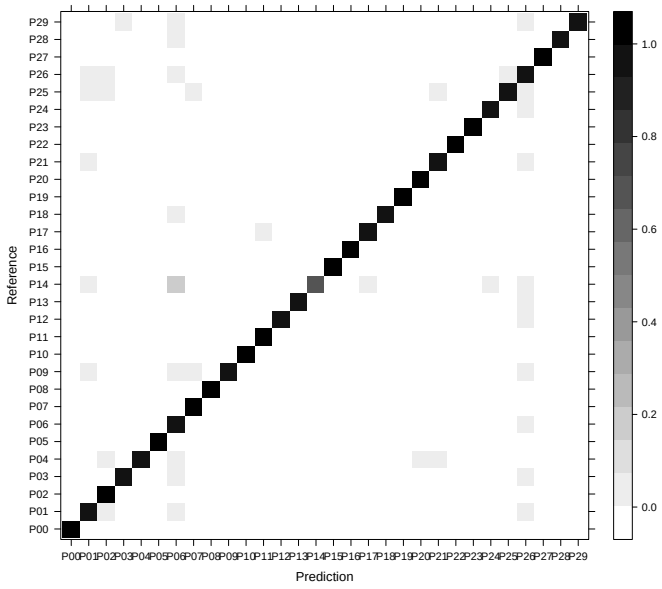



**Figure 28:** RawData NB model training levelplot

### gMag Data

The output of the model training (figure 29) is one of the worst, compared to other methods used. There were many mismathes despite the accuracy was still high.

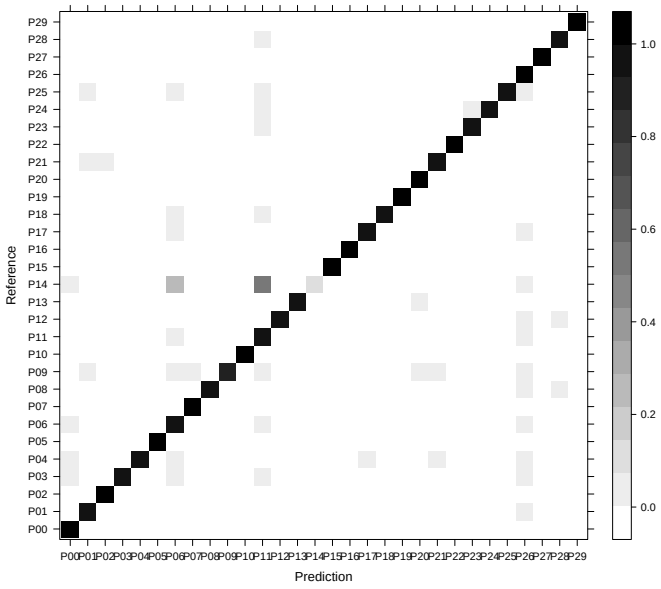



**Figure 29:** gMagData NB model training levelplot

SW

In general, NB perform less effective compare to LDA, KNN, RF for all three data sets. In figure 30 we can see that the P14 is predicted extremely bad.

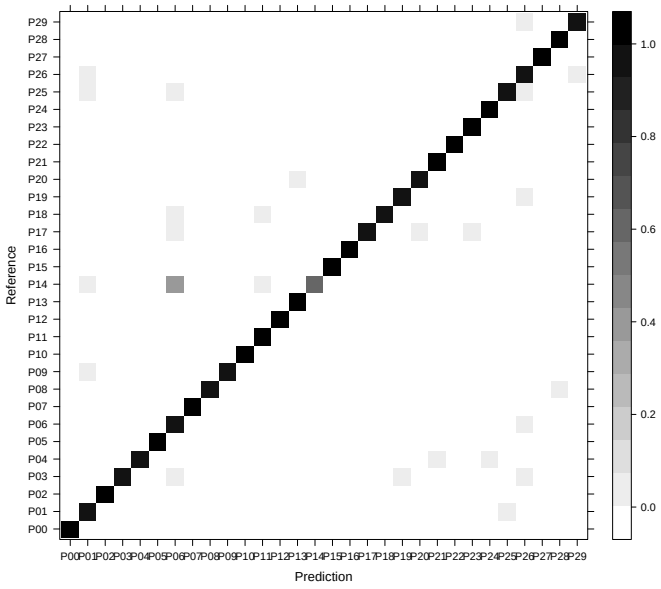



**Figure 30:** SW NB model training levelplot

For further exploring the correlation between features and accuracy, model training output. It was decided to repeat training for SW approach but with mean and compare results. To reduce running time of the training process, we choose only one model RF (figure 31) because it has the best accuracy.

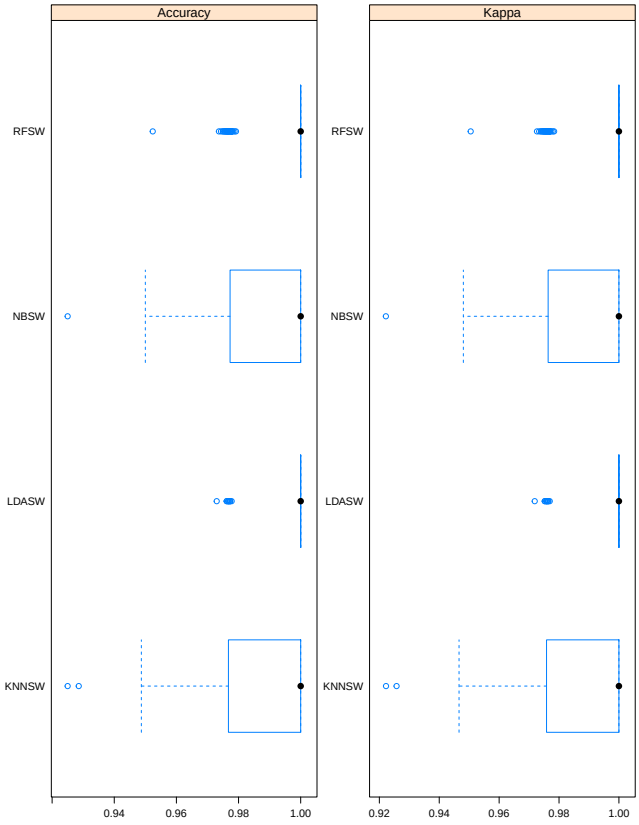



**Figure 31:** SW models comparison

There is no really correlation between accuracy and feature selected for SW (figure 32). With mode detailed look we can see that the difference in accuracy and kappa is tiny (figure 33).

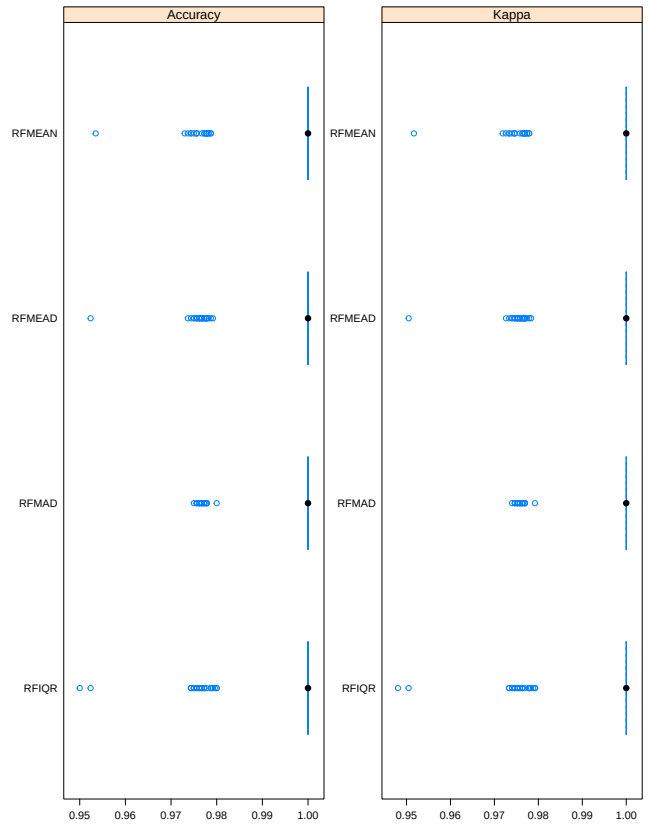



**Figure 32:** SW different features models comparison

```
Accuracy
             Min. 1st Qu. Median      Mean 3rd Qu. Max. NA's
RFMEAD 0.9523810       1      1 0.9965019       1    1    0
RFMEAN 0.9534884       1      1 0.9974242       1    1    0
RFIQR  0.9500000       1      1 0.9969945       1    1    0
RFMAD  0.9750000       1      1 0.9975688       1    1    0

Kappa
             Min. 1st Qu. Median      Mean 3rd Qu. Max. NA's
RFMEAD 0.9505009       1      1 0.9963646       1    1    0
RFMEAN 0.9516854       1      1 0.9973229       1    1    0
RFIQR  0.9480519       1      1 0.9968777       1    1    0
RFMAD  0.9740091       1      1 0.9974730       1    1    0
```

**Figure 33:** Kappa Accuracy models comparison

In general, all models provide a very high accuracy and kappa. But as it was seen on the RF and NB methods, the accuracy does not guarantee clear face recognition.
Models computed while training: 3 879 000

## Model testing

For the testing, the random forest method was chosen (figure 34) because it provides the highest accuracy which is not 1 (should be doubtful).

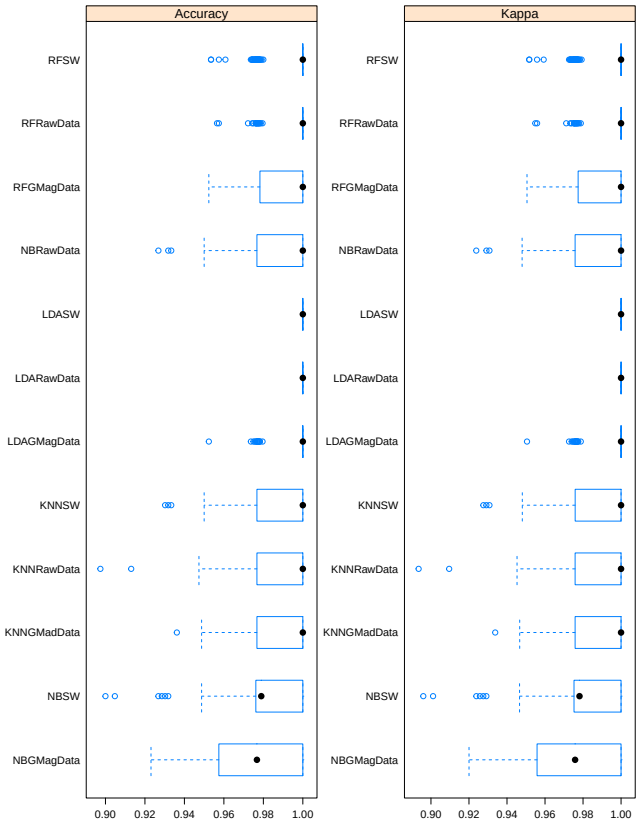



**Figure 34:** Models comparison

After providing the predict function with the model to test and test dataset (excluding ids), the output result could be seen in figure 35. The RF method did a very good job, there was only 1 mismatch between P10 and P27.

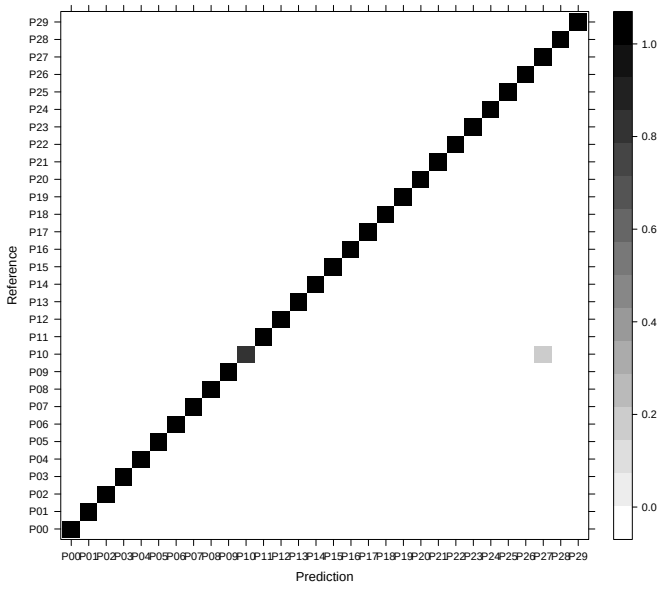



**Figure 35:** Prediction based on RF

## Conclusion

### What is the best approach (features, model, . . . ) you can come up with to successfully distinguish people?

The best model would be LDA with PCA and K-Fold Cross Validation in combination with pixels as features. But pixels as features are only good if all pictures taken are nearly the same. In the real world we would need more pictures from very different people in different positions in our dataset. Therefore pixels wouldn't be a good feature but edge detection seems promising to determine positions of eyes and other characteristics of a person.

### What do you think of the data/the results?

The raw image data is preprocessed very good already, meaning that all images have the same size, are in grayscale and are cut to only contain the face of the person. But taking this raw data directly for training is not really feasible, since the training takes very long, and not all the information (all the 2500 pixels) is really needed.

So the best course of action is to somehow reduce the information. This way, the

time to train a model can be reduced drastically, while the performance still is good. There are many different ways in which this information reduction can be achieved, e.g. by doing edge detection, by using a sliding window approach or by downsampling the image.

### What happens if we build our model from such a data set, then somebody not part of it uses a system where the model is deployed? What could you do about it/how could such systems possibly work?

One thing we found to be troublesome is that we can't know which preprocessing steps are taken exactly when it is done as part of the training process. This is insofar a problem, since the same transformations would have to be applied to a held-back test set. If we don't know which steps are taken, we can't redo them and the testing would yield invalid results.

The model will always try to match the given data to one of the images in the population. If the image of an unknown person is used, the result will be the person from the population that has the closest resemblance to the unknown one – a false positive. To prevent this, we thought about adding a "default face" to the population, which will match all faces that are not within the population. Faces within the population would not be matched with it, since there are better matches present. This way, persons that are not part of the system could be recognized as such.

Another solution is not to use the model as standalone solution. Instead, it could be part of a bigger biometrical trait recognition system. Along with the face image, other demographic attributes should be used to distinguish persons. Additionally, it should be ensured that the dataset used for training is representative of the population and environment where the system will be used. Labels associated with individual samples should not only include subject identity, but also some other demographic attributes. The accuracy of the face recognition will rely not only on the model but also on additional parameters which will prevent (or at least reduce) overlapping.