Logbook ML Competition

Groep 15
Anthony Beerens
Simeon Bruyland
Miguel Dagrain
Joris Vandenbroeck
Ward Vercouter

1 Data analysis

The goal of this project is to classify a finite set of 16 hand gestures as accurately as possible. We are given a training set of 2191 samples and a test set of 541 samples. Each sample consists of data points of a person who is doing a hand sign. These data points are divided into a variable number of frames (1 to 117). Each frame consists of 125 data points and has a fixed structure. The first 23 data points in a frame represent the body, the next 60 data points represent the face, and the last 42 data points represent the left and right hand. Also, each hand gesture in the data set is executed by a certain signer.

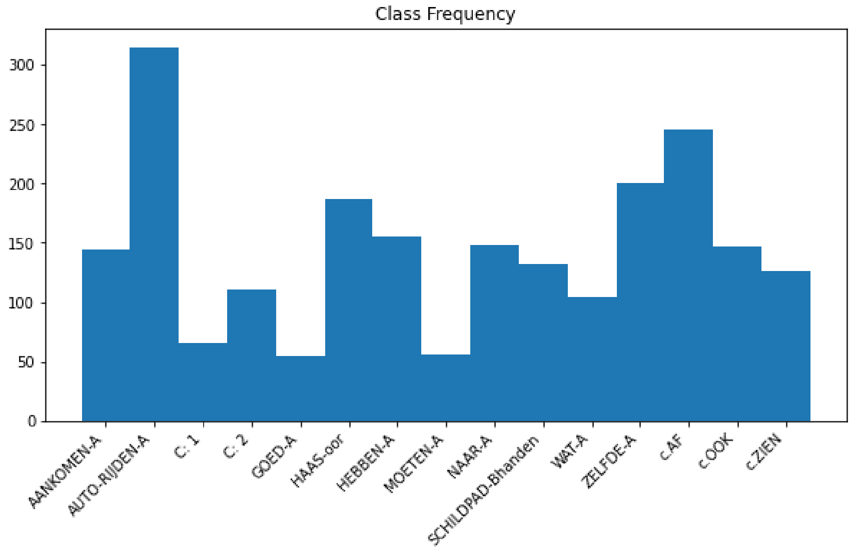


Figure 1: Class frequencies

When looking at how many key points are equal to zero over all samples, it can be seen that it depends on the whole body part. It does not occur that a single key point is equal to zero, it will always be the entire body part, i.e. body, left hand or right hand. The key points of the face are never equal to zero. It often occurs that a body part was not detected any more during one or more frames. In those cases, the coordinates, and hence all features derived from it, would suddenly drop, which is not wanted behaviour and has a negative impact on the training of a model. Therefore, it is beneficial to correct these scenarios.

Another scenario that is present in the provided data set is where the hand position suddenly changes completely. When plotting such a sample, it can be seen that the hand that has changed is very similar to the hand that remains in its original position.

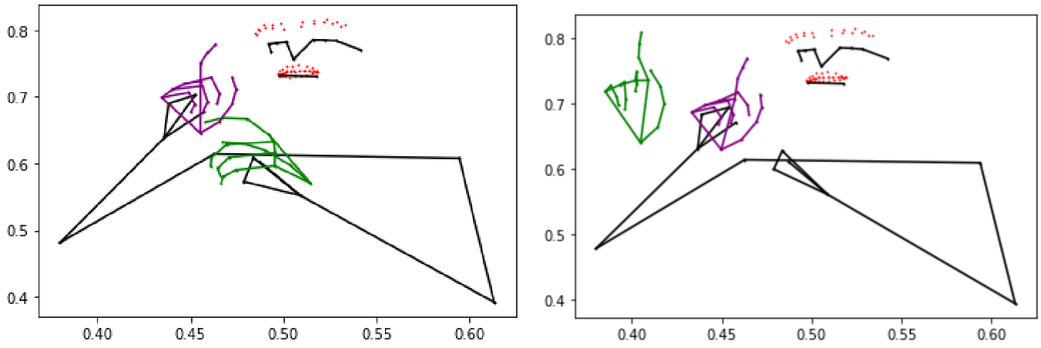


Figure 2: Misplaced hand

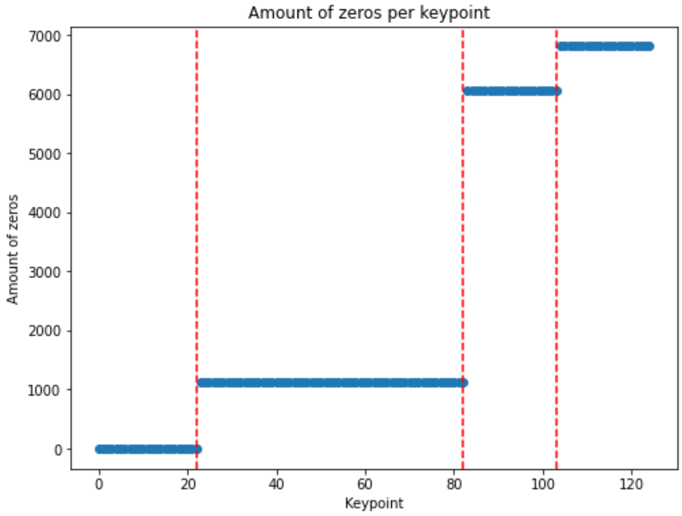


Figure 3: Zero values per keypoint

2 Pipeline

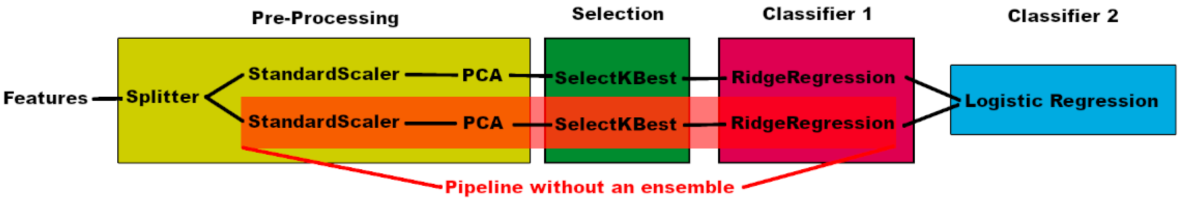
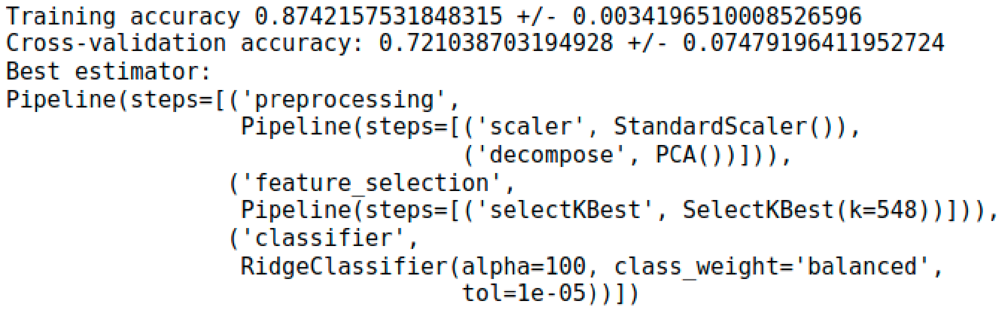


Figure 4: Pipeline

1. (For the ensemble) Feature separation to create diversity between the classifiers, one classifier has access to body, the other only access to the hands
2. To perform standardisation so that large numeric features don't dominate. StandardScaler was used instead of RobustScaler because there were few large outliers in the data set. MinMaxScaler would overfit on outliers because it would cram most features in a small range.
3. PCA is used to make features orthogonal and reduce cross-correlation between features. There were no features selected based on the PCA, this was left to the feature selection.
4. Feature selection using SelectKBest, based on variance of features between classes. Hypertuning on k (number of features).
5. RidgeRegression classifier(s) , hypertuned on alpha (to improve generalization), class weight is set to balanced to compensate for the bad class distribution.
6. (For the ensemble) LogisticRegression classifier based on probabilities sent by previous stage to make final decision, hypertuned on C. Also tested several solvers; best results in cross-validation with liblinear but "lbfgs' had less accuracy variation between classes.

To do feature engineering, a baseline is needed to compare results and to decide if a new set of features improves the accuracy of the model. For the baseline model, all samples are simply transformed into two frames by averaging and combining half the frames into one (*transform_to_K_frames*). This gives the same amount and kind of features as in the first stage of the competition. The hyperparameter grid consists of different K-values for *SelectKBest* and several alphas for the *RidgeClassifier*. The Cross Validation accuracy and confusion matrix are shown in figure 5 and 6.



```
Training accuracy 0.8742157531848315 +/- 0.0034196510008526596
Cross-validation accuracy: 0.721038703194928 +/- 0.07479196411952724
Best estimator:
Pipeline(steps=[('preprocessing',
                 Pipeline(steps=[('scaler', StandardScaler()),
                                 ('decompose', PCA())])),
                ('feature_selection',
                 Pipeline(steps=[('selectKBest', SelectKBest(k=548))])),
                ('classifier',
                 RidgeClassifier(alpha=100, class_weight='balanced',
                                 tol=1e-05))])
```

Figure 5: Class frequencies

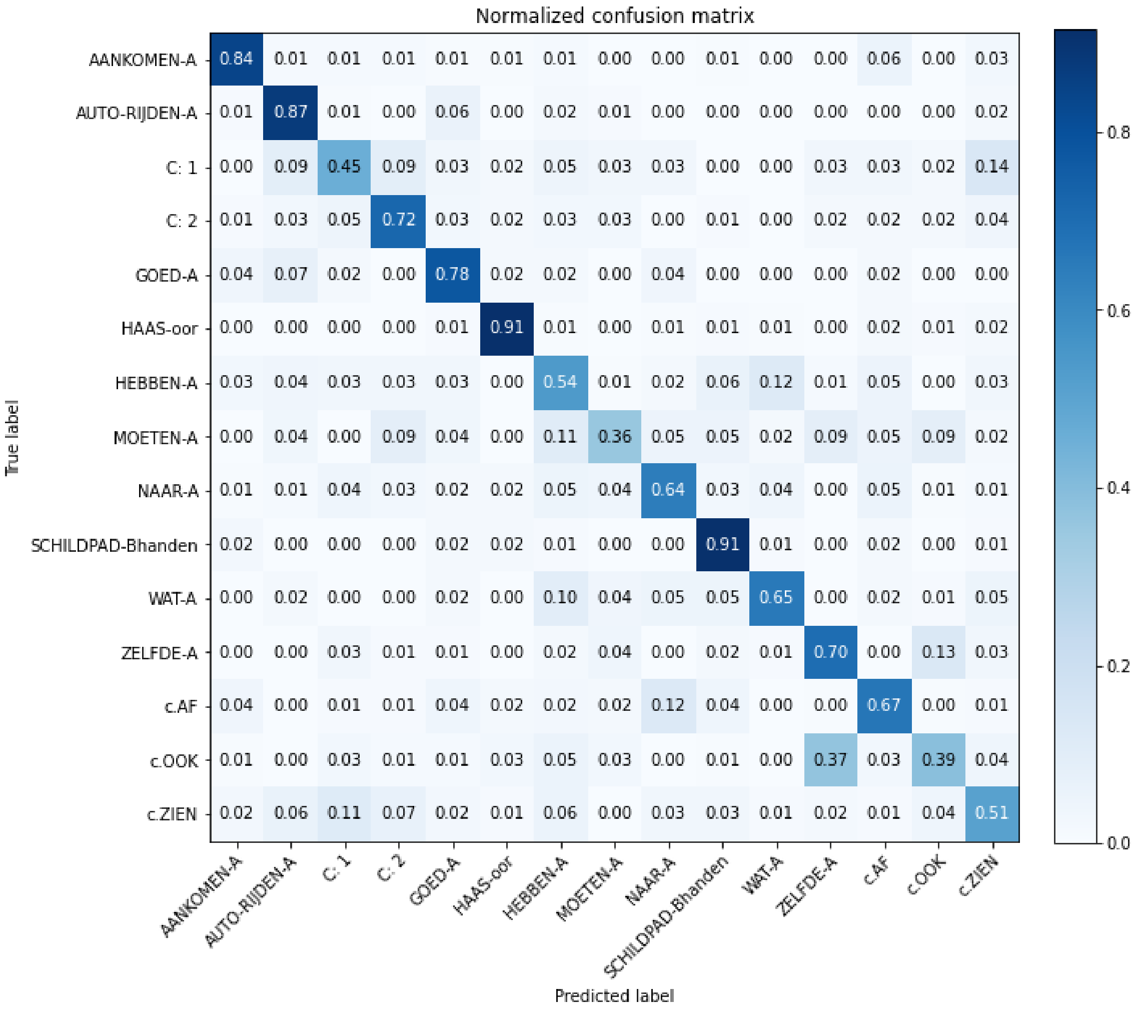


Figure 6: Confusion matrix

3 Validation strategy

As seen in figure 1, there is a large imbalance between the classes. If the original distribution would be used, the model would be biased towards the most represented classes in the training set. This was confirmed by a very low accuracy for *C:1*, *Moeten-A* and *Goed-A* in the confusion matrix. To prevent this imbalance, the “balanced” mode in the *RidgeClassifier* is used to automatically adjust weights inversely proportional to class frequencies.

As a validation strategy, *Stratified Grouped K-Fold Cross-Validation* is used. The reason for this is the information leakage between samples of the same signer, the groups are set on the signers. Also, the already existing imbalance between the number of samples is handled to a certain degree. Because stratification is used and thus, the *Cross Validation* object tries to split the samples in identically distributed folds as good as possible while keeping the signers separated.

K is set on 14 given that there are 38 signers and not all signers have the same amount of samples or the same distribution of labels in those samples. 14 gives the possibility to have 2 or 3 signers per group depending on how many samples they have.
Of course, there are other K's around 14 on which the same argument applies. Those were tested and found to have a larger difference between cross validation score and test score.

4 Data Cleaning

The quality of the predictions is determined by the quality of the input data. As seen in figure 3, there are a lot of zero values in the training dataset. In such a sample, either a complete body part was missing over the whole sequence of frames or a complete body part was missing in a few consecutive frames. Multiple approaches were taken to clean the data: Correcting zero values in frames by interpolating, throwing away bad samples, moving hands to wrist of the body and straightening the body pose and centring the data points around the shoulders.

Zero correction with interpolation

When a sample consists of some frames with missing body parts, neighbouring frames can be used to approximate these missing body parts. By looking at the surrounding frames and selecting the closest ones that are not equal to zero, the sample can be restored. Furthermore, by interpolating these body parts over the amount of frames that the non-zero body part is away, a sample can be restored more precisely. In the implementation, this is possible when each body part appears in at least one frame.

Throwing away bad samples

In some samples, a body part is missing over all frames. This sample is then less representative; e.g.if the hand raising the index is missing for a *C:1* sample . The problem is that it is not known which hand is important for recognizing the given sample. By throwing away these less representative samples, the model could be improved as it would be less subject to noise. This was tested and the overall *Cross Validation* score became approximately 1% better. However, in this dataset, almost all samples still contain at least partial information (e.g at least the body points are present, even if the hands are absent). If these samples were thrown away, the dataset would no longer be representative, as test data can be imperfect as well. Also, the most underrepresented classes would contain even less training samples. Therefore, all samples were kept in order to make the model more robust against these imperfections.

Moving hands to the wrist

There are some samples in which a hand is far away from wrist data points of the body, an example of this can be seen in figure 2. Also, many samples have incorrect Z-values which do not have a positive impact on classifying hand gestures. By moving the wrist data points of the hand to the wrist data points of the body, these unrepresentative samples can be corrected. This translation has positive effects on other extracted features that make use of the position and movement of the hand data points, e.g. distance from eye to hand.

Straightening and moving the body pose

To reduce variability in the data, the whole body pose was rotated such that the body of the person has the same orientation in each sample, i.e. shoulders in line with the X-axis. Also, all data points are translated such that the left shoulder is located at *(0,0)*. A comparison of an example can be seen in figure 7.

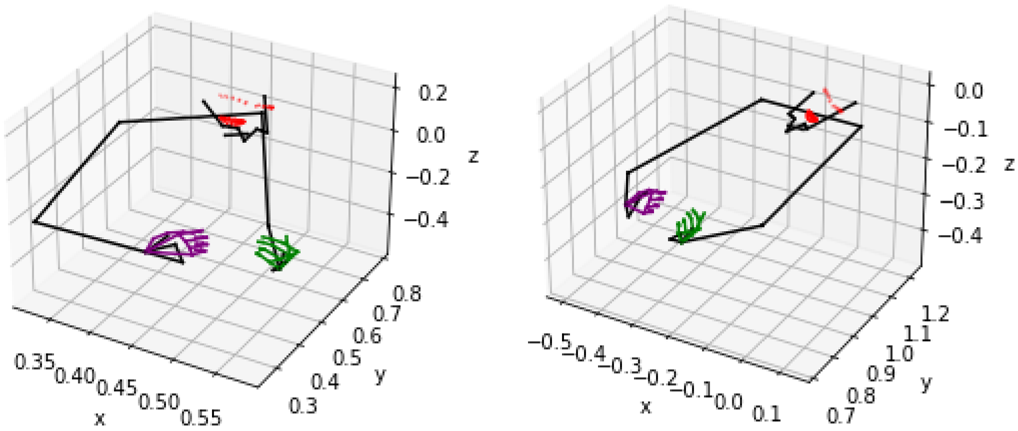


Figure 7: Straightening and moving body pose

5 Labels to improve

From our baseline model, a confusion matrix is constructed to get an idea of which labels are getting classified incorrectly. From this matrix, it is clear that there are a number of labels which are predicted (relatively) accurately, and a number of labels which are getting confused with other labels.

The first two classes that are very similar are *C:1* and *c.ZIEN.* 14% of the samples from class *C:1* are predicted as *c.ZIEN* and 11% from *c.ZIEN* are predicted as *C:1.* These signs have indeed a lot of similarity when looking at the videos provided. The only big difference is that the signer points to the eye when performing *c.ZIEN* and comes a bit closer to the face, whereas a signer performing *C:1* would keep the index finger more straight.
To distinguish these classes from each other, the idea was to design a feature that represents the minimum distance during a sequence between the index finger and the eye. Difficulties here are that the z-axis of many key points is not very accurate in the data set, some signers do not come very close to the eye or do not always use the same hand. Due to these difficulties, the model did not improve in distinguishing these classes.

The hardest classes to distinguish are by far *c.OOK* and *ZELFDE-A*. These signs are completely the same when it comes to the position of the body and hands. The only difference is in the amount of times that the hands come together: one time for *c.OOK* and two times for *ZELFDE-A*. Due to the lack of precision in the time domain, it is very difficult to design a good feature that can separate these classes.

Certain labels like *HEBBEN-A, WAT-A* and *MOETEN-A* are quite similar to each other. For these labels, the movements and positions of the hands and fingers are often quite similar.

As an example, with *WAT-A* and *HEBBEN-A*, the main hand moves in an up and down fashion, and the positions of the fingers are extremely similar. The only real difference is the angle of the hand compared to the body.

In order to try to differentiate between similar classes, a classifier was trained which was only rewarded when able to differentiate between these pairs. Even these hyper-specialized classifiers were not able to consistently differentiate these classes, leading to the conclusion that the current feature set is simply not good enough to differentiate these pairs.

In addition, a small script was developed in order to see whether humans could consistently differentiate between *c.OOK* and *ZELFDE-A*, in which participants generally scored 60-70%. This indicates that even for humans, the difference in the data is difficult to perceive.

6 Features

6.1 Domain Knowledge

Transforming into K frames

The first idea was to extend the methodology from stage one, which consisted of splitting the data into two by averaging over the first and second halves. This was done by creating a function which split the data into K frames by averaging or duplicating, depending on how many frames are given.
Setting K to two or three resulted in the best results. However, going even higher than this resulted again in a worse model, as too many insignificant features were added in again.

Polar coordinates

Some signs have more of a circular movement to them. So a conversion from Cartesian to polar coordinates was tried, because polar coordinates are better for circular shapes. However, in practice this did not result in any improvements, because most signs appear to have more of a translational movement than circular movements.

Remove facial features

Not all body parts are equally helpful in correctly labelling signs. By removing unrepresentative features in the face, the linear model would not be biased trying to recognize correlations where there are none.

Motion vectors

The signer's position across samples is not consistent. By translating absolute positions into movement vectors, the dependency on the initial position is strongly reduced, albeit not entirely, since the original position was also transmitted. When doing this without adding the starting position, this resulted in a severely worse model as too much information was thrown away. Putting a starting position back in resulted in a model which did not improve the model, likely due to variance between samples of the same label.

Variance

As another attempt to encode motion whilst reducing the amount of data, taking the variance of the positions across frames was attempted. This reduced the data to a relatively small set of features. This feature had a negative effect on the Cross Validation score, as a lot of information was discarded.

Angles

In the original data, there is a lot of correlation in the positions between body parts. To reduce this correlation, encoding the data as the angles between key points was attempted. Features like angles of shoulders, wrists and forearms were tried but did not result in an improvement of the Cross Validation score. It gave very similar results to the original features. Which could be expected as no information was discarded (besides the exact positions of the key points) and no information was added.

Stretched Fingers

By computing the correlation between the *x* and *y* key point of each finger, a measure of how stretched or curbed each finger was on average over the whole sequence could be computed. This feature was consistently discarded by SelectKBest as it did not present enough variance between classes.

6.2 Based on results

Minimum distance eye to finger

Based on the often high misclassification of *c.ZIEN* and *C:1* (in both directions) one of the ideas was to look at the distance between the finger and the eye. This assumes that *c.ZIEN* moves closer to the eye than *C:1*. However small improvements. Visualizing the misclassified samples showed that not all samples of *c.ZIEN* make the entire movement to the eye. By example the following misclassified sample of *c.ZIEN*:

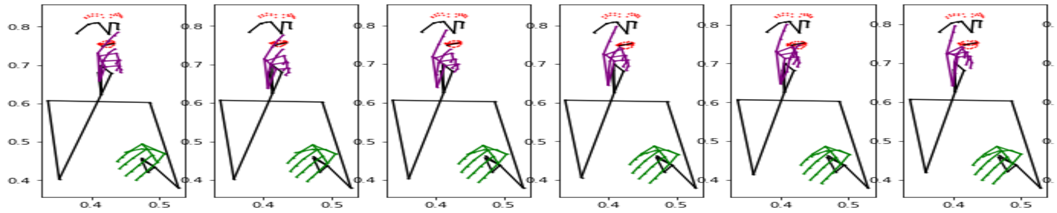


Figure 8: Distance to eye

Range of curviness of fingers

Given that, the previous one was to look at the range between which the fingers curve. Seen that the samples of *c.ZIEN* tend to bend throughout the video towards the eye. This is not the case for *C:1* in which the finger is kept in relatively the same position, not bending more or less during the video (not necessarily straight though).

The curviness of a finger has been extracted from the distance of the wrist to the top of the finger. Then to get the range, we look at the difference between the maximum and the minimum of this value over all frames.

Layered 1, 2 averages for pose of hands

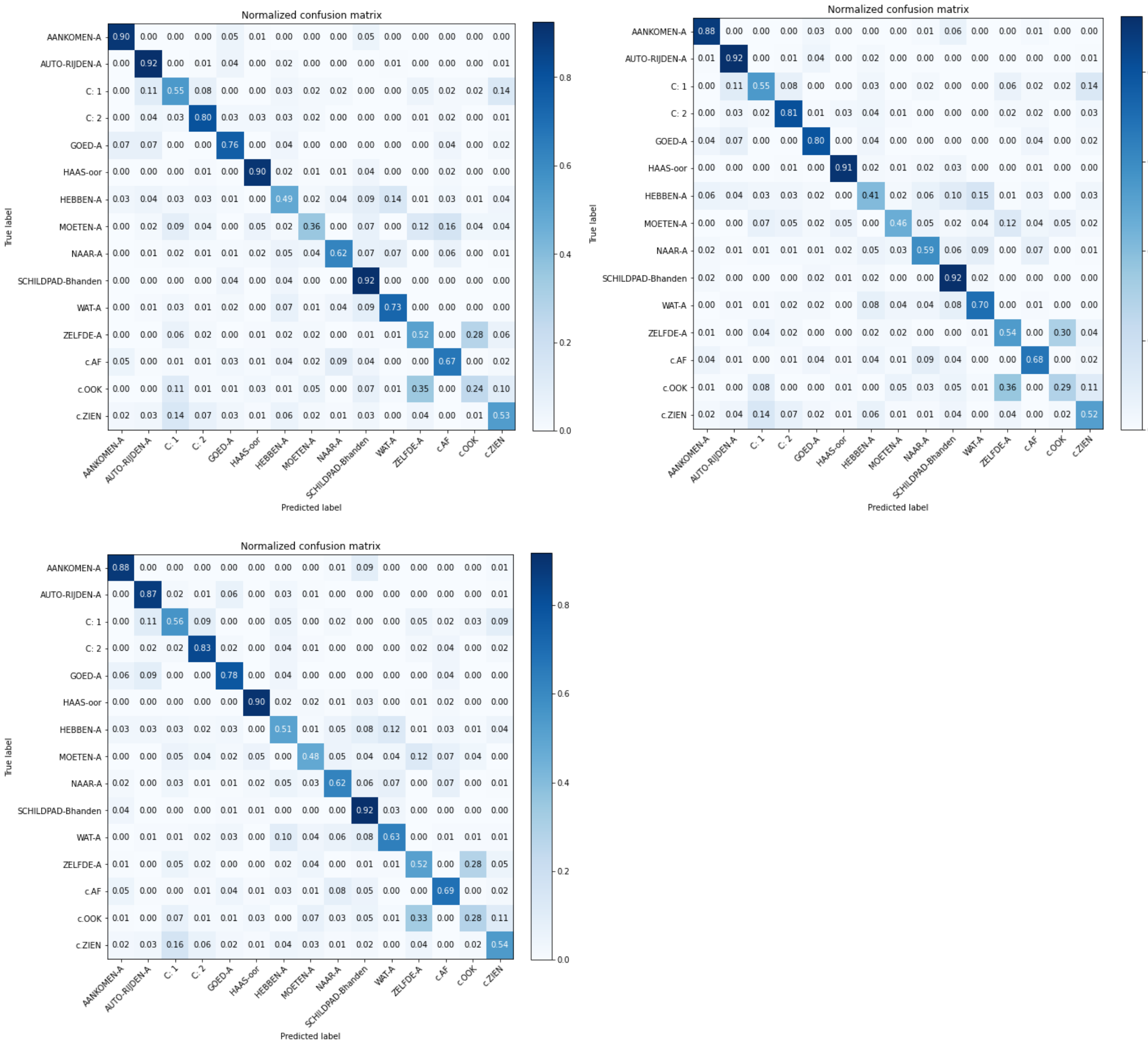


Figure 9:
(top left) confusion matrix of 1 average of hand pose
(top right) confusion matrix of 2 averages of hand pose
(bottom) Confusion matrices of 1 and 2 average(s) of hand pose

From the top 2 matrices, it is clear that 1 average is better for *HEBBEN-A*, 2 averages is better for *MOETEN-A*. This can be explained by the fact that the pose of the hand in *HEBBEN-A* is normally the same (not position). Where in *MOETEN-A* this changes during the sign. Thus, it could be logical that using both the features from 1 and 2 average(s) improves results on both labels. Which is also visible in the result.

Movement of hands

ZELFDE-A and *c.OOK* are often confused, as is visible in all previous confusion matrices. The first solution we wanted to try was to look at the total movement of the hands. The underlying assumption that *ZELFDE-A* moves more than *c.OOK*.
This did not necessarily improve *ZELFDE-A* and c.OOK due to the sometimes small movements of the hand samples of either of these labels, by example the misclassified *ZELFDE-A* . However the overall result did improve thus the feature was kept.

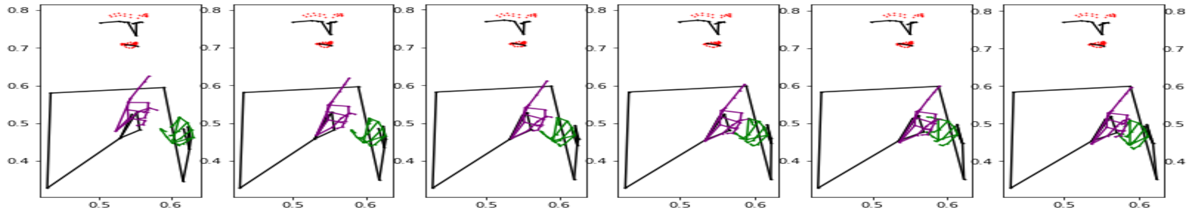


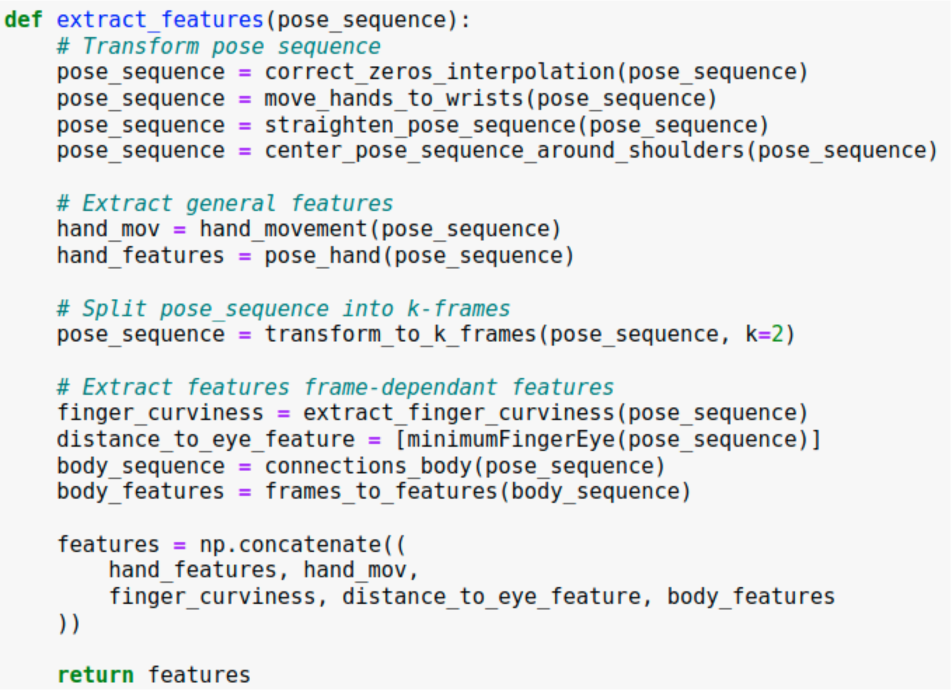
Figure 12: Movement of hands

Movement of index fingers

The same problem was also tried to be tackled by looking at solely the movement from the index finger. The idea is that *c.OOK* the fingers would move to each other one time and in *ZELFDE-A* two times. While not necessarily everyone moves their hands for this movement, which can be derived from looking at misclassified samples. This worked on the *c.OOK* and *ZELFDE-A* labels, yet it did worsen the overall model. Likely due to variance between samples of the same label in some labels. Therefore it was not kept.

7 Best models

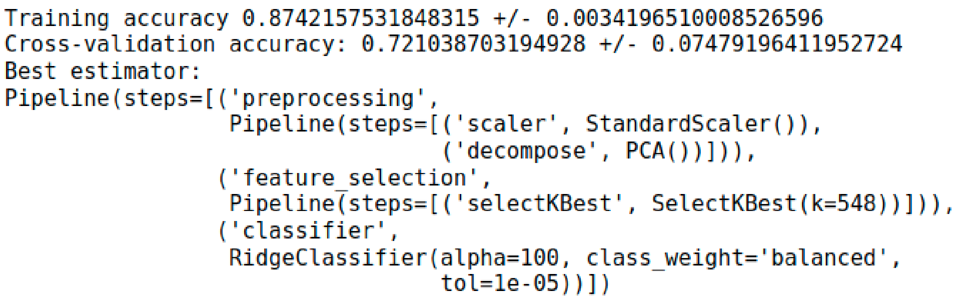
7.1 Linear model



```
def extract_features(pose_sequence):
    # Transform pose sequence
    pose_sequence = correct_zeros_interpolation(pose_sequence)
    pose_sequence = move_hands_to_wrists(pose_sequence)
    pose_sequence = straighten_pose_sequence(pose_sequence)
    pose_sequence = center_pose_sequence_around_shoulders(pose_sequence)

    # Extract general features
    hand_mov = hand_movement(pose_sequence)
    hand_features = pose_hand(pose_sequence)

    # Split pose_sequence into k-frames
    pose_sequence = transform_to_k_frames(pose_sequence, k=2)

    # Extract features frame-dependant features
    finger_curviness = extract_finger_curviness(pose_sequence)
    distance_to_eye_feature = [minimumFingerEye(pose_sequence)]
    body_sequence = connections_body(pose_sequence)
    body_features = frames_to_features(body_sequence)

    features = np.concatenate((
        hand_features, hand_mov,
        finger_curviness, distance_to_eye_feature, body_features
    ))

    return features
```

Code snippet 1: Linear model extract features

In the best linear model, features are extracted as seen in code snippet 1. First, a pose sequence (sample) is transformed by correcting zeros with interpolation, hands are moved to the wrists, the pose sequence is rotated and translated such that the shoulders are always straight and at the origin. Next, features are extracted such as the *hand movement, finger curviness, distance to eye, ….* Finally, all these features are concatenated into a feature list of size 548. The Cross Validation accuracy, confusion matrix and learning curve can be seen in figure 13, 14 and 15.



```
Training accuracy 0.8742157531848315 +/- 0.0034196510008526596
Cross-validation accuracy: 0.721038703194928 +/- 0.07479196411952724
Best estimator:
Pipeline(steps=[('preprocessing',
                 Pipeline(steps=[('scaler', StandardScaler()),
                                 ('decompose', PCA())])),
                ('feature_selection',
                 Pipeline(steps=[('selectKBest', SelectKBest(k=548))])),
                ('classifier',
                 RidgeClassifier(alpha=100, class_weight='balanced',
                                 tol=1e-05))])
```

Figure 13: Linear model - Cross Validation score

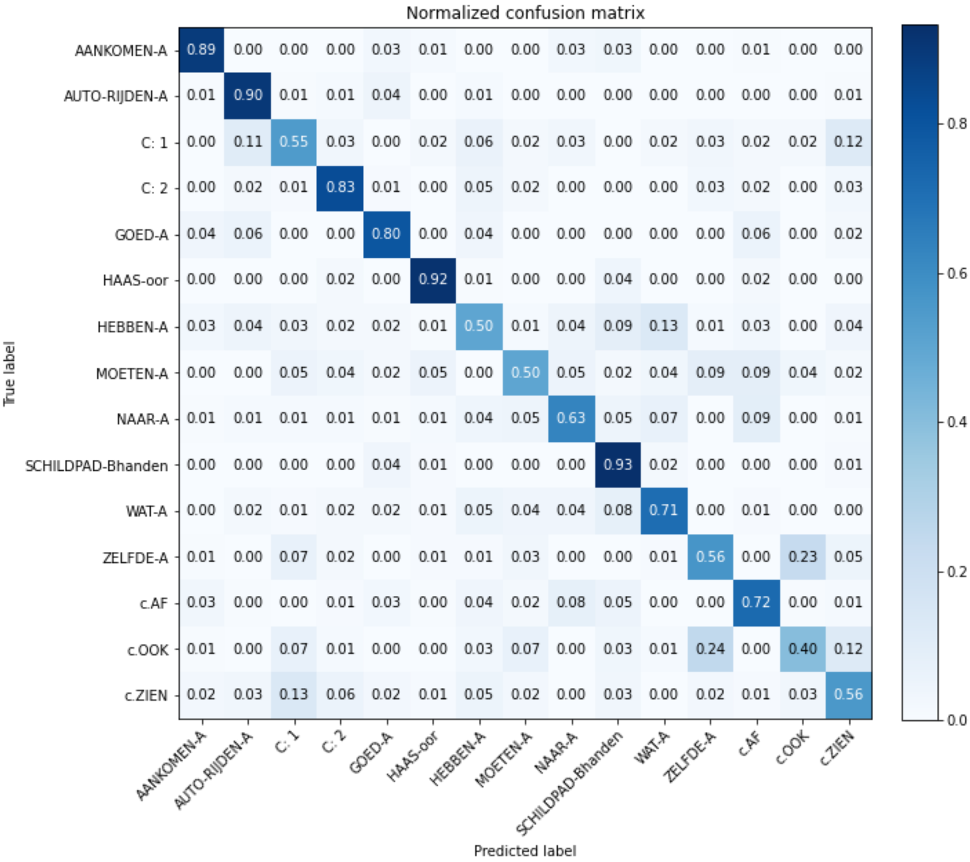


Figure 14: Linear model - Confusion matrix

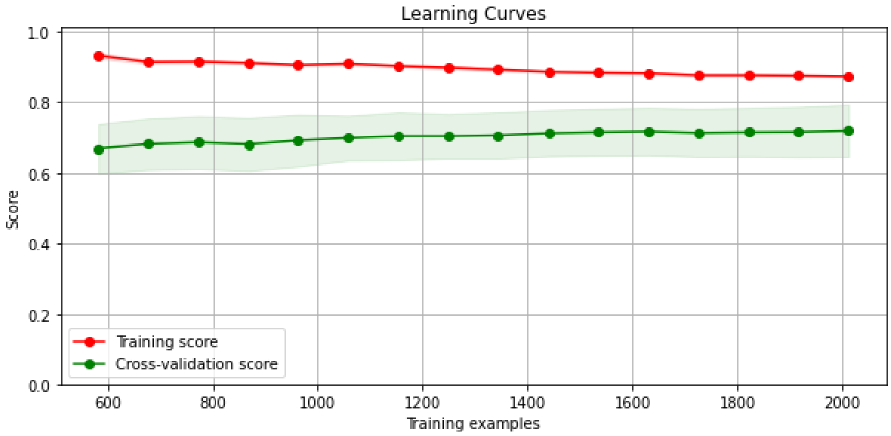
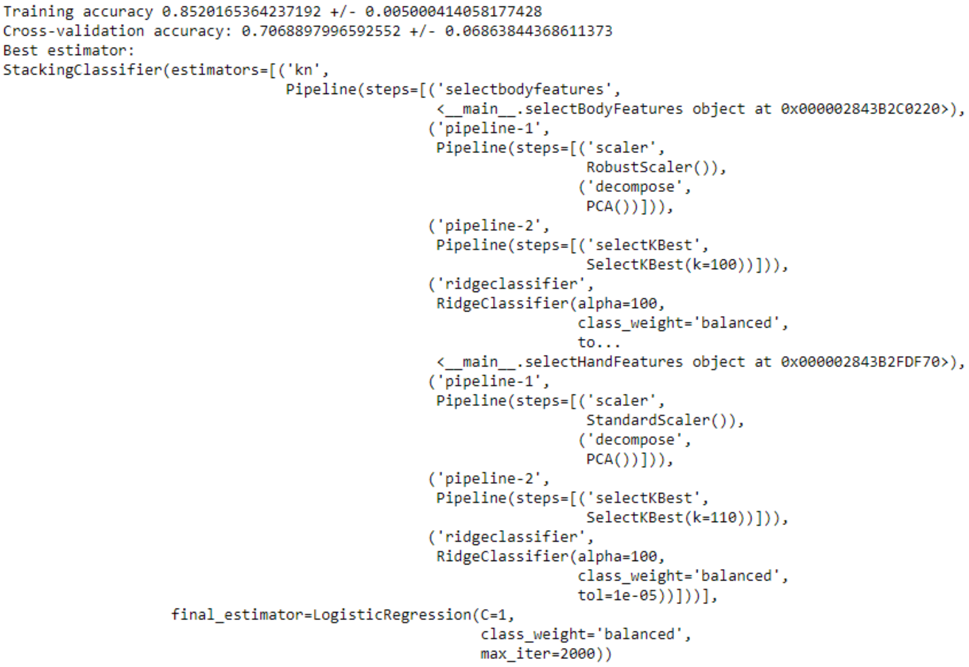


Figure 15: Linear model - Learning curve

7.2 Ensemble

The ensemble is a StackingClassifier built out of two RidgeClassifiers based on different features (body vs hand), followed up with a LogisticRegression classifier without feature passthrough. Compared to our best linear model, we can see it does not improve on total cross-validation, but it does improve on some of the weakest classes (+16% on *HEBBEN*, +14% on *MOETEN*) at the expense of some of the strongest classes. This different repartition of strong classes does lead to an improvement on the public kaggle score.

The main issue of the ensemble remains that is very weak when differentiating *c.OOK* & *ZELFDE-A*. The learning curve shows that the model is limited by the quality of features, and not by the amount of data - it is unlikely more data would have resulted in further improvements. The Cross Validation accuracy, confusion matrix and learning curve can be seen in figure 16, 17 and 18.



```
Training accuracy 0.8520165364237192 +/- 0.005000414058177428
Cross-validation accuracy: 0.7068897996592552 +/- 0.06863844368611373
Best estimator:
StackingClassifier(estimators=[('kn',
                                    Pipeline(steps=[('selectbodyfeatures',
                                                       <__main__.selectBodyFeatures object at 0x000002843B2C0220>),
                                                      ('pipeline-1',
                                                       Pipeline(steps=[('scaler',
                                                                          RobustScaler()),
                                                                         ('decompose',
                                                                          PCA())])),
                                                      ('pipeline-2',
                                                       Pipeline(steps=[('selectKBest',
                                                                          SelectKBest(k=100))])),
                                                      ('ridgeclassifier',
                                                       RidgeClassifier(alpha=100,
                                                                         class_weight='balanced',
                                                                         to...
                                                       <__main__.selectHandFeatures object at 0x000002843B2FDF70>),
                                                      ('pipeline-1',
                                                       Pipeline(steps=[('scaler',
                                                                          StandardScaler()),
                                                                         ('decompose',
                                                                          PCA())])),
                                                      ('pipeline-2',
                                                       Pipeline(steps=[('selectKBest',
                                                                          SelectKBest(k=110))])),
                                                      ('ridgeclassifier',
                                                       RidgeClassifier(alpha=100,
                                                                         class_weight='balanced',
                                                                         tol=1e-05))]))],
                     final_estimator=LogisticRegression(C=1,
                                                           class_weight='balanced',
                                                           max_iter=2000))
```

Figure 16: Ensemble model - Cross Validation score

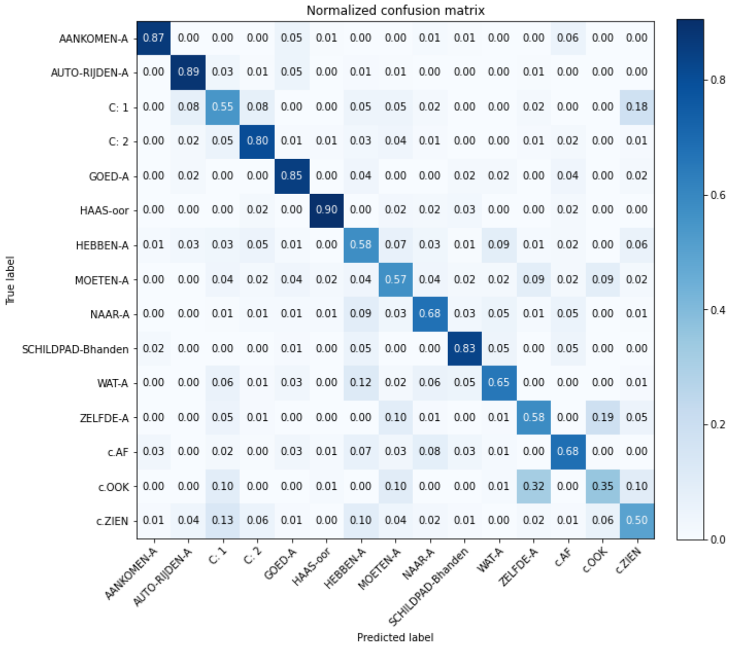


Figure 17: Ensemble model - Confusion matrix

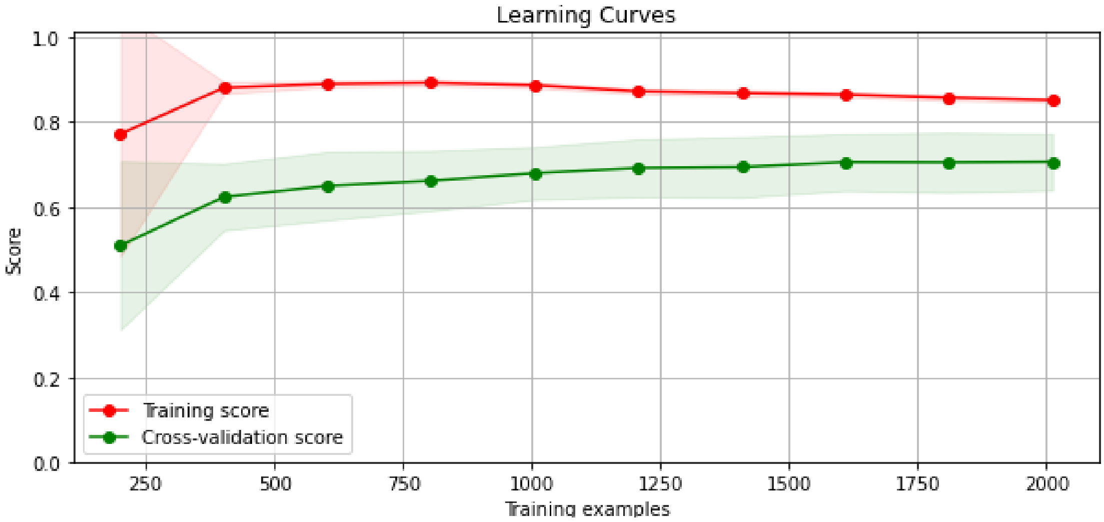


Figure 18: Ensemble model - Learning curve