Machine learning

Gesture recognition

November 2019

Introduction

The goal of this assignment is to find a gesture recognition model, that is able to classify every gesture correctly from a given set of samples. Figure 1 shows the flowchart explaining all the steps leading to choosing the best model.

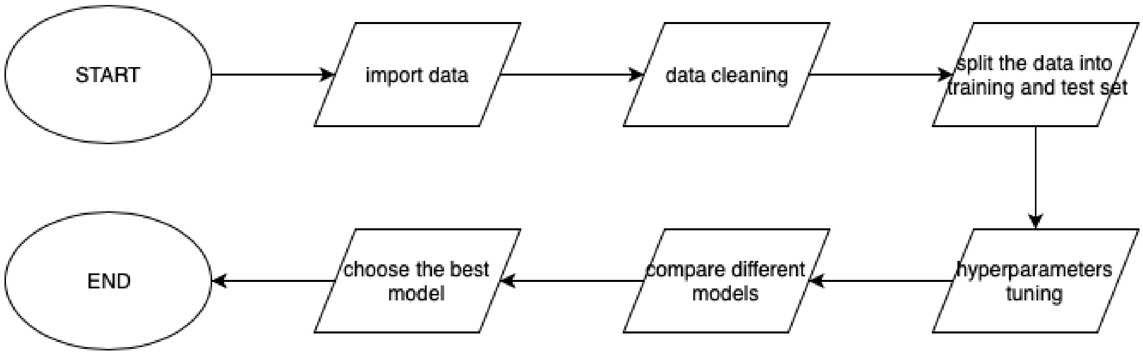

Figure 1: Flowchart

Preprocessing data

The first thing to do is to import the three given files of data, representing x, y, and z dimensions of samples. Next, we label every feature ("function_name" function in the code) and roll the data in order to reduce noise ("f_organize" function in the code). Rolling the data means, that we create a moving window, and within the window we substitute individual values with the mean of previous n values. We make sure to apply rolling only to columns with acceleration values, not labels. After that we concatenate all three files into one dataframe. Because the important acceleration information to distinguish gestures is likely well below 20Hz, we reduce the frequency ("reduce_frequency" function in the code). In our case, we drop every second column. Even after that procedure we still have enough data to successfully distinguish gestures. After that we create a new .csv file with data ready for the analysis. The next step is to apply principal component analysis (PCA), in order to reduce the size of the dataframe. After that we split the data into training set and test set. Training set contains 80% of the data and test set contains 20% of the data. At the end we tune our models and choose the best one. The details on PCA and model tuning are provided in the next sections.

PCA

The next task is to apply PCA. In order to check what is the best number of PCA for our dataset, we test ten different numbers of PCA (10, 20, ..., 100) and we are going to check the variance. In order to perform the decomposition we use the PCA algorithm included in sklearn library. As can be seen in the figure 2, the best number of PCA appears to be about 30.

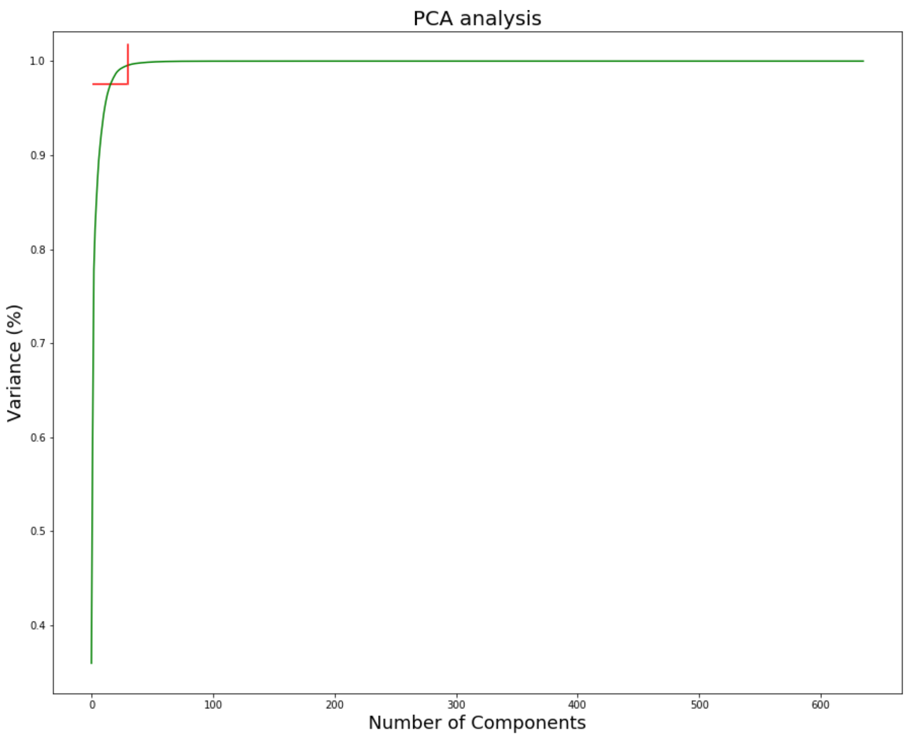

Figure 2: PCA selection

Hyperparameter tuning

Having the data ready for analysis, we test and compare different classification models. We selected 6 models: k nearest neighbours (kNN), decision tree classifier (DTC), support vector machine (SVM), random forest (RF), naive bias (GNB) and neural network (NN). We test different values of selected parameters for these models to choose the best combination:

- for **kNN** tuned parameters are number of neighbors and weights,
- for **DTC** tuned parameters are max depth and max features (number of features to split every node),
- for **SVM** tuned parameters are two versions of kernel and different values of gamma and c,
- for **RF** tuned parameters are number of estimators (trees), max depth and bootstrap,
- for **GNB** there are no parameters to tune, because this model is based on bias theorem,
- for **NN** tuned parameters are number of hidden layers, number of neurons and number of iterations.

For every combination of parameters we train the models and apply cross validation in order to compute the accuracy for every combination. At the end of the process it is possible to find the best set of values of parameters for every model. As the best set of parameters we consider the one providing the highest accuracy. Table 1 shows the tested and chosen values for tested parameters.

Table 1: Tested and chosen values of parameters

model	**parameter**	**tested values**	**chosen value**
kNN	number of neighbors	1, 50, 100, 150, 200	1
kNN	weights	'uniform', 'distance'	'distance'

DTC	max depth	5, 10	10
DTC	max features	1, 7, 13, 19, 23	23
SVM	kernel	'rbf', 'linear'	'rbf'
SVM	gamma	0.1, 0.01, 0.001, 0.0001	0.001
SVM	C	1, 10, 100, 1000	10
RF	number of estimators	10, 255, 500	500
RF	max depth	10, 55, 100	100
RF	bootstrap	True, False	False
NN	number of hidden layers	1, 2, 3, 4	1
NN	number of neurons	50, 100, 150, 200	150
NN	number of iterations	100, 500, 1000, 1500	100

Results and discussion

Figure 3 presents the accuracies reached by our models. As can be seen from the box plots, kNN, SVM and RF reach the highest accuracies, NN preforms middle range, while DTC and GNB perform much worse. For the latter it could have been expected, because GNB is the most suitable for binary problem classification.

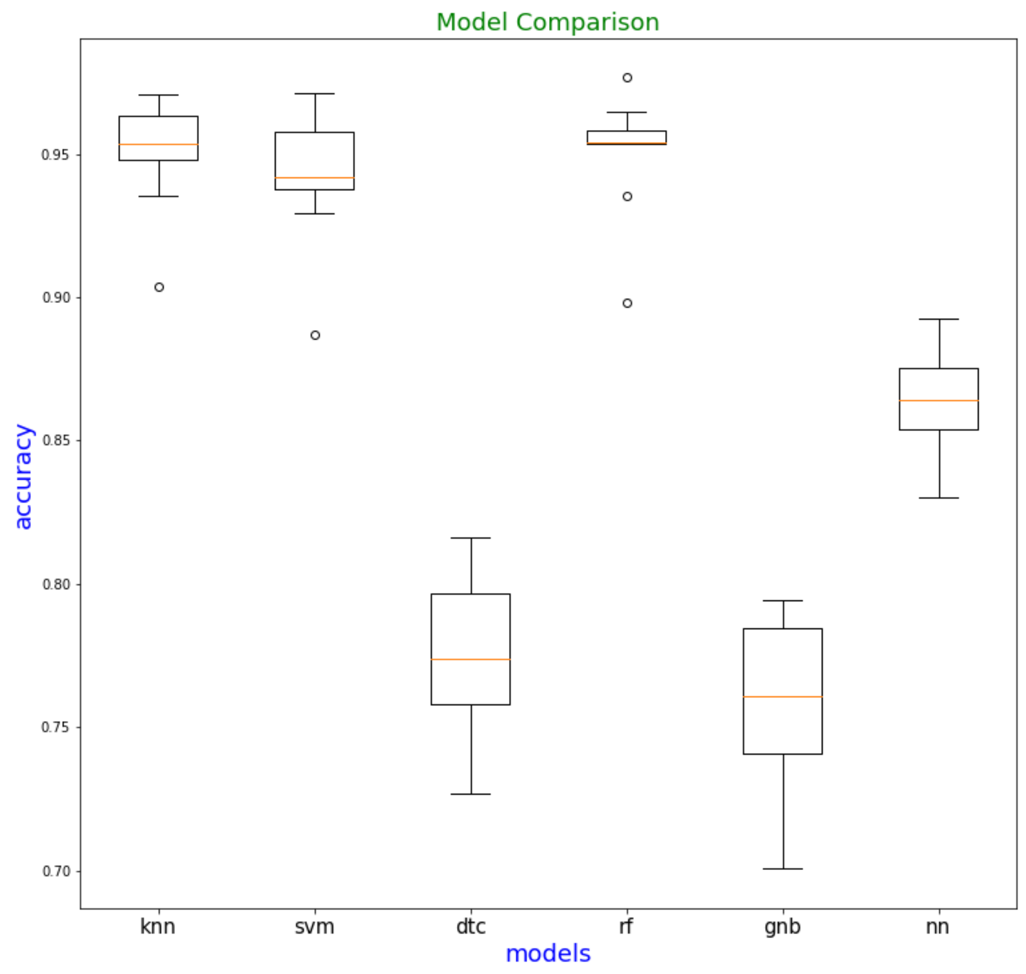

Figure 3: Models accuracies

Table 2 shows the average accuracy with a standard deviation of all tested models.

Table 2: Tested and chosen values of parameters

model	mean of accuracies	standard deviation
kNN	0.950	0.018
SVM	0.942	0.022
DTC	0.774	0.028
RF	0.950	0.020
GNB	0.758	0.030
NN	0.865	0.019

Figure 4 shows confusion matrices for the best PCA for tested models. Once again it can be seen, that GNB reaches the lowest accuracy. DTC also seems to give more wrong predictions than the remaining four models. NN performs slightly better than DTC. kNN, SVM and RF performed very similarly, giving the most correct classifications. We decide to choose kNN as our final model, because it is the easiest to implement and its time complexity is much lower than SVM or RF. That means, we can obtain similar results in shorter time and with less computational power.

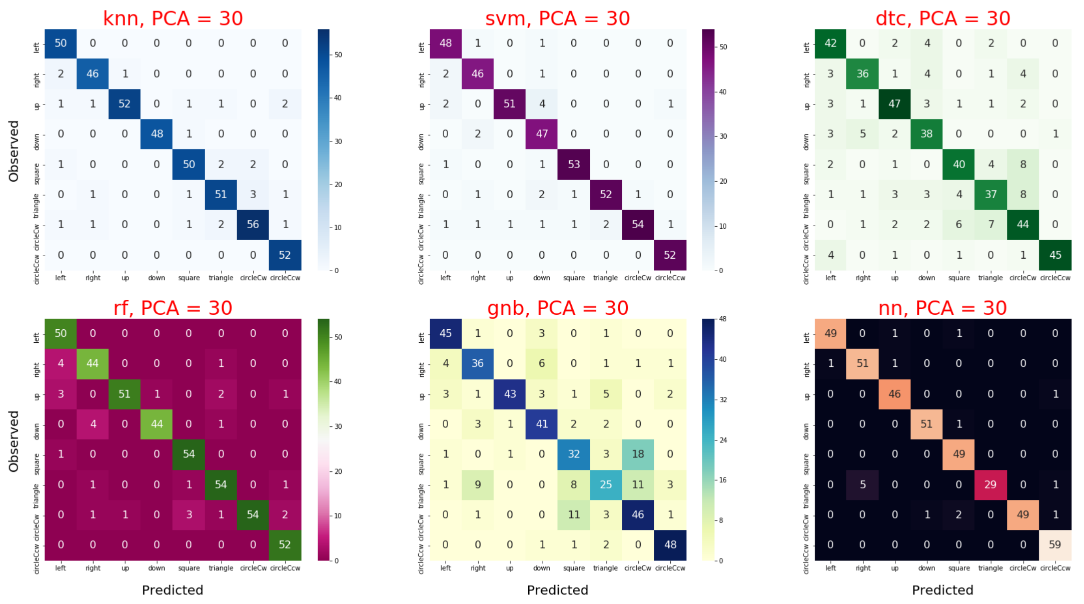

Figure 4: Confusion matrices for tested models

Leave one out cross validation

Leave one out cross validation (LOOCV) is a similar approach to cross validation, because per each iteration we are changing the training and validation sets. In particular, in every iteration, one datapoint is left out from the training. In this case, there are 2160 samples, so per each iteration *n-1* samples are used to train the model and the remaining sample is used as validation sample. This process is repeated until every sample has been used as validation sample. It is intuitively understandable, that the number of possible combinations is given by the number of samples included into the dataset, n. Of course, is computationally expensive, but should return more reasonable estimation of the accuracy of the model. We performed the LOOCV on our best model, kNN. We were able to obtain an accuracy of 0.95 with standard deviation of 0.21.

Conclusions

Thanks to the high quality of our dataset it has been possible to find many models that have been able to correctly predict the gestures. As explained before, our choice has been kNN, nevertheless, also the other models performed very well, meaning that they correctly classified gestures.