EVD3 MACHINE LEARNING PORTFOLIO_

By submitting this portfolio the authors certify that this is their original work, and they have cited all the referenced materials, in the forms of texts, models, and books properly.

Course: EVD-3

Date: 02 November 2020

Teacher: Jeroen Veen

Students: Martijn Visser, 661263

Tom Sievers, 598986

André Slokker, 595668

Jochem Grootherder, 598765

Table of Contents

1 INTRODUCTION

The purpose of this document is to learn to work with Machine Learning. This is done using different gestures. First the problem will be stated. After that the needed data will be acquired and explored. Then the two best Machine Learning models will be selected and trained. These models will be deployed and tested. Finally a conclusion will be written.

Image processing consists of different critical steps like acquisition, enhancement, segmentation, feature extraction and classification. The classification can be improved by using a Machine Learning algorithm. The first part of the EVD3 course is about learning the basic principles of Machine Learning and learning how to select and train an algorithm. The student should also be able to analyze a Machine Learning pipeline by validating, testing and evaluating quality measures.

The EVD project assignment is to create a pool aid. This pool aid must visualize the path of the cue ball in a game of pool based on the cue position and angle. Machine Learning will be used to detect gestures to start, stop and calibrate the pool aid. The gestures which are used in this document will also be used during the project.

In the process of computer vision there are multiple steps of which one deals with classification. Dealing with classifications in computer vision without machine learning ends in many written rules to find certain patterns. Because machine learning is based on the recognition of patterns, it can be applied to learn these patterns itself instead of painstakingly finding and writing those rules yourself.

The minor project has as criteria that all steps of image processing like acquisition, enhancement, segmentation, feature extraction and classification are included. As classification can include Machine Learning, it is an integral of the minor project to be able to use Machine Learning correctly when it is needed.

2 Problem statement

The machine learning objective for this assignment is to recognize three different gestures to control a pool assistant. These different gestures are used for starting, stopping and calibrating the system. In order to get to the classification of these hand gestures, data has to be acquired for each of the gestures. This data has to be explored to understand the dataset and its characteristics. After this the data has to be prepared so that a training set and a test set. Once there is a training and test set, a machine learning model can be selected and trained. Next up the model should be fine-tuned to remove any unwanted outcomes. When this is done the model is ready to be deployed and should be properly tested

The problem definition is: “A player must be able to start, stop and calibrate a pool assistant by the means of showing hand gestures in front of a camera.”

There are three different gestures that will be classified. These gestures will be used to control a pool aid. The gestures are used for starting, stopping and calibrating the aid. An image of the gestures can be found in appendix A.

Functional Requirements

Code	Description	Priority (MoSCoW)
FR-1	The system uses thresholding to remove the background.	Must
FR-2	The system creates a binary image.	Must
FR-3	The system extracts information from the image into preselected features.	Must
FR-4	The system can use two different classification algorithms to use the existing data to classify new images.	Must
FR-5	The system classifies into three gestures: start, stop and calibrate.	Must
FR-6	The system is able to classify the gesture in different positions.	Must
FR-7	The system is able to detect gestures from members of the project team.	Must
FR-8	The system is able to detect gestures on a	Must

	blue background.	
FR-9	The system is able to detect gestures which have a distance of 35 to 40 centimeter from the camera.	Must
FR-10	The system is able to detect hand gestures in a regular indoor environment lighting.	Must

Table 1: Functional requirements.

Non-functional requirements

Code	**Type**	**Description**	**Priority (MoSCoW)**
NFR-1	Performance	The system is able to detect and classify a gesture in 1 second.	Must
NFR-2	Reliability	The system has a minimum success rate of 90% for classifying gestures.	Must
NFR-3	Reliability	The system has a minimum success rate of 90% for detecting gestures.	Must

Table 2: Non functional requirements.

3 DATA ACQUISITION AND EXPLORATION

Test set-up

To collect data for exploration and feature engineering, the following set-up was used. A phone functioned as the camera and was connected to the pc using the application DroidCam. The same application can also be used on a Raspberry Pi with a bit of work. The phone was mounted on a car phone holder, which was positioned above a table. On this table a blue fabric was placed to fill the background of the camera view. All data was acquired using this specific set-up. For each of the classes 200 pictures have been captured, which then results in a total of 800 images (including pictures made for the ignore class). In Appendix A: Hand gestures an example for each class can be found.

Feature engineering

Area

Using the area of an object as a feature is useful. Although it does not help discern the three different gestures, because their areas are very similar, where it does help is finding gestures of the “ignore” category. For example the area of a full flat hand is a lot larger than a “start” gesture.

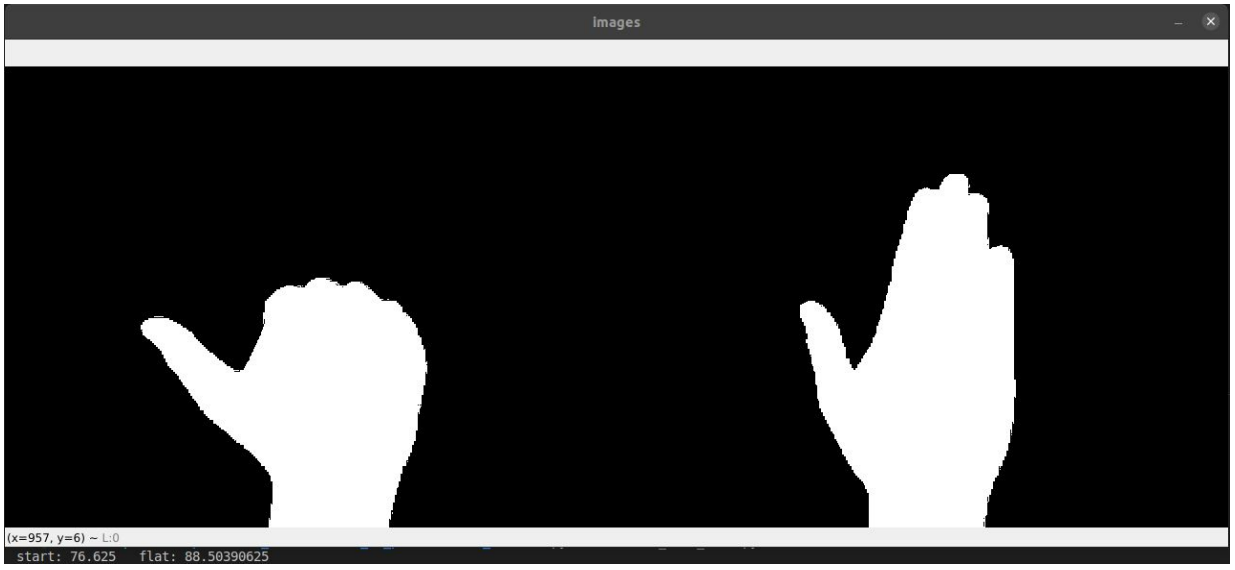

Figure 1: Example usage feature area (start gesture: 76.6, flat hand: 88.5).

Perimeter

The perimeter of an object can be used as a feature to detect when a gesture contains fingers or not. Gestures with fingers have a much larger perimeter compared to fists.

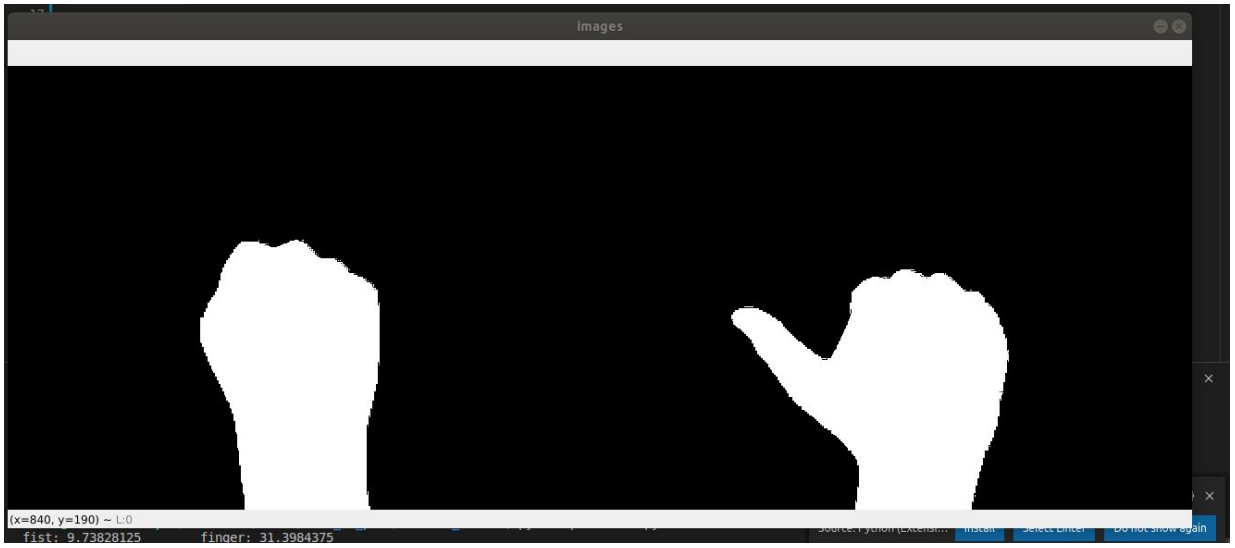

Figure 2: Example feature perimeter usage (fist: 9.73, start gesture: 31.4).

Aspect ratio

The aspect ratio of an image is the ratio of its width to its height. This could be used to differentiate the calibrate gesture from the start and stop gesture. This is because the calibrate gesture is most likely going to be longer than it's wide, this will mean the aspect ratio will be less than 1.0. Whereas the start and stop gesture are probably wider than that, it's long, thus the aspect ratio will be greater than 1.0.

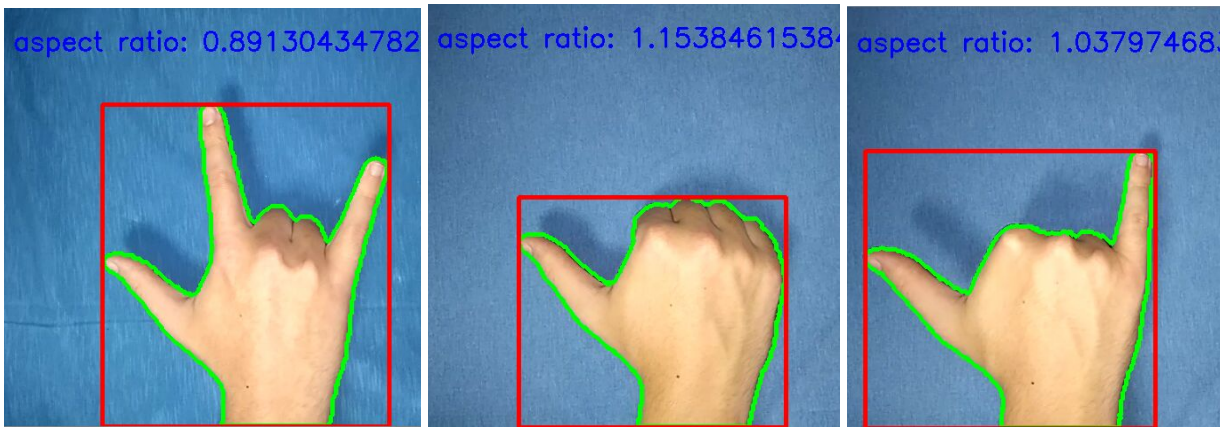

Figure 3A, 3B, 3C: Example feature aspect ratio usage.

Extent

By using the extent as a feature it means to compare the area of the actual object against what the area would be if it was surrounded by a rectangle. An example where area is not enough is if you were holding your hand flat the area of the hand would always be the same no matter if you were holding your fingers close together or not. By comparing the area against the area of a rectangle this can be differentiated. In our specific case the number of extended fingers and their directions differ, which means that the resulting extent will be different and can be used as a feature.

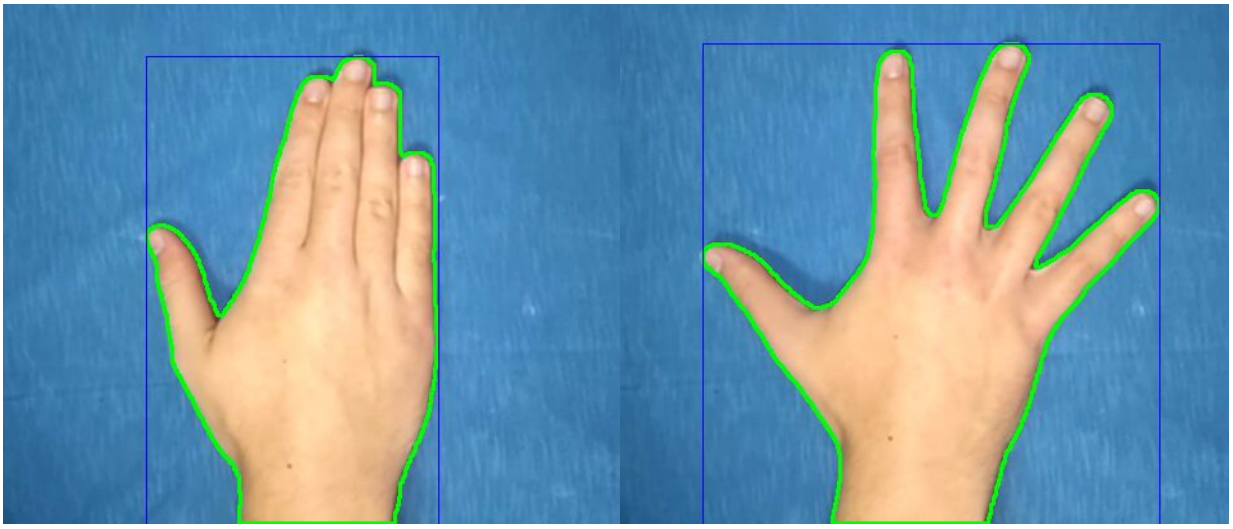

Figure 4: Example feature extent usage

Acute and shallow angles

To increase the likeliness of detecting the different gestures, a feature is added which counts the number of shallow and acute angles. The reason for this choice is because each of the gestures has a similar perimeter and area. This makes differentiating between them a hard thing to do even for a human. To get these features, a contour of the hand is taken and simplified using approxPolyDP. To further simplify, points that are within 20 pixels of each other are grouped into one to remove points that lie almost ontop of eachother in the contour. After this operation the simplified contour is grouped into triangles. Of these triangles the angle for the contour is computed using the Law of cosines. After this the centroid of a triangle is computed and check if this centroid lies within the polygon that is the contour, if this is not the case, the computed angle is subtracted from 360. Afterwards the real angle is checked to see if it is acute or shallow, which increments a value and is returned after the above steps are computed for every triangle. Below a visual representation of the triangles and their centroid.

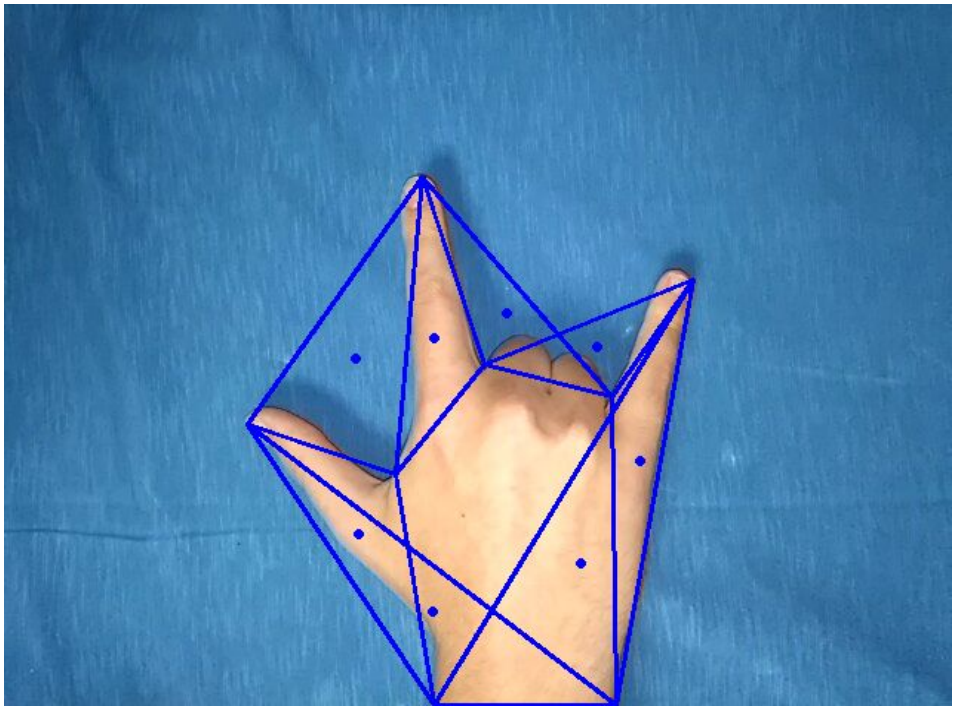

Figure 5: Example feature acute and shallow angles usage

Convexity

Another feature is looking at the number of farthest points which the convexityDefects function returns. This function calculates the farthest point between two points (two vertices). First the convex hull of the gesture is calculated. This is needed to flat the contour. This contour will be used by the convexityDefects function to find the convexity defects.

Data exploration

To explore the feature data a pandas scatter matrix in combination with seaborn was used. This matrix gives insight in all 2 combinations of different features with each other and how much overlap there is between the different categories. The image for this matrix can be found here on github for readability and is also listed in Appendix B: Scatter Matrix.

The area feature in combination with perimeter creates discernible groups between stop and start. It also shows some groups forming in combination with the number of acute angles. All the other feature combinations do not create separated groups with area. The perimeter feature creates two groups of ignore and not ignore together with the feature extent. With other features it does not create any discernible groups. Aspect ratio does not create any useful groupings. Extent does not create any extra groups, only the one previously mentioned. Number of acute angles does combine well with number of shallow angles and number of convexity defects. Both create discernible groups for all four classes. When looking at the number of shallow angles and number of convexity defects combinations, there are no new revelations not previously mentioned.

Data preparation

The only step needed is to scale the features to the same range. Cleaning is not needed because there are no real extreme outliers, or missing data for certain features. Also, none of the features are text based, which would need to be converted to numbers. For scaling of the features the scikit learn StandardScaler will be used. This scaler will be used in the prediction pipeline, because of the way it works. But for the preprocessing pipeline a self implemented transformer will be used which transforms an image into the feature data. This class is then put into a pipeline for use in other files. The implementation of the pipeline and transformer can be found in Code Appendices, Appendix B: pipeline.py.

4 Model selection and training

Model exploration

One simple constraint on the models is that the model has to be supervised because label data already exists. Next to this, the model also has to be a multiclass classification model. Using sklearn https://scikit-learn.org/stable/modules/multiclass.html as a resource, the following longlist of suitable machine learning models was established:

- sklearn.naive_bayes.BernoulliNB
- sklearn.naive_bayes.GaussianNB
- sklearn.tree.DecisionTreeClassifier
- sklearn.tree.ExtraTreeClassifier
- sklearn.ensemble.ExtraTreesClassifier
- sklearn.ensemble.GradientBoostingClassifier
- sklearn.ensemble.RandomForestClassifier
- sklearn.neighbors.RadiusNeighborsClassifier
- sklearn.neighbors.KNeighborsClassifier
- sklearn.neighbors.NearestCentroid
- sklearn.svm.LinearSVC
- sklearn.svm.NuSVC
- sklearn.svm.SVC
- sklearn.linear_model.LogisticRegression
- sklearn.linear_model.RidgeClassifier
- sklearn.linear_model.SGDClassifier
- sklearn.linear_model.Perceptron
- sklearn.linear_model.PassiveAggressiveClassifier
- sklearn.neural_network.MLPClassifier
- sklearn.discriminant_analysis.QuadraticDiscriminantAnalysis
- sklearn.gaussian_process.GaussianProcessClassifier
- sklearn.discriminant_analysis.LinearDiscriminantAnalysis

To shorten this list a script was created where every model was tested on 2 different train and test sets. This script can be found in the Code Appendices, Appendix A: longlist_test.py. The performance of each iteration was written to a file and then examined by a person on which classifiers have the best performance in those two test sets. The precision and recall scores were used as a metric for performance. The test results can be found in Appendix C: Classifier results. Using these performance

values a shortlist was created where the f1-score never fell below 0.99 for any of the labels. This resulted in the following shortlist:

- sklearn.ensemble.ExtraTreesClassifier
- sklearn.gaussian_process.GaussianProcessClassifier
- sklearn.neighbors.KNeighborsClassifier
- sklearn.ensemble.RandomForestClassifier
- sklearn.svm.SVC

From this list only two have to be used. The choice falls between the classifiers which are already familiar to the project team. This means that RandomForestClassifier and SVC will be used from here on out.

Data split

The data is split into a train and test set using the random state zero. Below is the distribution for each occurrence in the test and train set. The code to achieve this table can be found in Code appendices, Appendix E: split_test.py.

	Start	*Stop*	*Calibrate*	*Ignore*
Train set	126	120	114	120
Test set	74	80	86	80

Hyperparameter tuning

To tune the hyperparameters of these models first a scoring method has to be chosen. For scoring precision will be chosen, because FP should be at a minimum. The hyperparameters to be tuned for both of the classifiers will be listed below. Default hyperparameter values are marked in **bold**.

RandomForestClassifier hyperparameters

Hyperparameter	**Values**
n_estimators	20, 40, 60, 80, **100**, 150, 1000
criterion	**gini**, entropy
max_features	**auto**, log2

Table 3: Hyperparameter list RandomForestClassifier

SVC hyperparameters

Hyperparameter	Values
kernel	linear, poly, **rbf**
C	0.2, 0.5, **1**, 2
decision_function_shape	**ovr**, ovo
degree	2, **3**, 4

Table 4: Hyperparameter list SVC

Of all the hyper parameter values tested above with both RandomForestClassifier and SVC the default parameters almost always perform the best. On the occasions that those are not the best parameters, there is a difference in precision of 0.002. This difference is so small that it is insignificant.

Performance metrics

Like previously mentioned precision is preferred in the current machine learning problem, because one should be sure that a certain command has been issued. This means that in an ROC curve the value with the highest true positive rate at which the false positive rate is relatively low.

ROC curves

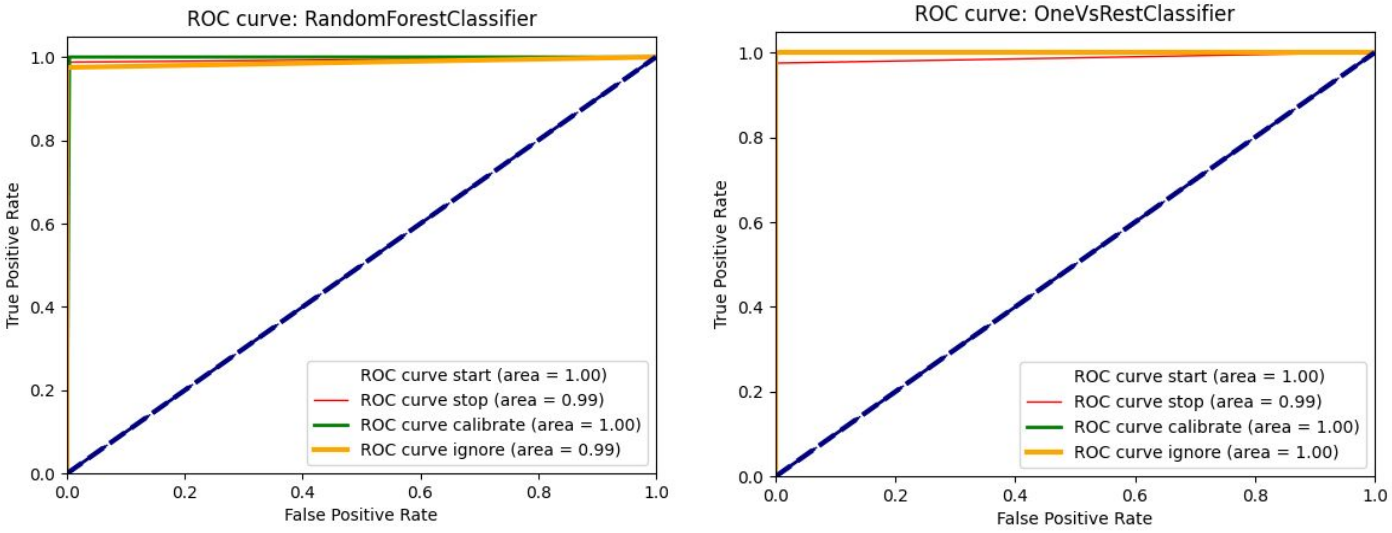

Figure 6A, 6B: ROC curves

In the above graphs the ROC curves are calculated for each of the classes and each of the classifiers. In both of the legends the AUC is also denoted for each curve. This shows that the performance overall

is slightly better with the SVC compared to RFC, this also holds true with different random states of the classifiers themselfs, SVC doesn't change and RFC does change. The overall performance of RFC sometimes matches SVC but not always.

Learning curves

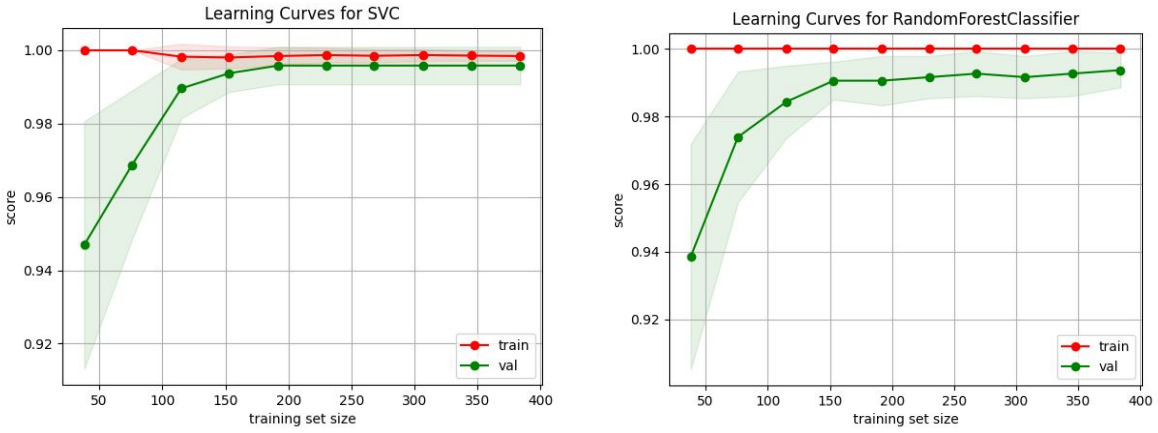

Figure 7A, 7B: Learning curves

The learning curves have been calculated for both classifiers. As can be seen in the figures both the bias and the variance are low. The algorithms are able to classify gestures well with different training set sizes. Also the training set score stays the same with different training set sizes which indicates low bias and low variance.

Confusion matrix

To evaluate the TP, FP, TN, FN a confusion matrix will be used. A set random state was used for these calculations to get the same results every time, but the random state was also removed at some point to test if the confusion stays similar when the train and test sets change.

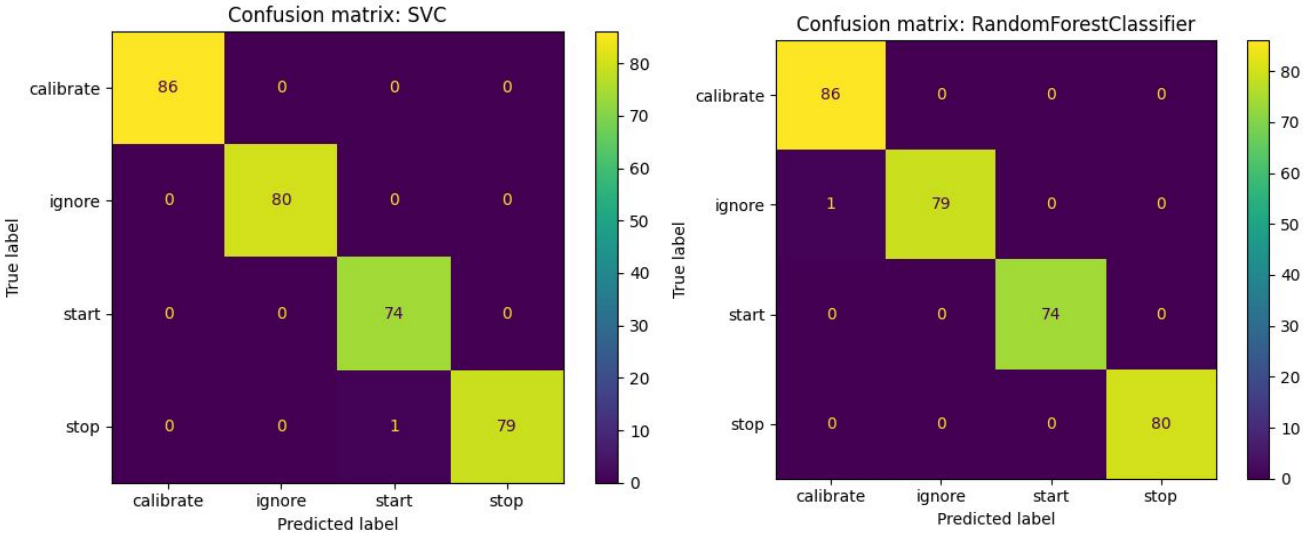

Figure 8A, 8B: Confusion matrices

In the scatter matrices above it can be seen that both classifiers perform very well on the test sets and have very little confusion, where only 1 class is confused with another class. Both perform similar in the same random state. When the random state setting is removed and executing the same code 100 times the maximum number of false predictions reaches a maximum of 2 classes that are confused. This shows that the overall precision and recall of both models are very high.

5 DEPLOY AND TEST

The models are deployed on the machine which is also used to gather the training data. The reasoning behind this is the ease of installation of the droidcam application on a Raspberry Pi. To test the Machine Learning algorithms different gestures are being held at different positions underneath the camera in different angles.

Test criteria	**Result SVC**	**Result Random Forest Classifier**
The start gesture is held at 5 different positions in 5 different angles.	22 out of 25 detected correctly	23 out of 25 detected correctly
The calibrate gesture is held at 5 different positions in 5 different angles.	20 out of 25 detected correctly	20 out of 25 detected correctly
The stop gesture is held at 5 different positions in 5 different angles.	18 out of 25 detected correctly	24 out of 25 detected correctly
A total of 25 “random” gestures are held at different positions and different angles.	20 out of 25 detected correctly	18 out of 25 detected correctly
The gestures are detected and classified within 1 second.	100 out of 100 gestures	100 out of 100 gestures

Table 5: Test criteria

The testing application makes use of machine learning models inside a pipeline, exported using joblib. The export of these pipelines can be found in the Code appendices, Appendix C learn_model.py.
The testing application itself can be found in Meaning the application could be deployed on a Raspberry Pi if needed. Also the preprocessing pipeline is implemented inside a script that can be imported by the user. This pipeline is found inside the Code appendices, Appendix B: pipeline.py.

6 Conclusion

The purpose of this Machine Learning portfolio is to learn working with different types of Machine Learning models and to evaluate their performance. According to the tests, the SVC and Random Forest Classifier models performed the best on the features in the test data. Both models didn't have to be tuned, since both gave an almost perfect score. Both models also have been deployed and tested. In these situations the models still performed well, but less compared to when they were trained. This is because of completely different positions and angles which were not included in the training data. In conclusion, both models perform well enough for daily use.

The Machine Learning project checklist (Géron, 2019) was used during this project. The first step was to think about how this project should be tackled. A problem statement was written and a list with SMART functional and non-functional requirements was made. The second step was acquiring the data. Then the data was explored using a Scatter matrix to find out which features were easy to separate with different gestures. After the data exploration the data was prepared by cleaning it and selecting the useful features. Also the features were scaled. The next step was finding promising models by creating a longlist with different Machine Learning models and finding out which models performed the best on the data. Finally two models were chosen and fine tuned (which was not really needed since both models performed really well with their default parameters). As a final step the models were deployed and tested.

7 REFERENCES

Géron, A. (2019). Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow. Sebastopol, Canada.: O'Reilly Media.

scikit-learn. (n.d.). API Reference. Retrieved October 8, 2020, from https://scikit-learn.org/stable/modules/classes.html

OpenCV. (n.d.). OpenCV: OpenCV modules. Retrieved October 8, 2020, from https://docs.opencv.org/4.4.0/

CODE APPENDICES

Appendix A: longlist_test.py

```
from sklearn.utils import all_estimators
from sklearn.model_selection import train_test_split
from sklearn.metrics import classification_report
from sklearn.preprocessing import StandardScaler

import joblib

output_file = open("classifier_test_output.txt","w")

dataset = joblib.load('hand_data.joblib')

X = StandardScaler().fit_transform(dataset["data"])

X_train, X_test, y_train, y_test = train_test_split(
    X, dataset['targets'], test_size=0.4, random_state=0)

X_train2, X_test2, y_train2, y_test2 = train_test_split(
    X, dataset['targets'], test_size=0.4, random_state=42)

names =
["BernoulliNB","GaussianNB","DecisionTreeClassifier","ExtraTreeClassifier","Extra
TreesClassifier","GradientBoostingClassifier","RandomForestClassifier","RadiusNei
ghborsClassifier","KNeighborsClassifier","NearestCentroid","LinearSVC","NuSVC","S
VC","LogisticRegression","RidgeClassifier","SGDClassifier","Perceptron","PassiveA
ggressiveClassifier","MLPClassifier","QuadraticDiscriminantAnalysis","GaussianPro
cessClassifier","LinearDiscriminantAnalysis"]
all_classifiers = all_estimators(type_filter="classifier")
for name, estimator in all_classifiers:
    if name in names:
        try:
            print("Name: {} iteration: 1\n".format(name))
            clf = estimator()
            clf.fit(X=X_train, y=y_train)
            y_true, y_pred = y_test, clf.predict(X_test)
            output_file.write("Name: {} iteration: 1\n".format(name))
            output_file.write(classification_report(y_true, y_pred))
            output_file.write("\n")
            print("Name: {} iteration: 2\n".format(name))
            clf2 = estimator()
            clf2.fit(X=X_train2, y=y_train2)
            y_true, y_pred = y_test2, clf2.predict(X_test2)
            output_file.write("Name: {} iteration: 2\n".format(name))
            output_file.write(classification_report(y_true, y_pred))
```

```
            output_file.write("\n")
        except:
            print("Unable to predict or fit")
            output_file.write("\n")
```

Code block 1: Script which tests different Machine Learning algorithms and writes the output scores to a file.

Appendix B: pipeline.py

```
import extract
import cv2 as cv
import numpy as np
import os
import glob
from segment import maskBlueBG
from sklearn.pipeline import Pipeline
from sklearn.base import BaseEstimator, TransformerMixin
from sklearn.preprocessing import StandardScaler
import joblib

class ImgDataConverter(BaseEstimator, TransformerMixin):
	def __init__(self, min_hsv_val = (100, 70, 0), max_hsv_val = (140, 255,
255)):
		self.min_hsv_val = min_hsv_val
		self.max_hsv_val = max_hsv_val
		def fit(self, X, y=None):
		return self
		def transform(self, X):
		if(X.ndim != 4):
			raise Exception('X should have 4 dimesions. X has : {}
dimensions'.format(X.ndim))

		array = np.empty([len(X), 7])
		i = 0
		for img in X:
			bw = maskBlueBG(img, self.min_hsv_val, self.max_hsv_val)
			kernel = cv.getStructuringElement(cv.MORPH_ELLIPSE, (5, 5))
			bw = cv.morphologyEx(bw, cv.MORPH_OPEN, kernel)
			contour = extract.getLargestContour(bw)
			if contour is not None:
			features = extract.getSimpleContourFeatures(contour)
			array[i] = np.array([features])
			i += 1
			else:
			array[i] = np.array([0, 0, 0, 0, 0, 0, 0])
		return array
```

```
conv = ImgDataConverter()

pre_process_p = Pipeline([
	('img_to_feature_data', ImgDataConverter())
])

if __name__ == "__main__":
	data_path = '../img_acquisition/training_data/'
	print("[INFO] loading images...")
	p = os.path.sep.join([data_path, '**', '*.png'])

	file_list = [f for f in glob.iglob(
	p, recursive=True) if (os.path.isfile(f))]
	print("[INFO] images found: {}".format(len(file_list)))

	feature_names = ['area', 'perimeter', 'aspect_ratio', 'extent', 'n_acute',
'n_shallow', 'n_convexity']
	X = np.empty([len(file_list), 480, 640, 3], np.uint8)
	y = ["empty"]*len(file_list)
	i = 0
	for filename in file_list:
	img = cv.imread(filename)
	X[i] = cv.imread(filename)
	y[i] = filename.split(os.path.sep)[-2]
	i = i + 1
	X = pre_process_p.fit_transform(X)

	unique_targets = ["start", "stop", "calibrate", "ignore"]

	hand_dataset = {'data' : X, 'targets' : y, 'unique_targets' :
unique_targets}

	print(X[0])
```

Code block 1: Script which extracts the features from data.

Appendix C: learn_model.py

```
import joblib
from sklearn.ensemble import RandomForestClassifier
from sklearn.svm import SVC
from sklearn.pipeline import Pipeline
from sklearn.preprocessing import StandardScaler

dataset = joblib.load("hand_data.joblib")
```

```
SVC_model_pipe = Pipeline([('std_scaler', StandardScaler()), ('SVC', SVC())])

RFC_model_pipe = Pipeline([('std_scaler', StandardScaler()), ('RFC',
RandomForestClassifier())])

X = dataset["data"]
y = dataset["targets"]

RFC_model_pipe.fit(X, y)
SVC_model_pipe.fit(X, y)

joblib.dump(RFC_model_pipe, "RFC_model.joblib")
joblib.dump(SVC_model_pipe, "SVC_model.joblib")
```

Code block 1: Script which exports Machine Learning pipelines.

Appendix D: test_app.py

```
import joblib
from sklearn.ensemble import RandomForestClassifier
from sklearn.svm import SVC
from sklearn.pipeline import Pipeline
from sklearn.preprocessing import StandardScaler

dataset = joblib.load("hand_data.joblib")

SVC_model_pipe = Pipeline([('std_scaler', StandardScaler()), ('SVC', SVC())])

RFC_model_pipe = Pipeline([('std_scaler', StandardScaler()), ('RFC',
RandomForestClassifier())])

X = dataset["data"]
y = dataset["targets"]

RFC_model_pipe.fit(X, y)
SVC_model_pipe.fit(X, y)

joblib.dump(RFC_model_pipe, "RFC_model.joblib")
joblib.dump(SVC_model_pipe, "SVC_model.joblib")
```

Code block 1: Script which is used to deploy the Machine Learning models.

Appendix E: split_test.py

```
from sklearn.model_selection import train_test_split
import joblib

dataset = joblib.load("hand_data.joblib")

X_train, X_test, y_train, y_test = train_test_split(
```

```
        dataset["data"], dataset["targets"], test_size=0.4, random_state=0)

train_split = {}

for train in y_train:
        if train not in train_split:
        train_split[train] = 1
        else:
        train_split[train] += 1
test_split = {}

for test in y_test:
        if test not in test_split:
        test_split[test] = 1
        else:
        test_split[test] += 1

print(train_split)
print(test_split)
```

Code block 1:Script used to check split of data.

All code is available at https://github.com/TomSievers/EVD3_ML_port

APPENDICES

Appendix A: Hand gestures

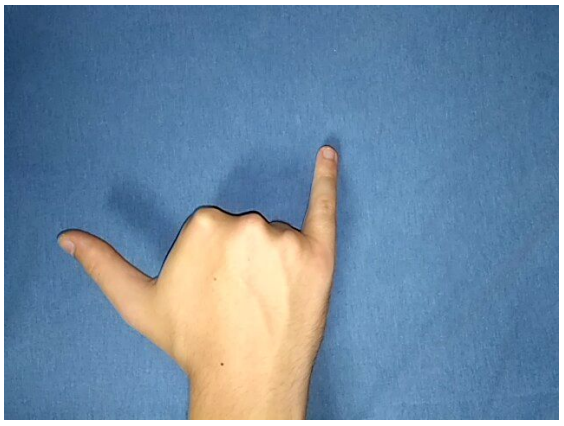

Figure 1: Stop gesture

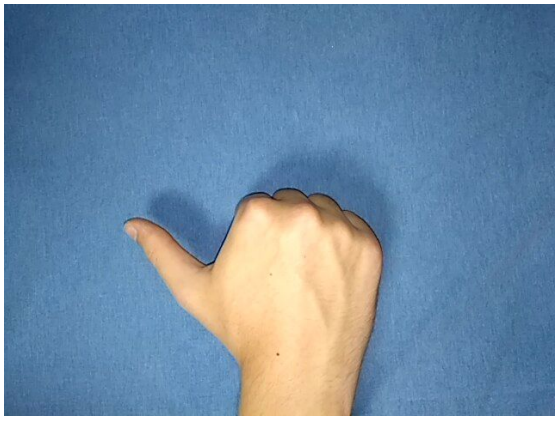

Figure 2: Start gesture

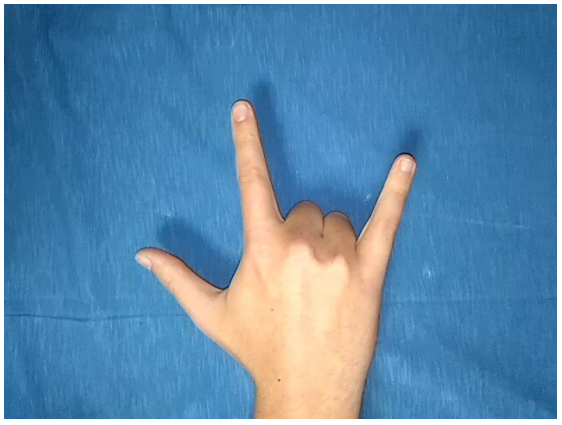

Figure 3: Calibrate gesture

Appendix B: Scatter matrix

Figure 1: Scatter matrix with all features.

Appendix C: Classifier results

This appendix contains the results of the tested classifiers. Each classifier has been tested in two iterations. The classifiers which scored the best are marked green to easily find the scores of the best classifiers.

Name: BernoulliNB iteration: 1

	precision	recall	f1-score	support
calibrate	0.67	0.94	0.78	86
ignore	0.70	0.29	0.41	80
start	0.84	0.65	0.73	74
stop	0.63	0.86	0.73	80
accuracy			0.69	320
macro avg	0.71	0.69	0.66	320
weighted avg	0.71	0.69	0.66	320

Name: BernoulliNB iteration: 2

	precision	recall	f1-score	support
calibrate	0.70	1.00	0.82	80
ignore	0.86	0.31	0.45	78
start	0.91	0.73	0.81	84
stop	0.63	0.88	0.73	78
accuracy			0.73	320
macro avg	0.77	0.73	0.70	320
weighted avg	0.77	0.73	0.71	320

Name: DecisionTreeClassifier iteration: 1

	precision	recall	f1-score	support
calibrate	0.99	1.00	0.99	86
ignore	1.00	0.96	0.98	80
start	1.00	1.00	1.00	74
stop	0.98	1.00	0.99	80
accuracy			0.99	320
macro avg	0.99	0.99	0.99	320
weighted avg	0.99	0.99	0.99	320

Name: DecisionTreeClassifier iteration: 2

	precision	recall	f1-score	support
calibrate	1.00	1.00	1.00	80
ignore	1.00	1.00	1.00	78
start	1.00	0.99	0.99	84
stop	0.99	1.00	0.99	78
accuracy			1.00	320

```
   macro avg       1.00      1.00      1.00       320
weighted avg       1.00      1.00      1.00       320
```

```
Name: ExtraTreeClassifier iteration: 1
              precision    recall  f1-score   support

   calibrate       0.99      1.00      0.99        86
      ignore       0.99      0.95      0.97        80
       start       0.97      1.00      0.99        74
        stop       0.99      0.99      0.99        80

    accuracy                           0.98       320
   macro avg       0.98      0.98      0.98       320
weighted avg       0.98      0.98      0.98       320

Name: ExtraTreeClassifier iteration: 2
              precision    recall  f1-score   support

   calibrate       1.00      1.00      1.00        80
      ignore       0.99      0.99      0.99        78
       start       1.00      0.99      0.99        84
        stop       0.99      1.00      0.99        78

    accuracy                           0.99       320
   macro avg       0.99      0.99      0.99       320
weighted avg       0.99      0.99      0.99       320
```

```
Name: ExtraTreesClassifier iteration: 1
              precision    recall  f1-score   support

   calibrate       0.99      1.00      0.99        86
      ignore       1.00      0.99      0.99        80
       start       1.00      1.00      1.00        74
        stop       1.00      1.00      1.00        80

    accuracy                           1.00       320
   macro avg       1.00      1.00      1.00       320
weighted avg       1.00      1.00      1.00       320

Name: ExtraTreesClassifier iteration: 2
              precision    recall  f1-score   support

   calibrate       1.00      1.00      1.00        80
      ignore       1.00      1.00      1.00        78
       start       1.00      0.99      0.99        84
        stop       0.99      1.00      0.99        78

    accuracy                           1.00       320
   macro avg       1.00      1.00      1.00       320
weighted avg       1.00      1.00      1.00       320
```

Name: GaussianNB iteration: 1

	precision	recall	f1-score	support
calibrate	1.00	1.00	1.00	86
ignore	0.98	0.99	0.98	80
start	1.00	1.00	1.00	74
stop	0.99	0.97	0.98	80
accuracy			0.99	320
macro avg	0.99	0.99	0.99	320
weighted avg	0.99	0.99	0.99	320

Name: GaussianNB iteration: 2

	precision	recall	f1-score	support
calibrate	1.00	1.00	1.00	80
ignore	1.00	0.95	0.97	78
start	0.98	0.99	0.98	84
stop	0.96	1.00	0.98	78
accuracy			0.98	320
macro avg	0.98	0.98	0.98	320
weighted avg	0.98	0.98	0.98	320

Name: GaussianProcessClassifier iteration: 1

	precision	recall	f1-score	support
calibrate	1.00	1.00	1.00	86
ignore	1.00	1.00	1.00	80
start	0.99	1.00	0.99	74
stop	1.00	0.99	0.99	80
accuracy			1.00	320
macro avg	1.00	1.00	1.00	320
weighted avg	1.00	1.00	1.00	320

Name: GaussianProcessClassifier iteration: 2

	precision	recall	f1-score	support
calibrate	1.00	1.00	1.00	80
ignore	1.00	0.99	0.99	78
start	0.99	0.99	0.99	84
stop	0.99	1.00	0.99	78
accuracy			0.99	320
macro avg	0.99	0.99	0.99	320
weighted avg	0.99	0.99	0.99	320

Name: GradientBoostingClassifier iteration: 1

	precision	recall	f1-score	support

```
   calibrate       0.99      1.00      0.99        86
      ignore       1.00      0.95      0.97        80
       start       0.99      1.00      0.99        74
        stop       0.98      1.00      0.99        80

    accuracy                           0.99       320
   macro avg       0.99      0.99      0.99       320
weighted avg       0.99      0.99      0.99       320

Name: GradientBoostingClassifier iteration: 2
              precision    recall  f1-score   support

   calibrate       1.00      1.00      1.00        80
      ignore       1.00      0.99      0.99        78
       start       0.99      0.99      0.99        84
        stop       0.99      1.00      0.99        78

    accuracy                           0.99       320
   macro avg       0.99      0.99      0.99       320
weighted avg       0.99      0.99      0.99       320
```

```
Name: KNeighborsClassifier iteration: 1
              precision    recall  f1-score   support

   calibrate       1.00      1.00      1.00        86
      ignore       1.00      1.00      1.00        80
       start       0.99      1.00      0.99        74
        stop       1.00      0.99      0.99        80

    accuracy                           1.00       320
   macro avg       1.00      1.00      1.00       320
weighted avg       1.00      1.00      1.00       320

Name: KNeighborsClassifier iteration: 2
              precision    recall  f1-score   support

   calibrate       1.00      1.00      1.00        80
      ignore       1.00      0.99      0.99        78
       start       0.99      1.00      0.99        84
        stop       1.00      1.00      1.00        78

    accuracy                           1.00       320
   macro avg       1.00      1.00      1.00       320
weighted avg       1.00      1.00      1.00       320
```

```
Name: LinearDiscriminantAnalysis iteration: 1
              precision    recall  f1-score   support

   calibrate       0.86      0.98      0.91        86
      ignore       1.00      0.80      0.89        80
       start       0.93      1.00      0.96        74
```

```
       stop       0.95      0.93      0.94        80

   accuracy                           0.93       320
  macro avg       0.93      0.93      0.92       320
weighted avg      0.93      0.93      0.92       320

Name: LinearDiscriminantAnalysis iteration: 2
              precision    recall  f1-score   support

   calibrate       0.89      0.99      0.93        80
      ignore       1.00      0.86      0.92        78
       start       0.98      1.00      0.99        84
        stop       0.97      0.97      0.97        78

   accuracy                           0.96       320
  macro avg       0.96      0.96      0.96       320
weighted avg      0.96      0.96      0.96       320
```

```
 Name: LinearSVC iteration: 1
              precision    recall  f1-score   support

   calibrate       0.88      0.88      0.88        86
      ignore       0.87      0.86      0.87        80
       start       0.94      1.00      0.97        74
        stop       0.99      0.94      0.96        80

   accuracy                           0.92       320
  macro avg       0.92      0.92      0.92       320
weighted avg      0.92      0.92      0.92       320

Name: LinearSVC iteration: 2
              precision    recall  f1-score   support

   calibrate       0.80      0.89      0.84        80
      ignore       0.88      0.88      0.88        78
       start       0.99      0.99      0.99        84
        stop       0.99      0.87      0.93        78

   accuracy                           0.91       320
  macro avg       0.91      0.91      0.91       320
weighted avg      0.91      0.91      0.91       320
```

```
Name: LogisticRegression iteration: 1
              precision    recall  f1-score   support

   calibrate       0.85      0.91      0.88        86
      ignore       0.89      0.82      0.86        80
       start       0.94      1.00      0.97        74
        stop       1.00      0.94      0.97        80

   accuracy                           0.92       320
  macro avg       0.92      0.92      0.92       320
```

```
weighted avg       0.92      0.92      0.92       320

Name: LogisticRegression iteration: 2
              precision    recall  f1-score   support

   calibrate       0.92      0.91      0.92        80
      ignore       0.91      0.90      0.90        78
       start       0.97      0.99      0.98        84
        stop       0.97      0.97      0.97        78

    accuracy                           0.94       320
   macro avg       0.94      0.94      0.94       320
weighted avg       0.94      0.94      0.94       320
```

```
Name: MLPClassifier iteration: 1
              precision    recall  f1-score   support

   calibrate       0.99      1.00      0.99        86
      ignore       1.00      0.99      0.99        80
       start       0.97      1.00      0.99        74
        stop       1.00      0.97      0.99        80

    accuracy                           0.99       320
   macro avg       0.99      0.99      0.99       320
weighted avg       0.99      0.99      0.99       320

Name: MLPClassifier iteration: 2
              precision    recall  f1-score   support

   calibrate       1.00      1.00      1.00        80
      ignore       1.00      1.00      1.00        78
       start       1.00      0.99      0.99        84
        stop       0.99      1.00      0.99        78

    accuracy                           1.00       320
   macro avg       1.00      1.00      1.00       320
weighted avg       1.00      1.00      1.00       320
```

```
Name: NearestCentroid iteration: 1
              precision    recall  f1-score   support

   calibrate       0.59      0.84      0.69        86
      ignore       0.45      0.17      0.25        80
       start       0.80      0.96      0.87        74
        stop       0.96      0.94      0.95        80

    accuracy                           0.73       320
   macro avg       0.70      0.73      0.69       320
weighted avg       0.70      0.72      0.69       320

Name: NearestCentroid iteration: 2
```

```
              precision    recall  f1-score   support

   calibrate       0.79      0.80      0.80        80
      ignore       0.68      0.49      0.57        78
       start       0.84      0.98      0.90        84
        stop       0.89      0.97      0.93        78

    accuracy                           0.81       320
   macro avg       0.80      0.81      0.80       320
weighted avg       0.80      0.81      0.80       320
```

```
Name: NuSVC iteration: 1
              precision    recall  f1-score   support

   calibrate       1.00      1.00      1.00        86
      ignore       1.00      1.00      1.00        80
       start       0.95      1.00      0.97        74
        stop       1.00      0.95      0.97        80

    accuracy                           0.99       320
   macro avg       0.99      0.99      0.99       320
weighted avg       0.99      0.99      0.99       320

Name: NuSVC iteration: 2
              precision    recall  f1-score   support

   calibrate       1.00      1.00      1.00        80
      ignore       1.00      0.99      0.99        78
       start       0.98      0.99      0.98        84
        stop       0.99      0.99      0.99        78

    accuracy                           0.99       320
   macro avg       0.99      0.99      0.99       320
weighted avg       0.99      0.99      0.99       320
```

```
Name: PassiveAggressiveClassifier iteration: 1
              precision    recall  f1-score   support

   calibrate       0.76      0.92      0.83        86
      ignore       0.89      0.70      0.78        80
       start       0.94      0.99      0.96        74
        stop       0.99      0.93      0.95        80

    accuracy                           0.88       320
   macro avg       0.89      0.88      0.88       320
weighted avg       0.89      0.88      0.88       320

Name: PassiveAggressiveClassifier iteration: 2
              precision    recall  f1-score   support

   calibrate       0.87      0.90      0.88        80
```

ignore	0.92	0.87	0.89	78
start	0.95	0.99	0.97	84
stop	0.93	0.91	0.92	78
accuracy			0.92	320
macro avg	0.92	0.92	0.92	320
weighted avg	0.92	0.92	0.92	320

Name: Perceptron iteration: 1

	precision	recall	f1-score	support
calibrate	0.92	0.76	0.83	86
ignore	0.78	0.94	0.85	80
start	0.90	1.00	0.95	74
stop	1.00	0.89	0.94	80
accuracy			0.89	320
macro avg	0.90	0.90	0.89	320
weighted avg	0.90	0.89	0.89	320

Name: Perceptron iteration: 2

	precision	recall	f1-score	support
calibrate	0.88	0.79	0.83	80
ignore	0.81	0.92	0.86	78
start	0.97	0.99	0.98	84
stop	0.99	0.92	0.95	78
accuracy			0.91	320
macro avg	0.91	0.91	0.91	320
weighted avg	0.91	0.91	0.91	320

Name: QuadraticDiscriminantAnalysis iteration: 1

	precision	recall	f1-score	support
calibrate	1.00	1.00	1.00	86
ignore	1.00	0.99	0.99	80
start	1.00	1.00	1.00	74
stop	0.99	1.00	0.99	80
accuracy			1.00	320
macro avg	1.00	1.00	1.00	320
weighted avg	1.00	1.00	1.00	320

Name: QuadraticDiscriminantAnalysis iteration: 2

	precision	recall	f1-score	support
calibrate	1.00	1.00	1.00	80
ignore	1.00	0.99	0.99	78
start	1.00	0.99	0.99	84
stop	0.97	1.00	0.99	78

```
   accuracy                         0.99       320
  macro avg       0.99      0.99      0.99       320
weighted avg       0.99      0.99      0.99       320
```

```
Name: RandomForestClassifier iteration: 1
              precision    recall  f1-score   support

   calibrate       0.99      1.00      0.99        86
      ignore       1.00      0.99      0.99        80
       start       1.00      1.00      1.00        74
        stop       1.00      1.00      1.00        80

   accuracy                           1.00       320
  macro avg       1.00      1.00      1.00       320
weighted avg       1.00      1.00      1.00       320

Name: RandomForestClassifier iteration: 2
              precision    recall  f1-score   support

   calibrate       1.00      1.00      1.00        80
      ignore       1.00      0.99      0.99        78
       start       0.99      0.99      0.99        84
        stop       0.99      1.00      0.99        78

   accuracy                           0.99       320
  macro avg       0.99      0.99      0.99       320
weighted avg       0.99      0.99      0.99       320
```

```
Name: RidgeClassifier iteration: 1
              precision    recall  f1-score   support

   calibrate       0.85      0.95      0.90        86
      ignore       1.00      0.81      0.90        80
       start       0.87      1.00      0.93        74
        stop       0.93      0.86      0.90        80

   accuracy                           0.91       320
  macro avg       0.91      0.91      0.91       320
weighted avg       0.91      0.91      0.91       320

Name: RidgeClassifier iteration: 2
              precision    recall  f1-score   support

   calibrate       0.86      0.97      0.91        80
      ignore       1.00      0.85      0.92        78
       start       0.90      0.95      0.92        84
        stop       0.91      0.86      0.88        78

   accuracy                           0.91       320
  macro avg       0.92      0.91      0.91       320
```

```
weighted avg       0.91      0.91      0.91       320
```

```
Name: SGDClassifier iteration: 1
              precision    recall  f1-score   support

   calibrate       0.88      0.88      0.88        86
      ignore       0.88      0.89      0.88        80
       start       0.95      0.99      0.97        74
        stop       0.97      0.93      0.95        80

    accuracy                           0.92       320
   macro avg       0.92      0.92      0.92       320
weighted avg       0.92      0.92      0.92       320

Name: SGDClassifier iteration: 2
              precision    recall  f1-score   support

   calibrate       0.78      0.91      0.84        80
      ignore       0.90      0.85      0.87        78
       start       0.97      0.99      0.98        84
        stop       0.99      0.85      0.91        78

    accuracy                           0.90       320
   macro avg       0.91      0.90      0.90       320
weighted avg       0.91      0.90      0.90       320
```

```
Name: SVC iteration: 1
              precision    recall  f1-score   support

   calibrate       1.00      1.00      1.00        86
      ignore       1.00      1.00      1.00        80
       start       0.99      1.00      0.99        74
        stop       1.00      0.99      0.99        80

    accuracy                           1.00       320
   macro avg       1.00      1.00      1.00       320
weighted avg       1.00      1.00      1.00       320

Name: SVC iteration: 2
              precision    recall  f1-score   support

   calibrate       0.99      1.00      0.99        80
      ignore       1.00      0.99      0.99        78
       start       1.00      0.99      0.99        84
        stop       0.99      1.00      0.99        78

    accuracy                           0.99       320
   macro avg       0.99      0.99      0.99       320
weighted avg       0.99      0.99      0.99       320
```

Table 6: Machine Learning models scores.