

University of Craiova
Faculty of Automation Computers and Electronics



Machine Learning

Project: Driving Behaviour

Student

Alex Smarandache

Computers and Information Technology

Study group S2

Year IV

January 2023

1 About this project

1.1 Context

Aggressive driving behavior is the leading factor of road traffic accidents. As reported by the AAA Foundation for Traffic Safety, 106,727 fatal crashes – 55.7 percent of the total – during a recent four-year period involved drivers who committed one or more aggressive driving actions. Therefore, how to predict dangerous driving behavior quickly and accurately?

1.2 Solution Approach

Aggressive driving includes speeding, sudden breaks and sudden left or right turns. All these events are reflected on accelerometer and gyroscope data. Therefore, knowing that almost everyone owns a smartphone nowadays which has a wide variety of sensors, we've designed a data collector application in android based on the accelerometer and gyroscope sensors.

2 Dataset

The training file contains the following columns:

- AccX: X acceleration axis
- AccY: Y acceleration axis
- AccZ: Z acceleration axis
- GyroX: X orientation axis
- GyroY: Y orientation axis
- GyroZ: Z orientation axis
- Class: classification label
- Timestamp: time in seconds

Based on the records in this file, we must specify the class to which a multitude of other records in the test file belong. It should be mentioned that the last two columns are not found in the test file.

The prediction file has the following format:

ID, CLASS

Where id is the number of the record in the test file and class is the predicted class for that record.

3 Code implementation

Initial imports:

```
[ ] import numpy as np # linear algebra
    import pandas as pd # data processing, CSV file I/O (e.g. pd.read_csv)
```

Parsing the training file:

```
✓ 0 sec. ▶ train_data = pd.read_csv("train_motion_data.csv")
    features = ['AccX', 'AccY', 'AccZ', 'GyroX', 'GyroY', 'GyroZ']
    #features = ['AccX', 'AccY', 'AccZ']
    x_train = train_data[features].dropna()
    y_train = train_data['Class']
```

The hyperparameters will be the columns found in the records in the test file: AccX, AccY, AccZ, GyroX, GyroY, and GyroZ. Also, I tried different combinations for them, other very good results were recorded when I included only the acceleration axes.

I have processed the data so that there are no unspecified elements.

Prepare the test data:

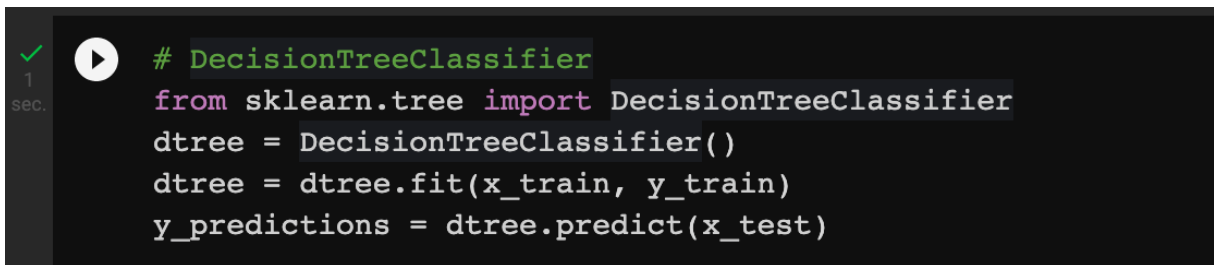
```
✓ 0 sec. [4] test_data = pd.read_csv("Test.csv")
    x_test = test_data[features].dropna()
```

I removed unspecified or null values from the test file as well.

3.1 Classification algorithms

3.1.1 Decision tree classifier

The decision tree classifier creates the classification model by building a decision tree. Each node in the tree specifies a test on an attribute, each branch descending from that node corresponds to one of the possible values for that attribute.



```

✓ 1 # DecisionTreeClassifier
sec. from sklearn.tree import DecisionTreeClassifier
dtree = DecisionTreeClassifier()
dtree = dtree.fit(x_train, y_train)
y_predictions = dtree.predict(x_test)

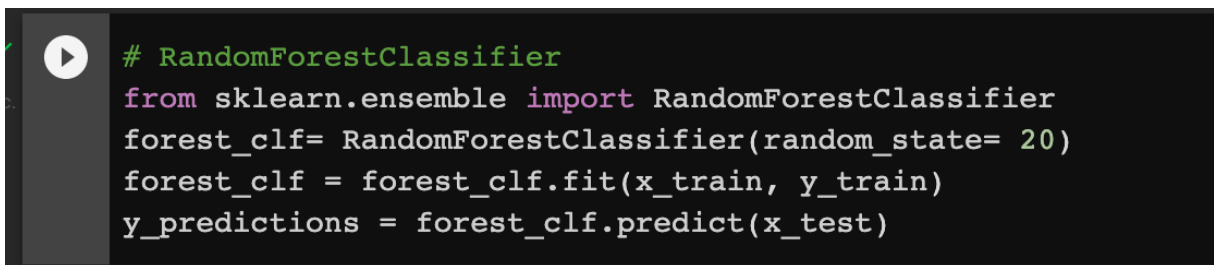
```

With this algorithm I obtained the following results on the entire test file:

- a) 0.32799(AccX, AccY, AccZ, GyroX, GyroY, and GyroZ)
- b) 0.32429(AccX, AccY and AccZ)

3.1.2 Random forest classifier

The random forest classifier can be used to solve for regression or classification problems. The random forest algorithm is made up of a collection of decision trees, and each tree in the ensemble is comprised of a data sample drawn from a training set with replacement, called the bootstrap sample.



```

✓ # RandomForestClassifier
from sklearn.ensemble import RandomForestClassifier
forest_clf= RandomForestClassifier(random_state= 20)
forest_clf = forest_clf.fit(x_train, y_train)
y_predictions = forest_clf.predict(x_test)

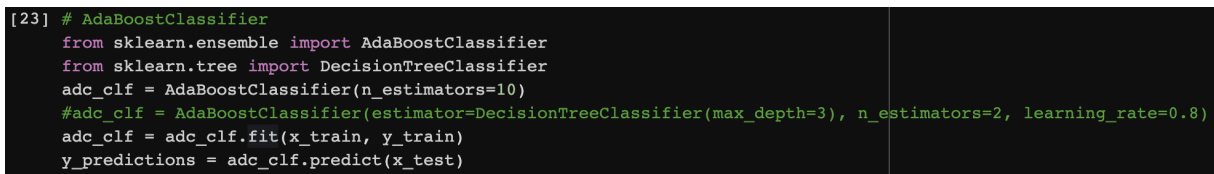
```

With this algorithm I obtained the following results on the entire test file:

- a) 0.35224(AccX, AccY, AccZ, GyroX, GyroY, and GyroZ)
- b) 0.34115(AccX, AccY and AccZ)

3.1.3 AdaBoost classifier

An AdaBoost classifier is a meta-estimator that begins by fitting a classifier on the original dataset and then fits additional copies of the classifier on the same dataset but where the weights of incorrectly classified instances are adjusted such that subsequent classifiers focus more on difficult cases.



```

[23] # AdaBoostClassifier
from sklearn.ensemble import AdaBoostClassifier
from sklearn.tree import DecisionTreeClassifier
adc_clf = AdaBoostClassifier(n_estimators=10)
#adc_clf = AdaBoostClassifier(estimator=DecisionTreeClassifier(max_depth=3), n_estimators=2, learning_rate=0.8)
adc_clf = adc_clf.fit(x_train, y_train)
y_predictions = adc_clf.predict(x_test)

```

With this algorithm I obtained the following results on the entire test file:

- a) 0.37854(AccX, AccY, AccZ, GyroX, GyroY, and GyroZ)
- b) 0.38183(AccX, AccY and AccZ)

3.1.4 MLP classifier

MLPClassifier supports multi-class classification by applying Softmax as the output function. Further, the model supports multi-label classification in which a sample can belong to more than one class. For each class, the raw output passes through the logistic function.

```
[ ] # MLPClassifier
    from sklearn.neural_network import MLPClassifier
    mlp_clf = MLPClassifier(early_stopping=True, batch_size=64, verbose=True,
                           learning_rate_init=0.001,
                           hidden_layer_sizes=(32, 64, 128), random_state=0)
    mlp_clf = mlp_clf.fit(x_train, y_train)
    y_predictions = mlp_clf.predict(x_test)
```

With this algorithm I obtained the following results on the entire test file:

- a) 0.38471(AccX, AccY, AccZ, GyroX, GyroY, and GyroZ)
- b) 0.38758(AccX, AccY and AccZ)

3.1.5 SVC classifier

SVC, or Support Vector Classifier, is a supervised machine learning algorithm typically used for classification tasks. SVC works by mapping data points to a high-dimensional space and then finding the optimal hyperplane that divides the data into two classes.

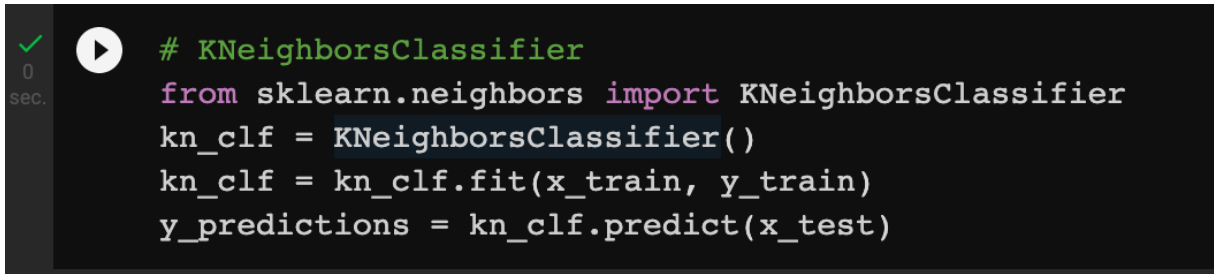
```
✓ [42] # SVC
1   from sklearn.svm import SVC
sec. svc = SVC(kernel='rbf')
     svc = svc.fit(x_train, y_train)
     y_predictions = svc.predict(x_test)
```

With this algorithm I obtained the following results on the entire test file:

- a) 0.40032(AccX, AccY, AccZ, GyroX, GyroY, and GyroZ)
- b) 0.39621(AccX, AccY and AccZ)

3.1.6 KNeighbors classifier

The k-nearest neighbors algorithm, also known as KNN or k-NN, is a non-parametric, supervised learning classifier, which uses proximity to make classifications or predictions about the grouping of an individual data point. While it can be used for either regression or classification problems, it is typically used as a classification algorithm, working off the assumption that similar points can be found near one another.



```

✓ 0 sec. # KNeighborsClassifier
from sklearn.neighbors import KNeighborsClassifier
kn_clf = KNeighborsClassifier()
kn_clf = kn_clf.fit(x_train, y_train)
y_predictions = kn_clf.predict(x_test)

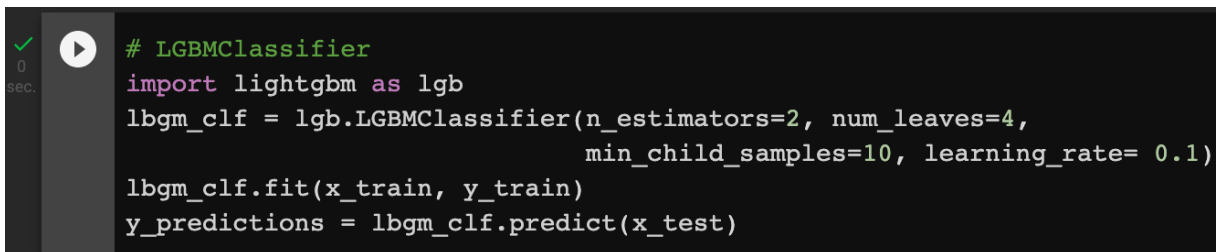
```

With this algorithm I obtained the following results on the entire test file:

- a) 0.34196(AccX, AccY, AccZ, GyroX, GyroY, and GyroZ)
- b) 0.33333(AccX, AccY and AccZ)

3.1.7 LGBM classifier

LightGBM is a gradient boosting ensemble method that is used by the Train Using AutoML tool and is based on decision trees. As with other decision tree-based methods, LightGBM can be used for both classification and regression. LightGBM is optimized for high performance with distributed systems.



```

✓ 0 sec. # LGBMClassifier
import lightgbm as lgb
lbgm_clf = lgb.LGBMClassifier(n_estimators=2, num_leaves=4,
                             min_child_samples=10, learning_rate= 0.1)
lbgm_clf.fit(x_train, y_train)
y_predictions = lbgm_clf.predict(x_test)

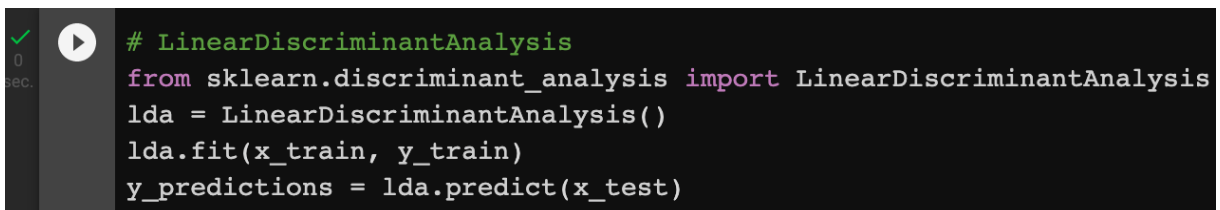
```

With this algorithm I obtained the following results on the entire test file:

- a) 0.38101(AccX, AccY, AccZ, GyroX, GyroY, and GyroZ)
- b) 0.38101(AccX, AccY and AccZ)

3.1.8 Linear Discriminant Analysis

LDA is mainly used in classification problems where you have a categorical output variable. It allows both binary classification and multi-class classification. The standard LDA model makes use of the Gaussian Distribution of the input variables.



```

✓ 0 sec. # LinearDiscriminantAnalysis
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis
lda = LinearDiscriminantAnalysis()
lda.fit(x_train, y_train)
y_predictions = lda.predict(x_test)

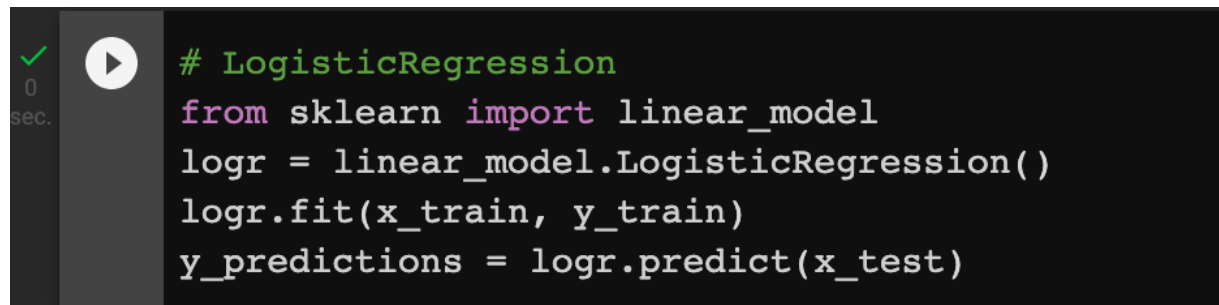
```

With this algorithm I obtained the following results on the entire test file:

- a) 0.37556(AccX, AccY, AccZ, GyroX, GyroY, and GyroZ)
- b) 0.37114(AccX, AccY and AccZ)

3.1.9 Logistic Regression

Logistic regression is a statistical method used to predict the outcome of a dependent variable based on previous observations. It's a type of regression analysis and is a commonly used algorithm for solving binary classification problems.

A screenshot of a Jupyter Notebook cell. On the left, there is a green checkmark and a play button icon. The text '0 sec.' is visible. The code in the cell is as follows:

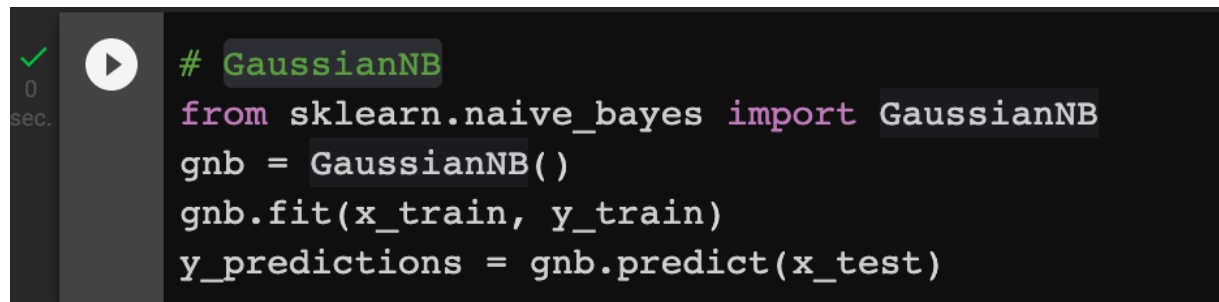
```
# LogisticRegression
from sklearn import linear_model
logr = linear_model.LogisticRegression()
logr.fit(x_train, y_train)
y_predictions = logr.predict(x_test)
```

With this algorithm I obtained the following results on the entire test file:

- a) 0.37713(AccX, AccY, AccZ, GyroX, GyroY, and GyroZ)
- b) 0.37115(AccX, AccY and AccZ)

3.1.10 Gaussian Naive Bayes

Gaussian Naive Bayes (GNB) is a classification technique used in Machine Learning (ML) based on the probabilistic approach and Gaussian distribution. Gaussian Naive Bayes assumes that each parameter (also called features or predictors) has an independent capacity of predicting the output variable.

A screenshot of a Jupyter Notebook cell. On the left, there is a green checkmark and a play button icon. The text '0 sec.' is visible. The code in the cell is as follows:

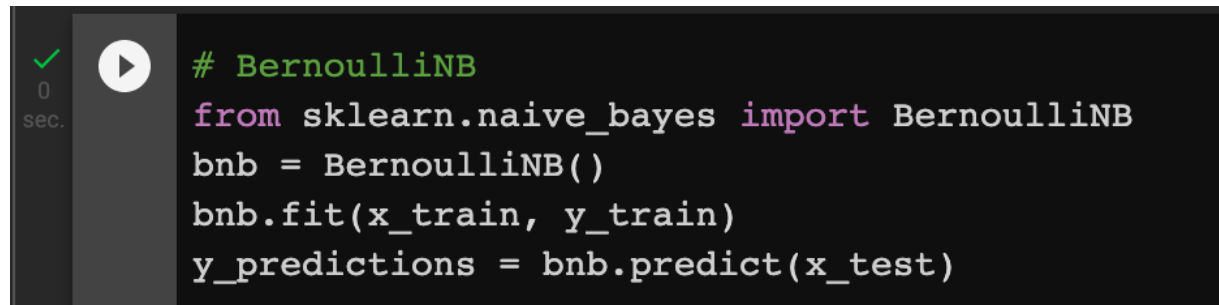
```
# GaussianNB
from sklearn.naive_bayes import GaussianNB
gnb = GaussianNB()
gnb.fit(x_train, y_train)
y_predictions = gnb.predict(x_test)
```

With this algorithm I obtained the following results on the entire test file:

- a) 0.40156(AccX, AccY, AccZ, GyroX, GyroY, and GyroZ)
- b) 0.40073(AccX, AccY and AccZ)

3.1.11 Bernoulli Naive Bayes

BernoulliNB implements the naive Bayes training and classification algorithms for data that is distributed according to multivariate Bernoulli distributions; i.e., there may be multiple features but each one is assumed to be a binary-valued (Bernoulli, boolean) variable.

A Jupyter Notebook snippet showing the use of BernoulliNB. It includes a green checkmark, a play button icon, and a timer showing '0 sec.'. The code is as follows:

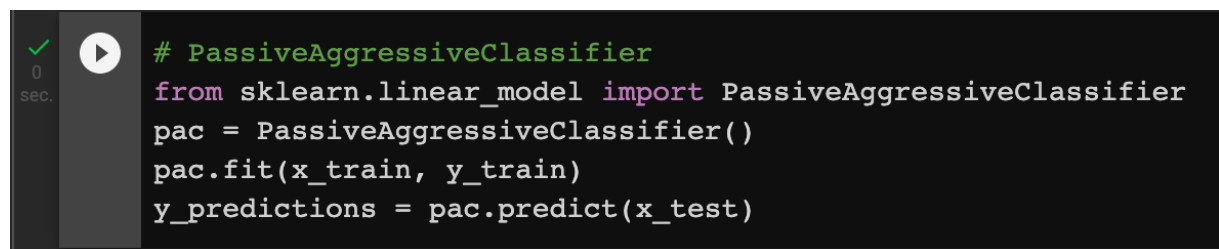
```
# BernoulliNB
from sklearn.naive_bayes import BernoulliNB
bnb = BernoulliNB()
bnb.fit(x_train, y_train)
y_predictions = bnb.predict(x_test)
```

With this algorithm I obtained the following results on the entire test file:

- a) 0.37731(AccX, AccY, AccZ, GyroX, GyroY, and GyroZ)
- b) 0.38060(AccX, AccY and AccZ)

3.1.12 Passive Aggressive Classifier

The Passive-Aggressive algorithms are a family of Machine learning algorithms that are not very well known by beginners and even intermediate Machine Learning enthusiasts. However, they can be very useful and efficient for certain applications.

A Jupyter Notebook snippet showing the use of PassiveAggressiveClassifier. It includes a green checkmark, a play button icon, and a timer showing '0 sec.'. The code is as follows:

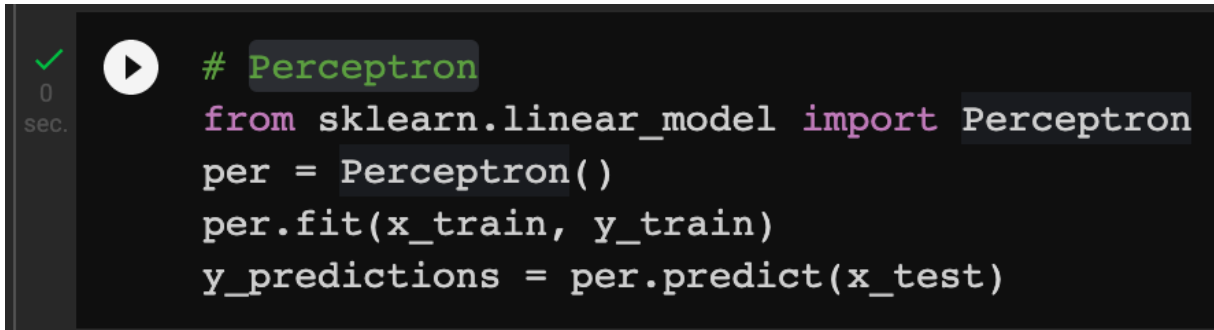
```
# PassiveAggressiveClassifier
from sklearn.linear_model import PassiveAggressiveClassifier
pac = PassiveAggressiveClassifier()
pac.fit(x_train, y_train)
y_predictions = pac.predict(x_test)
```

With this algorithm I obtained the following results on the entire test file:

- a) 0.35429(AccX, AccY, AccZ, GyroX, GyroY, and GyroZ)
- b) 0.38347(AccX, AccY and AccZ)

3.1.13 Perceptron

The Perceptron Classifier is a linear algorithm that can be applied to binary classification tasks. How to fit, evaluate, and make predictions with the Perceptron model with Scikit-Learn. How to tune the hyperparameters of the Perceptron algorithm on a given dataset.

A screenshot of a Jupyter Notebook code cell. On the left, there is a green checkmark and a play button icon. The code is written in Python and uses the Perceptron classifier from sklearn.linear_model. The code is as follows:

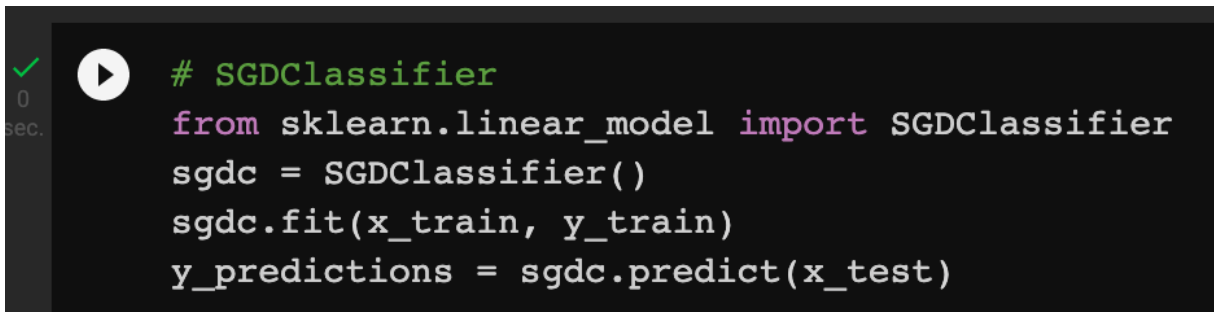
```
# Perceptron
from sklearn.linear_model import Perceptron
per = Perceptron()
per.fit(x_train, y_train)
y_predictions = per.predict(x_test)
```

With this algorithm I obtained the following results on the entire test file:

- a) 0.37443(AccX, AccY, AccZ, GyroX, GyroY, and GyroZ)
- b) 0.37443(AccX, AccY and AccZ)

3.1.14 SGD Classifier

SGD Classifier is a linear classifier (SVM, logistic regression, a.o.) optimized by the SGD. These are two different concepts. While SGD is a optimization method, Logistic Regression or linear Support Vector Machine is a machine learning algorithm/model.

A screenshot of a Jupyter Notebook code cell. On the left, there is a green checkmark and a play button icon. The code is written in Python and uses the SGDClassifier from sklearn.linear_model. The code is as follows:

```
# SGDClassifier
from sklearn.linear_model import SGDClassifier
sgdc = SGDClassifier()
sgdc.fit(x_train, y_train)
y_predictions = sgdc.predict(x_test)
```

With this algorithm I obtained the following results on the entire test file:

- a) 0.37603(AccX, AccY, AccZ, GyroX, GyroY, and GyroZ)
- b) 0.34648(AccX, AccY and AccZ)

3.1.15 Quadratic Discriminant Analysis

QDA, because it allows for more flexibility for the covariance matrix, tends to fit the data better than LDA, but then it has more parameters to estimate. The number of parameters increases significantly with QDA. Because, with QDA, you will have a separate covariance matrix for every class.

```

✓ [123] # QuadraticDiscriminantAnalysis
0      from sklearn.discriminant_analysis import QuadraticDiscriminantAnalysis
sec.   qda = QuadraticDiscriminantAnalysis()
       qda.fit(x_train, y_train)
       y_predictions = qda.predict(x_test)

```

With this algorithm I obtained the following results on the entire test file:

- a) 0.39745(AccX, AccY, AccZ, GyroX, GyroY, and GyroZ)
- b) 0.39827(AccX, AccY and AccZ)

Map the predictions into an output with the requirements specifications:

```

✓ 0      ids = test_data['ID'].dropna()
sec.   Final_predictions= pd.DataFrame(y_predictions, columns=['Class'])
       Final_predictions.index.name='ID'
       Final_predictions.index += 1
       Final_predictions= Final_predictions.to_csv(
           'Final_Predictions.csv', header= 'Class', index='true')

```

4 Observations

I tested a total of 15 algorithms, both on 2 sets of hyperparameters.

The five best algorithms with hyperparameters composed of: AccX, AccY, AccZ, GyroX, GyroY, and GyroZ:

1. Gaussian Naive Bayes(0.40156)
2. SVC(0.40032)
3. Quadratic Discriminant Analysis(0.39745)
4. MLP(0.38471)
5. LGBM(0.38101)

The five best algorithms with hyperparameters composed of: AccX, AccY and AccZ:

1. Gaussian Naive Bayes(0.40073)
2. Quadratic Discriminant Analysis(0.39827)
3. SVC(0.39621)
4. MLP(0.38758)
5. Passive Aggressive(0.38347)

5 Conclusions

This competition was an excellent start for me in the field of Machine Learning, although the code was not extremely complicated to make, I put a considerable effort into finding the most suitable algorithms for this data set. What surprised me was the difference between the score of the algorithms on the data set made up of approximately 30% of the recordings and the complete one, with rather large differences being recorded in some situations, both positive and negative. So, it was an interesting experience and it was definitely just the first of many Machine Learning competitions that I will participate in because it is an interesting and future field.

6 Bibliography

References

- [1] Wikipedia,https://en.wikipedia.org/wiki/Category:Classification_algorithms, accessed in January 2023.
- [2] Driving Behaviour,<https://www.kaggle.com/competitions/driving-behaviour/overview>, accessed in January 2023.