Olive Oil Classification

REALIZED BY:

NAIM DALI HASSEN

MED HEDI KHEMIRI

EMNA CHENIOUR

HATEM LAMINE

GL4 2023

github link: https://github.com/NaimDali/Deep_Learning_Project

General Introduction

Olive oil is the most popular cooking oil. It has been used for thousands of years. It is a key ingredient in Mediterranean cuisine and is widely used in cooking around the world.
One of the reasons olive oil is so popular is its health benefits. It is high in monounsaturated fatty acids, which are considered healthy fats that can help lower cholesterol and reduce the risk of heart disease. Olive oil also contains antioxidants, which can help protect against cell damage and inflammation.

Project Context

Olive oil is widely used in the fields of nutrition, health, and agriculture. Scientists and researchers are interested in understanding the health benefits of olive oil, its chemical composition, and the effects of different processing methods on its quality and nutritional value. Additionally, olive oil production is an important industry in many countries, and efforts are being made to improve the sustainability and efficiency of olive oil production processes. As a result, there is a wealth of information available on olive oil that can be explored and analyzed from various perspectives.

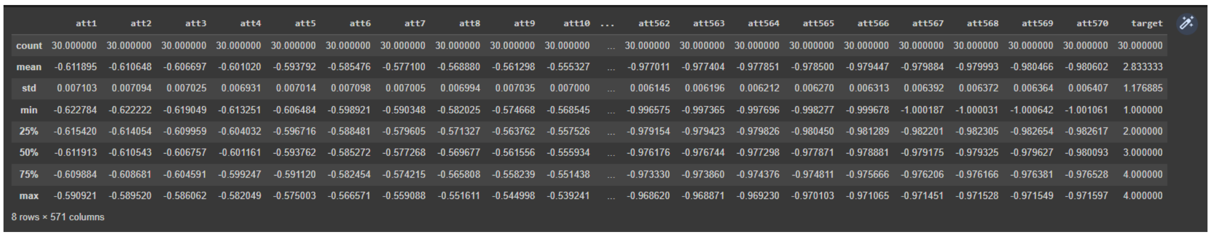
Data Collection

The data needed for this project was provided. All we had to do was to load and process data from two different ARFF files, which contain training and testing data for our project. We then converted it into Pandas dataframes.
Next, we focused on the 'target' column of both dataframes to convert the labels from bytes to integers. This is necessary because the target column contains the class labels of the different types of olive oil that are being classified. Finally, we counted the number of occurrences of each class label in the training data using the value_counts() method and creates a bar plot to

visualize the distribution of classes to help us identify any class imbalances or biases in the data, which can affect the performance of machine learning models.

Data understanding

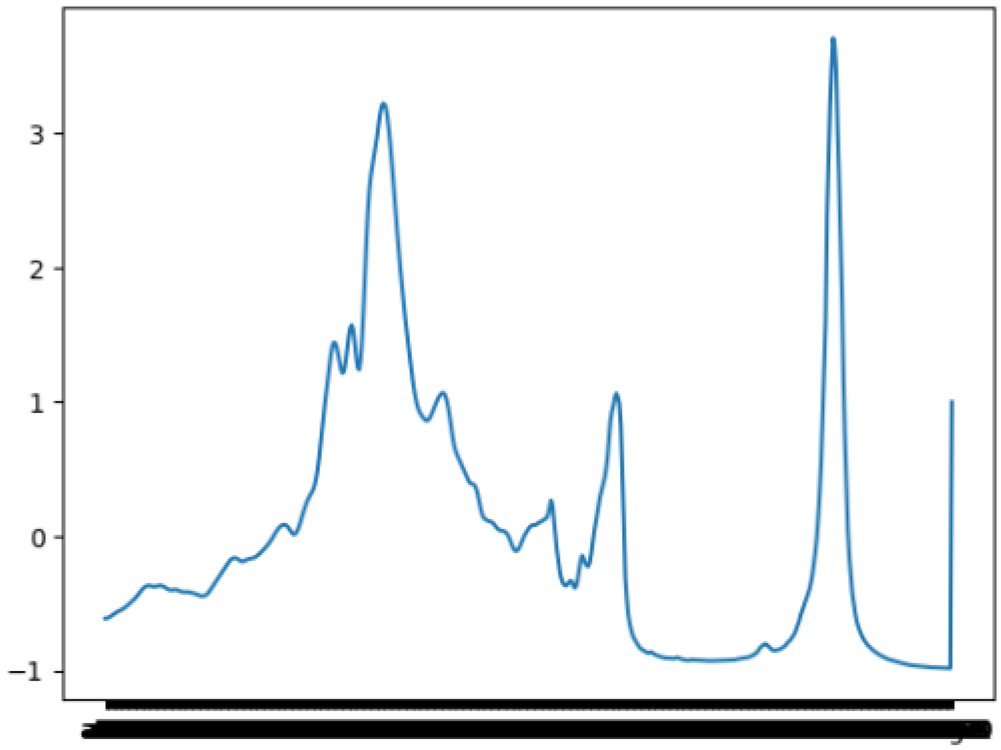
the shape of both datasets: train_df and test_df is (30, 571).
We use the method describe to describe our train dataset



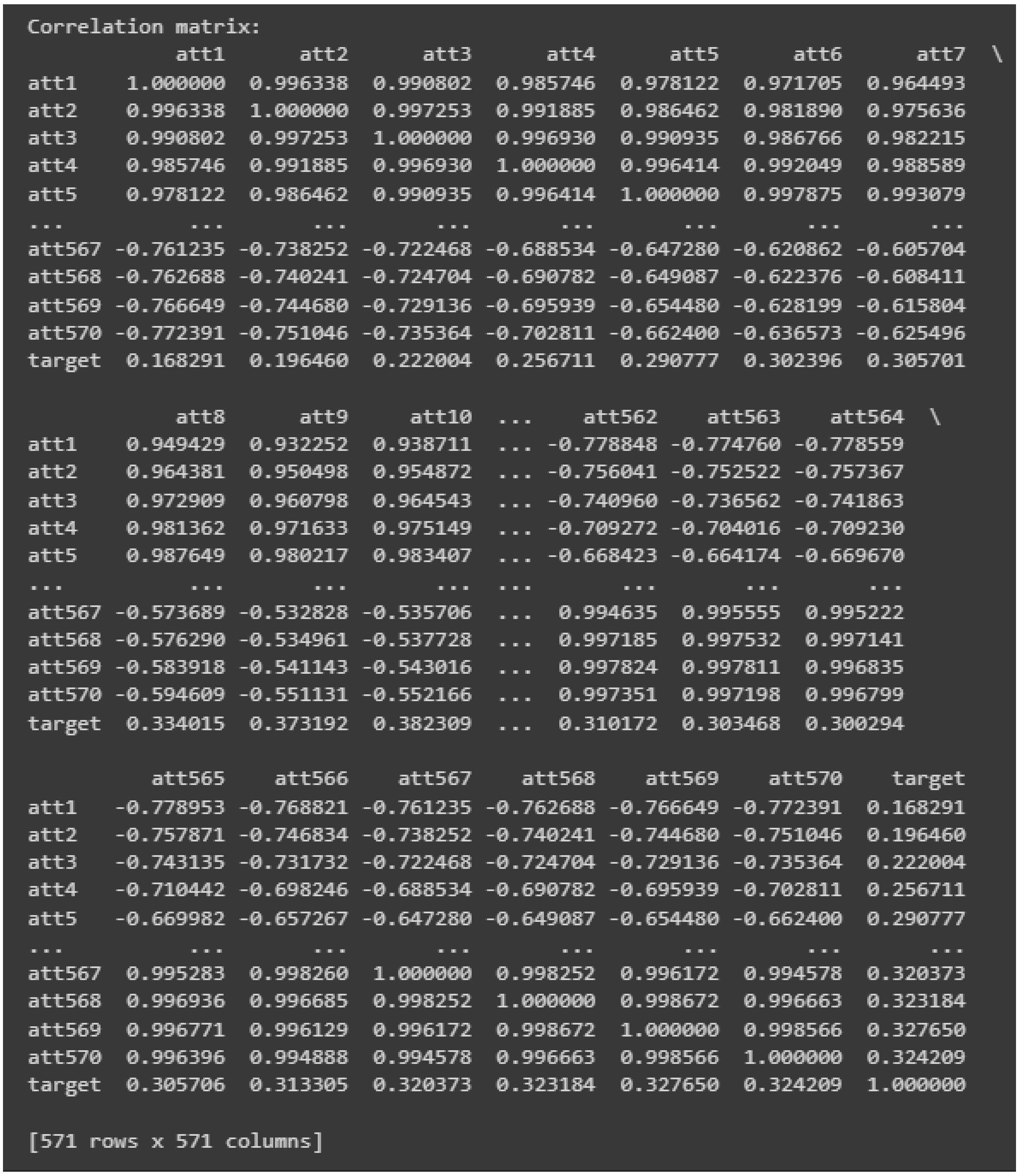
	att1	att2	att3	att4	att5	att6	att7	att8	att9	att10	...	att562	att563	att564	att565	att566	att567	att568	att569	att570	target
count	30.000000	30.000000	30.000000	30.000000	30.000000	30.000000	30.000000	30.000000	30.000000	30.000000	...	30.000000	30.000000	30.000000	30.000000	30.000000	30.000000	30.000000	30.000000	30.000000	30.000000
mean	-0.611895	-0.610648	-0.606697	-0.601020	-0.593792	-0.585476	-0.577100	-0.568880	-0.561298	-0.555327	...	-0.977011	-0.977404	-0.977851	-0.978500	-0.979447	-0.979884	-0.979993	-0.980466	-0.980602	2.833333
std	0.007103	0.007094	0.007025	0.006931	0.007014	0.007098	0.007005	0.006994	0.007035	0.007000	...	0.006145	0.006196	0.006212	0.006270	0.006313	0.006392	0.006372	0.006364	0.006407	1.176885
min	-0.622784	-0.622222	-0.619049	-0.613251	-0.606484	-0.598921	-0.590348	-0.582025	-0.574668	-0.568545	...	-0.996575	-0.997365	-0.997696	-0.998277	-0.999678	-1.000187	-1.000031	-1.000642	-1.001061	1.000000
25%	-0.615420	-0.614054	-0.609959	-0.604032	-0.596716	-0.588481	-0.579605	-0.571327	-0.563762	-0.557526	...	-0.979154	-0.979423	-0.979826	-0.980450	-0.981289	-0.982201	-0.982305	-0.982654	-0.982617	2.000000
50%	-0.611913	-0.610543	-0.606757	-0.601161	-0.593762	-0.585272	-0.577268	-0.569677	-0.561556	-0.555934	...	-0.976176	-0.976744	-0.977298	-0.977871	-0.978881	-0.979175	-0.979325	-0.979627	-0.980093	3.000000
75%	-0.609884	-0.608681	-0.604591	-0.599247	-0.591120	-0.582454	-0.574215	-0.565808	-0.558239	-0.551438	...	-0.973330	-0.973860	-0.974376	-0.974811	-0.975666	-0.976206	-0.976166	-0.976381	-0.976528	4.000000
max	-0.590921	-0.589520	-0.586062	-0.582049	-0.575003	-0.566571	-0.559088	-0.551611	-0.544998	-0.539241	...	-0.968620	-0.968871	-0.969230	-0.970103	-0.971065	-0.971451	-0.971528	-0.971549	-0.971597	4.000000

8 rows × 571 columns

After cheking the values, we found that Total number of missing values: 0 and number of duplicated values is also 0.
We visualize the first row of our train dataset:



In addition, we calculate the correlation matrix of the dataFrame train_df, and prints the correlation matrix that highlight the correlation of columns in the dataFrame.
Correlation between attr5 and attr6 is very high “0.997875 “ which means they have a big positive correlation between them.



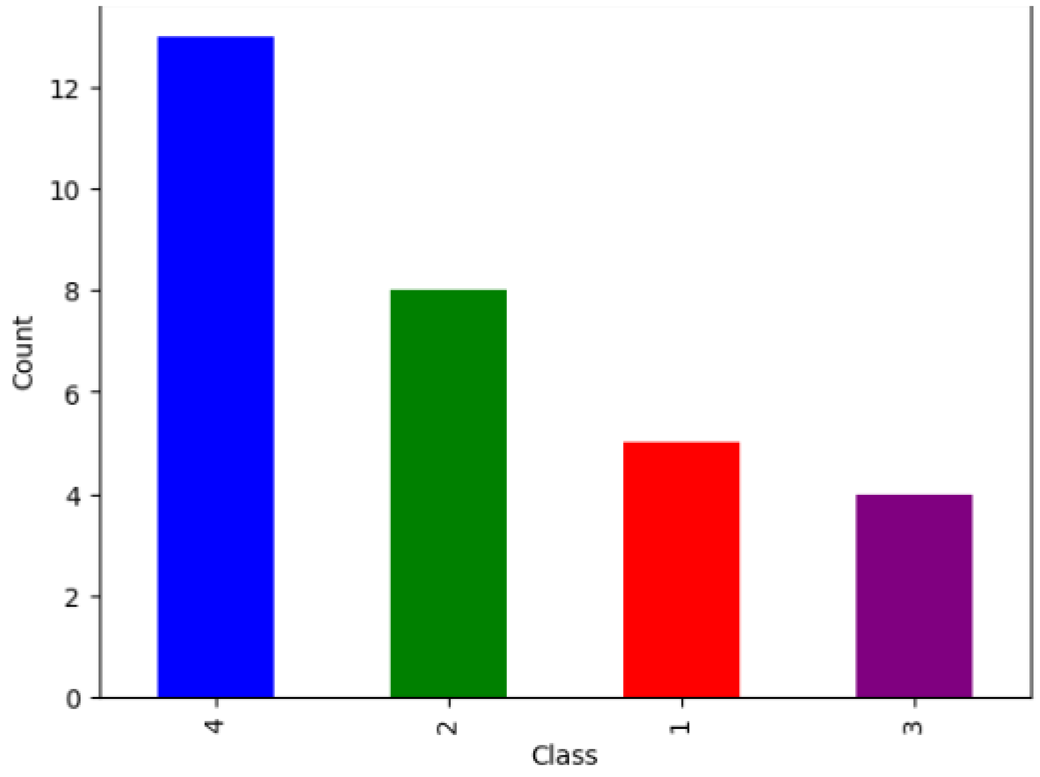
```
Correlation matrix:
            att1      att2      att3      att4      att5      att6      att7  \
att1    1.000000  0.996338  0.990802  0.985746  0.978122  0.971705  0.964493
att2    0.996338  1.000000  0.997253  0.991885  0.986462  0.981890  0.975636
att3    0.990802  0.997253  1.000000  0.996930  0.990935  0.986766  0.982215
att4    0.985746  0.991885  0.996930  1.000000  0.996414  0.992049  0.988589
att5    0.978122  0.986462  0.990935  0.996414  1.000000  0.997875  0.993079
...          ...       ...       ...       ...       ...       ...       ...
att567 -0.761235 -0.738252 -0.722468 -0.688534 -0.647280 -0.620862 -0.605704
att568 -0.762688 -0.740241 -0.724704 -0.690782 -0.649087 -0.622376 -0.608411
att569 -0.766649 -0.744680 -0.729136 -0.695939 -0.654480 -0.628199 -0.615804
att570 -0.772391 -0.751046 -0.735364 -0.702811 -0.662400 -0.636573 -0.625496
target  0.168291  0.196460  0.222004  0.256711  0.290777  0.302396  0.305701

            att8      att9     att10  ...    att562    att563    att564  \
att1    0.949429  0.932252  0.938711  ... -0.778848 -0.774760 -0.778559
att2    0.964381  0.950498  0.954872  ... -0.756041 -0.752522 -0.757367
att3    0.972909  0.960798  0.964543  ... -0.740960 -0.736562 -0.741863
att4    0.981362  0.971633  0.975149  ... -0.709272 -0.704016 -0.709230
att5    0.987649  0.980217  0.983407  ... -0.668423 -0.664174 -0.669670
...          ...       ...       ...  ...       ...       ...       ...
att567 -0.573689 -0.532828 -0.535706  ...  0.994635  0.995555  0.995222
att568 -0.576290 -0.534961 -0.537728  ...  0.997185  0.997532  0.997141
att569 -0.583918 -0.541143 -0.543016  ...  0.997824  0.997811  0.996835
att570 -0.594609 -0.551131 -0.552166  ...  0.997351  0.997198  0.996799
target  0.334015  0.373192  0.382309  ...  0.310172  0.303468  0.300294

          att565    att566    att567    att568    att569    att570    target
att1   -0.778953 -0.768821 -0.761235 -0.762688 -0.766649 -0.772391  0.168291
att2   -0.757871 -0.746834 -0.738252 -0.740241 -0.744680 -0.751046  0.196460
att3   -0.743135 -0.731732 -0.722468 -0.724704 -0.729136 -0.735364  0.222004
att4   -0.710442 -0.698246 -0.688534 -0.690782 -0.695939 -0.702811  0.256711
att5   -0.669982 -0.657267 -0.647280 -0.649087 -0.654480 -0.662400  0.290777
...          ...       ...       ...       ...       ...       ...       ...
att567  0.995283  0.998260  1.000000  0.998252  0.996172  0.994578  0.320373
att568  0.996936  0.996685  0.998252  1.000000  0.998672  0.996663  0.323184
att569  0.996771  0.996129  0.996172  0.998672  1.000000  0.998566  0.327650
att570  0.996396  0.994888  0.994578  0.996663  0.998566  1.000000  0.324209
target  0.305706  0.313305  0.320373  0.323184  0.327650  0.324209  1.000000

[571 rows x 571 columns]
```

Finally, we generate a bar plot of the counts of each class of the target in our train_df.



Data Preparation

To start with, we defined the train-test split.The training set `X_train` contains all the features except the target variable, while the target variable is stored in `y_train`. Similarly, the testing set `X_test` contains all the features except the target variable, while the target variable is stored in `y_test`.

We also created an empty DataFrame to store the evaluation metrics of different models. The `metrics_df` DataFrame is then created with columns for the model name, precision, accuracy, recall, and F1 score. This DataFrame will be used to store the evaluation metrics of different models. The `save_metrics` function is a helper function that takes in the model name, precision, accuracy, recall, F1 score, and the DataFrame to which the metrics will be added. This will allow us add the evaluation metrics of different models to the `metrics_df` DataFrame. Now, we are ready to start working on our models.

Modeling

We will work on different models: We train our models with the training data by calling the fit method of the model object with X_train and y_train as arguments. We then make predictions on the test data by calling the predict method of the fitted model object with X_test as an argument. The predicted values are stored in y_pred. Finally, we calculate evaluation metrics to compare our models' performance.

Metrics definition

To evaluate the performance of our models, we will compare some metrics :

- Accuracy: It is defined as the number of correct predictions made by the model divided by the total number of predictions made. It is a measure of the overall correctness of the model's predictions.
- Precision: It is defined as the proportion of predicted positive cases that are actually positive. It is a measure of the model's exactness or how many of the positive predictions are true.
- Recall: It is defined as the proportion of actual positive cases that are correctly identified by the model. It is a measure of the model's completeness or how many of the positive cases are identified.
- F1-score: It is a harmonic mean of precision and recall and is a balanced measure that combines both precision and recall. It is defined as F1-score = 2 x (precision x recall) / (precision + recall). It provides a single score that balances both precision and recall.

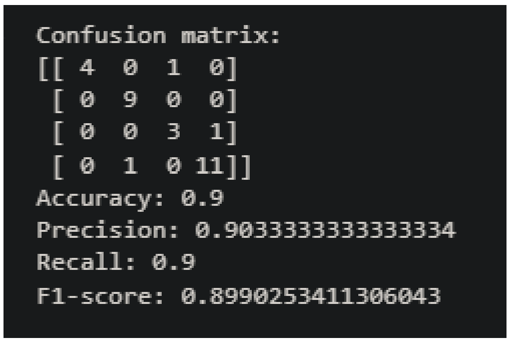
Evaluation

We calculate the evaluation metrics (accuracy, precision, recall, and F1 score) using the metrics module from the sklearn package. The accuracy_score, precision_score, recall_score, and f1_score functions are used to calculate these metrics. The average argument in the precision, recall, and F1 score functions is set to 'weighted', which means that the function computes the weighted average of the precision, recall, and F1 score for each class. This is useful when there are imbalanced classes in the data.
We also print the confusion matrix and evaluation metrics to calculate the confusion matrix of the model's predictions. This will allow us to have a look on the number of true positives, true negatives, false positives, and false negatives. We would also save the evaluation metrics of the Gaussian Naive Bayes model to the metrics_df DataFrame.

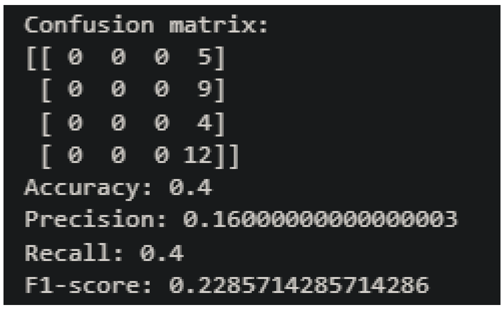
Naive Bayes

`GaussianNB`

We start by creating the GaussianNB object. We then fit our model to the training data by calling the fit method of the model object with X_train and y_train as arguments. After finishing training, we start generating predictions on the test data by calling the predict method of the fitted model object with X_test as an argument. The predicted values are stored in y_pred.

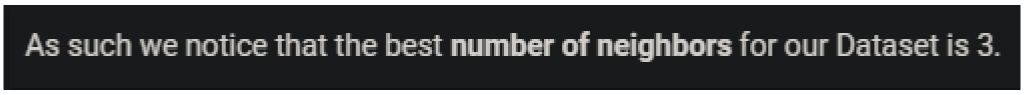

```
Confusion matrix:
[[ 4  0  1  0]
 [ 0  9  0  0]
 [ 0  0  3  1]
 [ 0  1  0 11]]
Accuracy: 0.9
Precision: 0.9033333333333334
Recall: 0.9
F1-score: 0.8990253411306043
```
`MultinomialNB`

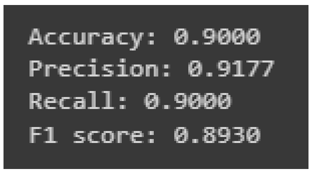
We train a Multinomial Naive Bayes model on the training data and evaluate its performance on the test data using the metrics defined above. The precision score is now included in the list of metrics. The confusion matrix is printed to show the distribution of actual vs predicted values, and the metrics are printed to evaluate the model's performance. Finally, the metrics are added to a global dataframe for later comparison with other models.


```
Confusion matrix:
[[ 0  0  0  5]
 [ 0  0  0  9]
 [ 0  0  0  4]
 [ 0  0  0 12]]
Accuracy: 0.4
Precision: 0.16000000000000003
Recall: 0.4
F1-score: 0.2285714285714286
```

K-Nearest Neighbors (KNN)

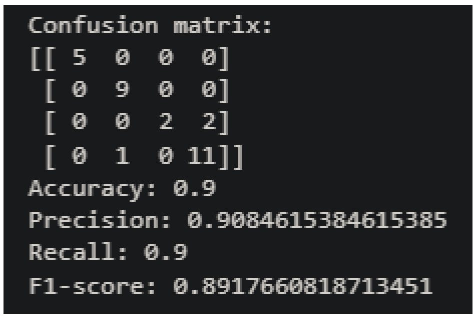
We try thz K-Nearest Neighbors model with different values of k and evaluates the performance of each model on the test set. we return a dictionary containing several metrics such as accuracy, precision, recall, and F1-score. We thencall the save_metrics function to add the metrics of the model with k=3 to the metrics_df dataframe for later comparison.


As such we notice that the best **number of neighbors** for our Dataset is 3.


```
Accuracy: 0.9000
Precision: 0.9177
Recall: 0.9000
F1 score: 0.8930
```

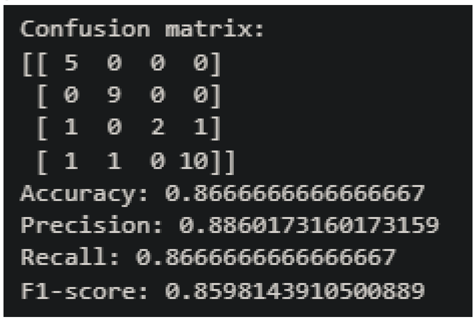
Linear Discriminant Analysis (LDA)

We now try training a Linear Discriminant Analysis (LDA) model: The `LinearDiscriminantAnalysis` function from the `sklearn.discriminant_analysis` module is used to create the LDA model. We the try training the model on the training set, predicting the test set and evaluating its performance on a test set using the `accuracy_score`, `precision_score`, `recall_score`, and `f1_score` functions from the `sklearn.metrics` module, and the confusion matrix using the `confusion_matrix` function.


```
Confusion matrix:
[[ 5  0  0  0]
 [ 0  9  0  0]
 [ 0  0  2  2]
 [ 0  1  0 11]]
Accuracy: 0.9
Precision: 0.9084615384615385
Recall: 0.9
F1-score: 0.8917660818713451
```

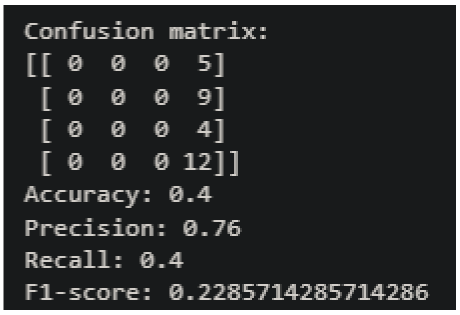
Decision Tree

We build a decision tree classifier model using the `DecisionTreeClassifier` class from scikit-learn library. The model is trained using the training set. Then, the model makes predictions on the test set, and the accuracy, precision, recall, F1-score, and confusion matrix of the predicted results are calculated and printe. Finally, we save the metrics in a dataframe.


```
Confusion matrix:
[[ 5  0  0  0]
 [ 0  9  0  0]
 [ 1  0  2  1]
 [ 1  1  0 10]]
Accuracy: 0.8666666666666667
Precision: 0.8860173160173159
Recall: 0.8666666666666667
F1-score: 0.8598143910500889
```

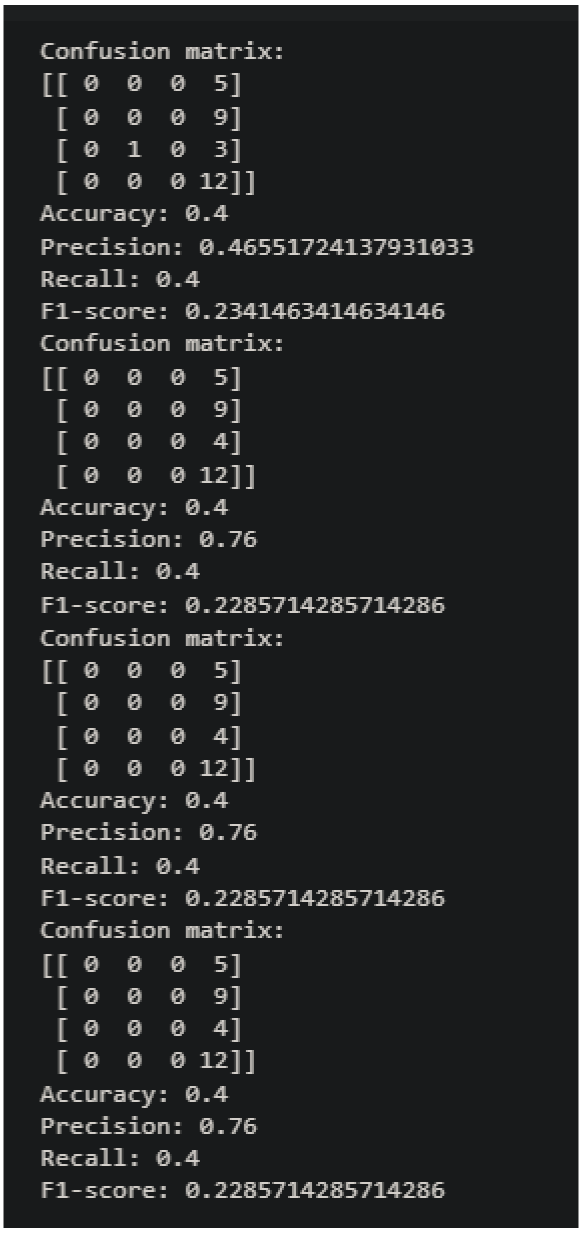
Artificial Neural Network (ANN)

We train an artificial neural network (ANN) model using the Multi-Layer Perceptron (MLP) algorithm. The MLPClassifier is initialized with the parameter "hidden_layer_sizes" set to (5,5). This neural network will have two hidden layers with 5 nodes each. The model is trained using the training data (X_train and y_train) and then used to make predictions on the test data (X_test). After finishing predictions, the code calculates various evaluation metrics, including accuracy, precision, recall, and F1-score. It also calculates the confusion matrix, which shows the number of true positive, false positive, true negative, and false negative predictions made by the model. As usual, the save_metrics() function is called to save the metrics of the ANN model in the metrics_df dataframe.


```
Confusion matrix:
[[ 0  0  0  5]
 [ 0  0  0  9]
 [ 0  0  0  4]
 [ 0  0  0 12]]
Accuracy: 0.4
Precision: 0.76
Recall: 0.4
F1-score: 0.2285714285714286
```

SVM

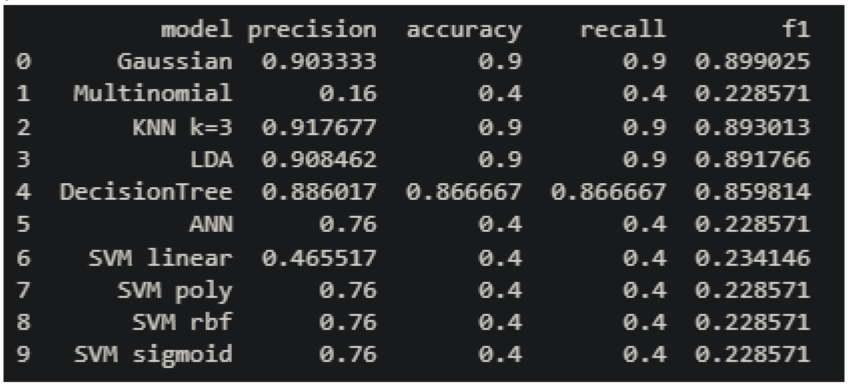
We define an SVM model that takes in a kernel and a metrics dataframe as inputs. Inside the function, a SVM model with the specified kernel is initialized and trained on the training data. The model is then used to make predictions on the test set, and various classification metrics (accuracy, precision, recall, f1-score, and confusion matrix) are calculated using scikit-learn's metrics module. The metrics are printed to the console, and the function returns an updated metrics dataframe with the calculated metrics added.



```
Confusion matrix:
[[ 0  0  0  5]
 [ 0  0  0  9]
 [ 0  1  0  3]
 [ 0  0  0 12]]
Accuracy: 0.4
Precision: 0.46551724137931033
Recall: 0.4
F1-score: 0.2341463414634146
Confusion matrix:
[[ 0  0  0  5]
 [ 0  0  0  9]
 [ 0  0  0  4]
 [ 0  0  0 12]]
Accuracy: 0.4
Precision: 0.76
Recall: 0.4
F1-score: 0.2285714285714286
Confusion matrix:
[[ 0  0  0  5]
 [ 0  0  0  9]
 [ 0  0  0  4]
 [ 0  0  0 12]]
Accuracy: 0.4
Precision: 0.76
Recall: 0.4
F1-score: 0.2285714285714286
Confusion matrix:
[[ 0  0  0  5]
 [ 0  0  0  9]
 [ 0  0  0  4]
 [ 0  0  0 12]]
Accuracy: 0.4
Precision: 0.76
Recall: 0.4
F1-score: 0.2285714285714286
```

Conclusion

To analyze the metrics for each trained model on this data, we printthe `metrics_df` dataframe. This will print the performance metrics of the models that were trained and evaluated on a dataset in a readable format. This will help us analyze the performance of each model and compare them to make a decision on which model to use for the given problem.



```
          model  precision  accuracy    recall        f1
0      Gaussian   0.903333       0.9       0.9  0.899025
1   Multinomial       0.16       0.4       0.4  0.228571
2       KNN k=3   0.917677       0.9       0.9  0.893013
3           LDA   0.908462       0.9       0.9  0.891766
4  DecisionTree   0.886017  0.866667  0.866667  0.859814
5           ANN       0.76       0.4       0.4  0.228571
6    SVM linear   0.465517       0.4       0.4  0.234146
7      SVM poly       0.76       0.4       0.4  0.228571
8       SVM rbf       0.76       0.4       0.4  0.228571
9   SVM sigmoid       0.76       0.4       0.4  0.228571
```

After conducting a thorough evaluation of various machine learning models, it has been observed that the K-Nearest Neighbors (KNN) algorithm has achieved the highest precision, accuracy, and recall values compared to other models.