Faculty of Electrical and Computer Engineering

Artificial Intelligence - ENCS434

Project 2 Report

Machine Learning for Classification

Group:

Farah Abu Lebdeh-1171456

Marah Anabatwi-1171326

Instructor: Dr. Adnan Yahya

Second Semester 2021-2022

Due Date: 10-6-2021

I. Abstract:

The aim of this project is to build a machine learning model to generation and testing the models for test data and reporting on the results. We used python libraries and Google Collaborator in order to achieve our goal.

II. Theory:

Many libraries and API's were used during the implementation of this project, so here we'll have a brief theory and description of what has been used and for what purpose.

- ✓ **Pandas:**

It's a software library written for the Python programming language for data manipulation and analysis. In particular, it offers data structures and operations for manipulating numerical tables and time series.

1. **read_csv():** function from pandas to deal with common problems when importing data.
2. **dataframe.info():** function is used to get a concise summary of the data frame.

Syntax: DataFrame.info(verbose=None, buf=None, max_cols=None, memory_usage=None, null_counts=None)

3. **dropna():**function allows the user to analyze and drop Rows/Columns with Null values in different ways.

Syntax: DataFrameName.dropna(axis=0, how='any', thresh=None, subset=None, inplace=False)

4. **drop_duplicates():**method helps in removing duplicates from the data frame.

Syntax: DataFrame.drop_duplicates(subset=None, keep='first', inplace=False).

5. **Groupby():**function allows the user to do operation involves some combination of splitting the object.

DataFrame.groupby(by=None, axis=0, level=None, as_index=True, sort=True, group_keys=True, squeeze=<object object>, observed=False, dropna=True)[source].

6. **dataframe.corr()** is used to find the pairwise correlation of all columns in the dataframe.

Syntax: DataFrame.corr(self, method='pearson', min_periods=1).

✓ Seaborn:

is a Python data visualization library based on **matplotlib**. It provides a high-level interface for drawing attractive and informative statistical graphics.

Heatmap(): Plot rectangular data as a color-encoded matrix.

Syntax: seaborn.heatmap(data, *, vmin=None, vmax=None, cmap=None, center=None, robust=False, annot=None, fmt='.2g', annot_kws=None, linewidths=0, linecolor='white', cbar=True, cbar_kws=None, cbar_ax=None, square=False, xticklabels='auto', yticklabels='auto', mask=None, ax=None, **kwargs).

✓ plotly.express

module (usually imported as px) contains functions that can create entire figures at once, and is referred to as Plotly Express or PX.

px.pie(): chart is a circular analytical chart, which is divided into region to symbolize numerical percentage

Syntax: plotly.express.pie(data_frame=None, names=None, values=None, color=None, color_discrete_sequence=None, color_discrete_map={}, hover_name=None, hover_data=None, custom_data=None, labels={}, title=None, template=None, width=None, height=None, opacity=None, hole=None)

✓ matplotlib.pyplot

is a state-based interface to matplotlib. It provides a MATLAB-like way of plotting.

✓ Removing stop words with NLTK in Python:

The process of converting data to something a computer can understand is referred to as pre-processing. One of the major forms of pre-processing is to filter out useless data. In natural language processing, useless words (data), are referred to as stop words.

Stop Words: A stop word is a commonly used word (such as "إلى" , "في" "على" , "من") that a search engine has been programmed to ignore, both when indexing entries for searching and when retrieving them as the result of a search query.

✓ sklearn.preprocessing

package provides several common utility functions and transformer classes to change raw feature vectors into a representation that is more suitable for the downstream estimators.

✓ Oversampling and undersampling

In data analysis are techniques used to adjust the class distribution of a data set (i.e. the ratio between the different classes/categories represented). These terms are used both in statistical sampling, survey design methodology and in machine learning.

✓ SMOTE:

There are a number of methods available to oversample a dataset used in a typical classification problem (using a classification algorithm to classify a set of images, given a labelled training set of images). The most common technique is known as SMOTE: Synthetic Minority Over-sampling Technique. To illustrate how this technique works consider some training data which has s samples, and f features in the feature space of the data. Note that these features, for simplicity, are continuous. As an example, consider a dataset of birds for classification. The feature space for the minority class for which we want to oversample could be beak length, wingspan, and weight (all continuous). To then oversample, take a sample from the dataset, and consider its k nearest neighbors (in feature space). To create a synthetic data point, take the vector between one of those k neighbors, and the current data point. Multiply this vector by a random number x which lies between 0, and 1.

Add this to the current data point to create the new, synthetic data point. Many modifications and extensions have been made to the SMOTE method ever since its proposal.

✓ TF-IDF: (term frequency-inverse document frequency) vectorizer:

It's a statistical measure that evaluates how relevant a word is to a document in a collection of documents. This is done by multiplying two metrics: how many times a word appears in a document and the inverse document frequency of the word across a set of documents. TF-IDF for a word in a document is calculated by multiplying two different metrics:

- The term frequency of a word in a document.

- The inverse document frequency of the word across a set of documents.

Tf idf (t,d,D) = tf (t,d) . idf (t,D)

tf (t,d) = log (1 + freq(t,d))

idf (t,D) = log (N / count (d € D:t € d))

✓ Cosine similarity:

a measure of similarity between two non-zero vectors. Which is a real-valued function that measure the similarity between two non-zero vectors of an inner product space.

A.B = |A| |B| Cos (θ)

$$Cos(\theta) = \frac{A.B}{|A||B|}$$

✓ Split Data to training and testing sets:

This module introduced the idea of dividing your data set into two subsets:

- **training set**—a subset to train a model.
- **test set**—a subset to test the trained model.

Split arrays or matrices into random train and test subsets

Quick utility that wraps input validation and next(ShuffleSplit().split(X, y)) and application to input data into a single call for splitting (and optionally subsampling) data in a oneliner.

- sklearn.model_selection.train_test_split(*arrays, test_size=None, train_size=None, random_state=None, shuffle=True, stratify=None)

✓ Random Forest Classifier:

an ensemble learning method and predict by combining the outputs from individual trees. It is a meta estimator that fits a number of decision tree classifiers on various sub-samples of the dataset and uses averaging to improve the predictive accuracy and control over-fitting. The sub-sample size is controlled with the max_samples parameter
if bootstrap=True (default), otherwise the whole dataset is used to build each tree. Random forests build each tree independently.

✓ Decision Tree Classifier:

a predictive modeling approach used in statistics, data mining and machine learning. It uses a decision tree (as a predictive model) to go from observations about an item (represented in the branches) to conclusions about the item's target value (represented in the leaves). Tree models where the target variable can take a discrete set of values are called classification trees; in these tree structures, leaves represent class labels and branches represent conjunctions of features that lead to those class labels. Decision trees where the target variable can take continuous values (typically real numbers) are called regression trees. Decision trees are among the most popular machine learning algorithms given their intelligibility and simplicity.

✓ KNeighbors (Nearest Neighbors):

Nearest neighbor classification is a machine learning method that aims at labeling previously unseen query objects while distinguishing two or more destination classes. As any classifier, in general, it requires some training data with given labels and, thus, is an instance of supervised learning. In the simplest variant, the query object inherits the label from the closest sample object in the training set. Common variants extend the decision set from the single nearest neighbor within the training data to the set of k nearest neighbors for any k > 1. The decision rule combines the labels from these kdecision objects, either by simple majority voting or by any distance-based or frequency-based weighting scheme, to decide the predicted label for the query object. Mean-based nearest neighbor classifiers group the training data and work on the means of classes rather than on the individual.

✓ KMeans:

K-means is an unsupervised classification algorithm, also called clusterization, that groups objects into k groups based on their characteristics. The grouping is done minimizing the sum of the distances between each object and the group or cluster centroid. The distance usually used is the quadratic or euclidean distance.

Initialization: once the number of groups, k has been chosen, k centroids are established in the data space, for instance, choosing them randomly.

Assignment of objects to the centroids: each object of the data is assigned to its nearest centroid.

Centroids update: The position of the centroid of each group is updated taking as the new centroid the average position of the objects belonging to said group.

✓ MLP Classifier:

A multilayer perceptron (MLP) is a class of feedforward artificial neural network (ANN). The term MLP is used ambiguously, sometimes loosely to any feedforward ANN, sometimes strictly to refer to network composed of multiple layers of perceptrons (with threshold activation). Multilayer perceptrons are sometimes colloquially referred to as "vanilla" neural networks, especially when they have a single hidden layer.

An MLP consists of at least three layers of nodes: an input layer, a hidden layer and an output layer. Except for the input nodes, each node is a neuron that uses a nonlinear activation function. MLP utilizes a supervised learning technique called backpropagation for training. Its multiple layers and non-linear activation distinguish MLP from a linear perceptron. It can distinguish data that is not linearly separable.

✓ NaiveBayes Classifier:

It is a classification technique based on Bayes' Theorem with an assumption of independence among predictors. In simple terms, a Naive Bayes classifier assumes that the presence of a particular feature in a class is unrelated to the presence of any other feature. For example, a fruit may be considered to be an apple if it is red, round, and about 3 inches in diameter. Even if these features depend on each other or upon the existence of the other features, all of these properties independently contribute to the probability that this fruit is an apple and that is why it is known as 'Naive'.

Naive Bayes model is easy to build and particularly useful for very large data sets. Along with simplicity, Naive Bayes is known to outperform even highly sophisticated classification methods.

Bayes theorem provides a way of calculating posterior probability P(c|x) from P(c), P(x) and P(x|c). Look at the equation below:

Probability of B occurring given evidence A has already occurred

Probability of A occurring

$$P(\mathrm{A}|\mathrm{B}) = \frac{P(\mathrm{B}|A) \cdot P(A)}{P(B)}$$

Probability of A occurring given evidence B has already occurred

Probability of B occurring

III. Formalization and Solution Procedure:

- **Preparing Data:**

As a start, we read the data from the CSV file using the Pandas library, to get our data frame random. In figure 1.1 we have the head of the data, which is 10 rows of the data frame. In figure 1.2 we show 12 authors in range 25-36 depending on part 3.

```
columns = set(newData['B'])
columns
```

```
{'GwadyM',
 'HHShkMohd',
 'L_Alobaidli',
 'LarissaAounSky',
 'MohamedBinZayed',
 'NawalElSaadawi1',
 'NidalSabeh',
 'Pontifex_ar',
 'QueenRania',
 'hussinalezzi5',
 'mohamedahou20',
 'monther72'}
```

Figure 1.1

```
newData = newData.sample(frac=1).reset_index(drop=True)
newData.head(10)
```

	A	B
0	ندم التقى شخص معوز حق تعرف وجه الله	Pontifex_ar
1	متي احتفال جمعيه جائزه تميز تربوي	QueenRania
2	حنين قصه و غصه و اغنيه و امان لغه طويل متعب	L_Alobaidli
3	...ئيس حكومه طلب تشكيل لجنه طارئه صحي متابعه قضيه	LarissaAounSky
4	...يس اقر نقطه عشر 1974 محمد ابن سلمان نظر فكره ت	NidalSabeh
5	مطر برد اعثر يد جيب معطف	L_Alobaidli
6	يها منبثق ذاكره شارد ان	L_Alobaidli
7	شتاق ل وش حيله الله يان فاقد	L_Alobaidli
8	...الله شهم اغتيل دم بارد اعلامي بارز سلاح قلم كم	monther72
9	...ولى راى اهل ايمان قتل صبر معتقل عسكر راى اهل س	GwadyM

Figure 1.2

Figure 2 shows info about data frame. We notice that we have 12564 rows for 12 Authors, and two columns A(Tweets) & B(Authors).

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 12564 entries, 0 to 12563
Data columns (total 2 columns):
 #   Column  Non-Null Count  Dtype
---  ------  --------------  -----
 0   A       12564 non-null  object
 1   B       12564 non-null  object
dtypes: object(2)
memory usage: 196.4+ KB
```

Figure 2

- **Text Processing**

✓ Remove Duplicates:

Next, we dropped the null and duplicated values by applying these functions on the data frame. See Figure 3

```
data.drop_duplicates(inplace = True)

data.info()
```

Remove Duplicates

```
[737] data.drop_duplicates(inplace = True)
      data.info()

      <class 'pandas.core.frame.DataFrame'>
      Int64Index: 12453 entries, 0 to 12563
      Data columns (total 2 columns):
       #   Column  Non-Null Count  Dtype
      ---  ------  --------------  -----
       0   A       12453 non-null  object
       1   B       12453 non-null  int64
      dtypes: int64(1), object(1)
      memory usage: 291.9+ KB
```

Figure 3

✓ Remove Stop Words & Nulls:

Then we remove the stop words and null from data as shown in Figure 4:

Remove StopWords

```
data['A'] = data['A'].apply(lambda c: " ".join(x for x in c.split() if x not in stop))
print(stop)
```

```
['يا', 'وهو', 'ومن', 'وما', 'ولو', 'ولكن', 'ولا', 'وإن', 'وإذا', 'وإذ', 'والذين', 'والذي', 
```

Figure 4

Also ,we calculated the following for each tweets:

✓ Number of Characters:

```
data['character_cnt'] = data['A'].str.len()
#data.groupby('B')['character_cnt'].mean()
```

✓ Number of Words:

```
data['word_counts'] = data['A'].str.split().str.len()
#data.groupby('A')['word_counts'].mean()
```

✓ Average Number of Characters per Word:

```
data['characters_per_word'] = data['character_cnt']/data['word_counts']
#data.groupby('B')['characters_per_word'].mean()
```

✓ Number of Numerics:

```
data['num'] = data['A'].apply(lambda x: len([x for x in x.split() if x.isdigit()]))
#data.groupby('B')['num'].mean()
```

Here we show 10 tweets and the calculation for it (NumOfCharacters, NumberOfWords, Average Number of Characters per Word, Number Of Numerics):

```
data.head(10)
```

	A	B	character_cnt	word_counts	characters_per_word	num
0	كلام مطران عربي عربي مسلم عقد صفقه عطاالله حنا ...	6	72	14	5.142857	0
1	رلي ايام شفت ولادي عائله عم اركض ليل نهار نقل ...	3	184	36	5.111111	1
2	عالم يوم واجه تحدي كبير ارهاب حرب هجر حل حقيقي...	1	95	18	5.277778	0
3	روك طالب محمد جلود جزائر مركز عربي مبروك مدرسه...	1	92	18	5.111111	1
4	ان شهيد قائد قصب سبق تنبيه خطوره مصافحه نقل كث...	9	160	34	4.705882	1
5	بؤس حصل دافع قضاء قدر مسبب انبغى اعترف ب عمل ا...	7	86	18	4.777778	0
6	دكتور عميد طب وزير سياحه واحد المع استاذ طب با...	0	110	21	5.238095	0
7	م حب استهلك شوق مايفوق ذره هواء مرحله	2	37	8	4.625000	0
8	ذكر يوم طوباوي شارل دو فوكو قال ايمان رؤيه يسو...	7	57	12	4.750000	0
9	دي قدره كبير تحويل شظيه داخل حقل ورد	2	36	8	4.500000	0

Figure 5

- ## Using Cosine Similarity:

We implemented an approach in building the model, by using **Cosine Similarity** and **TFIDF** to get features from the text in review content. Then we used the **heatmap** from **seaborn** library to draw the correlation of the features.

1. Using Cosine Similarity:

In this approach we used **Cosine Similarity** and **TFIDF** in order to extract features from the text in review content. But we couldn't apply Cosine Similarity on the whole dataset, because it needed extra memory, so we had to take a slice from the data.

Then we used TFIDF with max features of 500, in order to get the bag of words. We converted the output to a data frame as shown in figure 6.

	ابد	ابن	ابوظبي	اتحاد	اتفاق	اتى	اثر	اجاب	اجتماع	اجراء	احب	احترام	احتلال	احد	احمد	اخت	اخذ	اخر	اخوان	اخير
0	0.0	0.000000	0.0	0.0	0.0	0.28737	0.0	0.0	0.0	0.0	0.0	0.0	0.000000	0.0	0.0	0.0	0.0	0.0	0.0	0.0
1	0.0	0.000000	0.0	0.0	0.0	0.00000	0.0	0.0	0.0	0.0	0.0	0.0	0.000000	0.0	0.0	0.0	0.0	0.0	0.0	0.0
2	0.0	0.000000	0.0	0.0	0.0	0.00000	0.0	0.0	0.0	0.0	0.0	0.0	0.000000	0.0	0.0	0.0	0.0	0.0	0.0	0.0
3	0.0	0.000000	0.0	0.0	0.0	0.00000	0.0	0.0	0.0	0.0	0.0	0.0	0.000000	0.0	0.0	0.0	0.0	0.0	0.0	0.0
4	0.0	0.000000	0.0	0.0	0.0	0.00000	0.0	0.0	0.0	0.0	0.0	0.0	0.377791	0.0	0.0	0.0	0.0	0.0	0.0	0.0
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
12448	0.0	0.242876	0.0	0.0	0.0	0.00000	0.0	0.0	0.0	0.0	0.0	0.0	0.000000	0.0	0.0	0.0	0.0	0.0	0.0	0.0
12449	0.0	0.000000	0.0	0.0	0.0	0.00000	0.0	0.0	0.0	0.0	0.0	0.0	0.000000	0.0	0.0	0.0	0.0	0.0	0.0	0.0
12450	0.0	0.000000	0.0	0.0	0.0	0.00000	0.0	0.0	0.0	0.0	0.0	0.0	0.000000	0.0	0.0	0.0	0.0	0.0	0.0	0.0
12451	0.0	0.271356	0.0	0.0	0.0	0.00000	0.0	0.0	0.0	0.0	0.0	0.0	0.000000	0.0	0.0	0.0	0.0	0.0	0.0	0.0
12452	0.0	0.000000	0.0	0.0	0.0	0.32609	0.0	0.0	0.0	0.0	0.0	0.0	0.000000	0.0	0.0	0.0	0.0	0.0	0.0	0.0

12453 rows × 500 columns

Figure 6

Next, we took a slice from the data (e.g., `12453` samples) in order to apply Cosine Similarity on the features in figure 6. In figure 7 we can see the output matrix. After that we took the mean value for each row in the matrix and considered it as a feature and added it to the data frame as a column **'similarity'**.

```
array([[1.        , 0.09449112, 0.        , ..., 0.        , 0.07142857,
        0.05976143],
       [0.09449112, 1.        , 0.        , ..., 0.        , 0.09449112,
        0.        ],
       [0.        , 0.        , 1.        , ..., 0.        , 0.        ,
        0.        ],
       ...,
       [0.        , 0.        , 0.        , ..., 1.        , 0.23145502,
        0.        ],
       [0.07142857, 0.09449112, 0.        , ..., 0.23145502, 1.        ,
        0.05976143],
       [0.05976143, 0.        , 0.        , ..., 0.        , 0.05976143,
        1.        ]])
```

Figure 7

Figure 8 shows the heat map of the features, with similarity features added.

<matplotlib.axes._subplots.AxesSubplot at 0x7f633920bb50>

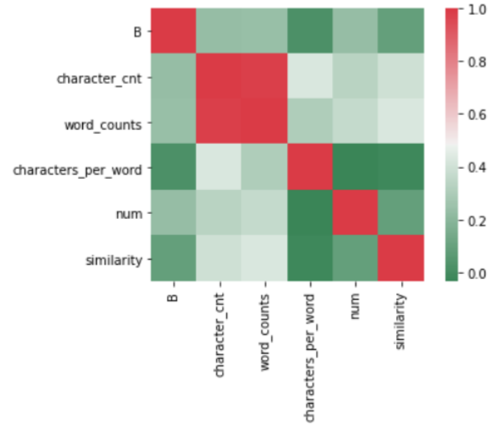


Figure 8

As the features are ready for the training, the data is unbalanced and needs to be resampled. We used the **Standard Scaler** from **sklearn** library, and **SMOTE** from **imblearn** library in order to resample the data. Then we apply splitting on the dataset to **80% training set** and **20% testing set**.

```
[346] from sklearn.model_selection import  train_test_split
      y = dat['B']
      x = dat.drop({'B'}, axis=1)
      x =x.drop({'A'}, axis=1)
      x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.20, random_state=27)
```

Figure 9

- ## Our Algorithms & Results:

What is Confusion Matrix and why you need it?

Well, it is a performance measurement for machine learning classification problem where output can be two or more classes. It is a table with 4 different combinations of predicted and actual values.It is extremely useful for measuring Recall, Precision and Accuracy.

Let’s understand TP, FP, FN, TN in terms of pregnancy analogy.

True Positive (TP)

- The predicted value matches the actual value.
- The actual value was positive and the model predicted a positive value.

True Negative (TN)

- The predicted value matches the actual value.
- The actual value was negative and the model predicted a negative value.

False Positive (FP) – Type 1 error

- The predicted value was falsely predicted.
- The actual value was negative but the model predicted a positive value.

False Negative (FN) – Type 2 error

- The predicted value was falsely predicted.
- The actual value was positive but the model predicted a negative value.

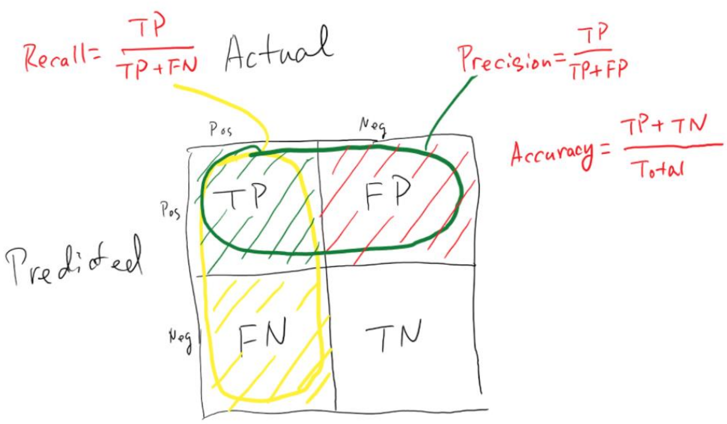


Figure 10

★ **Recall: Out of all the positive classes, how much we predicted correctly. It should be high as possible.**

★ **Precision: Out of all the positive classes we have predicted correctly, how many are actually positive.**

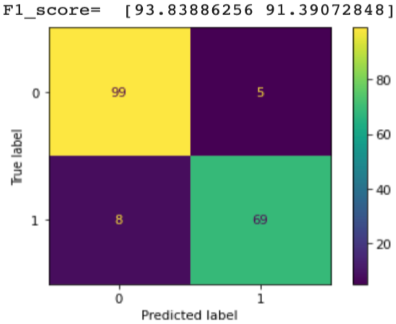
★ **Accuracy: Out of all the classes, how much we predicted correctly.It should be as high as possible.**

We tried many classifiers trying to get the best scores, and we found that the following classifiers achieved our goal:

Random Forest Classifier:

Random Forest Classifier for 2 Authors:

```
Accuracy =  92.81767955801105
Precision =  [92.52336449 93.24324324]
Recall =  [95.19230769 89.61038961]
F1_score=  [93.83886256 91.39072848]
```

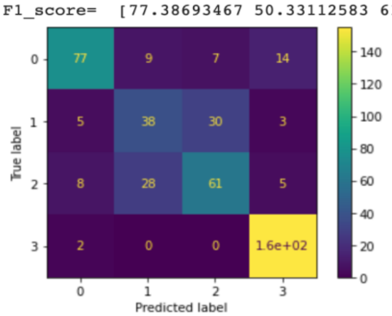


	precision	recall	f1-score	support
0	0.93	0.95	0.94	104
1	0.93	0.90	0.91	77
accuracy			0.93	181
macro avg	0.93	0.92	0.93	181
weighted avg	0.93	0.93	0.93	181

Figure 11.1

Random Forest Classifier for 4 Authors:

```
Accuracy =  74.8868778280543
Precision =  [83.69565217 50.66666667 62.24489796 87.57062147]
Recall =  [71.96261682 50.         59.80392157 98.72611465]
F1_score=  [77.38693467 50.33112583 61.         92.81437126]
```

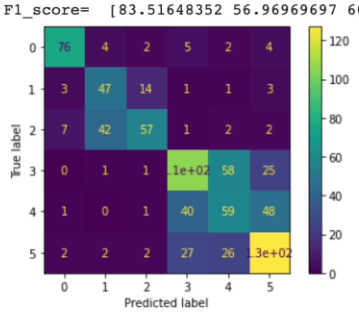


	precision	recall	f1-score	support
0	0.84	0.72	0.77	107
1	0.51	0.50	0.50	76
2	0.62	0.60	0.61	102
3	0.88	0.99	0.93	157
accuracy			0.75	442
macro avg	0.71	0.70	0.70	442
weighted avg	0.74	0.75	0.74	442

Figure 11.2

Random Forest Classifier for 6 Authors:

```
Accuracy =  59.073842302878596
Precision =  [85.39325843 48.95833333 74.02597403 58.88888889 39.86486486 60.76555024]
Recall =  [81.72043011 68.11594203 51.35135135 55.4973822  39.59731544 68.27956989]
F1_score=  [83.51648352 56.96969697 60.63829787 57.14285714 39.73063973 64.30379747]
```

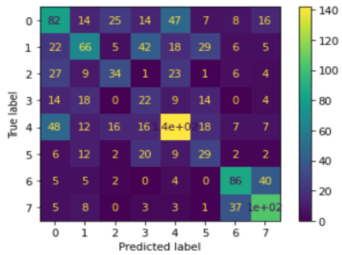


	precision	recall	f1-score	support
0	0.85	0.82	0.84	93
1	0.49	0.68	0.57	69
2	0.74	0.51	0.61	111
3	0.59	0.55	0.57	191
4	0.40	0.40	0.40	149
5	0.61	0.68	0.64	186
accuracy			0.59	799
macro avg	0.61	0.61	0.60	799
weighted avg	0.60	0.59	0.59	799

Figure 11.3

Random Forest Classifier for 8 Authors:

```
Accuracy =  45.45454545454545
Precision =  [39.23444976 45.83333333 40.47619048 18.6440678  55.68627451 29.29292929
 56.57894737 57.14285714]
Recall =  [38.49765258 34.19689119 32.38095238 27.16049383 53.38345865 35.36585366
 60.56338028 64.59627329]
F1_score=  [38.86255924 39.16913947 35.97883598 22.11055276 54.51055662 32.0441989
 58.50340136 60.64139942]
```

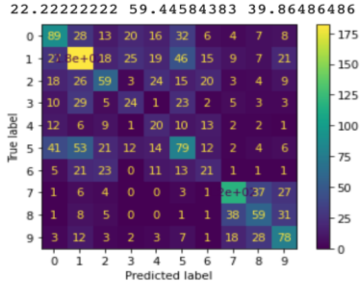



```
              precision    recall  f1-score   support

           0       0.39      0.38      0.39       213
           1       0.46      0.34      0.39       193
           2       0.40      0.32      0.36       105
           3       0.19      0.27      0.22        81
           4       0.56      0.53      0.55       266
           5       0.29      0.35      0.32        82
           6       0.57      0.61      0.59       142
           7       0.57      0.65      0.61       161

    accuracy                           0.45      1243
   macro avg       0.43      0.43      0.43      1243
weighted avg       0.46      0.45      0.46      1243
```

Figure 11.4

Random Forest Classifier for 10 Authors:

```
Accuracy =  40.7035175879397
Precision =  [42.99516908 49.05660377 36.875      27.5862069  18.51851852 34.49781659
 22.82608696 59.         38.81578947 42.16216216]
Recall =  [39.9103139  49.32249322 32.59668508 22.85714286 26.31578947 32.37704918
 21.64948454 59.89847716 40.97222222 50.32258065]
F1_score=  [41.39534884 49.18918919 34.60410557 25.         21.73913043 33.4038055
 22.22222222 59.44584383 39.86486486 45.88235294]
```

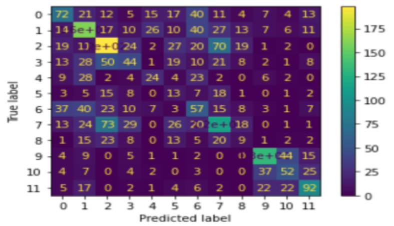



```
              precision    recall  f1-score   support

           0       0.43      0.40      0.41       223
           1       0.49      0.49      0.49       369
           2       0.37      0.33      0.35       181
           3       0.28      0.23      0.25       105
           4       0.19      0.26      0.22        76
           5       0.34      0.32      0.33       244
           6       0.23      0.22      0.22        97
           7       0.59      0.60      0.59       197
           8       0.39      0.41      0.40       144
           9       0.42      0.50      0.46       155

    accuracy                           0.41      1791
   macro avg       0.37      0.38      0.37      1791
weighted avg       0.41      0.41      0.41      1791
```

Figure 11.5

Random Forest Classifier for 12 Authors:

```
Accuracy =  39.02047370533922
Precision =  [37.11340206 43.98907104 47.94188862 28.75816993 30.37974684  9.48905109
 24.46351931 38.41059603 11.25       60.90909091 37.68115942 52.27272727]
Recall =  [32.57918552 47.07602339 50.38167939 21.46341463 23.07692308 17.80821918
 27.01421801 36.13707165  9.09090909 62.3255814  38.80597015 53.17919075]
F1_score=  [34.69879518 45.48022599 49.13151365 24.58100559 26.2295082  12.38095238
 25.67567568 37.23916533 10.05586592 61.6091954  38.23529412 52.72206304]
```



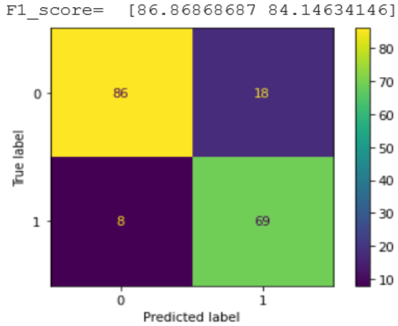
	precision	recall	f1-score	support
0	0.37	0.33	0.35	221
1	0.44	0.47	0.45	342
2	0.48	0.50	0.49	393
3	0.29	0.21	0.25	205
4	0.30	0.23	0.26	104
5	0.09	0.18	0.12	73
6	0.24	0.27	0.26	211
7	0.38	0.36	0.37	321
8	0.11	0.09	0.10	99
9	0.61	0.62	0.62	215
10	0.38	0.39	0.38	134
11	0.52	0.53	0.53	173
accuracy			0.39	2491
macro avg	0.35	0.35	0.35	2491
weighted avg	0.39	0.39	0.39	2491

Figure 11.6

Decision Tree Classifier:

Decision Tree Classifier for 2 Authors:

```
Accuracy =  85.6353591160221
Precision =  [91.4893617  79.31034483]
Recall =  [82.69230769 89.61038961]
F1_score=  [86.86868687 84.14634146]
```

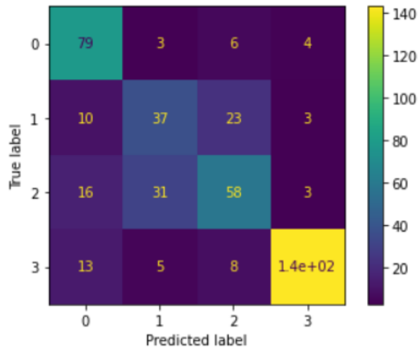


	precision	recall	f1-score	support
0	0.91	0.83	0.87	104
1	0.79	0.90	0.84	77
accuracy			0.86	181
macro avg	0.85	0.86	0.86	181
weighted avg	0.86	0.86	0.86	181

Figure 12.1

Decision Tree Classifier for 4 Authors:

```
Accuracy =  71.71945701357465
Precision =  [66.94915254 48.68421053 61.05263158 93.46405229]
Recall =  [85.86956522 50.68493151 53.7037037  84.61538462]
F1_score=  [75.23809524 49.66442953 57.14285714 88.81987578]
```

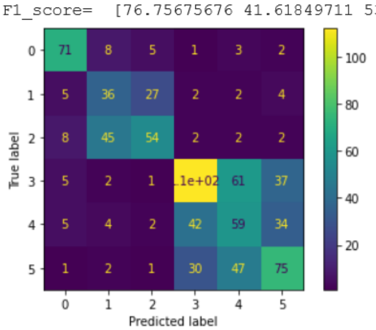


	precision	recall	f1-score	support
0	0.67	0.86	0.75	92
1	0.49	0.51	0.50	73
2	0.61	0.54	0.57	108
3	0.93	0.85	0.89	169
accuracy			0.72	442
macro avg	0.68	0.69	0.68	442
weighted avg	0.73	0.72	0.72	442

Figure 12.2

Decision Tree Classifier for 6 Authors :

```
Accuracy =  50.9386733416771
Precision =  [74.73684211 37.11340206 60.         59.25925926 33.90804598 48.7012987 ]
Recall =  [78.88888889 47.36842105 47.78761062 51.37614679 40.4109589  48.07692308]
F1_score=  [76.75675676 41.61849711 53.20197044 55.03685504 36.875      48.38709677]
```

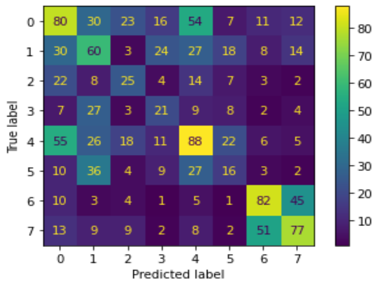


	precision	recall	f1-score	support
0	0.75	0.79	0.77	90
1	0.37	0.47	0.42	76
2	0.60	0.48	0.53	113
3	0.59	0.51	0.55	218
4	0.34	0.40	0.37	146
5	0.49	0.48	0.48	156
accuracy			0.51	799
macro avg	0.52	0.52	0.52	799
weighted avg	0.52	0.51	0.51	799

Figure 12.3

Decision Tree Classifier for 8 Authors :

```
Accuracy =  36.12228479485117
Precision =  [35.24229075 30.15075377 28.08988764 23.86363636 37.93103448 19.75308642
 49.39759036 47.82608696]
Recall =  [34.33476395 32.60869565 29.41176471 25.92592593 38.0952381  14.95327103
 54.30463576 45.02923977]
F1_score=  [34.7826087  31.33159269 28.73563218 24.85207101 38.01295896 17.0212766
 51.73501577 46.38554217]
```

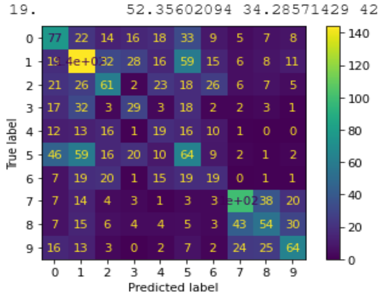



```
              precision    recall  f1-score   support

           0       0.35      0.34      0.35       233
           1       0.30      0.33      0.31       184
           2       0.28      0.29      0.29        85
           3       0.24      0.26      0.25        81
           4       0.38      0.38      0.38       231
           5       0.20      0.15      0.17       107
           6       0.49      0.54      0.52       151
           7       0.48      0.45      0.46       171

    accuracy                           0.36      1243
   macro avg       0.34      0.34      0.34      1243
weighted avg       0.36      0.36      0.36      1243
```

Figure 12.4

Decision Tree Classifier for 10 Authors :

```
Accuracy =  35.23171412618649
Precision =  [33.62445415 40.33613445 34.85714286 27.88461538 17.11711712 26.44628099
 19.3877551  52.91005291 37.5        45.07042254]
Recall =  [36.84210526 42.6035503  31.28205128 26.36363636 21.59090909 27.94759825
 18.62745098 51.8134715  31.57894737 41.02564103]
F1_score=  [35.15981735 41.43884892 32.97297297 27.10280374 19.09547739 27.17622081
 19.         52.35602094 34.28571429 42.95302013]
```

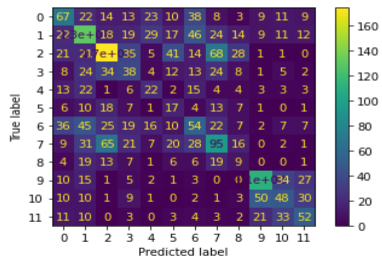



```
              precision    recall  f1-score   support

           0       0.34      0.37      0.35       209
           1       0.40      0.43      0.41       338
           2       0.35      0.31      0.33       195
           3       0.28      0.26      0.27       110
           4       0.17      0.22      0.19        88
           5       0.26      0.28      0.27       229
           6       0.19      0.19      0.19       102
           7       0.53      0.52      0.52       193
           8       0.38      0.32      0.34       171
           9       0.45      0.41      0.43       156

    accuracy                           0.35      1791
   macro avg       0.34      0.33      0.33      1791
weighted avg       0.36      0.35      0.35      1791
```

Figure 12.5

Decision Tree Classifier for 12 Authors :

```
Accuracy =  32.87836210357286
Precision =  [30.87557604 36.38888889 47.8021978  20.87912088 19.81981982 12.23021583
 23.78854626 33.80782918  8.91089109 53.58851675 30.96774194 35.86206897]
Recall =  [29.5154185  37.21590909 42.54278729 21.96531792 22.44897959 20.
 21.6        32.20338983 10.58823529 53.33333333 29.09090909 36.61971831]
F1_score=  [30.18018018 36.79775281 45.01940492 21.4084507  21.05263158 15.17857143
 22.64150943 32.98611111  9.67741935 53.46062053 30.         36.2369338 ]
```



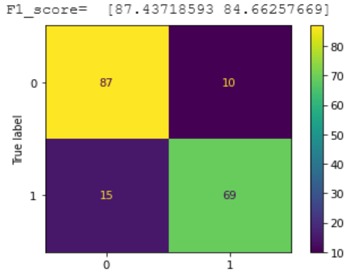

```
              precision    recall  f1-score   support

           0       0.31      0.30      0.30       227
           1       0.36      0.37      0.37       352
           2       0.48      0.43      0.45       409
           3       0.21      0.22      0.21       173
           4       0.20      0.22      0.21        98
           5       0.12      0.20      0.15        85
           6       0.24      0.22      0.23       250
           7       0.34      0.32      0.33       295
           8       0.09      0.11      0.10        85
           9       0.54      0.53      0.53       210
          10       0.31      0.29      0.30       165
          11       0.36      0.37      0.36       142

    accuracy                           0.33      2491
   macro avg       0.30      0.30      0.30      2491
weighted avg       0.34      0.33      0.33      2491
```

Figure 12.6

K-Neighbors Classifier:

K-Neighbors Classifier for 2 Authors with 1000 samples:

```
Accuracy =  86.1878453038674
Precision =  [85.29411765 87.34177215]
Recall =  [89.69072165 82.14285714]
F1_score=  [87.43718593 84.66257669]
```

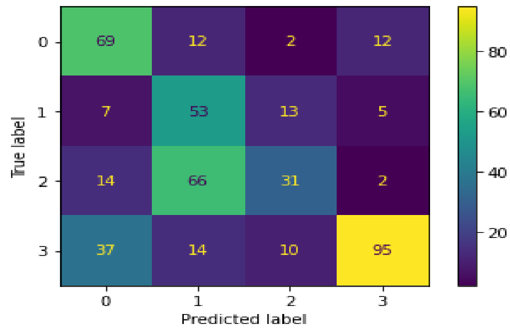



```
              precision    recall  f1-score   support

           0       0.85      0.90      0.87        97
           1       0.87      0.82      0.85        84

    accuracy                           0.86       181
   macro avg       0.86      0.86      0.86       181
weighted avg       0.86      0.86      0.86       181
```

Figure 13.1

K-Neighbors Classifier for 4 Authors with 500 samples

```
Accuracy =  56.10859728506787
Precision =  [54.33070866 36.55172414 55.35714286 83.33333333]
Recall =  [72.63157895 67.94871795 27.43362832 60.8974359 ]
F1_score=  [62.16216216 47.53363229 36.68639053 70.37037037]
```

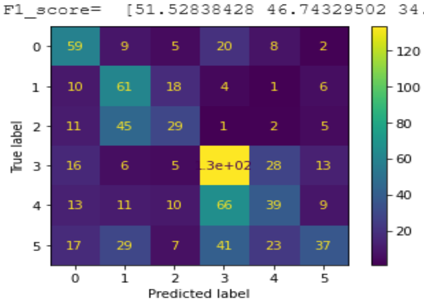



```
              precision    recall  f1-score   support

           0       0.54      0.73      0.62        95
           1       0.37      0.68      0.48        78
           2       0.55      0.27      0.37       113
           3       0.83      0.61      0.70       156

    accuracy                           0.56       442
   macro avg       0.57      0.57      0.54       442
weighted avg       0.62      0.56      0.56       442
```

Figure 13.2

K-Neighbors Classifier for 6 Authors with 500 samples :

```
Accuracy =  44.80600750938674
Precision =  [46.82539683 37.88819876 39.18918919 50.18867925 38.61386139 51.38888889]
Recall =  [57.2815534  61.         31.1827957  66.16915423 26.35135135 24.02597403]
F1_score=  [51.52838428 46.74329502 34.73053892 57.08154506 31.3253012  32.74336283]
```

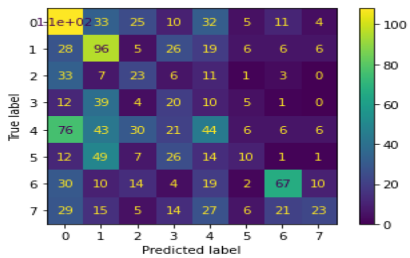



```
              precision    recall  f1-score   support

           0       0.47      0.57      0.52       103
           1       0.38      0.61      0.47       100
           2       0.39      0.31      0.35        93
           3       0.50      0.66      0.57       201
           4       0.39      0.26      0.31       148
           5       0.51      0.24      0.33       154

    accuracy                           0.45       799
   macro avg       0.44      0.44      0.42       799
weighted avg       0.45      0.45      0.43       799
```

Figure 13.3

K-Neighbors Classifier for 8 Authors with 500 samples:

```
Accuracy =  31.456154465004023
Precision =  [32.92682927 32.87671233 20.3539823  15.7480315  25.         24.3902439
 57.75862069 46.        ]
Recall =  [47.36842105 50.         27.38095238 21.97802198 18.96551724  8.33333333
 42.94871795 16.42857143]
F1_score=  [38.84892086 39.66942149 23.35025381 18.34862385 21.56862745 12.42236025
 49.26470588 24.21052632]
```

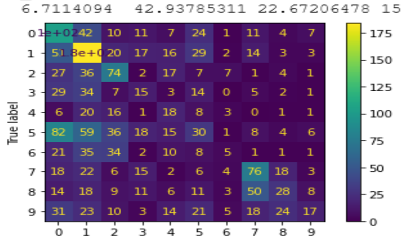


	precision	recall	f1-score	support
0	0.33	0.47	0.39	228
1	0.33	0.50	0.40	192
2	0.20	0.27	0.23	84
3	0.16	0.22	0.18	91
4	0.25	0.19	0.22	232
5	0.24	0.08	0.12	120
6	0.58	0.43	0.49	156
7	0.46	0.16	0.24	140
accuracy			0.31	1243
macro avg	0.32	0.29	0.28	1243
weighted avg	0.33	0.31	0.30	1243

Figure 13.4

K-Neighbors Classifier for 10 Authors :

```
Accuracy =  30.764935790061422
Precision =  [27.154047   38.90063425 33.33333333 15.78947368 16.66666667 18.98734177
 16.12903226 41.30434783 31.46067416 35.41666667]
Recall =  [47.05882353 54.27728614 42.04545455 13.63636364 24.32432432 11.58301158
  4.23728814 44.70588235 17.72151899 10.24096386]
F1_score=  [34.43708609 45.32019704 37.18592965 14.63414634 19.78021978 14.38848921
  6.7114094  42.93785311 22.67206478 15.88785047]
```

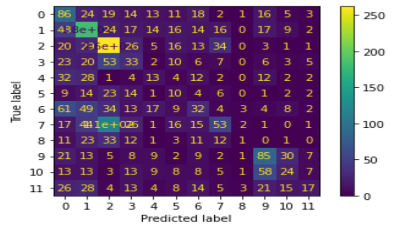


	precision	recall	f1-score	support
0	0.27	0.47	0.34	221
1	0.39	0.54	0.45	339
2	0.33	0.42	0.37	176
3	0.16	0.14	0.15	110
4	0.17	0.24	0.20	74
5	0.19	0.12	0.14	259
6	0.16	0.04	0.07	118
7	0.41	0.45	0.43	170
8	0.31	0.18	0.23	158
9	0.35	0.10	0.16	166
accuracy			0.31	1791
macro avg	0.28	0.27	0.25	1791
weighted avg	0.29	0.31	0.28	1791

Figure 13.5

K-Neighbors Classifier for 12 Authors:

```
Accuracy =  32.03532717784022
Precision =  [23.43324251 38.97216274 45.72425829 17.0984456  14.60674157  8.84955752
 20.51282051 35.81081081  8.33333333 37.94642857 24.          34.69387755]
Recall =  [40.56603774 50.69637883 63.90243902 19.64285714 11.60714286 11.62790698
 13.55932203 18.40277778  0.92592593 44.27083333 14.81481481 10.75949367]
F1_score=  [29.70639033 44.06779661 53.30620549 18.28254848 12.93532338 10.05025126
 16.32653061 24.31192661  1.66666667 40.86538462 18.32061069 16.42512077]
```



	precision	recall	f1-score	support
0	0.23	0.41	0.30	212
1	0.39	0.51	0.44	359
2	0.46	0.64	0.53	410
3	0.17	0.20	0.18	168
4	0.15	0.12	0.13	112
5	0.09	0.12	0.10	86
6	0.21	0.14	0.16	236
7	0.36	0.18	0.24	288
8	0.08	0.01	0.02	108
9	0.38	0.44	0.41	192
10	0.24	0.15	0.18	162
11	0.35	0.11	0.16	158
accuracy			0.32	2491
macro avg	0.26	0.25	0.24	2491
weighted avg	0.30	0.32	0.30	2491

Figure 13.5

K-Means Classifier:

K-Means Classifier for 2 Authors with max_features=100:

```
Accuracy =  83.42541436464089
Precision =  [89.41176471 78.125     ]
Recall =  [78.35051546 89.28571429]
F1_score=  [83.51648352 83.33333333]
```

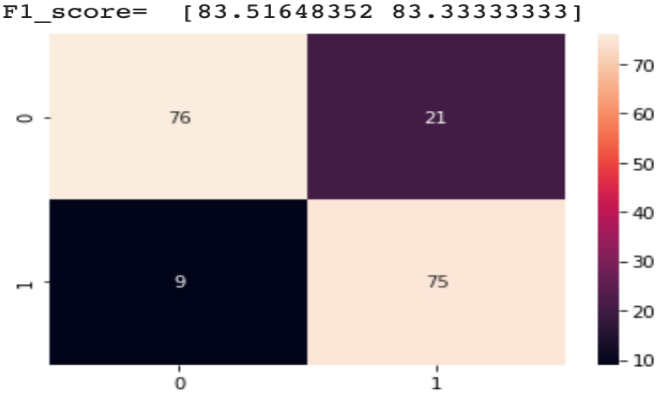


Figure 14.1

K-Means Classifier for 4 Authors:

```
Accuracy =  33.4841628959276
Precision =  [39.5         28.51239669  0.          0.        ]
Recall =  [74.52830189 86.25         0.          0.        ]
F1_score=  [51.63398693 42.85714286  0.          0.        ]
/usr/local/lib/python3.7/dist-packages/sklearn/metrics/_classifi
  _warn_prf(average, modifier, msg_start, len(result))
```

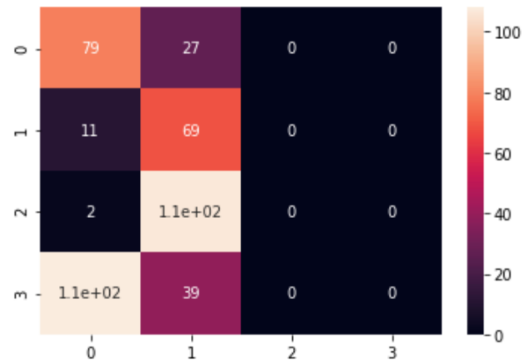


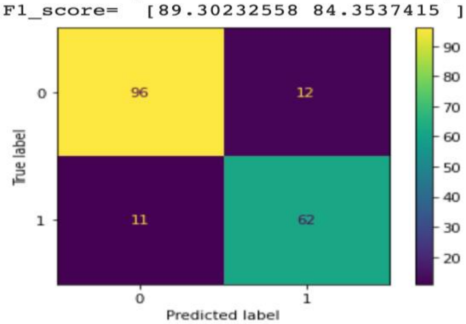
Figure 14.2

Notice: We have an error in this algorithm when tested on 4 authors, some value was zero although our code was run correctly for other algorithms.

MLP Classifier:

MLP Classifier for 2 Authors:

```
Accuracy =  87.29281767955801
Precision =  [89.71962617 83.78378378]
Recall =  [88.88888889 84.93150685]
F1_score=  [89.30232558 84.3537415 ]
```

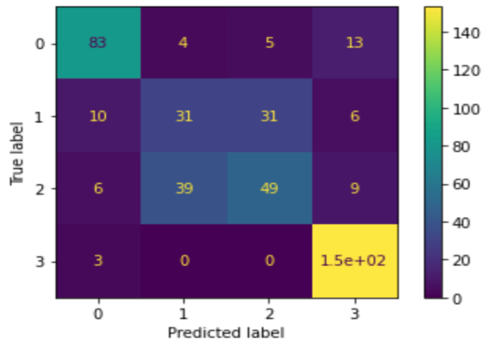



```
              precision    recall  f1-score   support

           0       0.90      0.89      0.89       108
           1       0.84      0.85      0.84        73

    accuracy                           0.87       181
   macro avg       0.87      0.87      0.87       181
weighted avg       0.87      0.87      0.87       181
```

Figure 14.1

MLP Classifier for 4 Authors:

```
Accuracy =  71.49321266968326
Precision =  [81.37254902 41.89189189 57.64705882 84.53038674]
Recall =  [79.04761905 39.74358974 47.57281553 98.07692308]
F1_score=  [80.19323671 40.78947368 52.12765957 90.80118694]
```

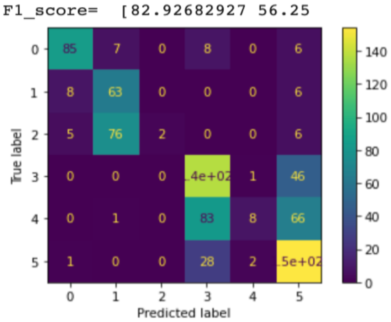



```
              precision    recall  f1-score   support

           0       0.81      0.79      0.80       105
           1       0.42      0.40      0.41        78
           2       0.58      0.48      0.52       103
           3       0.85      0.98      0.91       156

    accuracy                           0.71       442
   macro avg       0.66      0.66      0.66       442
weighted avg       0.70      0.71      0.70       442
```

Figure 14.2

MLP Classifier for 6 Authors:

```
Accuracy =  56.19524405506884
Precision =  [ 85.85858586  42.85714286 100.          53.515625    72.72727273
  54.22535211]
Recall =  [80.18867925 81.81818182  2.24719101 74.45652174  5.06329114 83.24324324]
F1_score=  [82.92682927 56.25        4.3956044  62.27272727  9.46745562 65.67164179]
```

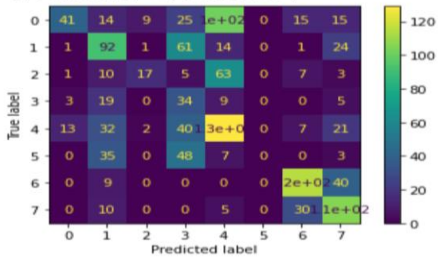



```
              precision    recall  f1-score   support

           0       0.86      0.80      0.83       106
           1       0.43      0.82      0.56        77
           2       1.00      0.02      0.04        89
           3       0.54      0.74      0.62       184
           4       0.73      0.05      0.09       158
           5       0.54      0.83      0.66       185

    accuracy                           0.56       799
   macro avg       0.68      0.55      0.47       799
weighted avg       0.66      0.56      0.48       799
```

Figure 14.3

MLP Classifier for 8 Authors:

```
Accuracy =  42.96057924376508
Precision =  [69.49152542 41.62895928 58.62068966 15.96244131 39.20972644  0.
 65.71428571 48.84792627]
Recall =  [18.5520362  47.42268041 16.03773585 48.57142857 52.86885246  0.
 70.12195122 70.1986755 ]
F1_score=  [29.28571429 44.3373494  25.18518519 24.02826855 45.02617801  0.
 67.84660767 57.60869565]
```

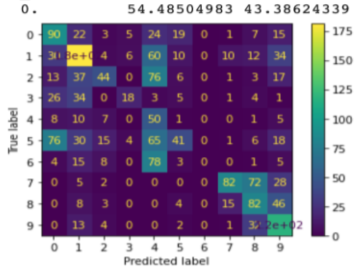


	precision	recall	f1-score	support
0	0.69	0.19	0.29	221
1	0.42	0.47	0.44	194
2	0.59	0.16	0.25	106
3	0.16	0.49	0.24	70
4	0.39	0.53	0.45	244
5	0.00	0.00	0.00	93
6	0.66	0.70	0.68	164
7	0.49	0.70	0.58	151
accuracy			0.43	1243
macro avg	0.42	0.40	0.37	1243
weighted avg	0.47	0.43	0.40	1243

Figure 14.4

MLP Classifier for 10 Authors:

```
Accuracy =  39.419318816303736
Precision =  [36.43724696 50.98591549 48.88888889 54.54545455 14.04494382 45.05494505
  0.          73.21428571 37.27272727 41.11498258]
Recall =  [48.38709677 52.16138329 22.33502538 19.56521739 60.97560976 16.015625
  0.          43.38624339 51.89873418 69.41176471]
F1_score=  [41.5704388  51.56695157 30.66202091 28.8         22.83105023 23.63112392
  0.          54.48504983 43.38624339 51.64113786]
```

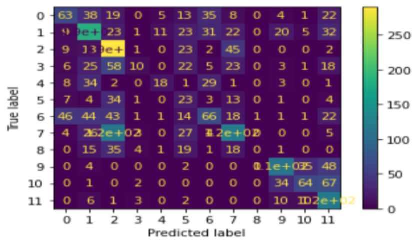


	precision	recall	f1-score	support
0	0.36	0.48	0.42	186
1	0.51	0.52	0.52	347
2	0.49	0.22	0.31	197
3	0.55	0.20	0.29	92
4	0.14	0.61	0.23	82
5	0.45	0.16	0.24	256
6	0.00	0.00	0.00	114
7	0.73	0.43	0.54	189
8	0.37	0.52	0.43	158
9	0.41	0.69	0.52	170
accuracy			0.39	1791
macro avg	0.40	0.38	0.35	1791
weighted avg	0.44	0.39	0.38	1791

Figure 14.5

MLP Classifier for 12 Authors:

```
Accuracy =  43.0349257326375
Precision =  [41.44736842 46.96969697 46.77419355 38.46153846 50.         13.60946746
 37.5        44.98141264  0.          59.04255319 54.7008547  35.19061584]
Recall =  [30.28846154 51.23966942 75.32467532  5.84795322 18.55670103 25.55555556
 25.58139535 39.67213115  0.          55.5         38.0952381  78.94736842]
F1_score=  [35.         49.01185771 57.71144279 10.15228426 27.06766917 17.76061776
 30.41474654 42.16027875  0.          57.21649485 44.9122807  48.68154158]
```



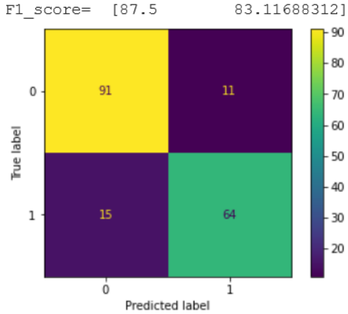

```
              precision    recall  f1-score   support

           0       0.41      0.30      0.35       208
           1       0.47      0.51      0.49       363
           2       0.47      0.75      0.58       385
           3       0.38      0.06      0.10       171
           4       0.50      0.19      0.27        97
           5       0.14      0.26      0.18        90
           6       0.38      0.26      0.30       258
           7       0.45      0.40      0.42       305
           8       0.00      0.00      0.00        94
           9       0.59      0.56      0.57       200
          10       0.55      0.38      0.45       168
          11       0.35      0.79      0.49       152

    accuracy                           0.43      2491
   macro avg       0.39      0.37      0.35      2491
weighted avg       0.43      0.43      0.40      2491
```

Figure 14.6

Naïve-Bayes Classifier:

Naïve-Bayes Classifier for 2 Authors:

```
Accuracy =  85.6353591160221
Precision =  [85.8490566  85.33333333]
Recall =  [89.21568627 81.01265823]
F1_score=  [87.5        83.11688312]
```

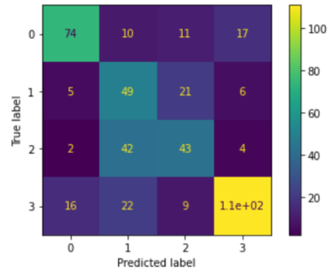



```
              precision    recall  f1-score   support

           0       0.86      0.89      0.88       102
           1       0.85      0.81      0.83        79

    accuracy                           0.86       181
   macro avg       0.86      0.85      0.85       181
weighted avg       0.86      0.86      0.86       181
```

Figure 15.1

Naïve-Bayes Classifier for 4 Authors with F=1000

```
Accuracy =  62.66968325791855
Precision =  [76.28865979 39.83739837 51.19047619 80.43478261]
Recall =  [66.07142857 60.49382716 47.25274725 70.25316456]
F1_score=  [70.81339713 48.03921569 49.14285714 75.        ]
```

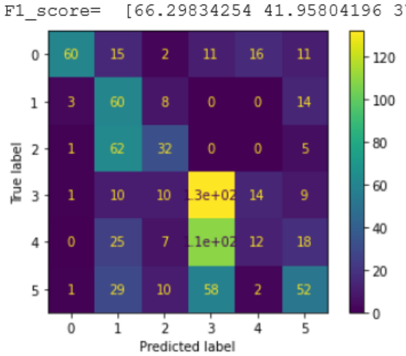



```
              precision    recall  f1-score   support

           0       0.76      0.66      0.71       112
           1       0.40      0.60      0.48        81
           2       0.51      0.47      0.49        91
           3       0.80      0.70      0.75       158

    accuracy                           0.63       442
   macro avg       0.62      0.61      0.61       442
weighted avg       0.66      0.63      0.64       442
```

Figure 15.2

Naïve-Bayes Classifier for 6 Authors:

```
Accuracy =  43.55444305381727
Precision =  [90.90909091 29.85074627 46.37681159 42.58064516 27.27272727 47.70642202]
Recall =  [52.17391304 70.58823529 32.         75.          7.01754386 34.21052632]
F1_score=  [66.29834254 41.95804196 37.86982249 54.32098765 11.1627907  39.8467433 ]
```

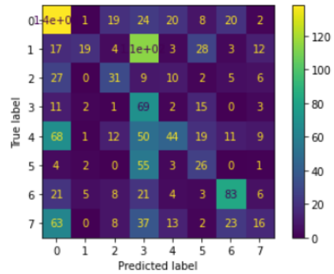



```
              precision    recall  f1-score   support

           0       0.91      0.52      0.66       115
           1       0.30      0.71      0.42        85
           2       0.46      0.32      0.38       100
           3       0.43      0.75      0.54       176
           4       0.27      0.07      0.11       171
           5       0.48      0.34      0.40       152

    accuracy                           0.44       799
   macro avg       0.47      0.45      0.42       799
weighted avg       0.46      0.44      0.41       799
```

Figure 15.3

Naïve-Bayes Classifier for 8 Authors:

```
[436] Accuracy =  34.27192276749798
      Precision =  [39.54154728 63.33333333 37.34939759 18.20580475 44.44444444 25.24271845
       57.24137931 29.09090909]
      Recall =  [59.48275862  9.5        34.44444444 66.99029126 20.56074766 28.57142857
       54.96688742  9.87654321]
      F1_score=  [47.50430293 16.52173913 35.83815029 28.63070539 28.11501597 26.80412371
       56.08108108 14.74654378]
```

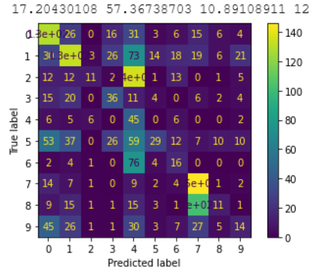


	precision	recall	f1-score	support
0	0.40	0.59	0.48	232
1	0.63	0.10	0.17	200
2	0.37	0.34	0.36	90
3	0.18	0.67	0.29	103
4	0.44	0.21	0.28	214
5	0.25	0.29	0.27	91
6	0.57	0.55	0.56	151
7	0.29	0.10	0.15	162
accuracy			0.34	1243
macro avg	0.39	0.36	0.32	1243
weighted avg	0.42	0.34	0.32	1243

Figure 15.4

Naïve-Bayes Classifier for 10 Authors:

```
Accuracy =  31.825795644891127
Precision =  [40.95238095 46.66666667 45.83333333 33.33333333  9.27835052 46.03174603
 19.27710843 45.20123839 26.19047619 22.22222222]
Recall =  [54.66101695 38.7755102   5.69948187 36.73469388 64.28571429 11.93415638
 15.53398058 78.49462366  6.875       8.80503145]
F1_score=  [46.82395644 42.3566879  10.13824885 34.95145631 16.21621622 18.95424837
 17.20430108 57.36738703 10.89108911 12.61261261]
```

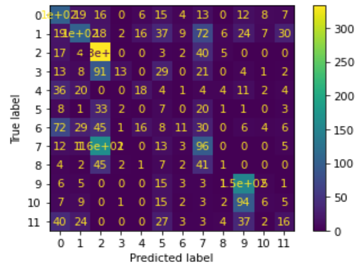


	precision	recall	f1-score	support
0	0.41	0.55	0.47	236
1	0.47	0.39	0.42	343
2	0.46	0.06	0.10	193
3	0.33	0.37	0.35	98
4	0.09	0.64	0.16	70
5	0.46	0.12	0.19	243
6	0.19	0.16	0.17	103
7	0.45	0.78	0.57	186
8	0.26	0.07	0.11	160
9	0.22	0.09	0.13	159
accuracy			0.32	1791
macro avg	0.33	0.32	0.27	1791
weighted avg	0.38	0.32	0.30	1791

Figure 15.5

Naïve-Bayes Classifier for 12 Authors:

```
Accuracy =  35.21703521703522
Precision =  [30.56379822 44.06779661 44.69798658 59.09090909 31.57894737  3.88888889
 27.5        27.74566474  4.16666667 44.57478006 17.14285714 20.25316456]
Recall =  [50.73891626 30.23255814 82.42574257  7.14285714 17.30769231  9.21052632
  4.8245614  31.47540984  0.95238095 79.58115183  4.16666667 10.25641026]
F1_score=  [38.14814815 35.86206897 57.96344648 12.74509804 22.36024845  5.46875
  8.20895522 29.49308756  1.5503876  57.14285714  6.70391061 13.61702128]
```




```
              precision    recall  f1-score   support

           0       0.31      0.51      0.38       203
           1       0.44      0.30      0.36       344
           2       0.45      0.82      0.58       404
           3       0.59      0.07      0.13       182
           4       0.32      0.17      0.22       104
           5       0.04      0.09      0.05        76
           6       0.28      0.05      0.08       228
           7       0.28      0.31      0.29       305
           8       0.04      0.01      0.02       105
           9       0.45      0.80      0.57       191
          10       0.17      0.04      0.07       144
          11       0.20      0.10      0.14       156

    accuracy                           0.35      2442
   macro avg       0.30      0.27      0.24      2442
weighted avg       0.34      0.35      0.30      2442
```

Figure 15.6

IV. Conclusion:

Using Cosine Similarity on 500-1000 Sample:

Algorithm	Language	Number Authors	Accuracy	Precision	Recall
Random Forest	Python	2	92.817	93.000	92.390
		4	74.886	71.041	57.622
		6	59.073	61.333	60.759
		8	45.454	42.875	43.125
		10	40.703	37.300	38.000
		12	39.020	35.0	35.0

Decision Tree	Python	2	85.635	85.0	86.0
		4	71.719	68.0	69.0
		6	50.938	52.0	52.0
		8	36.122	34.0	34.0
		10	35.231	34.0	33.0
		12	32.878	30.0	30.0
K-Neighbors	Python	2(1000F)	86.187	86.0	86.0
		4	56.108	57.0	57.0
		6	44.806	44.0	44.0
		8	31.456	32.0	29.0
		10	30.764	28.0	27.0
		12	32.035	26.0	25.0
K-Means	Python	2(100F)	83.425	83.765	83.817
		4	33.484	19.5	40.194
MLP	Python	2	87.292	87.0	87.0
		4	71.49.3	66.0	66.0
		6	56.195	68.0	55.0
		8	42.960	42.0	40.0

		10	39.419	40.0	38.0
		12	43.034	39.0	37.0
Naive Bayes	Python	2	85.635	86.0	85.0
		4(1000F)	62.669	62.0	61.0
		6	43.554	47.0	45.0
		8	34.271	39.0	36.0
		10	31.825	33.0	32.0
		12	35.217	30.0	27.0

Table 1

Testing Data:

In this part we tested some tweets for random authors to check if our algorithms guess the correct author if given a tweet of an unknown author. As shown in Figure below we noticed that the random algorithm could guess the correct author for the entered tweet which was “hussinalezzi5” and returned number “9” this number is the order of “hussinalezzi5”in array data.

{'GwadyM', 'HHShkMohd' , 'L_Alobaidli' , 'LarissaAounSky' , 'MohamedBinZayed' , 'NawalElSaadawi1' , 'NidalSabeh' , 'Pontifex_ar' , 'QueenRania' , 'hussinalezzi5' , 'mohamedahou20' , 'monther72'}

[0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11]

▾ Test Data

```
#pred_data ="ور مباشره مركز قومي ترجمه بدره اوبرا مصري قاعه طه حسين مكان ندوه ناقش كتاب دكتور"  #NawalElSaadawi1
#pred_data ="، عاجل صديق بوريس جونسون رئيس وزير بريطانيا امل ل شعب بريطاني صحه سلامه تحدي فيروس كورونا مستجد سنواجهه الله اراده عزيمه تضامن بشريه
#pred_data ="ذني دائم محمل استثناء اشبه بقيه" #L_Alobaidli
pred_data ="ادله تفاوضي ماكانت مكائد خطا قدم انتقالي داخل خارج كيان ساذج اسفومجرد متعهد اوسمسار بيع مجند تضحيهبهم خدمه جهه باستثناء الجنوب طبع 5
#pred_data ="كيه روج ل مضى امر اراد تكشف كتيبه سلاح خرج مصنف جمعيه ارهابي سقط مرحله سريع انتقل ثاني صرف نظر هجوم شخصي تكفير وهانحن مرحله ثالث 72

pred_data = pred_data.split(',')
pred_col = ['A']
z = {pred_col[0]:pred_data[0]}
k = pd.DataFrame(z,index = [0])
# clf = load('drive/MyDrive/RandomForest.joblib')
#predction = estimator.predict(k)
k['character_cnt'] = k['A'].str.len()
k['word_counts'] = k['A'].str.split().str.len()
k['characters_per_word'] = k['character_cnt']/dat['word_counts']
k['num'] = k['A'].apply(lambda x: len([x for x in x.split() if x.isdigit()]))
k['similarity'] = 0
k = k.drop({'A'}, axis=1)

clf = load('drive/MyDrive/Colab Notebooks/Output/RandomForest/RandomForest12.joblib')
# clf = load('drive/MyDrive/Colab Notebooks/Output/DecisionTree/DecisionTree12.joblib')
# clf = load('drive/MyDrive/Colab Notebooks/Output/KNeighbors/KNeighbors12.joblib')
# clf = load('drive/MyDrive/Colab Notebooks/Output/KMeans/KMeans12.joblib')
# clf = load('drive/MyDrive/Colab Notebooks/Output/MLP/MLP12.joblib')
# clf = load('drive/MyDrive/Colab Notebooks/Output/NaiveBayes/NaiveBayes12.joblib')

predction = clf.predict(k)
predction[0]
```

```
9
```

Figure 16: Test data

V. Links For Our Work:

★ https://colab.research.google.com/drive/19ABq0fEHbgKAIEn2M758G4x9Y_TLBTEJ#scrollTo=Rzb9YVv76wlB

★ https://drive.google.com/drive/folders/1TsWMEiWds3qUw1C21VL4kea0BTOj9tuJ?usp=sharing