Creating WordCloud

In this query 5 wordCloud was created, 1 for each category. Before these files were created, the special words stopWords = 'english' were removed from the articles, plus some others that were meaningless as per the question. To do this, the wordcloud + STOPWORDS library was used

Implementation of Clustering

In this query, the k-means algorithm was implemented to group our data, that is, the articles into 5 groups, and then this result to be evaluated by the clustering_Kmeans.csv output file, which is the job's demand. We first pre-processed the data, that is, our articles were counted in vectors where each vector is a dictionary and as the number of times the word is counted in the text (CounterVectorizer). We then modified the data to add an additional parameter, which is the weight of each word in the text (TFIDF-Vectorizer).

The next step was to reduce the dimensions of the vector through the LSI technique, where the TruncadedSVD library with a selected number of dimensions 20 was used (other values of this parameter were tested). And at this time we are ready to execute k-mean me k = 5. Let us stress that the implementation of the algorithm was made from a ready library, nltk.cluster with the KmeansClusterer function. This function enables the cosine similarity metric method to be used. Finally, a step was taken to evaluate the algorithm where the requested clustering_Kmeans.csv file is created at the end.

Classification Implementation

In the query, the 4 required algorithms were designed to categorize the data. Three of the four algorithms were implemented from python ready libraries except for KNN. Before performing the algorithms, the preprocessing step was preceded. For all algorithms, the same methodology was used. The preprocessing sequence is first to make a vector for each text with the number of occurrences of words in the text (CounterVectorizer). We then modified the data to add an additional parameter, which is the weight of each word in the text (TFIDF-Vectorizer).

The next step was to reduce vector dimensions through the LSI technique, where the TruncadedSVD library was used. For each of these 3 steps, several tests were performed on the parameters of these functions for the best result. For example, the meaningless words from the stopword library were removed, the min_df kai max_df variables were used to exclude the very common words or the very rare ones, that is, words that do not give us some extra information for analyzing the articles.
Also, more emphasis was put on the analysis and experimentation with the algorithm parameters in order to better understand them. In the next question, that is, in the implementation of our own classifier, more emphasis was placed on preprocessing (the title of articles was used) in order to achieve the best result.

1) Naive bayes

In this algorithm, the dimensionality reduction step was omitted since as we know this algorithm is not based on distance calculation and generally does not need to define our data in a d dimension but the only thing that counts is probability. Multinomial and Bernulli were tested.

2) Random Forest

Virtually here we use a lot of decision makers gathered. This number was tested at 10,20,50,100. The conclusion from these tests was that the result did not have any major alteration.
For example from 10 to 50 had a better result but from 50 to 100 the result remained the same.

3) Support Vector Machine

In this algorithm, although good results have been observed, we can observe that it was lagging behind

with others based on time. The experiments were done
In this case they mainly concerned the kernel, C and gamma parameters that were also requested in the speech of the exercise. For the most efficient research on these parameters, an attempt was made to use the GridSearch library simply because of the lack of knowledge of the bound parameters that must be taken, and because of the great runtime delay, this option was rejected. Generally it was observed in the kernel that rbf had better performance than linear.
4) KNN
The implementation of this algorithm was done without the help of python. Initially we defined K, the number of neighbors. The algorithm then implemented the 2 simple steps. First, for each testData data we calculated the distance from each trainData and we kept these results in a sorted list and from this list we kept the K shorter distances. The distance meter used is the Euclidean distance. The second step is the predicted step. We use the Majority voting method to select the category, that is, we count among these neighbors what categories are displayed more times, and we predict that the article belongs.

Finally, we will also mention what we have done to evaluate the algorithms. For each algorithm, we calculate accuracy / precision / recall / f1-score through 10-cross-validation and for the final result to be written and to the requested EvaluationMetric_10fold.csv file, we will keep the average of the final results.

Beat the Benchmark (bonus)

In this question a combination of algorithms, basically 2, a naive bayes and a machine learning algorithm of SVM were used. Initially, in the pre-processing step, the stemming method was tested, which can also cut some words of the same meaning, eg playing, play and puts it like a word, this of course has the risk when the words are not of the same importance and due to enough time The process was rejected in the execution of the program. Generally, one observation is that while several experiments were attempted in the pre-processing process, the difference in yield was negligible, and the reason was mainly that the algorithms from their own without any precession had a fairly high rate of accuracy. Also the title of the articles for the categorization decision was used. After performing the algorithms in the previous steps it was observed that the random forest had the best performance and the reason for this is assumed to be because while it has approximately 96% accuracy in the next 10 steps (10-cross-validation) it touches the 100% because it's probably rebuilding a new model where it corrects its old model, but that may involve the risk of overfitting if we test the model on new data. It is natural and next is this performance you can not overcome but there is no reason to use it in your own algorithm. The basic idea of this algorithm is as follows. We run the Naive bayes algorithm and extract it based on content and title. We do the same with SVM. So basically we have 4 predictions and the final prediction comes out on some occasions. The initial case is that if predictors of articles have predicted the same category then this is the final forecast for this article, we also take into account the predictors of the title. This has resulted in an increase in the performance of these algorithms. For example, NB had about 94% accuracy and SVM 95% and now it reached 96%. The code produces as output the testSet_categories.csv that is requested for the job in the strict format requested.

Execution
To run all the code files a main one is created in the src / default folder in order to execute and produce all the requested files of the exercise. These files are created in the output folder. Firstly, you run myClassifier (MINE.py) to make the testSet_categories.csv file, then the k-means that produces the clustering_Kmeans.csv file, and finally the classifiers that produce the EvaluationMetric_10fold.csv file.