All associated work for this coursework can be found at my GitHub repository which can be found here: https://github.com/c21112297/CMT316-Coursework-1-Part-2

Introduction

For this coursework, I had to create a model that has to correctly place over 2000 articles into 5 different categories. I had to utilise a variety of techniques to gather the data, perform feature engineering and produce a readable result.

The Process

The first thing I considered was how to get the data from the .txt files located in subfolders in the bbc folder. I achieved this by looping through each subfolders location to retrieve the .txt files name, extract the text from it, format it, and add it to a data frame to be operated on. I then combined each category's data frame into one data frame.

Now that the data is located in a centralised location, I split the data into the training and testing sets, storing the category label and the contents of the article with an 80/20 split of training and testing. The model I used to classify this data was support vector machines, which I trained using three different features; tf-idf vectorization, count vectorization, and count vectorization using bigrams.

I tested each feature's effect on the model by displaying the model's total accuracy as a percentage, the total amount, and the classification report.

I displayed the data in three distinct ways; accuracy percentage, the raw data amount, and the classification report. I also included the cross validation score and a confusion matrix.

Justifications

After the data is gathered it is stored in a DataFrame from the pandas module, which I used due to its flexibility and accessibility to easily pull the articles ready to be operated on. The data was then split into the training and testing sets using the train_test_split method from sklearn, which allowed me to easily split the data into the training and testing sets to my desired ratio.

A support vector machine was used to train these models, due to its relatively high accuracy and fast training time, meaning I did not have to wait a long time training a model on three different features.

The first model I used is count vectorization, which simply counts the number of times a word appears in a given text by turning each word into a vector, with each vector representing the number of times a word appears in each text. This was also used as a benchmark for when I used tf-idf vectorization, which generally performs better than count vectorization.

The second model I trained used the tf-idf vectorization feature. I used tf-idf vectorization because it focuses on word frequency and word importance, which then allows the removal of words that have less importance when generating a model.

And finally, I used count vectorization with bigrams as a feature, as bigrams are useful when extracting more meaning from segments of language, to look at the wider context behind a statement in a given text.

To evaluate the model's performance, I use several different methods:

Firstly, I calculated the models overall accuracy, which is simply the number of correct labels predicted, shown as a percentage and in raw numbers, and then displayed the classification report, showing several measures of a models performance such as the precision, recall, and f1-score. It shows these scores for every category in the dataset from business to tech. These scores are important as they are some of the fundamental key indicators of a models performance.

The second evaluation measure I utilised was cross validation, which takes a specified number of samples (folds) and tests the model. The final score is the average number across all folds. This was implemented to verify if the models suffer from overfitting.

The final measure of performance was a confusion matrix, which is another way to evaluate the accuracy of a classification. I used this because it gives me a better idea of the types of errors each model is making.

Performance

Count vectorization performance:

Count Overall Accuracy: 96.707%

Count Vectorization Classification Report:

	precision	recall	f1-score
Business	0.99	0.93	0.96
Entertainment	0.96	0.96	0.96
Politics	0.94	0.96	0.95
Sport	0.99	1.00	0.99
Tech	0.95	1.00	0.97

Cross validation score for training data: 95.875%
Cross validation score for testing data: 88.014%

Confusion Matrix:
[[80 1 4 0 1]

[0 45 0 1 1]
[1 1 68 0 1]
[0 0 0 76 0]
[0 0 0 0 54]]

Tf-idf Vectorization Performance:

Tf-idf Vectorization Overall Accuracy: 98.503%

Count Vectorization Classification Report:

	precision	recall	f1-score
Business	0.99	0.97	0.98
Entertainment	0.98	1.00	0.99
Politics	0.97	0.99	0.98
Sport	1.00	1.00	1.00
Tech	0.98	0.98	0.98

Cross validation score for training data: 97.356%
Cross validation score for testing data: 93.116%

Confusion Matrix:
[[83 0 2 0 1]
 [0 47 0 0 0]
 [1 0 70 0 0]
 [0 0 0 76 0]
 [0 1 0 0 53]]

Bigram Count Vectorization Performance:

Bigram Count Vectorization Overall Accuracy: 94.91%

Count Vectorization Classification Report:

	precision	recall	f1-score
Business	0.93	0.98	0.95
Entertainment	0.96	0.91	0.93
Politics	0.97	0.92	0.94
Sport	0.94	0.97	0.95

Tech	0.96	0.94	0.95

Cross validation score for training data: 91.908%
Cross validation score for testing data: 77.531%

Confusion Matrix:
[[84 0 2 0 0]
[1 43 0 2 1]
[1 1 65 3 1]
[1 1 0 74 0]
[3 0 0 0 51]]

By looking at the evaluation measures, we can see that the feature engineering has generated model that have a high accuracy rate of above 90%. However, looking at these models some clearly perform better than others.

The best performing model was the tf-idf vectorizer, due to the additional pre-processing functionalities that come with this model. This is followed by the count vectorizer, and finally then the bigram count vectorizer. The results all point to tf-idf vectorization being the best overall classifier.

Critical Reflection

Word Embedding

To improve the performance further I would implement word embedding, checking the number of types of words such as nouns, verbs, and adjectives, to see which types of words have a greater effect on the classification and a higher weight .

Automate Code

Another improvement would be to better automate the code by using functions. This would ensure the code doesnt repeat itself, preventing the wastage of memory, and the increase of run time.