Exercise 2

Leong Eu Jinn

April 2023

1 Introduction

This report outlines the plan of work for developing a new spam detector in Python from scratch. The methods used will be a naive Bayes classifier which handles both bag-of-words(BoW) and tf-idf Features as input. Stretch goals.

- A seperate kNN classifier can be made as a stretch goal in order to compare results with both classifiers. It's input features to compare similarities between data points being word counts and their respective euclidean distance.[1].
- Another stretch goal that can be achieved would be a confusion matrix to calculate accuracy etc.
- Results comparison with other freely available classification datasets.
- cross-validation approach in model evaluation

2 Data

The data used will be a SMS Spam Collection dataset, it has 5574 text messages collected. It is available here:
`http://archive.ics.uci.edu/ml/machine-learning-databases/00228/`

The following dataset will have a train/test split of 80/20 to be implemented. We will simply shuffle our data and then use indexing to achieve our different datasets or we will use the following module if allowed:
`https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.train_test_split.html`

The dataset has 4827 objects of the class "ham" and 747 objects labelled "spam". It has 86.6% and a 13.4% split. As we are not allowed to use other modules to load the dataset, we shall load it using regex.
The structure is planned as to have a dictionary with 2 keys(ham and spam) and to have the pre-processed text messages as the value pairs.

3 Methods

The process of making the Bayes classifier would be first to preprocess the data. The dataset text messages will be pre-processed first by lower casing all words and removing punctuation.
Following that we will put it into a dictionary and then split it into 4 lists. X_test,X_train, y_test and y_train.
After that we will tokenize and lemmatize the training set and removing stop words to prevent there to be too many tokens.
After processing all the data, we will make an tfidf matrix in the x_training data set.
We create a predict function using the training dataset's tfidf matrix. We know that each row represents a sentence and each column represents a term in the document's vocabulary. And so we can use that to seperate each row (sentence) into its own class in order to find the P(word given class). We can make 2 new tfidf matrixes, we index them such that each class has its own tfidf matrix and can use that calculate the new probability.
After training, we will evaluate the model using an the accuracy metric by having plotting down all correctly predicted answers and meaning it.

4 Experiments

By running the model's predict command on a reduced dataset. We got an accuracy of 0.93894 and with the original dataset, we got a high accuracy of 0.94125 even though it took a longer time.

5 Evaluation and discussion

It runs very slowly and had to be ran on a reeduced dataset that is 1/5th its original size in order to be tested and constantly debugged with issues such as an empty list after lemmatization with sentences full of stop words such as "but youre not here". Maybe implement a custom lemmatizer next time instead so that the sentence does not become completely empty. Another problem is that the program is inflexible and must be changed in order to fit the boss' needs so that the input will be a new test string and not one that was already in the class. But that is an easy fix and can be changed easily. The solution to calculate P(word given class) was (word count in class + 1)/ (total words in class + 1* set(vocab)). This was transformed to fit fidf such as:
Word count in class = sum(tfidf of the word for all documents in the class)
total words in the class = sum (tfidf of all words in the class)

6 Conclusion

Further implementation of the stretch goals such as a knn model to be done in parallel and be evaluated. That said, the project is done and fulfills requirements

References

[1] https://towardsdatascience.com/spam-email-classifier-with-knn-from-scratch-python-6e68eeb50a9e

[2] https://norma.ncirl.ie/4490/1/alanchavez.pdf

[3] https://iq.opengenus.org/naive-bayes-on-tf-idf-vectorized-matrix/

[4] https://towardsdatascience.com/tf-idf-for-document-ranking-from-scratch-in-python-on-real-world-dataset-796d339a4089

[5] https://www.analyticsvidhya.com/blog/2021/09/creating-a-movie-reviews-classifier-using-tf-idf-in-python/

[6] https://arxiv.org/pdf/1410.5329