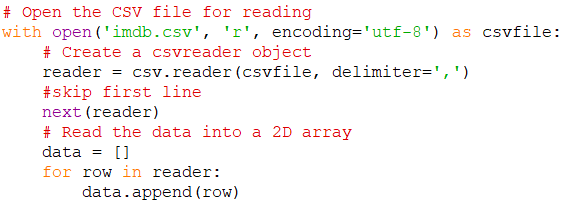
**Text Classification Model On IMDB Database**

**CE314 Assignment 2, Ben MacIver**

**Abstract**

The code implements a text classification model using the IMDB sentiment classification dataset. This dataset contains a large quantity of reviews and then a “positive” or “negative” tag. The code executes data pre-processing, linguistic feature analysis, and then the construction of a Support Vector Classification model. The performance of the model is evaluated on a test set taken from the dataset, and the accuracy returned. The data is read from a csv file then is pre-processed, with stop words and non alphabetical words removed. Some linguistic features of this cleaned data are analysed and output to the user. Then the data is trained on. The code then transforms the text data using TF-IDF vectorization and trains a Multinomial SVK model on the training set. The model's predictions are evaluated on the test set, and the accuracy is reported, along with a graph demonstration.

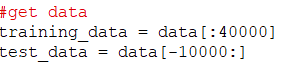
**1.Introduction**

When it comes to Natural Language Modelling sentiment analysis is extremely important. It’s what allows for machines to interpret emotions. . This code endeavours to navigate through the intricacies of text classification using a dataset of different movie or film reviews found on IMDB which of course vary in sentiment greatly, making it a very appropriate dataset. Textual context is often a very complicated metric to obtain, making this a challenging yet important task. And fortunately enough I have managed to create a model with very high accuracy on this data set.

1. **Reading the file and separating into test and training data**

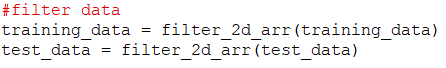
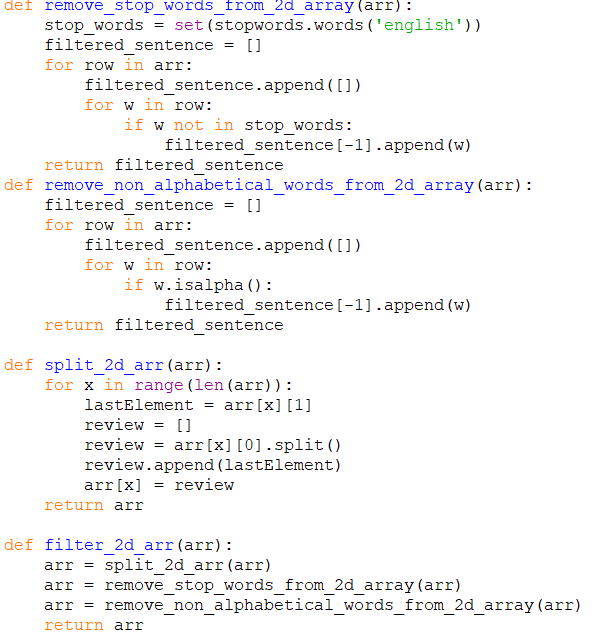
First there is the task of reading the data, which is in csv format. To do this I used the csv python library to open the csv file and then write it in to a 2d array called ‘data’.

Then we have to separate into test and training data. Like specified in the assignment brief, the training data is the first 40,000 reviews and the test data is the last 10,000 reviews.



**3.Pre Processing**

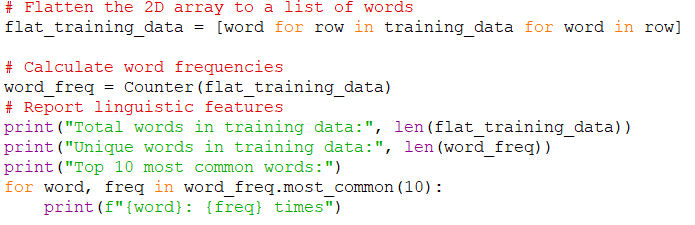
Then we need to pre process the data. For my code this involves splitting the reviews so one word corresponds to one element in an array and also removing stop words and non alphabetical words. This is done by calling a method called filter 2d array.



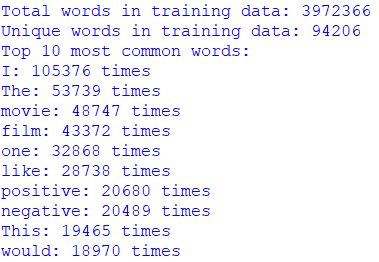
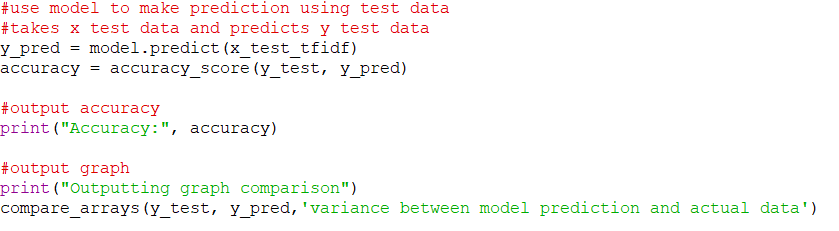
**4.Analysing the training set**

This step involved reporting the linguistic

features of the training data set. I began doing this by reporting the total words in the training data and then the total number of unique words. Then once these figures where reported I reported the 10 most common words in the training data along with the frequency of times that word appeared. Here is the code demonstrating this.



Below you can also find the output to this piece of the code.

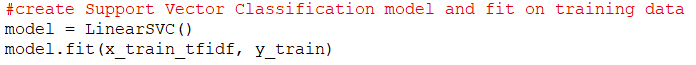


**5.Build and train the model**

This step was probably one of the most important ones and here was how I went about doing it. First I split both the training data and the test data into two separate arrays for the x and y. X being the first column of the data set (the reviews) and y being the second (the sentiment). Then I prep this raw text data for training in the text classification model by vectorizing the data and converting it into a TD-IDF matrix. This might sound complicated, but to understand the code all you need to know is this essentially formats the data so it is compatible with the text classification model.



Then to train the model we first create a Support Vector Classification Model object and save it in a variable called model, and then give it the training data to train itself on. Here is the code for this.



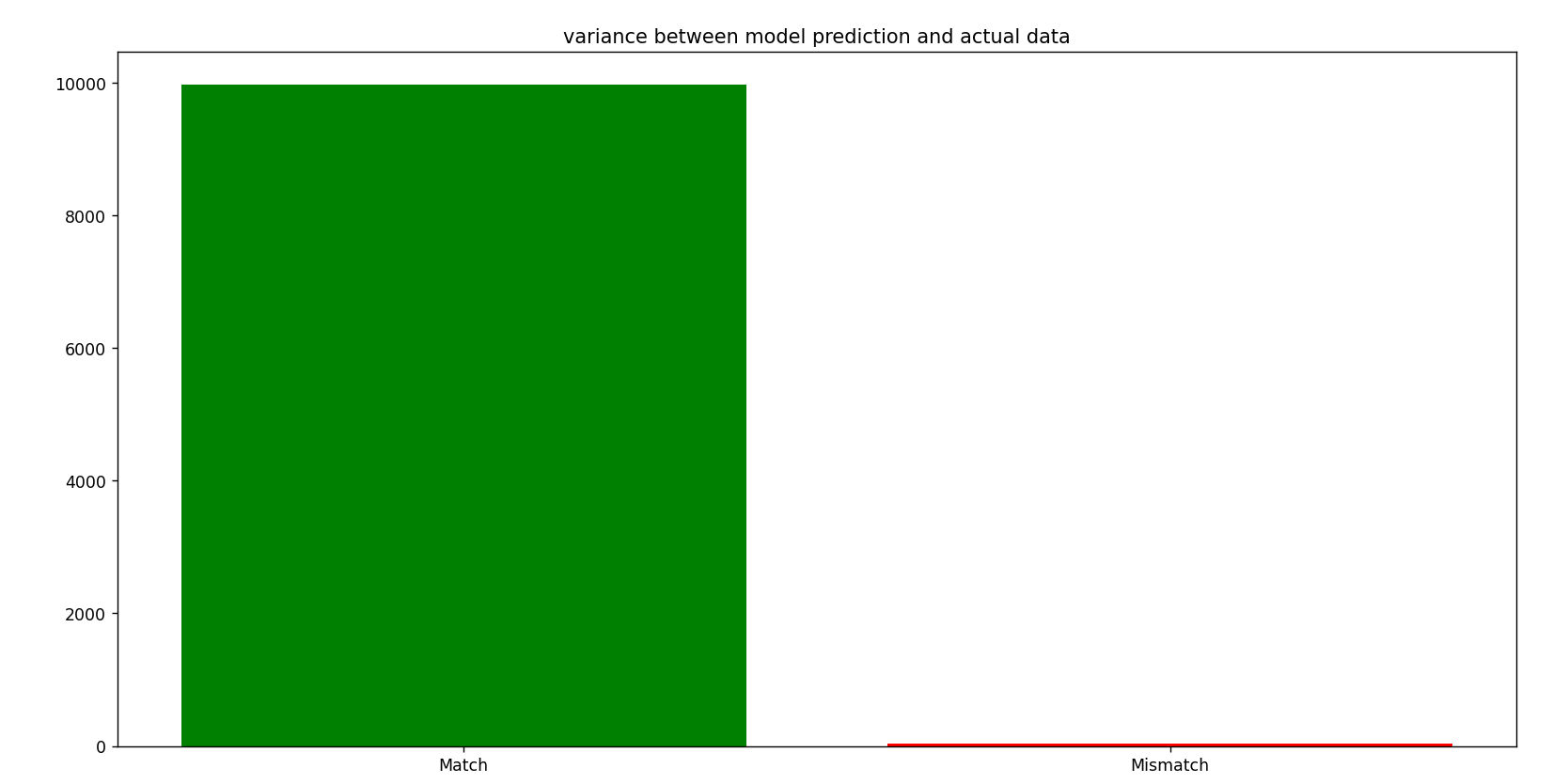
**5. Testing the model and reporting performance**

Now that we have trained our model it should be able to make predictions on what it thinks the sentiment of a set of reviews is. To put this to the test we now give the 1st row of the test data and ask the model to predict the 2nd row which contains all the sentiments. Then we take the actual 2nd row which we got from the imdb csv file we have been using this whole time and compare them. We do this in two ways. First by reporting the percentage of times they match and then by displaying a bar chart to the user that tells them how often the prediction and the actual data matches vs how often it mismatches.

Below is the code for this

This code returns an accuracy of : 0.9968 which is an extremely high percentage indicating a very successful predicition and strong model.

And below is the aforementioned graph



As you can see the model is very accurate since there is many instances of a match and hardly any of a mismatch.

**Conclusion**

In conclusion, I have successfully found a very accurate text classification model in the Support Vector Classification model. After I had coded all the necessary preparation and pre-processing on the data set the model was able to predict the sentiment on the test data with a very high accuracy of 99.6%. Now what I would do in the future if I continued work on this is give the model even more data to analyse to really test its limits rather than just using the provided IMDB database.

**References**

While I have no documents to reference, there is a number of python libraries that are required for the code to run. Below is a list of all of these and there associated import statements.

