**Report**

The dataset is the IMDB sentiment classification dataset. It consists of 50000 rows. The aim is to build a model to train using the 40000 rows of the dataset and predict the remaining 10000 rows of the dataset and thereby calculating the accuracy of the model. The performance of the model should also be assessed based on different criteria like recall, precision etc.

The data contains two columns, one with the review and the other the corresponding sentiment, i.e., whether the review is positive or if it is a negative one. First of all, the sentiment part is converted to Boolean value. That is positive is taken as 1 and negative is taken as 0.

Then pre-processing is done to the data. This includes removing non alphanumeric characters and removing stopwords. English stopwords are available in nltk corpus. Also, the words are converted to lowercase.

Either stemming or lemmatization could be done to the data. I had tried PorterStemmer as well as WordNetLemmatizer separately. In both cases I could not find much increase in accuracy and in some cases I could even find the accuracy decreasing. So, I am not using stemming or lemmatization for both the models I have tried out. I have commented out the code for both stemming and lemmatization. This is basically what I have done for pre-processing.

While analysing the linguistic features of the data, I came to some conclusions. I formed two lists of words: one for words from positive reviews and other one for negative reviews. Then I found the most frequent words in each case. Not surprisingly most of the frequent words in both cases were same. This means most of the words in both positive and negative reviews are same. Some of them were related to the film industry. So, we can conclude that most words we see in reviews don’t help us to distinguish between positive and negative reviews.

But some words occur more in positive reviews and some other words occur more in negative reviews. This was a useful observation. For example, the word ‘great’ occurs more in positive reviews than negative reviews. This means that suppose if the review that is supposed to be classified contains the word ‘great’, it is more likely to be a positive review than a negative review. However, it is not compulsory that it should be positive. It could also be negative.

Similarly, the word ‘worst’ occurs more in negative reviews than it occurs in positive reviews. So, if a review to be classified has the word ‘worst’, it is more likely a negative review than a positive review. But it is not compulsory that it is a negative review. It could also be a positive one. But considering many words like this in a review, it becomes easier for a trained model to classify the review.

This was a very useful observation in the dataset. Another observation was that the average number of words in a positive review and that of the average negative review are almost same i.e., almost 100 words. This means that the number of words in the review does not have much affect on whether it is positive or negative.

The next step is to divide the pre-processed dataset into training set and testing set. This was already given in the requirement and it was done accordingly. The first 40000 rows were used for training and the rest 10000 was used for testing. Thereby the performance of the model was calculated by calculating the accuracy, precision, recall etc.

Two models were tried for the purpose of classification: using Support Vector Machine and using XGBoost algorithm.

**Support Vector Machine**

Support Vector Machines are supervised learning methods which are really useful for sentiment analysis.

For using this model, first I converted the data into a vector format using the TfidfVectorizer. Then the model was defined with a linear kernel. The model was trained with the training data. The test data was input to the model to obtain the predictions. The performance of the model was assessed by calculating the precision, recall, accuracy, etc. All of them were in the range 0.87-0.88.

The accuracy obtained was 0.8769.

**XGBoost**

XGBoost is a tree-based machine learning algorithm. It is short for Extreme Gradient Boosting and it is designed for speed and performance.

For the case of this problem, I have converted the data to a vector format using CountVectorizer. I have then converted it into a format that is accepted by the model by using the function xgb.DMatrix(). Then I have defined the parameters of the model and then trained the model using the training data. Then I have passed the test data to the model to make the predictions. The output or prediction of the model is in the form of scores for each class. The class with the higher score is considered the prediction or result.

The performance of the model is assessed by calculating accuracy, precision, recall, etc. All were found to be in the range in and around 0.84. The accuracy of the model was found to be 0.8434.

Although both SVM and XGBoost provide accuracy above 80% we can see that SVM provides a better accuracy than XGBoost.

Even so, XGBoost takes much less time than SVM to train. So, if time is the main concern, it is better to go with XGBoost. But if time is not a big concern and more accuracy is required then SVM is the solution.

**References**

https://www.geeksforgeeks.org/python-stemming-words-with-nltk/

https://www.geeksforgeeks.org/python-lemmatization-with-nltk/

https://medium.com/@vasista/sentiment-analysis-using-svm-338d418e3ff1

https://ai.plainenglish.io/sentiment-classification-using-xgboost-7abdaf4771f9