CLASSIFICATION OF MOVIE REVIEWS

REQUIRED INFORMATION

Mason User Id: jbidhan Miner2 Username: flash Best Public Score: 0.66

DESCRIPTION OF APPROACH:

I approached the problem reading the question and making notes for the steps required. The approach I used can be summarized as below:

1. **Reading and storing the Training file** using pandas.read_csv : I created 2 methods (function: readTrainfile() and readtestfile()) which needed a parameter for filepath to read the input. Using pandas.read_csv() function read the file and stored as a pandas' DataFrame having column names: [label , sentence]. This sentences column was then passed on to another function to process the text (function: fixdata(sentences)). Then the filtered text was returned in the DataFrame format.
2. **Preprocessing of the sentences** (cleaning) using various libraries like SnowballStemmer, RegexpTokenizer (used for splitting sentence into words using regular expression) and unicodedata and regular expressions (re). Firstly, using regular expression I removed all the numbers and then rejoined the sentences. Then I made all the characters to lowercase and then removed the English language stop words (lnltk.corpus) . Using regular expression, I replaced multiple repeating characters with a single character like rrrrr with r. I also removed special characters and html tags. Then again using regular expression words having length less than 2. Then using unicodedata normalize removed diacritics(accents) from the sentences. Using the SnowballStemmer (nltk.stem.snowball) stemmed words and then stored back the sentence on the same index. I used SnowballStemmer(Porter2) as it is more accurate than Porter Stemmer. I preferred stemmer over lemmatizer as it was fast and serves our purpose well.
3. **Cross Validation**: Dividing the training data using StratifiedKFold (sklearn.model_selection.StratifiedKFold) and initially by creating 80:20 Training and test data manually out of Training data. I used StratifiedKFold to ensure that each fold of dataset has the same proportion of observations with a given label and the datasets are balanced.
4. **Vectorization of the sentences** using CountVectorizer (sklearn.feature_extraction.text): By using the fit_transform() function we got sparse vectors (scipy.sparse) of the tokenized sentences. I created a function named vectorizingData() which took testdata (transform()) and training data: returned the vectorized data. The parameters set for CountVectorizer were -> min_df=0, lowercase=True, stop_words="english", and strip_accents="unicode" which are for minimum frequency for words, lowering the case, removing stopwords, and striping accents respectively
5. **Finding distance** between the test data and training data using various distance like cosine distance, Euclidean distance, and linear kernel (all from sklearn.metrics.pairwise). The core logic of finding distance involved iterating over the index of test-data and, then for each index of test data's sparse matrix I found distance from training data's sparse matrix indexes. In the prediction finally used linear kernel as it was the most efficient and had better accuracy.
6. **Prediction**: Using the Majority voting, predicting the label for the test data using basic python and itemgetter, max and numpy.argsort(), etc. The Distance found in the previous step was sorted using numpy.argsort() and top K neighbors were taken out using various python functions like itemgetter(), max() and numpy.argsort(). Thus, the prediction was done based on maximum occurring label value i.e., +1 or -1. Using the function saveOutput() saved the text file using numpy.savetxt().
7. **Finding accuracy** for the algorithm and determining value of K nearest Neighbors and other parameters involved. I had created a function which took test original label and the predicted labels and returned the percentage of correct predictions based on correct predictions/ total predictions *100.
8. **Real Test and Prediction:** Reading test data file and applying Step 2, 3, 5, and 6

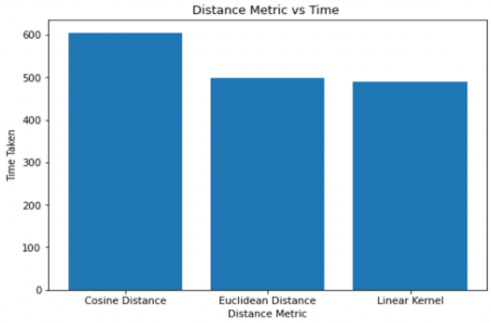
*The library modules which were used were wrapped for modularity and the sake of reusability.

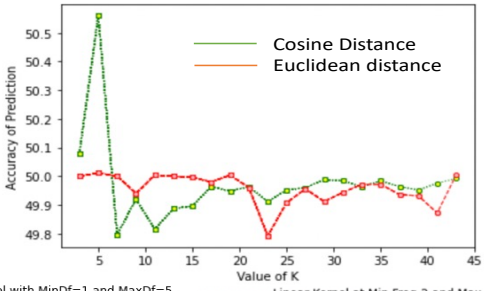
Methodology of Choosing the Approach and Associated Parameters and Graphs

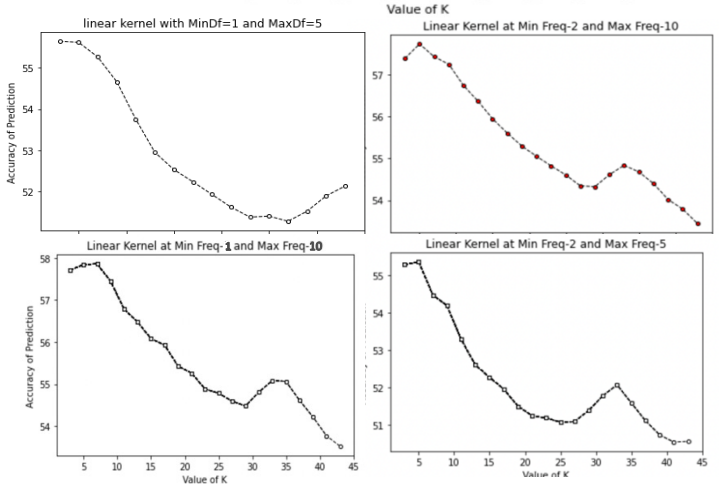
I Initially started writing the code on Jupyter notebook after resolving initial issues of setting up the environment and created two files, one for analysis and other for actual prediction. There were serval factors involved in choosing following steps:

min_df	max_df	No. of Words
Default (1)	Default (1.0)	48293
2	Default	28232
Default	5	32261
Default	10	36544
2	5	12200
2	10	16483

1. **Preprocessing**: After reading the file and preprocessing it without using custom methods, there were over a hundred thousand words in the bag of words. (That is, after deleting stop words and applying the basic filtering offered by the CountVectorizer in the sklearn package). Thus, the custom text processing was necessary. After processing the text on various parameters, the data that was obtained as shown in the table. I did certain operation specially like removing repeating words like rrrrr to r. etc. I also removed letters having length less than 2 as they don't add much weight to the sentence and mostly looked like typing mistake. I also converted accents to normal English characters like å to a, etc. Doing so, the bag of words had 48k words initially and then I tried various frequency limits (upper and lower) to track the changes in the words.
2. **Distance Metric**: The key factor in deciding which metric to use was speed of operation and accuracy of prediction on the test data. To do so, I ran an experiment for a fixed value of K and found that Linear Kernel was faster and achieved better accuracy (<55 in all the cases) in the training phase. Linear Kernel can be used when the data is linearly separable, it is most utilized when a data set contains many features like Text which has a lot of features, as each alphabet is a new feature. Also, cosine similarity is equivalent to linear kernel for scipy.sparse matrices.
3. **Cross Validation:** There were initial issues in splitting data, so I used arrays to split data by adding [0 to 5000] indexes to test set and rest to training set (80:20 split or 0.2 split) to check the working of code and testing faster. Then, I decided to use StratifiedKFold to split data into 5 folds for better validation. I then created sets out of data using a modular function. I also faced an issue iterating over the sets as it was not in the same format as the file so had to change the iterator by a bit in the core KNN function. I wrote a loop ranging from 3 to 45 (value of K) for validating and tuning the model.
4. **Frequency of words**: CountVectorizer parameters such as min df and max df, which represent the least and maximum frequency of words occurring in the document were used and plotted for various values of K and found that highest accuracy was achieved with min_df=2. On testing the same over test data the algorithm achieved maximum accuracy of 0.66.

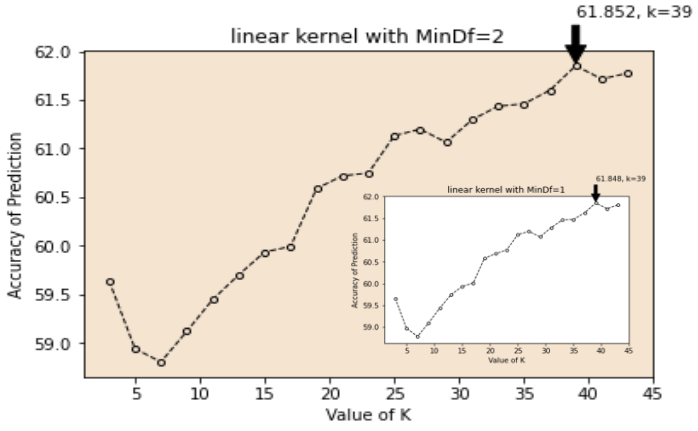






CONCLUSION

On seeing the results of the various experiments, it was thus decided to go with the value of K as 39 and minimum frequency of words (min_df) as: 2 and maximum frequency of words as the default value(max_df). Linear kernel was taken as distance metric as it was way faster than other algorithms and gave same results as provided by cosine similarity.



REFERENCES

Documentation from https://scikit-learn.org/ , https://www.nltk.org/, https://www.scipy.org , and https://matplotlib.org/