NLP Assignment 3

Contribution :- Megha Mathur (MT19104) --> Question1
Sonali (MT19081) -->Question2

Q1. Sentiment Analysis

Features Implemented from Paper Provided:-

- **All_capital words count**

  Firstly tweets are pre-processed, i.e. Tokenized using nltk.word_tokenize and then checked whether tokens are alphabet or not. And Then it is verified whether it is uppercase or not. Then count of all such tokens are reported.

  Preprocessing steps:- Tokenized using nltk.word_tokenize
  :- Check whether it is an alphabet or not using isalpha().
  :- Checked whether it is uppercased or not.

  Assumptions :- Whole training and testing data is considered.

- **POS- count of occurrence of each part of speech**

  Firstly tweets are pre-processed, i.e. Tokenized using nltk.word_tokenize. Then using nltk.pos_tag, tags for each token is computed. Then features of all unique pos_tags are created and for each tweet presence or absence of pos_tag is reported.

  Preprocessing steps:- Tokenized using nltk.word_tokenize
  :- Using nltk.pos_tag, tags of each token are determined.

  Assumptions :- Whole training and testing data is considered.
  :- nltk.pos_tag is used for generating pos_tag for each word or token.

- **Hashtag count**

  Firstly tweets are pre-processed, i.e. Tokenized using nltk.word_tokenize. Then using regex (reg_hash='^#'), all such tokens that begin with hashtags are checked and their count is computed.

  Preprocessing steps:- Tokenized using nltk.word_tokenize
  :- Using regex ('^#') count of hashtags are computed.

  Assumptions :- Whole training and testing data is considered.

- **Punctuation**

  - **Number of contiguous sequence of !, ? or both**

    Firstly tweets are pre-processed, i.e. Tokenized using nltk.word_tokenize. Then if Token comprises ? or !, is counted only if its successive tokens also contain ? ! or both. Count of such contiguous pairs is reported.

    Preprocessing steps:- Tokenized using nltk.word_tokenize

    Assumptions :- Whole training and testing data is considered.

  - **Whether the last token contains an exclamation mark or question mark.**

    Firstly tweets are pre-processed, i.e. Tokenized using nltk.word_tokenize. Then if Token comprises ? or ! or both at the end then it is reported as 1 otherwise 0.

    Preprocessing steps:- Tokenized using nltk.word_tokenize

    Assumptions :- Whole training and testing data is considered.

- **Emoticons**
  - ➢ **Presence or absence of emoticons at any location.**

    Firstly tweets are pre-processed, i.e. Tokenized using nltk.word_tokenize. Then after matching the regex for emoticons at any position of the text, their presence or absence is reported.

    Preprocessing steps:- Tokenized using nltk.word_tokenize

    Assumptions :- Whole training and testing data is considered.

    :- All such emoticons that start with colon(:) and end with),@,#,&,*,),<,>,3,8,/ are considered.
  - ➢ **Presence or absence of emoticons at the end.**

    Firstly tweets are pre-processed, i.e. Tokenized using nltk.word_tokenize. Then after matching the regex for emoticons at the end of the text, their presence or absence is reported.

    Preprocessing steps:- Tokenized using nltk.word_tokenize

    Assumptions :- Whole training and testing data is considered.

    :- All such emoticons that start with colon(:) and end with),@,#,&,*,),<,>,3,8,/ are considered.

- **Elongated Words**

  Firstly tweet data is preprocessed i.e. tweet is word tokenized and checked whether it is an alphabet. Then for each token, characters are analyzed and any character in the token is repeated more than two times is considered as an elongated word or tokens. Their count is reported.

  Preprocessing steps:- Tokenized using nltk.word_tokenize

  :- Check whether it is an alphabet or not using isalpha().

  Assumptions :- Whole training and testing data is considered.

  :- words/ tokens with any particular character repeated more than 2 times is considered as an elongated word/token.

- **Negation-Count of negated contexts**

  Firstly tweet data is preprocessed i.e. tweet is word tokenized and then checked against the regex(reg_neg = re.compile (r' (? : ^ (? : never | no | nothing | nowhere | noone | none | not | havent | hasnt | hadnt | cant | couldnt | shouldnt | wont | wouldnt | dont | doesnt | didnt | isnt | arent | aint) $) | n\ 't')) and if it matches then successive tokens are matched against the regex (reg_clause=re.compile(r'^[.:;!?]$')) and if matched then all the word/tokens between are counted as negative context tokens.

  Preprocessing steps:- Tokenized using nltk.word_tokenize

  Assumptions :- Whole training and testing data is considered.

  :- regex is taken from the paper reference itself for matching.

- **Word-Ngram**
  - ➢ **Contiguous N-Gram**

    Firstly tweet data is preprocessed i.e. tweet is word tokenized and checked whether it is an alphabet or not and stop words are removed. Then for N=1,2,3,4 separate dictionaries are created. Then presence or absence of N-Grams are reported.
  - ➢ **Non-Contiguous N-Gram :-**

For N=2,3 non contiguous tokens are considered. With skip value as 1, features are computed. Then it is checked against an already created contiguous N-Gram, i.e. if that token is already present then it is ignored.

Preprocessing steps:- Tokenized using nltk.word_tokenize
:- Check whether it is an alphabet or not using isalpha().
:- Stop words are removed.

Assumptions :- For training, only 10000 data samples are taken.
:- Words with occurrence < 50 are considered as rare and not considered as tokens.
:- testing is done for the whole corpus.
:- Skip value is taken as 1 for non-contiguous N-Gram.

Due to the computation issue, Count-vectorizer is used for Contiguous N-Gram and Maximum features were taken as 100 but whole training corpus is considered for feature generation.

- **Lexicons**
  - ➢ **NRC-Hashtag sentiment Lexicon**

    Initially, after reading the NRC- hashtag sentiment lexicon text file, a dictionary is created with key as word and value as sentiment score. Then for each training and testing tweet, the score of each token is taken from the dictionary and summation is reported as a score.

    Preprocessing steps:- Tokenized using nltk.word_tokenize
    :- Check whether it is an alphabet or not using isalpha().

    Assumptions :- Whole training and testing data is considered.

  - ➢ **NRC-Sentiment140 Lexicon**

    Initially, after reading the NRC-Sentiment140 lexicon text file, a dictionary is created with key as word and value as sentiment score. Then for each training and testing tweet, the score of each token is taken from the dictionary and summation is reported as a score.

    Preprocessing steps:- Tokenized using nltk.word_tokenize
    :- Check whether it is an alphabet or not using isalpha().

    Assumptions :- Whole training and testing data is considered.

  - ➢ **MPQA**

    Initially, after reading the MPQA lexicon text file, a data frame is created with columns as word and polarity. Then for each training and testing tweet, count of positive and negative words are computed after checking against the MPQA list.

    Preprocessing steps:- Tokenized using nltk.word_tokenize
    :- Check whether it is an alphabet or not using isalpha().

    Assumptions :- Whole training and testing data is considered.

  - ➢ **Bing Liu Lexicon**

    Initially, after reading the positive and negative text file of Bing Liu Lexicon, each token generated after tokenization is checked against each file and found in the particular list, its count increases. Therefore for each tweet, count of positive and negative features and their difference is computed.

Preprocessing steps:- Tokenized using nltk.word_tokenize
:- Check whether it is an alphabet or not using isalpha().

Assumptions :- Whole training and testing data is considered.
:- Positive-negative count is taken for difference.

Feature Vector Shape:- `(1120000, 162) ------> Train`
`(480000, 162) ------> Test`

Additional Feature Implemented :-

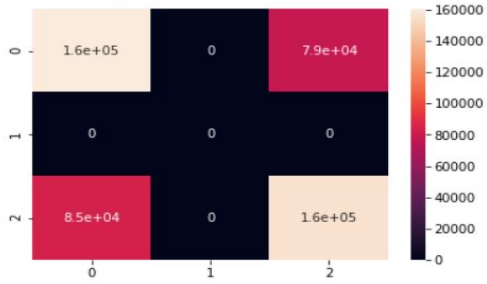
- **Tf-idf based Bag of words**

  Initially tweets are tokenized and checked isalpha or not and then using TfidfCountVectorizer, tf-idf for each tweet is computed and reported.

```
Confusion_matrix is :-
[[160523.      0.  78681.]
 [     0.      0.      0.]
 [ 84522.      0. 156274.]]


 Classes  Precision    Recall
       0   0.655076  0.671072
       4   0.665123  0.648989


Accuracy is :-  0.65999375
Macro average F1 score is:-  0.6600648599211598
```



Models implemented :-

- **Naive Bayes**

  Initially vocabulary of the whole corpus is created for each unique token and then for training data i.e. for each class separate dictionaries were created with keys as word and value as its count in that particular class. Then for each key class probability is calculated with add-1 smoothing. Then for each testing data probability with each class is computed and the label of maximum probability class is given and then reported.

  Evaluation :-

```
Confusion_matrix is :-
[[193711.   1390.  44103.]
 [     0.      0.      0.]
 [ 60433.   2334. 178029.]]

 Classes  Precision    Recall
       0   0.762210  0.809815
       4   0.801456  0.739335

Macro average F1 score is:-  0.7781870612270346
Accuracy is :-  0.7744583333333334
```

```
Precision is :-  0.5212218348464259
Recall is :-  0.5163834747598104
```

Assumptions:- For confusion matrix all three labels are considered but for precision, Recall , F1-score and accuracy only 2 labels are considered.

- **SVM**

  Using sklearn, SVM is implemented. Firstly x_train is taken after removing polarity and id from the feature vector and then y_train is taken as polarity and model is fitted and using predict function test labels are predicted.

  Evaluation :-

```
Confusion_matrix is :-
[[179082.      0.  60122.]
 [     0.      0.      0.]
 [ 47723.      0. 193073.]]


     Classes  Precision    Recall
       0   0.789586  0.748658
       4   0.762547  0.801811

Accuracy is :-  0.7753229166666666
Precision is :-  0.7760662106372096
Recall is :-  0.7752347705408744
Macro average F1 score is:-  0.7756502677784326
```

  Assumptions:- C is taken as 1

  :- For confusion matrix all three labels are considered but for precision, Recall , F1-score and accuracy only 2 labels are considered.

- **MLP**

  Using sklearn, MLP is implemented. Firstly x_train is taken after removing polarity and id from the feature vector and then y_train is taken as polarity and model is fitted and using predict function test labels are predicted.

  Evaluation :-

```
Confusion_matrix is :-
[[179237.      0.  59967.]
 [     0.      0.      0.]
 [ 54003.      0. 186793.]]

 Classes  Precision    Recall
       0   0.768466  0.749306
       4   0.756982  0.775731

Accuracy is :-  0.7625625
Precision is :-  0.7627242254612054
Recall is :-  0.7625186780561583
Macro average F1 score is:-  0.7626214379085156
```

  Assumptions:- max_iter is taken as 200.

:- For confusion matrix all three labels are considered but for precision, Recall , F1-score and accuracy only 2 labels are considered.

- **Decision Tree**

  Using sklearn, Decision-tree is implemented. Firstly x_train is taken after removing polarity and id from the feature vector and then y_train is taken as polarity and model is fitted and using predict function test labels are predicted.

  Evaluation :-

```
Confusion_matrix is :-
[[158535.      0.  80669.]
 [     0.      0.      0.]
 [ 86702.      0. 154094.]]

 Classes  Precision    Recall
       0   0.646456  0.662761
       4   0.656381  0.639936
Accuracy is :-  0.6513104166666667
Precision is :-  0.6514187007096428
Recall is :-  0.6513482677549309
Macro average F1 score is:-  0.6513834823283389
```

  Assumptions:- max_iter is taken as 200.

  :- For confusion matrix all three labels are considered but for precision, Recall , F1-score and accuracy only 2 labels are considered.

Q2. EmoINT

Preprocessing Steps:

1. **Mentions are** removed using regular expression. (The complete username is removed not only @)
2. **URLs** present in the tweets are removed**.**
3. **Punctuations** are removed from tweets
4. **Elongations** are handled**.**
5. All tweets are **converted to lowercase.**
6. **Hashtags** have been removed**.**
7. **Contractions are expanded.**

FEATURE SET:

1. VADER sentiments are implemented. On this basis, we get 4 features : pos , neu , neg and compound values.
2. Lexicon based features:
   - POLAR WORD COUNT : This is implemented using Bing Liu lexicon and MPQA lexicon. Both lexicons give two features each i.e. positive and negative score.
   - AGGREGATE POLARITY SCORE : This implemented using three lexicons : Sentiment 140, Afinn word and Sentiment Wordnet. The words present in every lexicon are different. So a combined data frame has been formed from the three lexicons. The positive are values are replaced

by positive and the values less than zero are replaced by negative. Then two features are built i.e. positive and negative score.

- AGGREGATE POLARITY SCORE (HASHTAGS) : This is implemented using NRC hashtag sentiment lexicon. In this sentimentScore is a feature built. For every word in the tweet, the score is added up and a final score is generated for the tweet.
- EMOTION WORD COUNT : This is implemented using NRC Word emotion lexicon. This lexicon contains 10 emotions and word mapping of them with words. So total 10 features are built using this for 10 unique emotions.
- AGGREGATE EMOTION SCORE: This is implemented using NRC 10 expanded lexicon. One feature is built by adding the values for all emotions associated with each word of a tweet.
- AGGREGATE EMOTION SCORE (HASHTAG) : This is implemented using NRC hashtag association lexicon. In this a feature vector is built with the scores given and value is generated by adding up the values for words in a particular tweet.
- EMOTICONS SCORE : This is implemented using AFINN lexicon. The mapping is created for emoji to emoticon and then the value is taken for the respective emoticon from the csv and the total value is assigned to the feature vector value.
- NEGATION SCORE : A list of negating words is created. Then the number of such words in each tweet are counted.
- N GRAM FEATURES : The n gram features are generated for values 1,2

Models implemented are:

- Decision Trees
- MLP
- SVM

The evaluation metric used is:

Pearson Coefficient.

REFERENCES :

https://scikit-learn.org/stable/modules/generated/sklearn.feature_extraction.text.CountVectorizer.html
https://github.com/pmbaumgartner/text-feat-lib/tree/master/notebooks
https://gist.github.com/lorenzOliveto/f20a89e9f68276cae21497a177ad8a4c
https://github.com/fnielsen/afinn
https://www.geeksforgeeks.org/python-pandas-merging-joining-and-concatenating/
https://towardsdatascience.com/basic-tweet-preprocessing-in-python-efd8360d529e
https://www.geeksforgeeks.org/python-pandas-dataframe/

https://docs.python.org/3/library/re.html
https://github.com/pemagrg1/Text-Pre-Processing-in-Python/blob/master/notebook/preprocess_notebook.ipynb
https://www.geeksforgeeks.org/create-a-pandas-dataframe-from-list-of-dicts/
https://www.geeksforgeeks.org/create-a-pandas-dataframe-from-lists/
https://stackoverflow.com/questions/176918/finding-the-index-of-an-item-in-a-list
https://www.geeksforgeeks.org/count-nan-or-missing-values-in-pandas-dataframe/
https://www.geeksforgeeks.org/get-unique-values-from-a-column-in-pandas-dataframe/
https://stackoverflow.com/questions/36684013/extract-column-value-based-on-another-column-pandas-dataframe
https://www.geeksforgeeks.org/how-to-get-rows-index-names-in-pandas-dataframe/
https://www.programiz.com/python-programming/list#:~:text=In%20Python%20programming%2C%20a%20list,float%2C%20string%20etc.).&text=A%20list%20can%20also%20have,is%20called%20a%20nested%20list.
https://www.geeksforgeeks.org/how-to-rename-columns-in-pandas-dataframe/
https://stackoverflow.com/questions/13295735/how-to-replace-nan-values-by-zeroes-in-a-column-of-a-pandas-dataframe
https://github.com/ugis22/analysing_twitter/blob/master/Jupyter%20Notebook%20files/Analysis%20of%20Twitter.ipynb
https://www.geeksforgeeks.org/how-to-drop-one-or-multiple-columns-in-pandas-dataframe/
https://scikit-learn.org/stable/modules/feature_extraction.html
https://www.geeksforgeeks.org/python-sentiment-analysis-using-vader/
https://www.geeksforgeeks.org/how-to-drop-one-or-multiple-columns-in-pandas-dataframe/
https://kanoki.org/2019/07/17/pandas-how-to-replace-values-based-on-conditions/
https://pandas.pydata.org/pandas-docs/stable/reference/api/pandas.DataFrame.reindex.html
https://heartbeat.fritz.ai/support-vector-regression-in-python-using-scikit-learn-89cc18e933b7
https://scikit-learn.org/stable/modules/tree.html