CSE556: Natural Language Processing

Assignment 03 Report

Naman Jain
2018347

Navneet Agarwal
2018348

Assignment 3a: Sentiment Analysis in Twitter

Naive Bayes Classifier

Preprocessing

- Convert all the tweets into lower case.
- Split every tweet on space to normalize the usernames and URLs, assuming that these two are always separated by spaces from other words. Every username was converted to ***@someusername*** and every URL was converted to ***http://someurl.***
- After normalizing, convert the tweets back to string format.
- All punctuations were removed from the tweets and every word was lemmatized.

Assumptions

- The training was done only for 50k tweets due to computation and memory constraints.
- Implemented Multinomial NB with Laplace Correction.

Methodology

- After the preprocessing was done, we constructed a vocabulary considering both the train and test set. We also removed the stop words from this vocabulary.
- We use this vocabulary in the CountVectorizer( ) to create the unigrams feature matrix for both the train and test set. The constructed feature matrix was converted into a 2d NumPy array.
- Now to train our Naive Bayes Classifier, we first calculated the number of positive and negative instances in the train set to get the **Prior Probability** of both the classes.
- After this, we constructed two dictionaries, one for the positive class and one for negative class. These dictionaries had the unigrams features as the keys and the values as the occurrences of a unigram feature in a particular class.
- The values of the dictionaries were then used to calculate and store the Likelihood Probabilities. (The counts were added by 1 and divided by the sum of vocabulary length and the number of occurrence of tokens in a class)

- For any sentence that needs to be tested, we use it's feature vector, if a particular feature is non-zero, we use the dictionaries to get the likelihood probability for each class. Log function was used while calculating the posterior probability for both negative and positive classes to handle the underflow.
- The test tweet was assigned the label of the class which had more posterior probability than the other, the predicted label was stored in a NumPy array, which coil be further used for evaluation.

MLP / DT / SVM

Preprocessing

- The preprocessing steps are different for every feature.
- Some of the functions used to extract the features use tweets as it is.
- While some of them used it after lowering the case, removing the punctuations, URLs, hashtags and usernames.

Assumptions

- Every feature except the n-grams was extracted for the whole training set. Due to RAM constraints, the n-gram feature was extracted only for 50 k training tweets.
- For the Emoticons feature extraction, we have used the AFINN-emoticon-8 file provided in the assignment.
- "I" was not considered to be a word with all letters in upper case.
- A word is elongated if it contains more than 2 continuous repetitions of letters.

Methodology

- We have constructed a function for every feature extraction, which are mentioned below.
- **FuncAllCapsCount(GivenTweets):**
    - This function is used to calculate the count for words with all letters in Upper case for each tweet in the list of **GivenTweets.**
    - The tweets from the training set and testing set are provided as it is.
    - It tokenizes the tweets using word_tokenize and then finds the desired count using a regex.
    - It returns a NumPy array containing the desired count for every tweet in the GivenTweets**.**

- **FuncHashtagsCount(GivenTweets):**
    - This function is used to calculate the count of Hashtags for each tweet in the list of **GivenTweets.**
    - The tweets from the training set and testing set are provided as it is.
    - It splits the tweets on space and then finds the desired count using a regex.

    - It returns a NumPy array containing the desired count for every tweet in the GivenTweets**.**

- **FuncPunctuationsCount(GivenTweets):**
    - This function is used to calculate the count for continuous punctuations(? And !) and whether the Last token is ? or ! , for each tweet in the list of **GivenTweets.**
    - The tweets from the training set and testing set are provided as it is.
    - It tokenizes the tweets using word_tokenize and then finds the desired count using a nested loop and the presence using equality operation on the last token.
    - It returns a NumPy array containing the desired count and the presence of punctuation symbols ? and ! in the last token for every tweet in the GivenTweets**.**

- **Construction of POS tag as a feature**
    - We construct a set which contains all the possible tags in both the train and test set. This set contains all the POS tag features.
    - We also calculate the counts of the POS tags in this set for every tweet.
    - Using this count and set we construct the feature vector for the count of POS tags.

- **FuncBingLiu_MPQA_NRCemotion(GivenTweets):**
    - This function is used to calculate the count of positive and negative words for each tweet in the list of GivenTweets.
    - The Tweets here are passed after preprocessing.
    - We used MPQA, NRC emotion association and Bing Liu Lexicon data sets to create two sets, one for positive words and one for negative words.
    - Then for every tweet, we check if a particular word is in the positive/negative set and then increase the count accordingly.
    - It returns a numpy array containing the count of positive and negative words for each tweet in list of GivenTweets.

- **Func_NRCHashtagSentiment(Giventweets)**
    - This function is used to calculate the total score of hashtags for each tweet in the list of GivenTweets.
    - The Tweets here are passed as it is.
    - We used the NRC Hashtag Sentiment Lexicon data set in this function.
    - Then for every tweet, we check if a particular word is hashtag or not, is present in NRC Hashtag Sentiment data set and then increase/decrease the score accordingly.
    - It returns a NumPy array containing the total score for Hashtags for each tweet in the list of GivenTweets.

- **Func_Sentiment140Lexicon(Giventweets)**
    - This function is used to calculate the total score of words for each tweet in the list of GivenTweets.

    - The Tweets here are passed as it is.
    - We used the Sentiment 140 Lexicon data sets in this function.
    - Then for every tweet, we check if a particular word is in the Sentiment 140 Lexicon or not and then increase/decrease the score accordingly.
    - It returns a NumPy array containing the total score of words for each tweet in the list of GivenTweets.

- **elongated_words(Giventweets)**
    - This function is used to calculate the count of elongated for each tweet in the list of GivenTweets.
    - The Tweets here are passed as it is.
    - We tokenize the words using word_tokenize and then use a regex for every word in a tweet, to check if it is elongated or not. If it is an elongated word we increase the count.
    - It returns a NumPy array containing the count of elongated for each tweet in the list of GivenTweets.

- **emoticon_score(Giventweets)**
    - This function is used to calculate the total score for emoticons used for each tweet in the list of GivenTweets.
    - The Tweets here are passed as it is.
    - We use the AFINN emoticon data set for this function.
    - We split the words using space and then check if this emoticon is in the AFINN data set, increment/decrement the score according to the score of the emoticon.
    - It returns a NumPy array containing the total score for emoticons used for each tweet in the list of GivenTweets.

- **negation_count(GivenTweets)**
    - This function is used to calculate the count of negated context for each tweet in the list of GivenTweets.
    - The Tweets here are passed as it is.
    - We split the tweets using sent_tokenize and then for every sentence we use word_tokenize.
    - For every token in a sentence, we check if the first token is a negation word and the last token is { **, . : ; ! ?** } , then we increment the count of negated context count for that tweet.
    - It returns a NumPy array containing the count of negated context for each tweet in the list of GivenTweets.

- **N-Gram construction (n= 1,2,3,4)**
    - We use the train set and test set for constructing the vocabulary of unigrams, bigrams, trigrams and 4-grams. This vocabulary was constructed using the Countvectorizer fit( ) function.

    - Using this Vocabulary we construct the n-gram feature for both the train set and test, using Countvectorizer fit_transform( ) function
    - This gives out a sparse matrix containing the n-gram features.

- Now after using the above-mentioned features we construct a data frame for all the features except the n-gram features and then save it.
- We then convert the above-mentioned DataFrame into a sparse matrix and then concatenate this sparse matrix and the n-gram sparse matrix using the hash_stack function.
- These features of the training set were then fitted to all the three Classification models, SVM/ Decision Trees / MLP and then the values were predicted using the predict functions for these models.
- For the Evaluation, we created a function func_Eval(Yactual, Ypred), which takes the actual labels of the testing set and the predicted set. Then we calculate the count of TruePositives, FalsePositives, TrueNegatives and FalsePositives. These counts then can be used to calculate the accuracy, precision, recall and F1 scores. These counts also help in constructing the confusion matrix.

Files Attached

- There are 4 python notebooks for question 1
    - **Ques1_NaiveBayes** contains the unigrams feature extraction and the implementation of NB from scratch.
    - **Ques1_features_excluding_ngrams** contains the feature extraction except for the n-grams features.
    - **Ques1_NgramConstruction_DT_MLP_SVM** contains the ngram construction, training and testing part for DT,SVM and MLP.
    - **Question1_evaluation** contains the code for calculating the accuracy, precision, recall,f1 score, macro f1 score and the confusion matrix.
    - **Bonus** contains the implementation of bonus feature.
- The train and test feature vectors were very big in size so we have attached google drive link for them: https://drive.google.com/drive/folders/150uqiPPjrweEj0TRSuvZRrWeoe3JgQU6?usp=sharing
- The models are saved in the **SavedModels** folder.
- The predicted labels are saved in the **Predictions** folder.

Bonus features

- We went through the usernames and found that three particular usernames were very frequent @mileycyrus,@tommcfly and @ddlovato. So we created three additional features for the presence of these usernames in a particular tweet. And then used the Decision Trees Classifier to test these additional features. The macro-f1 score increased from 0.7348 to 0.7366. This feature is relevant because the majority of the tweets might be positive or negative for a particular username.

Assignment 3b: Emotion Intensity Prediction

Preprocessing

- All the characters in tweets were converted to lower case
- URLs, usernames, punctuations and hashtags were removed from tweets while a duplicate set containing hashtags and punctuations was maintained for feature extraction from hashtags and emoticons
- All the tweets and features were maintained in pandas dataframe

Methodology for features extraction

- Separate py notebooks and models were created for ‘anger’ and ‘joy’ emotions.
- Positive, negative, neutral and compound VADER sentiment score was calculated for each tweet using a python library
- Polar word count feature using ‘MPQA’ and ‘Bing Liu’ corpora were extracted for each tweet by storing words in respective sentiment sets. Union of both the corpus was recorded.
- Aggregate polarity scores: Sentiment140 (d1) was computed by storing scores of each term in the dictionary/Hash table and aggregate score for all words in a tweet was computed. For Sentiwordnet(d2) multiple python libraries were used to fetch sentiment score from synset of the word. For AFFIN (d3), Affin library was imported which directly provided sentiment score for each tweet.
- Aggregate polarity scores (Hashtags): Sentiment scores of each hashtag word from tweets were extracted using NRC Hashtag Sentiment lexicon file.
- Aggregate emotion score using NRC-10 Expanded lexicon and Aggregate emotion score (Hashtags) using NRC Hashtag Emotion Association Lexicon were computed using similar steps as above for respective emotion X (anger/joy)
- Emoticons score using the AFINN project (Nielsen, 2011) were extracted for each emoji (identified by using regEx) in a tweet and sum of positive or negative values were reported for each data point.
- Count of Negating words for each tweet was computed using regEx.
- Emotion word count: Count of the number of words matching respective emotion X from NRC Word-Emotion Association Lexicon were reported for each tweet.
- N-grams (1 and 2) features were computed using CountVectorizer function from Sklearn library. Vocabulary terms having document frequency less than 5 were ignored while making feature vectors.

Bonus features

- Slang sentiment scores- Sentiment in tweets expressed by slang words were captured using Slang Sentiment Dictionary (SlangSD) for computing sentiment score of each slang term in a tweet. The aggregate score of slang terms in a tweet was reported as a feature. It improved the errors and Pearson & Spearman coefficients.

Files Attached

- There are 2 Python notebooks for question 2. Q2_anger.ipynb and Q2_joy.ipynb contain all the code for emotion anger and joy respectively.
- All the extracted features from tweets are dumped into files Features_anger.txt and Features_joy.txt
- Output predictions from DT/SVM/MLP models are stored in files dt_anger.txt, dt_joy.txt,SVM_anger.txt etc.
- Evaluation_Screeshot.jpg contains the output from evaluation.py script
- Lexicons folder contain all the corpora for lexicon feature extraction.

Contribution

- Both the members had an equal contribution to this assignment.
- Functions for the lexicon features were created by Naman.
- Functions for Features other than lexicon in task 1 were created by Navneet.
- The prediction and model fitting for task 1 were done by Navneet.
- The prediction and model fitting for task 2 were done by Naman.
- We both discussed the approach for Naive Bayes implementation and then the code was written.
- The functions for evaluation metrics in question 1 was done by Navneet.