CS918: 2018-19, Exercise two: Sentiment classification for social media

Student ID: 1896548

#Sentiment Classifier Development Steps

1. Dataset Explorations
2. **Build baseline model with tf-idf features**
    - Pre-processing (with baseline, this is kept as simple as possible):
        - Replace URL with tag "URLLINK"
        - Replace @someuser with tag "USERMENTION"
        - Remove other non-alphaneumeric characters
    - Features:
        - Using lowercasing on each token
        - Using unigram and bigrams
        - Using term-frequency and inverse-document-frequency
    - Classifier:
        - Support Vector Machine with Linear kernel
            - *Accuracy:* ***0.6575***
            - *Macro F1 Average Score:* ***0.63***
3. **Build baseline model with word-embedding and sentiment lexicons**
    - Pre-processing: same as in baseline model with tf-idf features
    - Features:
        - Using sentiment lexicons for positive/negative word counts
        - Using pre-trained word-vectors
            - sum on each dimension of each tweet
            - average on each dimension of each tweet
        - Classifier:
            - Logistic Regression,
                - *Accuracy:* ***0.644***
                - *Macro F1 Average Score:* ***0.62***
4. **Combine baselines models and optimizations**
    - 1st. Stage
        - Pre-processing: as done in baseline models
        - Features:
            - baseline tf-idf + baseline world-embedding + sentiment lexicons
        - Classifier:
            - Logistic Regression,
                - *Accuracy:* ***0.687***
                - *Macro F1 Average Score:* ***0.67***
    - 2st. Stage, improvement based on errors
        - Pre-proecssing:
            - Add function to handle emojis
            - Add function to handle long words, e.g. looooooong
        - Features:
            - Add neutral word counts, excluding stop words
        - Classifier:

- Logistic Regression,
  - *Accuracy:* ***0.689***
  - *Macro F1 Average Score:* ***0.67***

#Dataset Explorations

Before modeling, first I start with some explorations with the tweets dataset, hoping to gather some ideas for future developments.

Senetiment class counts

The class count is **NOT balanced**, with almost half of the tweets are neutral(about 46.1%), and positive tweets have the second highest counts (about 35.4%), while the negative tweets have fewer number (about 18.5%).

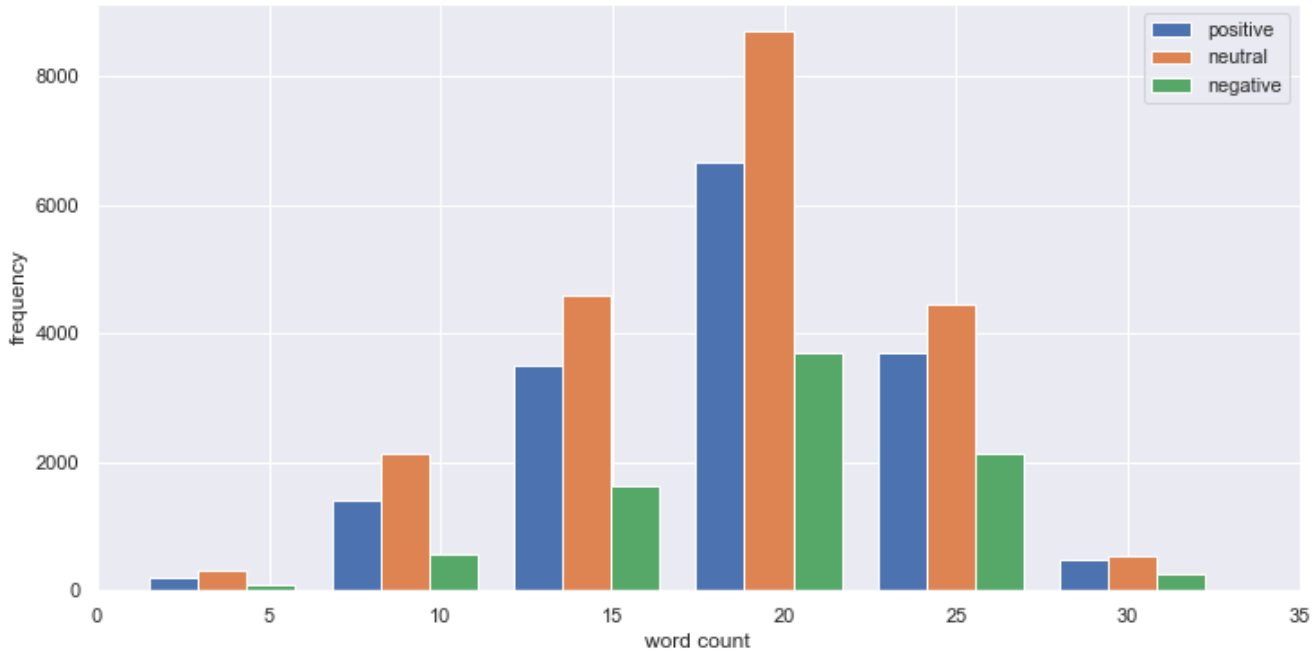
Number of words in each tweet

> Are longer tweets likely to be netural?

If each tweet is split by space, **the majority of tweets have less than 35 words**. In fact, among all the tweets in train set, only about 3.5% has more than 35 words, that is 16 out of 45026 tweets. Of the 16 tweets: 43.75% is neutral, 31.25% is positive and 25.00% is negative; which doesn't vary much from the distribution of overall counts.

> Will length of tweets correlated with its sentiment?

Another view on tweets are the comparison of each sentiment on different word counts. The histogram between word-count and count of sentiments is plotted (as below), and it looks like *length does not have strong correlation with sentiments*, perhaps the effects can be further tested when modelling.



Emojis used in tweets

> Do emojis correlate with the sentiment expressed in a tweet? What are the most used emojis?

With python function `emoji.UNICODE_EMOJI`, we can get a dictionary of emojis for checking if emoji is in a tweet. The following codes are to build a list of emojis of each sentiment.

```
pos_emoji = []; neu_emoji = []; neg_emoji = []
for idx, row in data.iterrows():
    tmp = set(row.tweet).intersection(set_emoji)
    if len(tmp) > 0:
        if row.label == 'positive':    pos_emoji.extend(list(tmp))
        elif row.label == 'negative':  neg_emoji.extend(list(tmp))
        elif row.label == 'neutral':   neu_emoji.extend(list(tmp))
```

Some findings are:

1. loudly_crying_face (😭) and face_with_tears_of_joy (😂)
    - Not meaningful for predicting sentiments, since they both have higher count in all three classes
2. thinking_face(🤔)
    - Not meaningful in differentiating between neutral and negative
3. red_heart (❤), smiling_face_with_heart-eyes (😍), raising_hands (🙌)...etc
    - Have higher counts with positive sentiment
4. pouting_face (😡) and face_with_rolling_eyes (🙄)
    - Have higher counts with negative sentiment
5. weary_face (😩) and backhand_index_pointing_down (👇)
    - Have higher counts with neutral sentiments

#Baseline Model: tf-idf features

Baseline 1st Model Intuitions

This basline model is constructed to check the performance of Bag-Of-Words approach. Using `sklearn` functions for faster implementations.

1st Model Pre-Processings

Data pre-processing is done with three functions provided during seminar: `replaceURLs()` is to replace URLs in tweet to a tag 'URLLINK', `replaceUserMentions()` is to replace any mention of another user to a tag 'USERMENTION', and `replaceRest()` will remove any other non-alphaneumeric charaters. The idea is to have basic tokens at the start of modelling, and *it's noted that some meaningful features will be removed with these processings, such as emojis*.

1st Model Featue Extractions

The following steps are performed with `sklearn.feature_extraction.text.TfidfVectorizer`, on each tweet.

1. **lowering cases** of each tweet, then tokenizing it by **uni-gram and bi-gram**
2. construct a sparse matrix (`scipy.sparse`) of **counts**
    - columns: each unigram and bigram
    - rows: each tweet
    - each cell in the matrix represents the count of each token in a tweet

3. convert to **term-frequency** and **inverse-document-frequency** matrix

1st Model Classifiers

Four classifiers are tested (using devset) with this baseline model:

1. Support Vector Machine, with linear kernel; ***Accuracy: 0.6575***
2. Random Forest, with `estimators=10`; *Accuracy: 0.572*
    - Random Forest cannot perform well with whole sparse matrix of tf-idf
        - Have to limit the number of featues used
3. Naive Bayes; *Accuracy: 0.577*
4. Logistic Regression, with one vs. rest approach; *Accuracy: 0.6515*

#Baseline Model: word-embedding + sentiment lexicons features

2nd Model Intuitions

This model will be using *word-embedding features* and generate *sentiment word counts from lexicons*. The goal here is to compare the performance with Bag-Of-World by using pre-trained vectors. As for sentiment lexicons, we'll be counting the number of positive/negative words in each tweet.

2nd Model Pre-Processings

Pre-processing is done as in the first model, allowing for easy comparison between the features.

2nd Model Featue Extractions: Sentiment Lexicons and Word Vectors

The first part of feature extraction is to create counts of positive/negative words. Here I will be using the same dictionary provided in first exercise. The function `count_opinon_lexicons()` create counts for both positive and negative words in a tweet.

The second part is using `gensim.models.KeyedVectors.load_word2vec_format` to load a *pre-trained word-vectors on Google News*, and compute statistics of a tweet on each dimension with `numpy`. Alternatively, there's another available pre-trained word-vector which is trained on tweets with 400 dimension, but its size is 3-times larger than the one on Google News. The second pre-trained vectors doesn't have obvious accuracy boost on devset, so in this exercise the default is using the one trained on Google News.

```
# build features with word vectors
def tweet_vectors(sent, dim=300, method='sum'):
    tokens = list(filter(None, sent.split(' ')))  # filter out ' ' token
    tmp = np.empty((len(tokens), dim))  # default dimension is 300
    for idx, token in enumerate(tokens):
        try:
            tmp[idx, :] = wv_model.get_vector(token)
        except KeyError:
            tmp[idx, :] = 0
    if method == 'sum': return np.sum(tmp, axis=0)
    if method == 'max': return np.max(tmp, axis=0)
    if method == 'min': return np.min(tmp, axis=0)
    if method == 'mean': return np.mean(tmp, axis=0)
```

2nd Model Classifiers

Two classifiers are tested (using devset) with this baseline model:

1. Support Vector Machine, with linear kernel; *Accuracy: 0.849*
    - this result is *not reliable*, the algorithm fails to coverge
2. Logistic Regression, with one vs. rest approach; ***Accuracy: 0.644***

#Combined Model

Combined Model Pre-processing

With combined model, some new pre-processings are implemented. Firstly, two new functions are added to **convert emojis into string tags that can be analyzed by sentiment lexicons and word vectors**, and the tags are repeated 3 times for better weight when processing. Here the positive and negative emojis are collected from dataset explorations and online resources. After testing, the feature does contribute to some slight improvement to the overall performance.

```
#reference on some top common emojis http://emojitracker.com/
def replace_happyf(tweet):
    return re.sub(r'♡|❤|👌|😍|😊|💞|😘|☺|😁|👍|😀|😇|🎉|(:[-]?\))', 'happy
happy happy', tweet)
def replace_sadf(tweet):
    return re.sub(r'🤔|😞|😠|😒|😩|😔|😢|💔|💩|(:\()', 'hate hate hate',
tweet)
```

Secondly, some words typed by users may have repeated lettes, such as 'looooooooong', which is actually 'long'. A new function `replace_long()` is to handle this kind of spellings.

Combined Model Feature Exrations

During error analysis, it's noticeable from confuision matrix that ***most of the incorrct classification are due to classifying "Real" positive or negative values into "Predicted" neutral class***. To address this problem, here I add a new feature for sentiment lexicons that is **the number of neutral word counts, excluding stopwords in a tweet**. The function used in 2nd model `count_opinon_lexicons()` is modified to compute the number of neutral worlds.

```
def count_opinon_lexicons(tweet, words_pos, words_neg):
    tmp = [token.lower() for token in tweet.split(' ')]
    count_pos = len(set(tmp).intersection(words_pos))
    count_neg = len(set(tmp).intersection(words_neg))
    count_neu =
len(set(tmp).difference(stopWords).difference(words_pos).difference(words_neg))
    return (np.float64(count_pos), np.float64(count_neg), np.float64(count_neu))
```

Other features have tried including: adding number of word counts on each tweet, adding length of each tweet (number of characters), count number of exclamations in a tweet, if URL is present or not..etc. However,

these features didn't contribute to the model performance, thus not included.

Combined Model Classifier

So far as in baseline models, Logistic Regression has better performance and training time comparing to other classifiers. So for the combined model, I would be using `sklearn.linear_model.LogisticRegression` for tuning and submission.

The final result

- Logistic Regression,
  - tuned model parameters:
    - `tol=0.0001`, `C=1.0`, `solver='liblinear'`
  - Accuracy: **0.689**