**Programming Assignment 2 – Semi-supervised Text Classification**

**Ankit Batra (PID: A53252501)**

1. **Code**

Code and README present

1. **Supervised: Improve the Basic Classifier**

The basic classifier uses a naïve approach based on representation of features as a bag-of-words. That is the matrix of features is constructed based solely on the count of words in reviews. This is clearly not a good approach, as some useless words like ‘the’, ‘a’ etc. which appear majority of the times play greater role in deciding the sentiment of reviews as compared to other less frequent words which actually decide the sentiment. Therefore, to improve upon this basic classifier, we’ve used **TF-IDF** weighting (using only the training data) which is a better version of this classifier.

Broadly speaking, the importance increases proportionally to the number of times a word appears in the document but is offset by the frequency of the word in the corpus.

**Computation of TF-IDF**

Tf-Idf is a weighting scheme that assigns each term in the document a weight based on its term frequency and inverse document frequency. The terms with higher weight scores are considered to be more important. Typically, the tf-idf weight is composed of two components:

1. Normalized Term Frequency (tf): This indicates the number of occurrences of a particular term *t* in document *d*. Therefore,

where *tf(t, d)* = term frequency for a term *t* in document *d*

And *N(t, d)* = number of times a term *t* occurs in document *d*

1. Inverse Document Frequency (idf): It typically measures how important a term is. IDF of a term *t* is given by:

where *df(t)* = number of documents containing the term *t*

**Changes to the code**: Instead of X being a feature vector based on the count of words, it’s constructing a tf-idf vector corresponding to each sentiment. Thus, sentiment.trainX is a matrix constructed from tf-idf vector of all the reviews.

1. **Performance**

Using tf-idf weighting of reviews, the accuracy (by taking reviews consisting of top 8 positive words and negative words) is shown in the figure below. Though it could be noticed that the accuracy is slightly lesser than the baseline on dev set, however, the performance of this model is much better on the test set.

Possible reasons why baseline accuracy is higher on dev set:

1. The dev set is quite small, it just contains 458 reviews. It might be by chance that the top 8 words (positive and negative) are seen most of the times in positive and negative reviews.

We also analyzed the top 8 features (that is, positive and negative words) of both versions of the classifier. As could be noticed from the figure, the baseline classifier doesn’t select good enough features as compared to the classifier with TF-IDF weighting. For instance, one of the negative feature (word) selected by baseline classifier is ‘*excited*’, which clearly doesn’t make any sense. Whereas all the top 8 features selected using TF-IDF weighting do make much more sense.

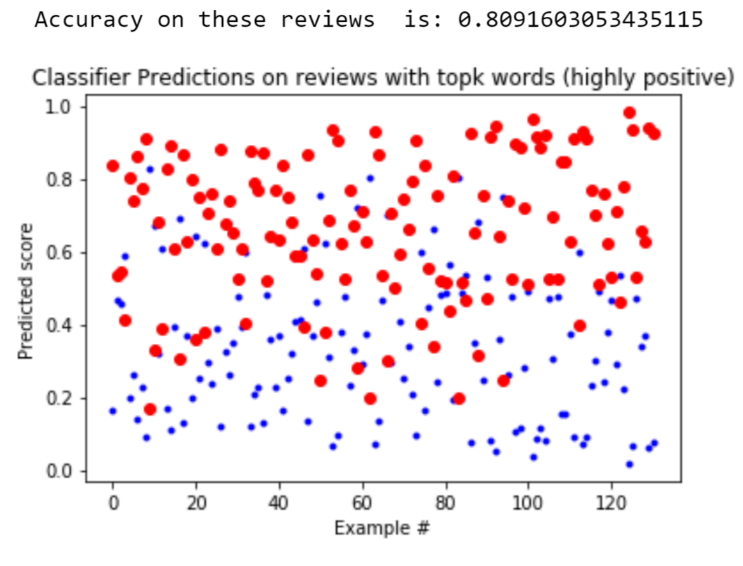
****

Figure Accuracy on highly positive reviews (based on tf-idf weighting)

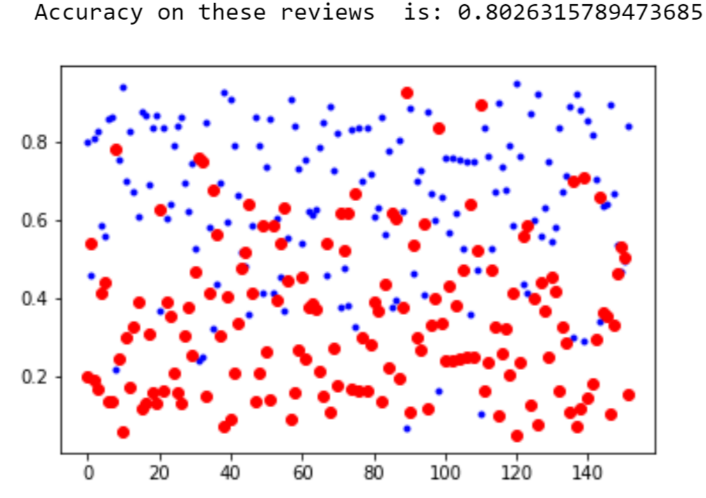


Figure Accuracy on highly negative reviews (based on tf-idf weighting)

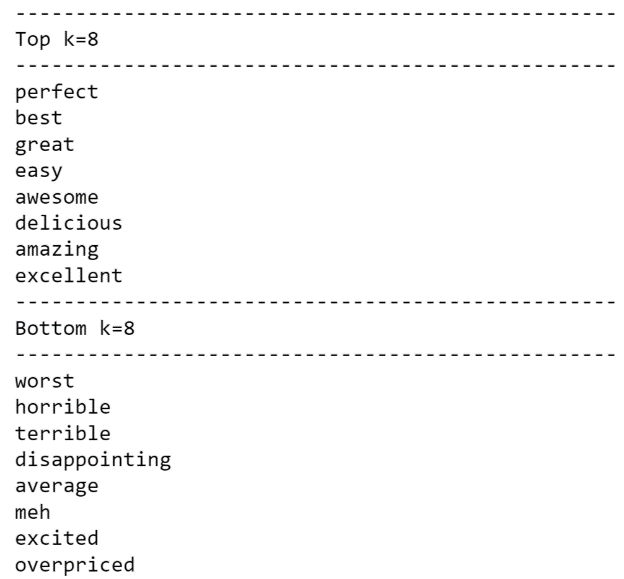


Figure Top 8 positive and bottom 8 negative words using Baseline classifier

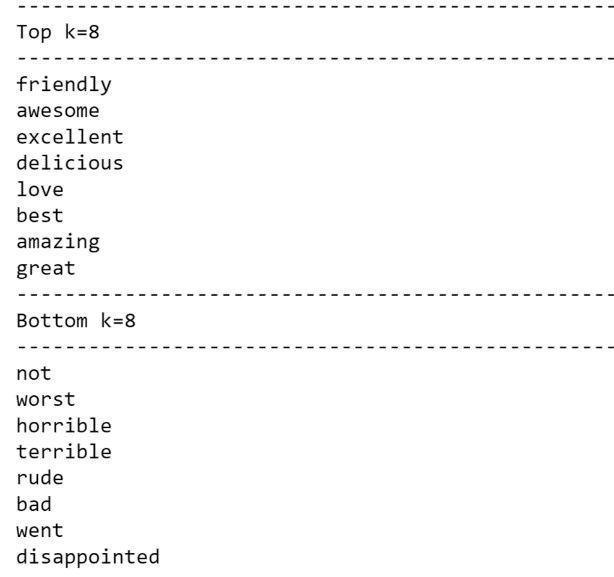


Figure Figure 3 Top 8 positive and bottom 8 negative words using TF-IDF weighting

1. **Strategies for feature engineering**
2. **L1 regularization**: Also known as ‘Lasso Regression’, it adds absolute value of magnitude of coefficient as penalty term to the loss function. If 𝞴 is 0, we get ordinary regression and if 𝞴 is very large, that will make coefficients zero and hence will under-fit.

Basically, Lasso shrinks the less important feature’s coefficients to 0, thus removing some features altogether. Since in our case we a lot of features, this works well as it selects relevant features.

The accuracy on dev set using L1 regularization is lesser than using TF-IDF weighting.

1. **Lemmatization**: It is a process of converting a word to its base form. We’ve used wordnet lemmatizer with NLTK in this assignment.

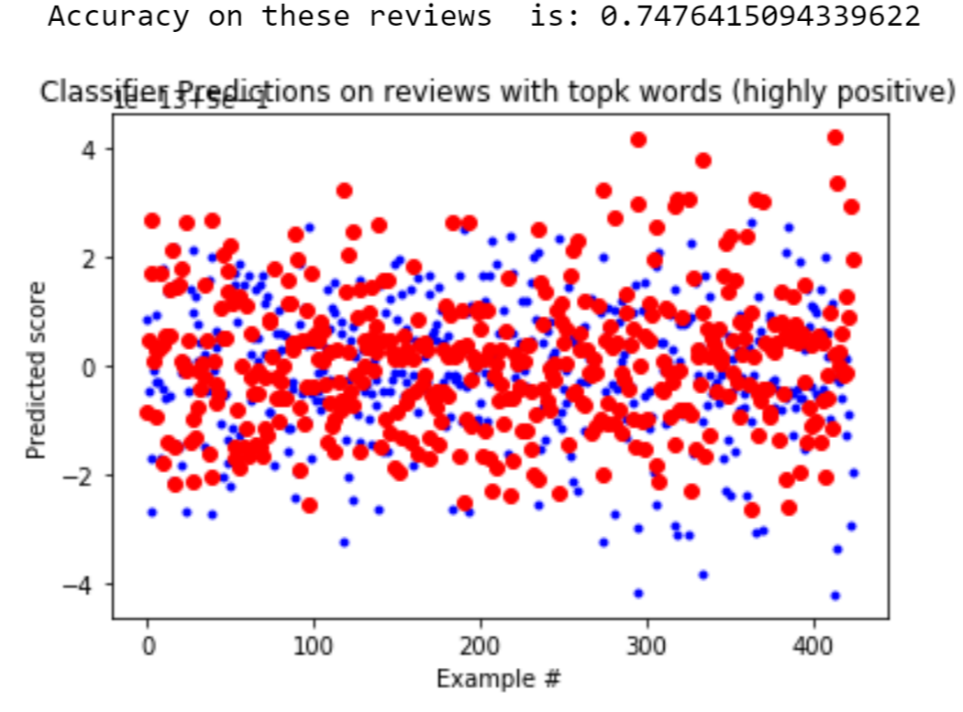
This approach however doesn’t perform well with this dataset as could be seen in the figure.

Figure Accuracy on top 8 positive words containing reviews using Lemmatization

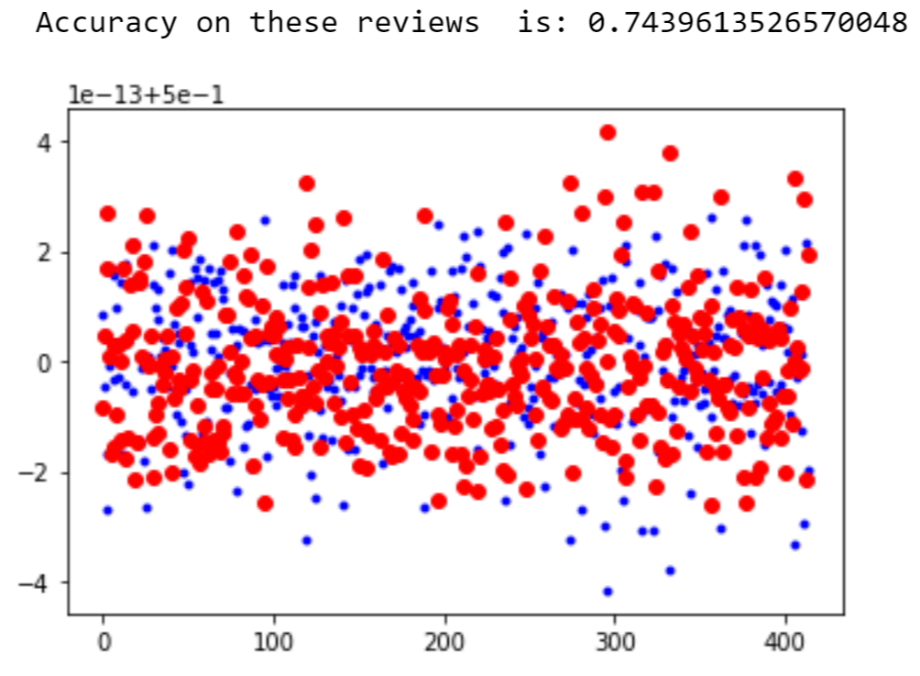


Figure Accuracy on top 8 negative words containing reviews using Lemmatization

1. **Choice of hyper-parameters**

For L1-regularization, C is the hyperparameter which is actually the inverse of regularization parameter 𝞴. We did a number of experiments based on the value of C. Below table shows how accuracy varies with the value of C. This experiment was done in order to find out the scale of C, which should be suitable for our task.

Clearly from the table, the scale of C should be in between 1-1000, thus we **choose C by cross-validation**. We have used 10 fold cross validation using sklearn.

Best value of C that we obtain from cross-validation is approximately 10.

|  |  |  |
| --- | --- | --- |
| Value of C | Accuracy on top 8 positive words | Accuracy on top 8 negative words |
| 1 | 0.808 | 0.796875 |
| 10 | 0.82857 | 1.0 |
| 100 | 0.87 | 0.759 |
| 1000 | 0.88235 | 0.78 |
| 10000 | 1.0 | 0.79255 |

Figure Variation of accuracy with the value of C

When C becomes very large ~10000, the features extracted by the classifier doesn’t make sense as seen from the figure below.

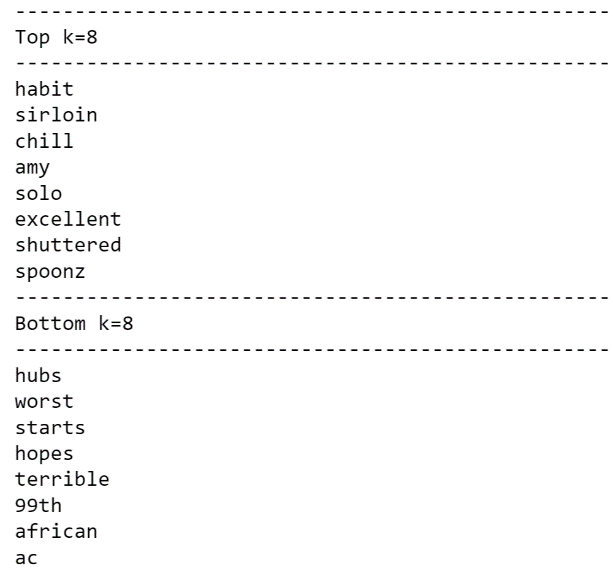


Figure Top 8 positive and bottom 8 negative words when C is very large

1. **Prediction results on Kaggle**

The results of the improved model are uploaded on Kaggle. Obtained accuracy is 78.355%.

1. **Semi-supervised**
2. **Exploiting Unlabeled Data**
3. **Comparison with the supervised classifier**
4. **Approach for utilizing unlabeled data**

Following approach is taken to utilize the unlabeled data:

1. Percentage of unlabeled data taken is varied from 0% (only using labeled data) to 100%.
2. The classifier trained on the current labeled data is used for predictions on the unlabeled data.
3. From the unlabeled data, the review corresponding to the most confident prediction (using sklearn log\_proba) is added to the labeled data, and is removed from the unlabeled data.
4. We repeat steps 2-3 for a fixed number of iterations (1000).
5. **Quantitative analysis of the effect of size of labeled data**

Figure Effect of size of labeled data

1. **Identify features whose weight changes significantly and hypothesize the reason(s)**
2. **Comparison between supervised and semi-supervised classifiers**
3. **Error analysis on semi-supervised classifier**
4. **Semi-supervised prediction results on Kaggle**

The semi-supervised prediction results have been uploaded on Kaggle. Obtained accuracy is 0.76632.

1. **Designing better features**
2. **Approach for designing better features**

We can design better features using pre-trained embeddings such as FastText.

**FastText**: This is an extension to Word2Vec.

* Instead of feeding the individual words to the neural net, FastText breaks words into several n-grams(sub-words). For example, the trigrams for the word apple is app, ppl, and ple.
* The word embedding vector for apple is the sum of all these n-grams.
* After training the neural network, we have word embeddings for all the n-grams given the training set.
* In this approach, rare words can be properly represented as it is highly likely that some of their n-grams also appears in other words.

1. **Performing relevant comparisons**

The accuracy on the dev data using pre-trained word embeddings (FastText) is better as compared to the other techniques that we’ve used in this assignment.