Midterm report

Nancy Gu and Sophia Zhu

October 26, 2017

1 Dataset

The dataset contains 1600000 tweets in train.txt and there are 800000 negative tweets and 800000 negative tweets. For easier computation, we split them into pos_train.txt file and neg_train.txt file. For each example in the training set, there are 5 features: id, date (e.g. Mon Apr 06 22:22:45 PDT 2009), querying, author, and tweet string. After processing the data, we decided that the id of the tweet gives the relative order of the tweet which can be derived from the date so this feature can be discarded. We also decided to ignore the querying since every example has "NO QUERY" for the querying column. We are going to apply feature engineering to date, author, and tweet content to see how they correlate with tweet polarity (0 = negative, 4 = positive).

We noticed that all dates are in the range of April 2009 - June 2009, so from the date column we derived month, day, hour, and minute features which we want to apply the regression to. Figure 1 illustrates the distribution of positive and negative comments by these features.

2 Some examples to get started

To take consideration of the author name (type string), we decided to hash the string using the standard string hash function provided by Python. There are 376569 authors in the positive training set, and 415671 in the negative training set, with an overlap of 132465 authors. The hashed author names has range -9223333653995923800 to 9223353268156471434.

The positive and negative training sets share 168280 common words in their tweet string vocabularies. We apply feature engineering to the tweet string by building ngram models on the training set and computing the negative and positive perplexities for the evaluation tweet string. Therefore positive and negative perplexities are two new features. For details on ngram models, refer to the next section. Figure 2 contains statistics on the tweet strings.

Figure 3 and 4 illustrate the most common words in the positive and negative training sets.

There is no missing or corrupted data since all date dates, author names, and tweet strings have reasonable length and appropriate distribution. Although it is possible for users to misspell words, we do not consider that as data corruption but simply as part of the natural language.

3 Linear and Nonlinear Regression Model

The features we are considering are: month of the tweet, day of the tweet, hour of the tweet, minute of the tweet, hashed author number, positive perplexity, and negative perplexity. We plan to fit both linear and nonlinear models (monomial with degree up to 3) to these features. To prevent overfitting, we plan to split our training dataset into training and evaluation set. In order to do that, a technique we will use is cross validation (we plan to use 5-fold cross validation). Therefore even if a model is doing very well on the training set, it might not do as well on the evaluation set, and therefore we would not use the model on the testing set. Also, since cross validation allows the model to train on separate parts of the training set, we avoid the situation that we will have too few data to work with, thus reduce the probability of underfitting. In addition, since ngram models are known to do reasonably well (above 50%) on sentiment analysis tasks, combining with other features should only make the accuracy better because if other features reduce the accuracy then they would have close to zero weight when we perform regression. Therefore we know the model will do better than a baseline model which assigns random polarity to a tweet string (should have around 50% accuracy) and should not be classified as "underfitting." Also, since our dataset is fairly large, underfitting should not become a problem in this case. The evaluation of the model is straightforward - the percentage of the polarity it gets right. Before running the model against the testing set, we would also evaluate the model against the evaluation set to find the model in our hypothesis set that performs the best (percentage of the polarity it gets right in the evaluation set), and our final accuracy will be the the model's accuracy on the testing set.

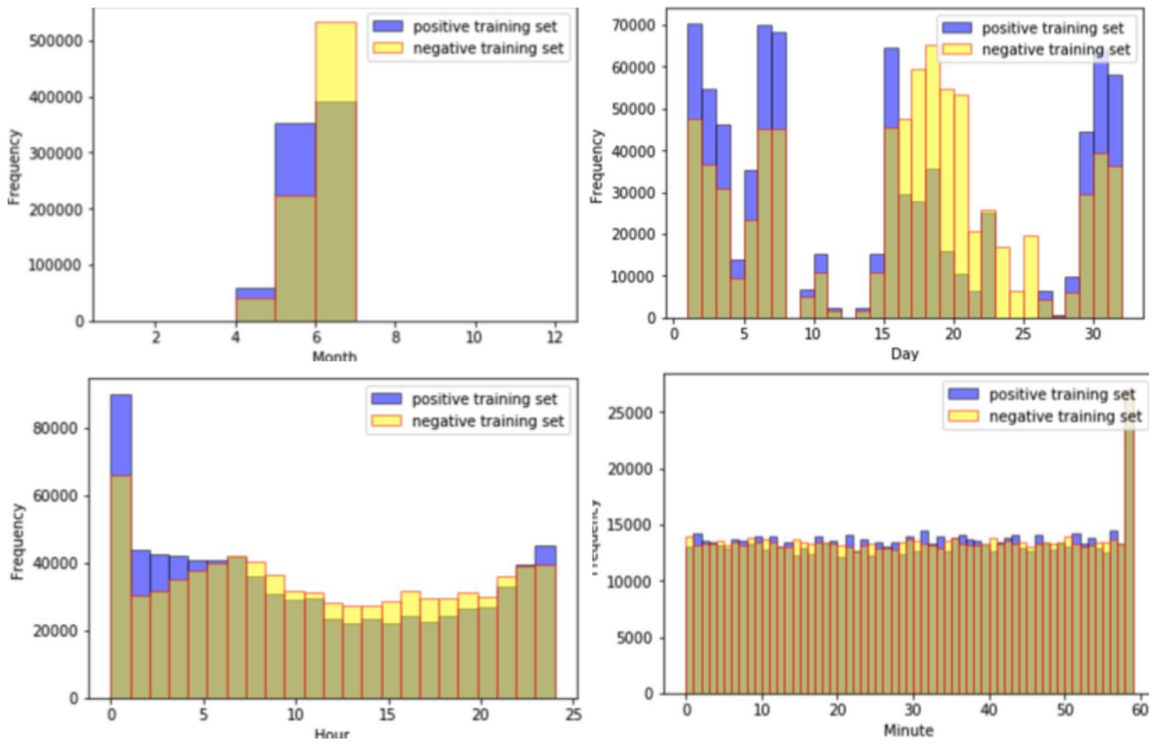
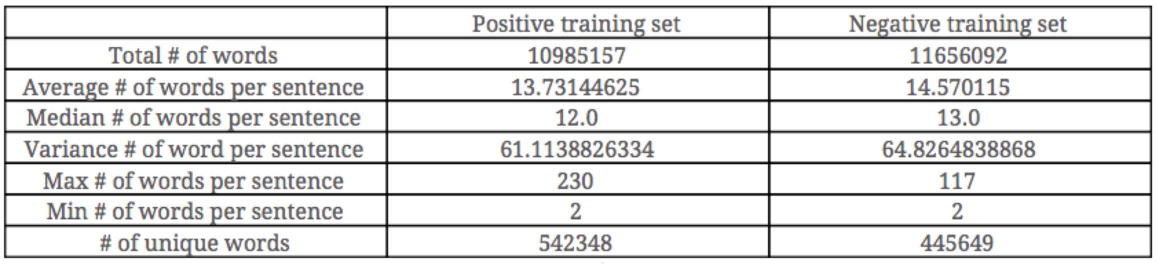


Figure 1: Distribution of positive and negative comments



	Positive training set	Negative training set
Total # of words	10985157	11656092
Average # of words per sentence	13.73144625	14.570115
Median # of words per sentence	12.0	13.0
Variance # of word per sentence	61.1138826334	64.8264838868
Max # of words per sentence	230	117
Min # of words per sentence	2	2
# of unique words	542348	445649

Figure 2: Summary statistics for tweet strings

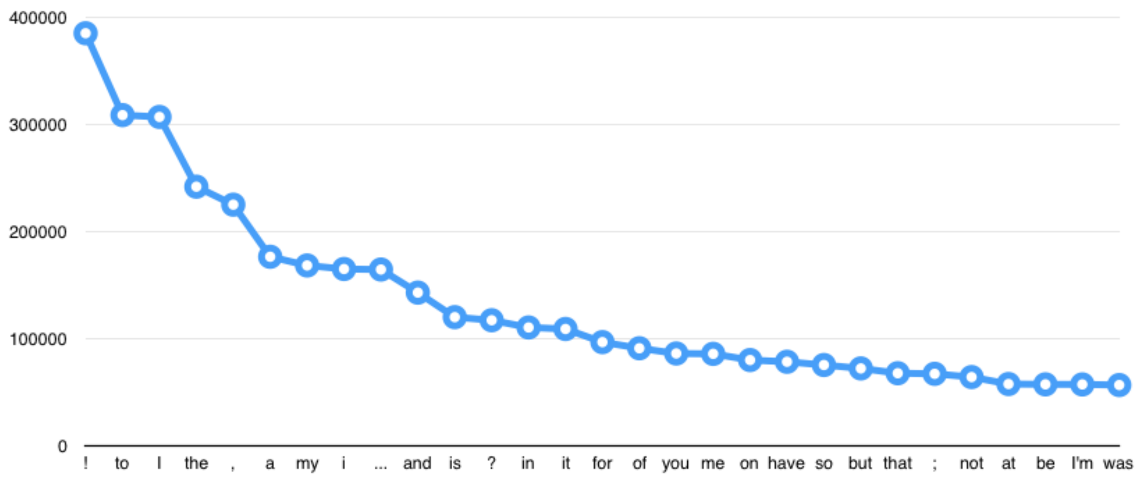


Figure 3: The 30 most frequent word in negative tweets

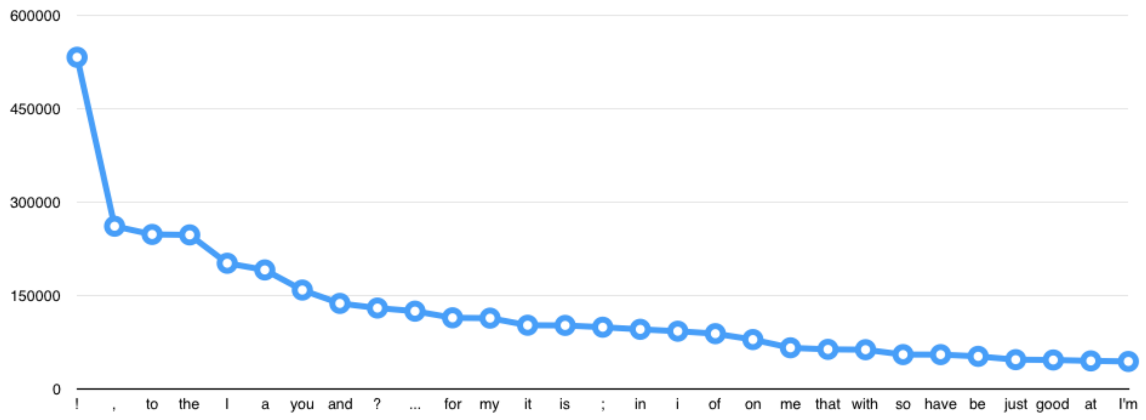


Figure 4: The 30 most frequent word in positive tweets

4 N-gram Model

In n-gram model, we first add start and end token to each tweet. For the unigram model, we count the number of occurrence of each word in the training set and store it in unigram model. For the bigram model, we count the number of occurrence of each (previous word, current word) pair in the training set and store it in bigram model. There is very likely to be words that are not in the training set during testing, so it is necessary to calculate a number of occurrence in the n-gram model for unknown words. In dealing with unknown words, we set all the words with count 1 as '<unknown>' in the unigram model and record the count of <unknown>'s in the unigram model multiplied by a certain offset since an unknown word should not be counted as much as a regular word in our opinion. For bigram, if the previous word does not exist in the unigram model, it is set to <unknown>, if the current word does not exist in the unigram model, it is set to <unknown>. Then we add these new word pairs(for example: (<unknown>, apple) into the bigram model. Because we use probabilities in the evaluation stage, we also need to smooth our model with parameter k in order to get good probability that can be used in multiplication. For smoothing in unigram, the smoothed value for a word A is (the count of A in unigram model + k) divided by (sum of all counts in unigram model + (the number of unique words in unigram model*k)). For bigram model, the smoothed value for a word pair (B,C) is (the count of (B,C) in bigram model + k) divided by (the count of B in unigram model + (the number of unique words in unigram model * kh)). In order to decide how much smoothing we would like to apply and what the offset is when we treat unknown on the ngram model we perform hyperparameter search on these two values. We will train ngram models on the training set (with cross validation) with different k smoothing values and different offsets, and evaluate these ngram models on the evaluation set (since ngram itself can be used to classify polarity of the tweets already) by comparing the positive and negative perplexity on the evaluation set. In general, positive perplexity represents how likely a tweet has positive polarity, and negative perplexity represents how likely a tweet has negative polarity. We choose the k value and offset that gives the best classification result (best accuracy).

5 Analysis Performed

We know that ngram model itself can be used on sentiment analysis tasks, therefore we built a simple bigram model on the training set with 5-fold cross validation technique. We simply used 1 as our offset and 0.01 for k smoothing, and we achieved on average 68% accuracy on the evaluation set.

6 Work To Be Done

First, we will need to perform the hyperparameter search in order to find the best offset and k smoothing value for the model (the source code for this part has been completed, but running the algorithm takes a fairly long time due to the size of the dataset). Second, we will need to fit the linear and nonlinear models described above combining ngram perplexities and other features. Third, we will need to save the best model and test for the performance on the testing set.