Sentiment Analysis on Tweets

Yu (Nancy) Gu, Jiangyue (Sophia) Zhu
yg234, jz552

Introduction

As social media become an important part of human life, there is a copious amount of information people can find from tweets, online reviews, blogs, and other channels. Although this information helps us gain knowledge in fields we are unfamiliar with and express our opinions openly, it also creates the problem of information overload, and the majority of us cannot filter out useful information among the gigantic sources of data. We want to explore the problem of sentiment analysis, analyzing people's attitudes towards a certain subject by identifying and extracting information from text pieces. This is an important problem because by automatically gathering information from people's written pieces with high accuracy, it can save a lot of manual work and effectively filter out information we want to see. In this project we are going to use people's tweets as our dataset since tweets are a form of informal communication containing vast amount of sass and idiom, and are the most similar to how people communicate in daily life. Details about our dataset can be found in the next section. Since the dataset is fairly big and in detail and is closely related to a relaxed form of communication, we believe that it can facilitate our initial analysis on humans' choice of words upon expressing positive or negative opinions. This project is has a lot of potential applications. For example, we can train our model to identify a majority of internet users' attitudes towards a certain subject such as "Donald Trump". It can also be trained to react to negative reviews on the Internet for users to be alerted for certain products.

Background

Historically sentiment analysis has been done through probabilistic language models, which assigns to a sequence of words a probability of having positive or negative sentiment. Given a large enough training corpus, we can train the model to compute the probability of each sentence appearing in the positive and negative training set. However, as there are many combinations of sentence structures, it is impossible to compute the exact probability for sentences that haven't been seen by the model before. Ngram model is a particularly effective language model based on Markov assumption - probability of some future event (next word) depends only on a limited history of preceding events. Therefore the probability of seeing the next word in the positive or negative training corpus depends only on the n words preceding it. In this project, we are using bigram model, which generates the likelihood of a sentence by assuming that the probability of the next word only depends on the preceding one word. Given a sentence $w_1 w_2 ... w_n$ where each w_k represents a word, bigram approximates

$$P(w_1 w_2 ... w_k) \approx P(w_n|w_{n-1}) \times P(w_{n-1}|w_{n-2}) \times ... \times P(w_2|w_1) .$$

We choose bigram because it has been proven to be more accurate than unigram, trigram, and other n-gram models in most contexts. We trained a positive bigram model on a positive training corpus including 800000 tweets that are classified as having positive sentiment, and a negative bigram model on a negative training corpus including 800000 tweets that are classified as having negative sentiment. In traditional bigram model, given a sentence in the test set, we will compute the probability of it being positive using the positive training model and the probability of it being negative using the negative training model, therefore obtaining the probabilities of it having positive and negative sentiment. Then the two probabilities will be compared to see if the sentence is more aligned with the positive or negative training corpus to determine its sentiment.

However, in this project, we want to use the combination of the result from bigram model

with information such as when the tweet is created and who the author is to have a more accurate classification because these features are also considered predictive factors for determining if a sentence has positive or negative sentiment. For example, if an author always produce negative posts, it is likely that a post by him or her in the testing set has negative sentiment. Therefore we want to fit different regression models on attributes such as author, date, probability that the tweet has positive sentiment from positive bigram model, and probability that the tweet has negative sentiment from negative bigram model. However, instead of using the probability above, we are going to use the negative log likelihood $-\sum_{i=1}^{n} \log P(w_i|w_{i-1})$ since minimizing the negative log likelihood is equivalent to maximizing the probability, and summation of the log probabilities will avoid precision error produced by multiplication of float numbers. Please refer to the reference for links to papers on ngram models.

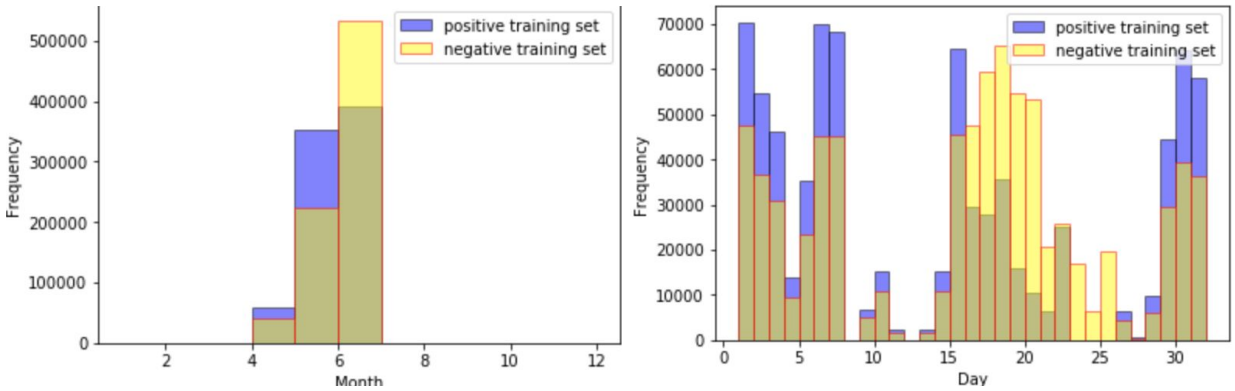
Dataset

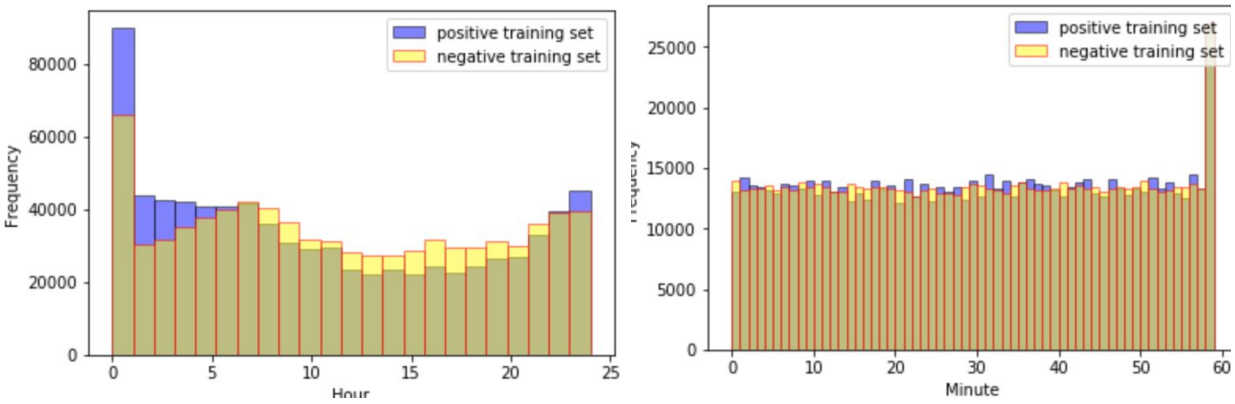
The training dataset contains 1600000 tweets, including 800000 positive tweets and 800000 negative tweets. The testing set involves 359 tweets, including 182 positive tweets and 177 negative tweets. For each tweet in the training and testing set, there are 5 features: id, date (e.g. Mon Apr 06 22:22:45 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 (first technique learned in class) to date, author, and tweet string to see how they correlate with tweet polarity (-1 = negative, 1 = positive).

Feature engineering: Date

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. The following graphs illustrate the distribution of positive and negative comments by these features.

Therefore we used the function $\psi('yyyy-mm-dd\ HH:MM:SS') \rightarrow (mm, dd, HH, MM)$ and generated four columns for our input. Note that year does not matter since all training / testing data are produced from the same year. We also think seconds of time does not produce a significant result logically. For example, it is possible that an author’s sentiment varies by the time of the day (e.g. a person is more likely to make tweets with positive sentiment after dinner), but it is very unlikely that sentiment will correlate to seconds of time.





Feature engineering: Author

To take consideration of the author name (type string), we decided to represent each author by the number of positive tweets and the number of negative tweets he or she has produced. There are 376569 authors in the positive training set, and 415671 in the negative training set, with an overlap of 132465 authors. For those authors that do not exist in both set, we decided that it is unfair to simply assign zero to the number of positive / negative tweets he or she produced. For example, since our training set is randomly sampled, it is possible that we record many positive tweets from an author but no negative tweets from him or her at all, and this person could have made more negative tweets than positive tweets. Instead of filling in zero for number of negative tweets this author wrote, we will calculate the average number of negative tweets made per author and assign this value to fill in the missing data.

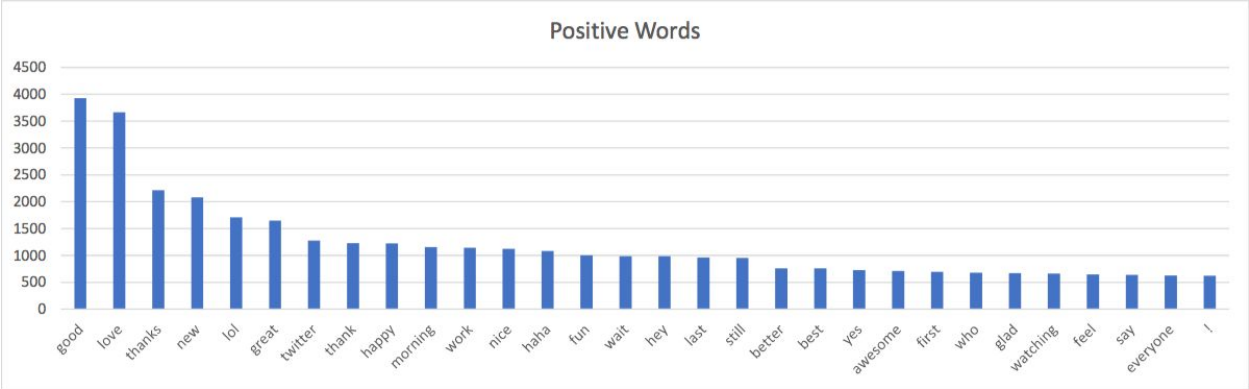
Feature engineering: Tweet String

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 log likelihood for the evaluation tweet string according to the algorithm described above. The following table contains statistics on the tweets.

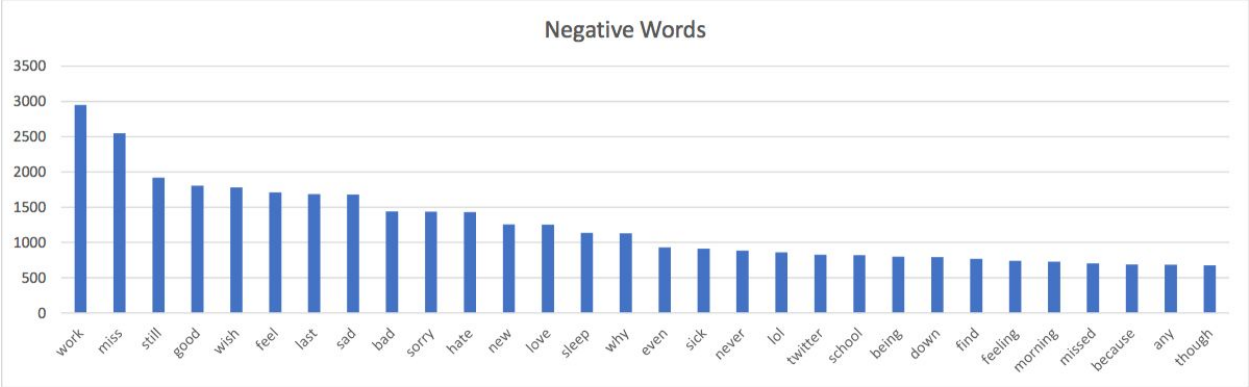
	Positive training set	Negative training set
Total # of words	10985157	11656092
Average # of words per sentence	13.73144625	14.570115
Median # of words per sentence	12.0	13.0
Variance # of word per sentence	61.1138826334	64.8264838868
Max # of words per sentence	230	117
Min # of words per sentence	2	2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

Summary statistics for tweet strings

The following graphs illustrate the most common words in the positive and negative training sets after excluding the commonly used words such as “I”, “the”, etc (which actually appear the most number of times in both positive and negative training set). Although there are some words that appear in both sets, we can definitely see that words with positive connotation appear more frequently in the positive training set than negative training set, and words with negative connotation appear more frequently in the negative training set than positive training set. Also, note that the probabilities for these words themselves constitute a unigram model, which is used for computations in bigram models.



The 30 most frequent word in positive tweets



The 30 most frequent word in negative tweets

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 misspelling as data corruption but simply as part of the natural language that our algorithm has to learn from.

Algorithm

Ngram model

There remains a problem with bigram model - if there is a new word in the testing set, the probability of seeing this word given the previous will be zero, and the negative log likelihood will approach negative infinity. To take consideration of new words, for each word in the training set that appears only once, we change them into a default <unknown> label, so that in our training set we will have some probability for the unknown words. We applied add k smoothing when we calculate the probabilities of a word appearing after a preceding word to further restrain the unknown probabilities. Also, the unknown words should not be counted equal weight with other words, therefore we multiple the number of <unknown> labels by a weight unknown_weight when calculating probabilities. In order to find the best k value and unknown_weight we used techniques such as hyperparameter searching and cross validation (second technique learned in class). We used five fold cross validation to search for the best configuration with unknown weight ranging from 0.1 to 1 with 0.1 step and k value ranging from 0.01 to 1 with 0.01 step. The evaluation technique here is to simply compare the probability obtained from positive model and negative model, make a prediction based on which one is higher, and count the number of correct predictions. The resulting best parameters are k = 0.58 and unknown_weight = 0.6. With these

parameters we are able to achieve a prediction accuracy of 66.81% without applying any regression model on author or date.

Regression Models

The input to a few regression models are 1600000 by 9 matrix such that the columns are month, day, hour, minute, number of positive posts made by author, number of negative posts made by author, negative log likelihood of positive sentiment, negative log likelihood of negative sentiment, and offset. Surprising, after performing a few trials, we realized that all the date columns have insignificant weights. In fact, by having them we are actually lowering our performance on our validation set. Therefore we reformed the input matrix to by 1600000 by 5 matrix, including number of positive posts made by author, number of negative posts made by author, negative log likelihood of positive sentiment, negative log likelihood of negative sentiment, and offset. The prediction we want to make are simply -1 or 1 with -1 being negative sentiment and 1 being positive sentiment.

The first model we used was Ordinary Least Squares (quadratic loss with no regularization) to fit our data (third technique learned in class). We chose this regression because it is scale invariant and easy to compute with QR factorization. Then we also tried several other models, namely, Ridge regression, Lasso regression, L1 regression, Huber regression, and Logistic regression. We chose Ridge regression and Lasso regression because it's possible that regularizers might help with the performance in our case since quadratic loss tends to penalize the outliers significantly, and regularizers might be able to reduce that loss. We also tried L1 regression, Huber regression, and Logistic regression for the same reason - to avoid the significant penalties on the outliers. For the later regression, we used proximal gradient method (fourth technique learned in class) with 600 max iterations.

To prevent overfitting, we split our training dataset into training and evaluation set by 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.

Results

For each model described above, we computed the test set error rate by finding both the number of correct predictions and the l2 norm error (squared loss) between the prediction and the correct labels. The following table shows the number of correct predictions and squared error for each regression.

Model	Accuracy	L2 norm error
Ordinary Least Squares	69.1%	339
Ridge Regression	69.1%	339
Lasso Regression	69.1%	339

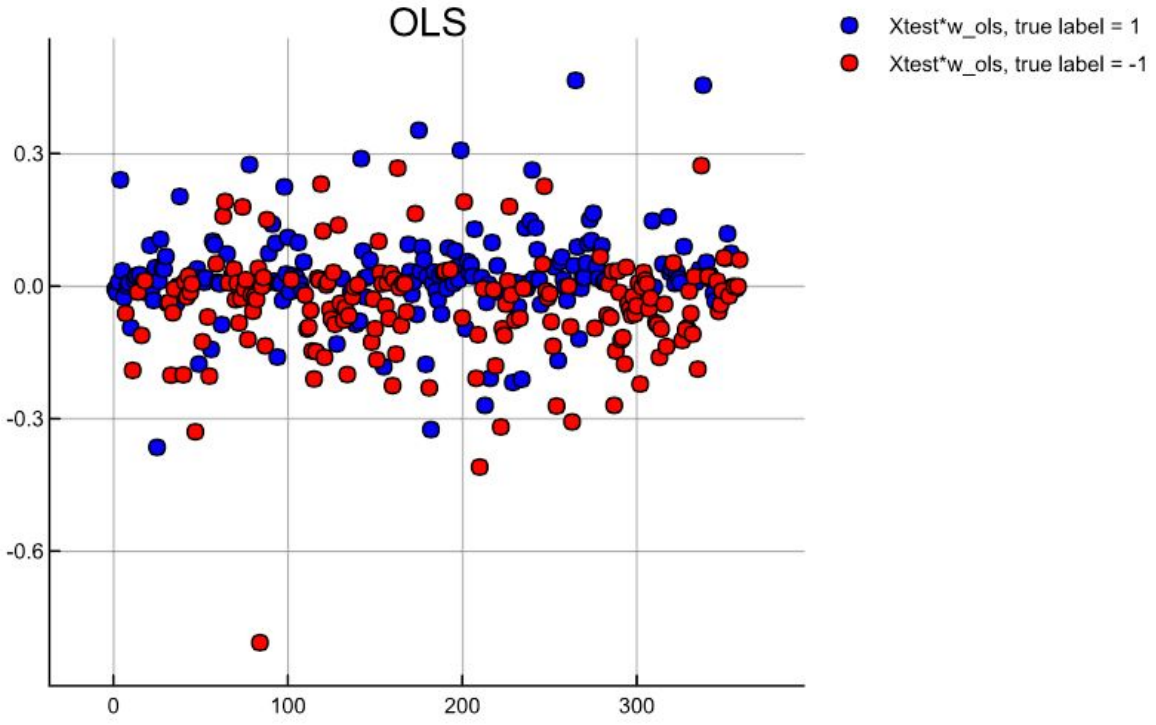
L1 regression	51.0%	363
Huber regression	49.3%	491
Logistic regression	69.1%	339

Accuracy and square loss for all regression models

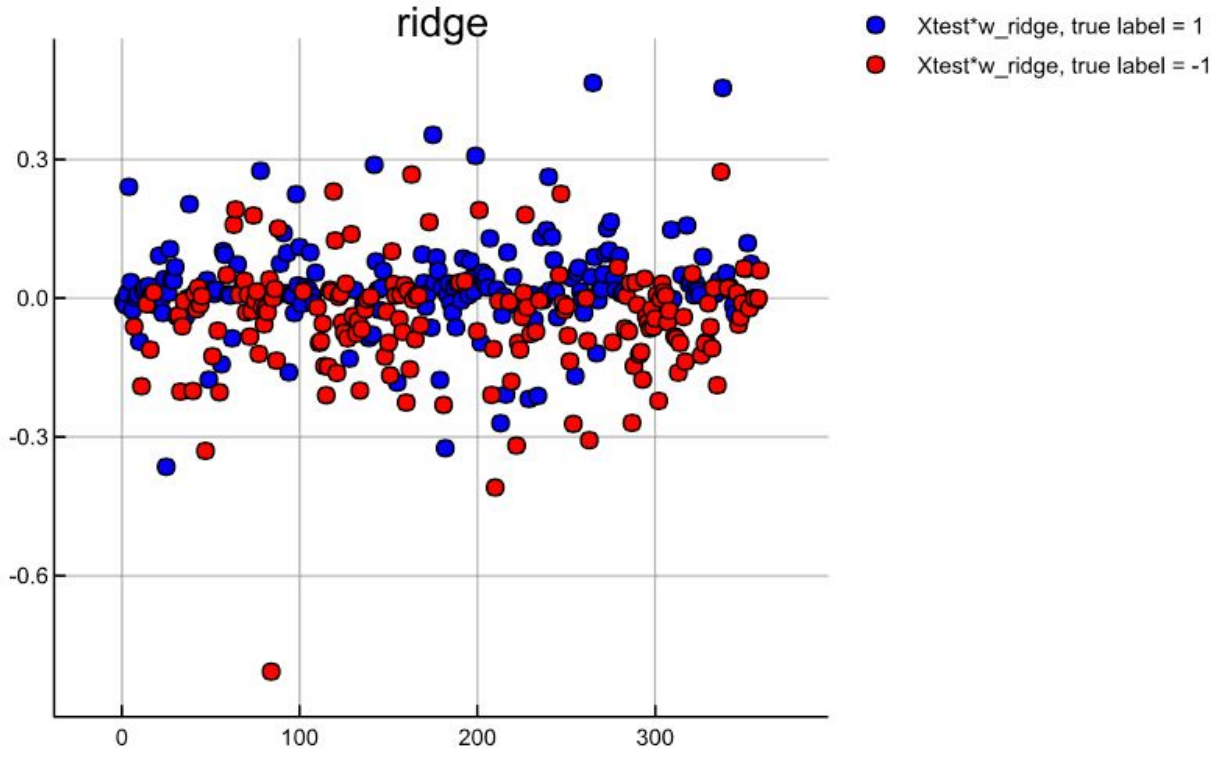
All the least squares models returns the same accuracy despite having different regularization. This is probably due to small w vector in the ordinary least squares result, In the w vector, the largest absolute value is 0.0208 and the smallest is 0.0206. So even with regularization, the regularizer is likely to have little effect on w because w is already small. In fact, the change in w vector after using regularizers are only in the sixth decimal point(0.000001). The fact that w is very small also shows that probably the columns we have is not enough to get a good prediction (at least when using the least squares models) and that we probably need more complicated methods such as word embeddings using neural networks to get a better model. However, something we observe is that both log likelihood for negative tweets and negative number of posts the author tweeted have negative weights, and both log likelihood for positive tweets and positive number of posts the author tweeted have positive weights. Therefore from the weight we can deduce that the features are indeed working as expected, with negative features contributing negatively to regression results, and positive features contributing positively to regression results.

Also, l1 regression and huber regression are not very good for classifying this data. This is probably due to the small values in w and the fact that l1 and huber regression favor sparse w solutions. L1 regression generated w vector that has one values(out of five) that has absolute value smaller than 0.001, and one value with absolute value smaller than 0.01. For huber regression, four out of five values with absolute value smaller than 0.002.

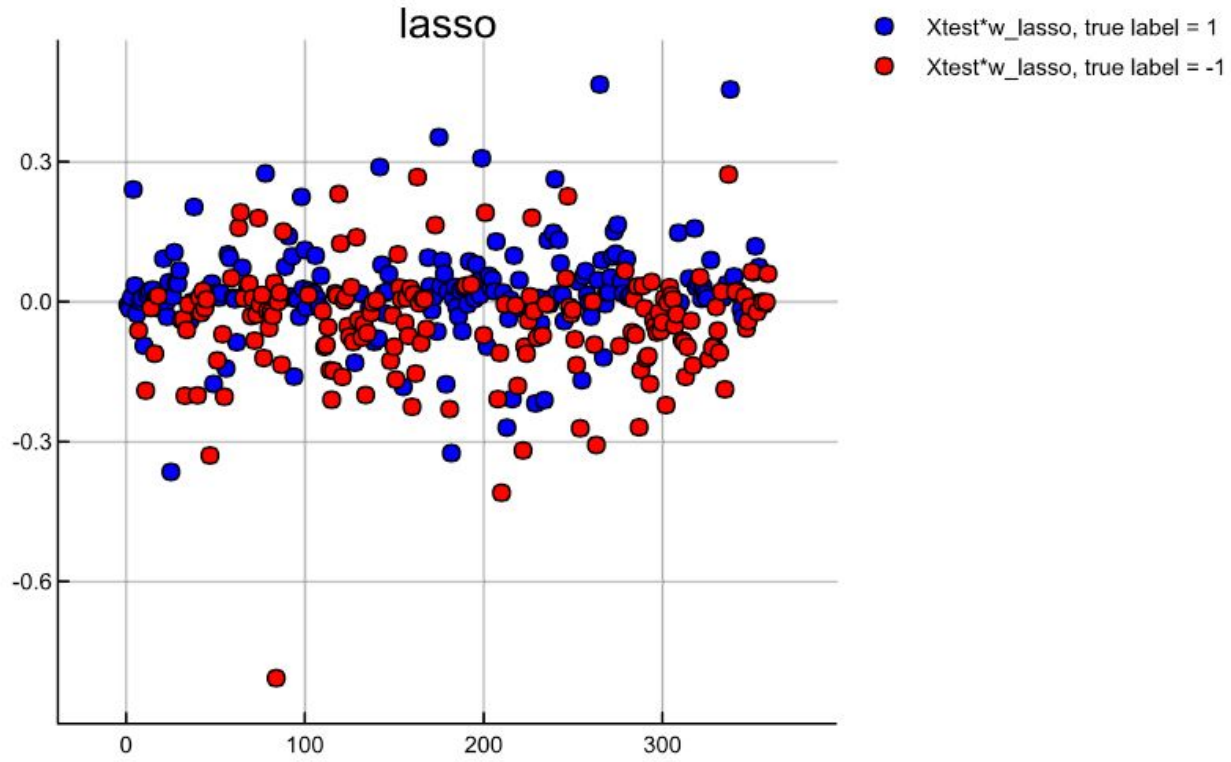
Among all of these models, the most accurate ones are the least squares models and the logistic regression, producing a close to 70% accuracy. The following graphs show prediction results of all regression models (before taking the sign of the result for prediction).



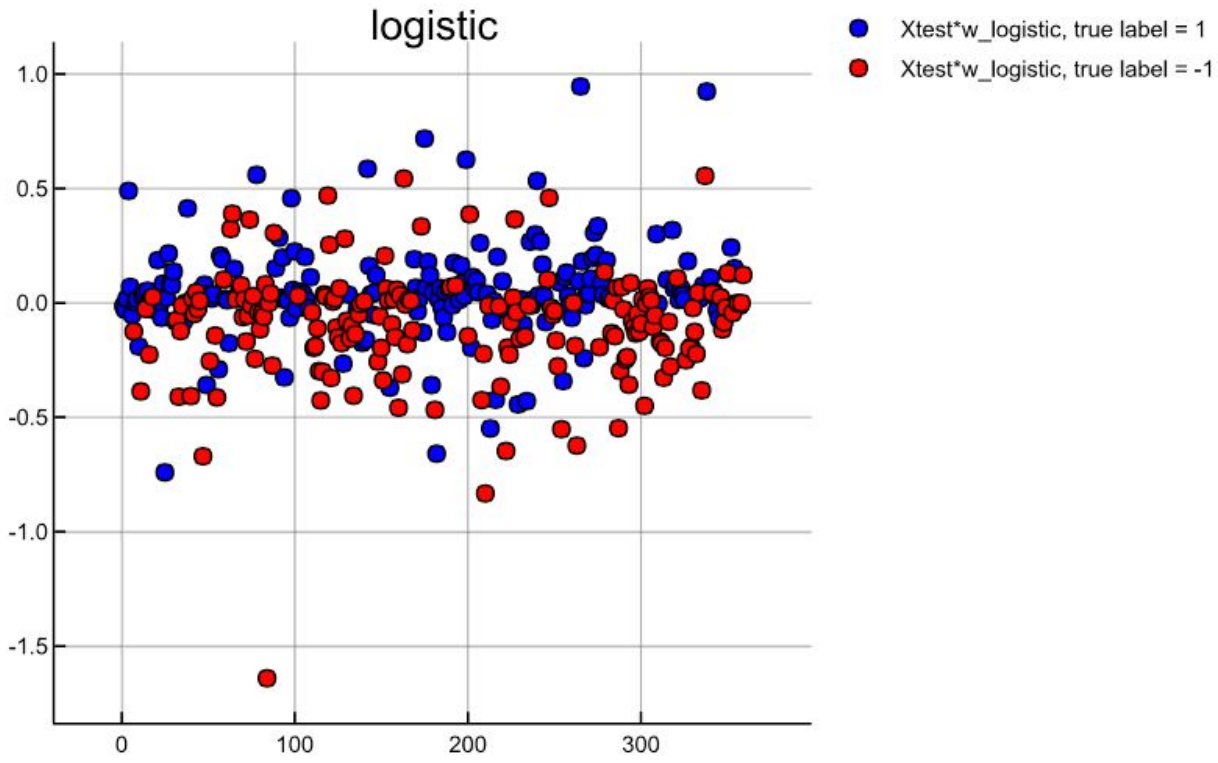
Regression result for Ordinary Least Square



Regression result for Ridge regression



Regression result for Lasso regression



Regression result for Logistic regression

Analysis

In our result, the best accuracy we achieved is about 69%. We are fairly confident in our results since we are close to 20% above a baseline model (randomly guess a tweet's polarity). Also we trained on many regressions with different loss functions and regularizers, and we eventually reach our conclusion that our model performs reasonably well. Compared to the industry level accuracy on sentiment analysis in general, which is about 80%, our current model seems a little underperforming. However, the industrial results are usually based on a more formal context, such as Wikipedia pages and online reviews. Our result is unique because it is based on tweets, a form of communication that is extremely concise and casual, which is also more prone to idiom and issues such as misspelling. Therefore we think our results obtained did pretty well. Also, the current industrial models in use are mostly based on neural networks. Since we have not learned about neural network in this class, we are not able to produce accuracy that matches the industry level. However, research has shown that with simple ngram predictions, the accuracies are in general between 70 and 75%, therefore our model performs reasonably well compared to industrial standards. The benefit of using our model for the company is that it is fairly simple to use and maintain, and it is very fast compared to the complicated multi-layer neural networks algorithm used in industry. We will be willing to use our algorithm in production for tasks like identifying users' sentiment in online review or feedback for a certain product (if users are providing positive or negative feedbacks) when computation resources are limited. However, this model needs to be used with precaution since the accuracy still requires improvement, but we think it could be a good substitute for manually checking for positive and negative online reviews for its efficiency, fast-computation, and minimal utilization of resources.

Reference

Speech and Language Processing, Chapter 4 Daniel Jurafsky & James H. Martin.