Movie Review Classification: Comparing Logistic Regression, Support Vector Machines, and Linear Discriminant Analysis.

Michal Golovanevsky
mgolovan@calpoly.edu

Jenna Landy
jlandy@calpoly.edu

Andrew Sulistio
apsulist@calpoly.edu

Abstract: *This project is two fold. Looking at a dataset of movie reviews, we first aim to develop a feature set that performs better than the bag of words provided to us. Second, we implement three linear classifiers and compare their performances on (1) our feature set, (2) the provided bag of words, and (3) how the accuracies of the classifiers compare between the datasets. Our final classifiers performed at about 70%-75% accuracy, compared to around 84% accuracy for the given features. However, our features reduced dimensionality significantly, with our dimensions being 0.01% the size of the original features. LDA and Logistic Regression performed very similarly, but LDA has the advantage of having a closed form solution.*

Key words: *logistic regression, linear discriminant analysis, support vector machine, IMDb movie reviews*

Introduction

This project is two fold. Looking at a dataset of movie reviews, we first aim to develop a feature set that performs better than the bag of words provided to us. Second, we implement three linear classifiers and compare their performances on (1) our feature set, (2) the provided bag of words, and (3) how the accuracies of the classifiers compare between the datasets.

Feature Set Creation

Structuring Given Files

The data provided was given in the `reviews` directory, with individual training set reviews in text files within a `train` directory in either `pos` or `neg` (and same for `test`). The first step in data processing was to extract the text from each file, record the sentiment (0 for negative, 1 for positive), and save all the data in full train and test DataFrames.

Feature Engineering

Once we structured the dataset, we engineered features from the raw text. Our original idea here was that we, as humans, know the type of language that will be used in positive or negative reviews (e.g. we know that the word "terrible" is more likely to be in a negative review and "amazing" is more likely to be in a positive one). The goal was to extract *sentiment*, not *context*, so we capitalized on this real-world knowledge and focused on strategic words rather than any word in the review.

This led us to find a dataset of positive and negative words, an *opinion lexicon*, created by Bing Liu and Minqing Hu at the University of Illinois at Chicago[1]. For each review we calculated the proportion of words that fell under a "positive" or "negative" list, reversing their weight if a negation word existed within the sentence.

	positive_words	negative_words
positive	0.044095	0.032046
negative	0.035301	0.041091
difference	0.008794	-0.009045

Figure 1. Average frequency of positive and negative words occurrence per document in train set.

With a similar motive to looking at positive and negative words, we also extracted positive and negative emoticons from the reviews. We accessed a list of emoticons and their meanings from Wikipedia. Then we calculated the frequency that both positive and negative emoticons appeared per document.

	positive_emoticons	negative_emoticons
positive	0.02808	0.00552
negative	0.01848	0.00568
difference	0.00960	-0.00016

Figure 2. Average frequency of positive and negative emoticon occurrence per document in train set.

While the positive and negative symbols showed how much the two symbol categories (words and emoticons) affected the document, we wanted to capture a document polarity score as well. By using the Flair[6] library, we were able to extract several types of word embeddings for each document. While we tried several word embeddings, such as ELMo[3], BERT[4], RoBERTa[5], and Flair[6] which takes in the context of the word into account, we ultimately settled on GloVe[7] which did not take context into account due to processing times and better accuracies. To generate the polarity scores we first ran SVM on our list of positive and negative words in order to extract the **w** vector, then by projecting the average of a document's positive and negative GloVe embeddings onto **w**, we were able to obtain a polarity score. Below is the pseudocode used to obtain polarity scores for all documents:

- $\forall$ x $\in$ {Positive word list Negative word list}
  - **x** = word vector of x
  - Add **x** to set X
- Perform SVM on X to determine the **w**, which best separates {**x** | **x** is positive word} from {**x** | **x** is a negative word}
- For D in documents
  - Document embedding DE = 0
  - For **x** $\in$ D such that **x** $\in$ {Positive word list $\cup$ Negative word list}
    - DE = DE + **x**
  - ADE = Average document embedding
  - Project ADE onto **w** to obtain polarity score.

Figure 3 shows the difference in polarity scores between positive and negative reviews.

	polarity
positive	0.951965
negative	0.183226
difference	0.768739

Figure 3. Polarity of documents in train set

We also added some generic natural language processing features to see if these were significantly different across positive and negative reviews. We extracted number of sentences and the average number of words per sentence. Figure 1 shows that in the training data, positive reviews have, on average, slightly fewer sentences but more words per sentence.

sentiment	num_sentences	words_per_sentence
0	12.74440	22.671113
1	12.16056	24.335304

Figure 4. NLP features by sentiment in train set.

Finally, we added two features to represent the text complexity. We looked at average word length as well as the Dale-Chall readability score, calculated using the py-readability-metrics package[8]. The Dale-Chall readability score estimates the reading level of a body of text, using a fourth grade vocabulary as a baseline. The idea behind these features was that perhaps a more angry or a more happy individual will write at a higher reading level or with more complex words, so these features may indicate the sentiment of a movie review. Figure 2 shows that in the training data, positive reviews have, on average, slightly longer words and a slightly higher reading level.

sentiment	avg_word_length	dale_chall_readability
0	3.963219	8.418689
1	4.017956	8.609225

Figure 5. Text complexity features by sentiment in train set.

Our final feature set has 9 features which we will compare to the given data from the Stanford research paper. The final predictors in our data set are:

- Positive Words
- Negative Words
- Positive Emoticons
- Negative Emoticons
- Polarity Score
- Number of Sentences
- Average Words per Sentence
- Average Word Length
- Dale Chall Readability Score

Working With the Stanford Dataset

The stanford dataset was given in the form of a sparse matrix, and the entire vocabulary was 89,527 words. Our implementations of the linear classifiers were intended to work on dense matrices. This dataset as a dense matrix of 89,527 was not feasible from a memory standpoint or a computational power one, so we decided to reduce the number of features we used.

We looked only at the 500 most frequent words, determined by looking at the first 500 words in the vocabulary file. As a baseline to see how much accuracy we would lose, we used sklearn's implementation of logistic regression[9]. This implementation can take in sparse *or* dense matrices. Using the full sparse matrix, we saw 87.7% accuracy. With the dense matrix, using the reduced feature set of 500 words, this decreases slightly to 84.1%. These accuracies are very close, indicating that the reduced feature set is very similar to the original, so we felt comfortable continuing with this data. However, it is important to keep this feature reduction in mind when comparing our feature set to Stanford's, because it is not an exact representation of their data.

Implementations

Logistic Regression

Our logistic regression implementation employs stochastic gradient descent, taking in as parameters the learning rate and tolerance, as well as an optional maximum iterations if we desire to stop the algorithm before it converges. Our stopping condition is when the number of iterations has reached the maximum iterations or when the difference in consecutive log losses drops below the given tolerance, whichever occurs first. We classify points by calculating `p`, the probability that a review is positive, and assigning "0" (negative) if `p` is less than or equal to 0.5 and assigning "1" (positive) otherwise.

Linear Discriminant Analysis

Our linear discriminant analysis implementation uses the closed form solution, finding the vector **w** as the eigenvector that corresponds with the largest eigenvalue of $S^{-1}B$. We then classify points by projecting them onto **w** and seeing which projected mean it is closest to.

Support Vector Machine

Our support vector machine implementation solves the primal problem and uses stochastic gradient descent. Like with logistic regression, we take as parameters the learning rate and tolerance, as well as an optional maximum iterations. Before running the support vector machine we made sure to transform the response variable from 0/1 binary to -1/1 binary so the classifier would work properly.

	Classifier	Our Features	Top 500 from Bag of Words	Combined Features
0	Logistic	0.74668	0.8412*	0.87836*
1	LDA	0.74856	0.8386	0.87336
2	SVM	0.696	0.84368*	0.88192*

* using SciKitLearn

Figure 6: Final Test Accuracies

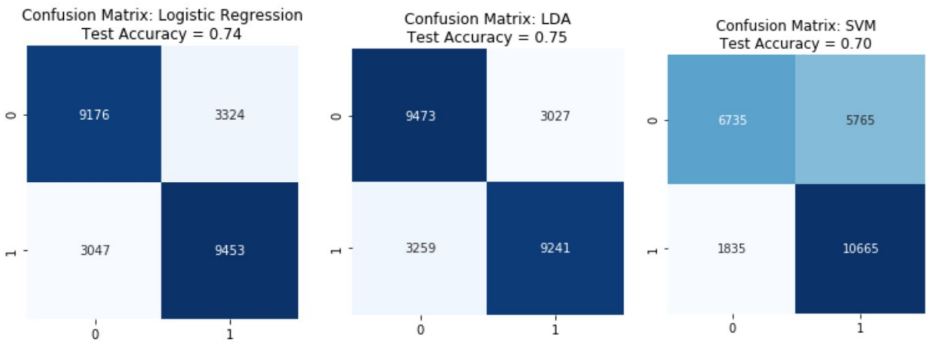

Figure 7: Confusion Matrices on Test Set of Our Features

Evaluation and Optimization

Evaluation

For all classifiers and with both datasets, we look at the accuracy of the model as the proportion of reviews in the test set that were classified correctly. We also looked at confusion matrices to see whether misclassifications were distributed equally across false negative and false positives or if our classifiers more frequently misclassified in one direction.

Optimization

Since SVM does not have a closed form solution, we had to run a grid search to optimize the combination of learning rate and the constant C. Our computing power was limited when trying to use really high values of C and really low values of so we used a somewhat narrow set of different values. We know that if data is easily linearly separable, we can use large values of C but when the data is not easily separable, smaller C values are more appropriate. Since large values of C, (>1), almost always crashed our kernel, this could be an indication that our data was not very separable. Nonetheless, it gave us good insight into what kind of values optimize SVM as shown in the heat map. The color is representing the accuracy, meaning the darker the color the higher the accuracy.

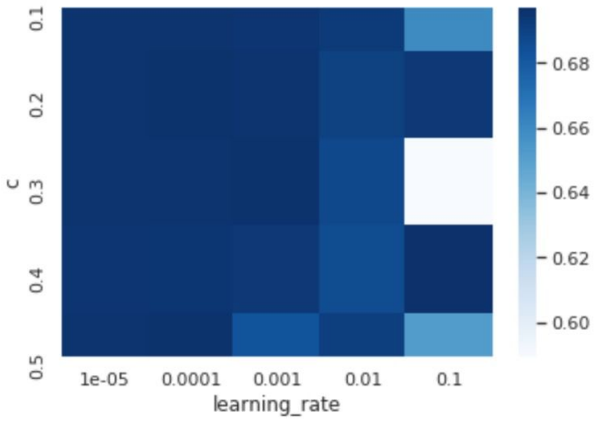

Figure 8: Optimizing C and Learning Rate for SVM

Results

Question 1: Can you extract a better feature set from the text than the creators of the dataset?

Our set of nine features performed fairly well, with between seventy and seventy-five percent accuracy for all classifiers. Figure 6 shows test accuracies for all models created. The features provided to us performed around ten percentage points better than our engineered variables for all models. Despite this, we're still proud of the accuracies we achieved because we did so with significantly fewer features (9 as opposed to 89,527).

But when combined a 500 word subset of the 89,527 features we saw a significant uptick in our accuracy scores, even when compared to only their existing features. As seen in figure 6, we consistently saw ~3-4% improvements when compared to their dataset.

Question 2. How do the three classification techniques you implemented compare?

Using our feature set, logistic regression and linear discriminant analysis performed the best with very similar accuracy scores of 0.747 and 0.749, respectively. The support vector machine classifier was about five percentage points worse with an accuracy of 0.696. The support vector machine also had very unbalanced incorrect predictions, as shown in the confusion matrix in Figure 7.

On the given dataset, all three classifiers had very similar scores around 84% accuracy. The support vector machine classifier had an accuracy of 0.844, compared to 0.839 for logistic and 0.844 for LDA.

The best classifier we created was support vector machine on a combined dataset of our nine features added to the provided set. This resulted in 88.2% correct classifications (followed by 87.8% for logistic and 87.3% for LDA). The fact that adding our features improved the given data's accuracy makes it clear that the variables we engineered are useful in predicting the sentiment of a movie review.

Conclusion

Our final classifiers performed at about 70% - 75% accuracy, compared to 84% accuracy for the given features. However, our features reduced dimensionality significantly, with our dimensions being 0.01% the size of the original features. LDA and Logistic Regression performed very similarly, but LDA has the advantage of having a closed form solution. Adding our features to the provided dataset improved accuracies, so we conclude that the features we engineered were useful in predicting sentiments of movie reviews.

Extension

Upon further readings and analysis of our dataset we noticed that there was a trend that reviewers would often give short plot summaries aka "factual" statements. But the key metric we wanted to extract from our dataset was "subjective" statements. If given more time and compute resources exploring the possibility of only extracting sentiment from purely "subjective" statements, in theory, would have improved our scores significantly.

In addition when analysing the Stanford dataset model we found similar performances when we only accounted for the presence of words compared to the amount of words that existed. This seems to suggest that the presence of the word lends more heavily towards how a user reviews a movie. Although this could be due to the previous point where counts do not help due to the possible existence of words being simple plot words.

Furthermore, we found that by only taking the top N most frequently occurring words performed similarly if not better than other statistical decomposition techniques. This might suggest that specific words don't matter much when writing a

review, and more so the frequency of frequent word usages might be more indicative towards the tone.

Other feature engineering techniques would be useful, in particular meta-analysis of other implementations has found that POS of words help accuracy scores significantly[10]. Because certain types of words, in particular adjectives convey a reviewers feelings towards a subject.

"Thwarted expectations"[10] are an interesting subject matter where if we're able to proper represent would add greatly to our dataset. One such example is "This film should be brilliant. It sounds like a great plot, the actors are first grade, and the supporting cast is good as well, and Stallone is attempting to deliver a good performance. However, it can't hold up". Our featureset acted on a per sentence basis, treating them independently. But as seen in this example each sentence ends up modifying the one previously, being able to capture this flow of sentiment based on each sentence in order would help greatly.

Sources

1. Minqin Hu and Bing Liu. "Opinion Lexicon". www.kaggle.com/nltkdata/opinion-lexicon/data.
2. Wikipedia. "List of Emoticons". en.wikipedia.org/wiki/List_of_emoticons.
3. Matthew E. Peters, Mark Neumann, Mohit Iyyer, Matt Gardner, Christopher Clark, Kenton Lee and Luke Zettlemoyer. "Deep contextualized word representations" https://arxiv.org/abs/1802.05365
4. Jacob Devlin, Ming-Wei Chang, Kenton Lee, Kristina Toutanova "BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding" https://arxiv.org/abs/1810.04805
5. Yinhan Liu, Myle Ott, Naman Goyal, Jingfei Du, Mandar Joshi, Danqi Chen, Omer Levy, Mike Lewis, Luke Zettlemoyer and Veselin Stoyanov. "RoBERTa: A Robustly Optimized BERT Pretraining Approach" https://arxiv.org/abs/1907.11692
6. Alan Akbik, Duncan Blythe and Roland Vollgraf. "Contextual String Embeddings for Sequence Labeling" https://alanakbik.github.io/papers/coling2018.pdf
7. Jeffrey Pennington, Richard Socher and Christopher D. Manning. "GloVe: Global Vectors for Word Representation" https://nlp.stanford.edu/pubs/glove.pdf
8. Carmine DiMascio. "py-readability-metrics". pypi.org/project/py-readability-metrics.
9. Scikit-learn: Machine Learning in Python, Pedregosa et al., JMLR 12, pp. 2825-2830, 2011.
10. Bo Pang, Lillian Lee, Shivakumar Vaithyanathan. "Thumbs up? Sentiment Classification using Machine Learning Techniques". https://www.cs.cornell.edu/home/llee/papers/sentiment.pdf