COMP 551: Applied Machine Learning - MiniProject 2 Report

Alexander Harris, Abbas Yadollahi, Swathi Srinivas Malagode

February 23rd, 2019

1 Abstract

Sentiment analysis is a sub-domain of opinion mining where the analysis is focused on the extraction of emotions and opinions of people towards a topic from structured, semi-structured or unstructured textual data. In this project, we focus on the task of predicting sentiment on a database of IMDB movie reviews. We perform text data processing for feature extraction and examine the sentiment expression to classify the IMDB reviews as positive or negative. A comparison on different classification approaches has been performed to determine the most suitable classifier to our problem. Our approach using the Linear SVC classifier has the best accuracy on the test set, achieving 89.36%.

2 Introduction

The present era of internet has become a huge database which hosts massive amounts of data, created and consumed by the users. This database has been growing exponentially, in which users express their opinions across various channels such as Facebook, Twitter, Instagram, etc. One such form of them expressing themselves are movie reviews, which are an unstructured form of grammar. The presence of such sentimental words to express a review inspired us to devise an approach to classifying them.

Sentiment analysis is a technology that will be very important in the future years. It is performed to extract opinion and subjectivity knowledge from user generated text content. With opinion mining, we can distinguish poor content from high quality content. With the technologies available we can know if a movie has a higher number of good or bad reviews and determine the reasons behind the reviews negativity or positivity. Feature selection is a critical task in sentiment analysis and effectively selecting representative features from subjective texts can improve sentiment-based classification. Our sentiment analysis study is focused on the IMDB reviews dataset.

In this project, we use binary, unigrams, bigrams and TF-IDF weighting features to process the text data from 25,000 IMDB reviews. We use four different machine learning classification techniques, namely Bernoulli Naive Bayes, Logistic Regression, Linear SVC, Linear SVM using SGD. Additionally, we also attempted a deep-learning based classification technique using Long Short Term Memory (LSTM) and Convolutional Neural Networks (CNN), since we believe that LSTMs are particularily adept at extracting information from sequences of words and not just simple occurences [1].

The goal of this project is to maximize the accuracy of our classifier model on predicting the sentiment associated to the 25,000 unlabeled reviews contained in the test set. The experimental results show that the highest accuracy decides the best performing model, and on our dataset, the Linear SVC classifier had an accuracy of 89.36% on the test set which beats the other techniques.

3 Related Work

Sentiment analysis is considered to be very challenging in the domain of natural language processing (NLP). Previously used techniques for sentiment classification can be split into three categories; these include machine learning algorithms, link analysis methods, and score-based approaches.

The effectiveness of machine learning techniques when applied to sentiment classification tasks is evaluated in the pioneering research by Pang [2]. Many studies have used machine learning algorithms, with support vector machines (SVM) and Naive Bayes (NB) being the most commonly used. SVM has been extensively used for movie reviews, while Naive Bayes has been applied to reviews and web disclosure. In comparison, SVM classifiers have outperformed other classifiers such as Naive Bayes.

Later, data mining was applied to movie classification. The 240 prototype movies from IMDB were used. The other work that used sophisticated feature selection was by Abbasi [3]. They found that using either the information gain (IG) or genetic algorithms (GA) resulted in an improvement in accuracy. They also combined the two in a new algorithm called Entropy Weighted Genetic Algorithm (EWGA), which achieved the highest accuracy of 91.7%. But the drawback of this method was that it was very computationally expensive to conduct the initial feature selection. The ensemble technique, which combines the outputs of several base classification models to form an integrated output, has become an effective classification method for many domains [4].

4 Dataset and Setup

Both the training and test datasets consist of 25,000 real world reviews taken from the IMDB website, where each entry is represented by a text file containing the raw review in a single line. For the training dataset, the files are equally separated into two folders, depending on whether the review has a negative or positive connotation to it. From the latter, we gathered two pieces of information for the data processing, the list of words used in the review and the sentimental connotation associated to them.

The training data was divided into two sets: a training set containing 17,500 reviews and a validation set containing the rest of the 7,500 comments. To avoid biasing, we randomized the selection when splitting the original training set, giving the two divided sets uneven amounts of negative and positive reviews. Before creating the feature vectors, we performed pre-processing on the reviews, namely decoding and lowercasing the raw text.

Once the pre-processing is done, we follow up by tokenizing the texts to create a feature set. These tokens are then vectorized to produce either word occurences or TF-IDF weightings. Once vectorized, we use the top features (measured by word count) to reduce the feature set size.

5 Proposed Approach

5.1 Validation Pipeline

To verify our results, we decided to use a held-out validation set rather than something like k-fold cross validation in order to save on computation time. Therefore the original training set was subject to a 70:30 training/validation split.

5.2 Feature Pipelines

To begin, we decided on two separate feature pipelines to test our chosen classifiers. We figured that the most logical separation would be word occurrences vs. TF-IDF weightings. To create our feature matrices, we used SKLearn's [5] built-in vectorizer methods, along with a custom tokenizer that was used to tokenize words, remove stopwords and perform lemmatization using NLTK [6]. We then performed cross-validation on these two pipelines using a simple logistic regression classifier to test different parameters such as unigram versus bigram, maximum number of word features, etc. We found that the TF-IDF pipeline with bigrams and a maximum word frequency of 50% gave the best result, with an accuracy of 88%. We also found that limiting the maximum number of word features to 5000 helped to improve accuracy. Further optimization can also come from tuning model hyper-parameters, which will be discussed in the next section.

5.3 Models

We performed grid search cross validation on the SKLearn classifiers in order to optimize model hyper-parameters like maximum number of iterations, tolerance and learning rates. The optimal hyper-parameters we obtained for each model can be found in the Results section.

5.3.1 Custom Bernoulli Naive Bayes

They main component to implementing a Bernoulli Naive Bayes algorithm from scratch is the Bayes' rule (Figure 1) that it follows. Seeing as it uses binary features for classifying, we created the vectorizer using binary unigrams. The main parameter used to tune the classifier was the maximum number of features used for fitting. Through multiple iterations, we found the highest precision model uses approximately the top 900 word features, giving an accuracy of 82.78% of the validation set.

$$P(y|x_j) = \frac{P(x_j|y) * P(y)}{P(x)},\ where\ P(x_j|y) = \frac{\#\ instances\ with\ x_j\ and\ y+1}{\#\ instances\ with\ y+2} \tag{1}$$

Figure 1: Bayes' Rule with Laplace Smoothing

The main issue with Naive Bayes using binarized features is that each feature has equal weighting when in reality this is not true. Therefore, the best way to remove useless features was by only using the highest occurring words in the dataset. This results in less noise from otherwise meaningless features when classifying the reviews, while substantially reducing computation time.

To help control the resulting probabilities when there's missing features in the training set, we use Laplace smoothing on the Naive Bayes classifier. This helped increase the overall accuracy since the complete feature set covers 80,000 words compared to the 900 used in our optimal setting.

5.3.2 Logistic Regression

The logistic regression algorithm is a form of discriminative learning that attempts to model the log-odds ratio with a linear function. It does this by using gradient descent to maximize the log-likelihood function. [7] This algorithm is quite adept when it comes to binary classification tasks, as we found in our feature pipeline testing that it can obtain an accuracy of 88% on our validation set with very little hyper-parameter tuning.

$$a = \ln \frac{P(y=1|x)}{P(y=0|x)} = w_0 + w_1 x_1 + ... + w_m x_m \tag{2}$$

Figure 2: Log-odds Ratio

$$l(D) = \ln L(D) = \sum_{i=1}^{n} y_i \ln(\sigma(w^T x_i)) + (1-y_i)\ln(1-\sigma(w^T x_i)) \tag{3}$$

Figure 3: Log-likelihood Function

5.3.3 Linear SVC

SKLearn's Linear SVC implements an SVC using a linear kernel to allow for a more flexible loss functions for larger samples. In our case, we are using the hinge loss function which corresponds to a linear SVM. [5]

5.3.4 SGD Classifier

SKLearn's SGD Classifier implements stochastic gradient descent training for a variety of different loss functions. Stochastic gradient descent performs the same function as gradient descent, however instead of computing the exact solution it attempts to estimate on a subset of the dataset. This means that computation is much faster and it can run even on large datasets that would not be able to fit into memory [8]. In our case, we are using the hinge loss function which corresponds to a linear SVM. [5]

5.3.5 LSTM

We also decided to try out a deep-learning approach for sentiment analysis. For this, we decided the use a CNN+LSTM based architecture, where the CNN layers would be dedicated to feature extraction and the LSTM layer would dedicated to sequence learning. LSTM's were designed to resolve the issue that recurrent neural networks (RNN) have with remembering information over long periods of time. This makes them particularly adept at processing sequence-based data such as speech and as such are one the state of the art algorithms used for NLP tasks like sentiment analysis [1].

Our model is implemented using Keras [9] and the Tensorflow [10] backend. We limited the number of word features to 5000, and used Keras preprocessing to tokenize and pad our training data. The model architecture itself consists of an embedding layer, followed by 1-dimensional convolutional and max-pooling layers. This is then followed by our LSTM layer with dropout to help combat overfitting, followed finally by a dense fully-connected layer using a sigmoid activation function, since this is a binary classification task. A visual representation of our model's architecture can be seen in Figure 4.

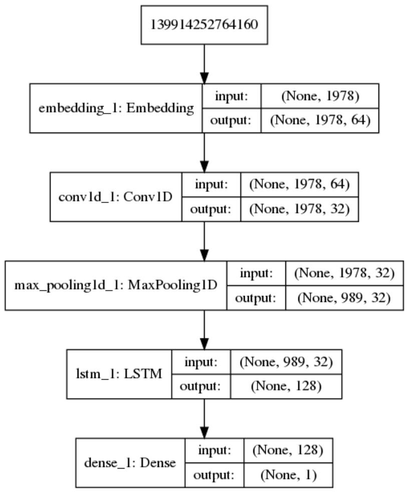

Figure 4: LSTM Model Architecture

6 Results

6.1 Validation Set Performance

As can be seen from Table 1, our best performing model was the Linear SVC with TF-IDF and bigram features, with an accuracy of 89.83% on our validation set, along with a training accuracy of 99.99% and runtime of 36.97 seconds. The SGD Classifier performance was fairly close to our best model, with a validation accuracy of 89.37%, but a higher runtime of 47.22 seconds. Our custom Bernoulli Naive Bayes model performed fairly poorly compared to the other models, with a validation accuracy of 82.31%. As for the LSTM, after training over 10 epochs we saw a maximum validation accuracy of 87.71% at 3 epochs, after which we start to overfit as can be seen from Figure 5.

Model	Hyper-parameters	Features	Training accuracy	Validation accuracy	Training runtime (s)
Bernoulli Naive Bayes	N/A	Binary+Unigram	0.8270	0.8231	34.49
Logistic Regression	C=1, solver='lbfgs', max_iter=1000	TFIDF+Unigram	0.9401	0.8859	40.50
Logistic Regression	C=1, solver='lbfgs', max_iter=1000	TFIDF+Bigram	0.9650	0.8820	61.90
SGD Classifier	loss='hinge', alpha=0.0001, max_iter=1000, tol=1e-4	TFIDF+Unigram	0.9562	0.8916	34.92
SGD Classifier	loss='hinge', alpha=0.0001, max_iter=1000, tol=1e-4	TFIDF+Bigram	0.9846	0.8937	47.22
Linear SVC	C=1, tol=1e-4, max_iter=1000	TFIDF+Unigram	0.9956	0.8904	34.84
Linear SVC	C=1, tol=1e-4, max_iter=1000	TFIDF+Bigram	0.9999	0.8983	36.97
LSTM	batch_size=128, epochs=3	Count	0.8971	0.8771	3714.81

Table 1: Results on Validation Set

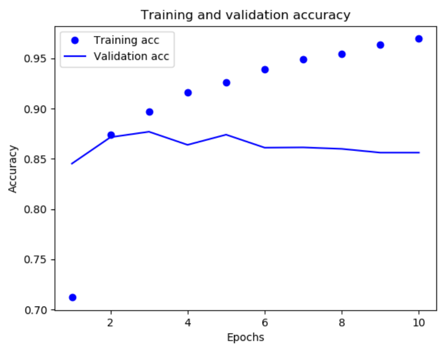

Figure 5: LSTM Training and Validation Accuracy

6.2 Test Set Performance

Our results obtained on the test set were fairly interesting. We decided to submit both our Linear SVC and SGD Classifier models, as they were our best performing. Our best performance on the test set was the Linear SVC model at 89.36%, while the SGD Classifier had an accuracy of 88.87%. Both of these results are slightly lower than what we obtained on the validation set, which suggests there could be some slight overfitting for both of these models on the validation set, due to our hyper-parameter tuning.

7 Discussion and Conclusion

Our main takeaways from this project are two-fold. First, we learned how to apply multiple different forms of classifiers, whether they be from scratch or from SKLearn, to a practical problem involving messy, real-world data. Also, we learned that choosing good hyper-parameters for our classifier is not always straightforward or obvious. We thought that using all word features would help improve our model, but in the end it depended on the classifier being used, as Naive Bayes benefited from a much lower count. Of all the models we tested, our Linear SVC resulted in the highest accuracy on the test, with a precision of 89.36%. Our accuracy on the test set is within 2.5% of the baseline set by our professor. In the future, we could apply a version of this model to predict the score on 10 given by the reviewer, or even use it for other websites such as YouTube where the sentiment of comments are important.

8 Contribution Statement

All team members have made significant personal contribution towards this project. The amount of work for each is described as follows:

- Alexander Harris: Dataset loading and preprocessing, implementation of SKLearn algorithms, model training, validation and testing, write-up contribution
- Abbas Yadollahi: Dataset loading and preprocessing, implementation of Naive Bayes algorithm, model training, validation and testing, write-up contribution
- Swathi Srinivas Malagode: Dataset loading and preprocessing, write-up contribution.

References

[1] S. Hochreiter and J. Schmidhuber, “Long short-term memory,” *Neural computation*, vol. 9, pp. 1735–80, 12 1997.

[2] B. Pang, L. Lee, and S. Vaithyanathan, “Thumbs up?: Sentiment classification using machine learning techniques,” *Proceedings of the ACL-02 conference on Empirical methods in natural language processing*, vol. 10, pp. 79–86, 2002.

[3] A. Abbasi, H. Chen, and A. Salem, “Sentiment analysis in multiple languages: Feature selection for opinion classification in web forums,” *ACM Transactions on Information Systems (TOIS)*, 2008.

[4] L. S. Larkey and W. B. Croft, “Combining classifiers in text categorization,” *SIGIR*, vol. 96, pp. 289–297, 1996.

[5] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay, “Scikit-learn: Machine learning in python,” *Journal of Machine Learning Research*, vol. 12, pp. 2825–2830, 2011.

[6] E. Loper and S. Bird, “Nltk: The natural language toolkit,” in *Proceedings of the ACL-02 Workshop on Effective Tools and Methodologies for Teaching Natural Language Processing and Computational Linguistics - Volume 1*, ETMTNLP ’02, (Stroudsburg, PA, USA), pp. 63–70, Association for Computational Linguistics, 2002.

[7] W. L. Hamilton, “Lecture 4 - linear classification,” *COMP 551*, Feb 2019.

[8] “Reducing loss: Stochastic gradient descent.” `https://developers.google.com/machine-learning/crash-course/reducing-loss/stochastic-gradient-descent`.

[9] F. Chollet *et al.*, “Keras.” `https://keras.io`, 2015.

[10] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G. S. Corrado, A. Davis, J. Dean, M. Devin, S. Ghemawat, I. Goodfellow, A. Harp, G. Irving, M. Isard, Y. Jia, R. Jozefowicz, L. K. andManjunath Kudlur, J. Levenberg, D. Mane, R. Monga, S. Moore, D. Murray, C. Olah, M. Schuster, J. Shlens, B. Steiner, I. Sutskever, K. Talwar, P. Tucker, V. Vanhoucke, V. Vasudevan, F. Viegas, O. Vinyals, P. Warden, M. Wattenberg, M. Wicke, Y. Yu, and X. Zheng, “Tensorflow: Large-scale machine learning on heterogeneous systems,” 2015. Software available from tensorflow.org.