COMP551 - MiniProject 2

John Flores, Benjamin MacLellan, Mahad Khan

Date Submitted: February 22, 2019

Abstract

Sentiment analysis is the computational task of learning and classifying human opinion and emotion from portions of text. A common application of this is classifying reviews, such as movie reviews, into positive and negative. Using a Kaggle dataset from IMDb of 25000 labelled reviews, evenly split between positive and negative, we test Bernoulli Naïve Bayes, logistic regression, and decision tree models – along with different feature extraction techniques – to learn binary classification of these reviews. Our best performing model, which is logistic regression using binary occurrences of TF-IDF normalized unigrams and bigrams with an inverse regularization strength (C-value) of 7500 with a maximum iteration number of 150, performs with a test accuracy of 91.253 % on the held-out test data on the Kaggle competition.

1 Introduction

Sentiment analysis (SA) learns to assign different emotions or sentiments to text documents, and can use a variety of machine learning tools. SA is ubiquitous in a variety of fields which rely on human interactions – where understanding the views and attitudes of a group are important, and is a classic classificiation problem. Such fields are politics, entertainment, customer products, and much more. The Internet Movie DataBase (IMDb) is an online database of critic- and user-generated movie reviews. In this project, we aim to develop models that can predict the sentiment of a given movie as positive or negative, using data from the IMDb. `Scikit-learn`, a Python package developed to aid machine learning, is used to build, train, and validate these models [3]. 80% of the data will be used as a training set while 20% will be used as a validation set. This report is structured as follows: in Section 2 we briefly summarize some related work on classifying movie reviews, Section 3 and 4 we outline the data set used and our proposed approach, and finally in Sections 5 and **??** we provide and discuss the results of our models and future directions.

2 Related Work

Sentiment analysis, or opinion mining, is the computational study of emotions, opinions, and attitudes towards a given subject or entity. With the wealth of readily available data from various online platforms and the huge advantage which knowing/understanding opinions can exert on any social-oriented entity (businesses, products, politicians, platforms, and so on), SA is a hugely important and active field. SA may encompass multiple aspects, but at its heart is composed of two aspects: **polarity** and **intensity** [2]. Polarity, in general, is a classification task in which portions of text (words, sentences, documents) are classified into opposing categories, such as positive or negative. More complex classifications can also be analyzed which represent more intuitive human emotions such as happiness or anger. Intensity is a continuous value of *how much* of the sentiment is estimated to the text – for example how much happiness does the text demonstrate.

SA is a very broad computational field. While it does not necessitate the use of machine learning algorithms, these are the most common and widely implemented. Both supervised and unsupervised learning approaches can be useful; and within supervised learning, algorithms including logistic regression, SVM, Naïve Bayes, decision trees, maximum entropy, (deep) neural networks [6], convolutional neural networks [11], k-nearest neighbours, and random forest [7]. Beyond the actual model which is employed, there a number of common feature extraction methods; for example bag-of-words, n-grams, stemming and lemmatization, negation handling, term frequency-inverse document frequency, part of speech tagging, and word to vector – among many other more advanced techniques. Movie review sentiment analysis is a very common application of these SA techniques and models, and is commonly used for teaching and proof-of-concept of new models. Thus, there is a plethora of literature and code repositories for this classification task. In Ref. [5], the authors demonstrate a broad variety of SA techniques for the binary movie review classification – including SVM, logistic regression, and random forest, and using bag-of-words and skip-gram word to vector. Accuracy

scores of their models range from 83% - 89%, varying the models, features, and hyperparameters. Moving beyond binary classification, the authors used recursive neural networks for multi-class analysis, and applying ensemble averaging to gain superior performance [5]. Performing a brief review of accuracy scores for similar IMDb review datasets, 87.8% [8] with a simple deep-neural network and binary word occurance, accuracy range of 74-88% using Naïve Bayes (Bernoulli and regular) and SVM [9], and 89% using a simple convolutional neural network and mapping words to integers [10].

3 Dataset and Setup

The dataset is comprised of 25 000 IMDb reviews in the form of text files, where half were negative and half were positive. The dataset was partitioned such that 80% served as test data while 20% served as validation data. Prior to feature extraction, the raw text files (which each represents one review) are preprocessed to improve feature quality and thus model accuracy. All text was converted to lowercase and excess punctuation removed using regular expression commands. The resulting text still contained HTML tags within it – most commonly, **break** tags of `<br>`. These are also removed. At this point the strings are prepared for the various feature extraction techniques used.

4 Proposed Approach

4.1 Custom Naïve Bayes Implementation

For efficiency and consistency, our custom implementation of the Naïve Bayes was developed to be compatible with all necessary `scikitlearn` validation and testing functions, such as `.fit()` and `cross_val_score()`. To this end, a custom class, inherited from the base `sklearn` class, `BaseEstimator`, is created. Inheriting from the base class allows smooth integration into `sklearn`'s functionality, but the estimator algorithm are custom written. Specifically, the class functions `.fit()`, `.predict()`, and `.score()` are custom built – and are the only functions used for our purposes in this project. As a sanity check, we compared our custom Naïve Bayes (`CustomBernoulliNaiveBayes()`) with the standard `sklearn` implementation, `BernoulliNB()`.

4.2 Feature Extraction and Model Comparison

Our approach will consist of two parts: feature extraction and model comparison. For model comparison, we compare three models to determine if one significantly out-performs the other: logistic regression, Naï, and decision trees.

Then, the best model will be tested with differing extracted features, including binary occurrence, a simple bag of words count, a "bag of n-grams" count, a TF-IDF (Term Frequency times Inverse Document Frequency) transformed count of each word, and a TF-IDF transformed count of various n-grams. These features are defined in Appendix B.

For all of these experiments, the feature vector will be normalized with the `Normalizer()` function. In addition, `GridSearchCV` will be used to tune hyperparameters, and the model will be cross-validated 5 times. A report function from the scikit-learn website was modified to print the hyperparameters of the top performing models within a trial according to the mean validation score (MVS) [4].

In logistic regression, `scikitlearn` defines the C parameter, which is the inverse of regularization strength. In order to measure its effect, we varied C from its default value, 0.001, to 10000. Convergence warnings appeared at higher values of C; as such, experiments increasing the maximum number of iterations the classifier would use were also performed.

5 Results

5.1 Custom Naïve Bayes Implementation

Our custom Naïve Bayes (`CustomBernoulliNaiveBayes()`) performed identical to the standard `sklearn` implementation, `BernoulliNB()`. Using binary occurrence of every word in the training data as the features

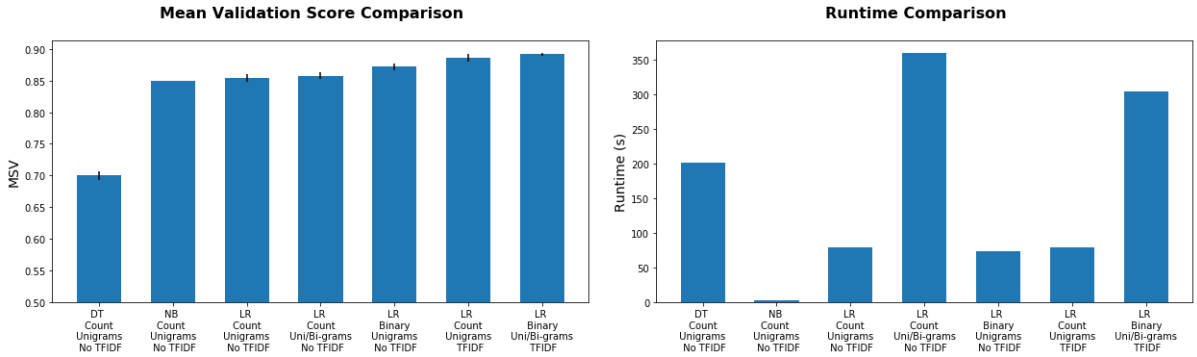

Figure 1: Mean Validation Score and Runtime Comparison between experiments (DT - Decision Trees, LR - Logistic Regression). From these trials, the best performing model was logistic regression running with TF-IDF transformed binary counts of unigrams and bigrams, attaining an accuracy of $89.15\% \pm 0.19\%$, running in 303 seconds. Decision trees performed significantly worse than logistic regression. The use of both unigrams and bigrams, TFIDF, and binary occurrence all improved the performance of our base LR model. However, the use of unigrams and bigrams significantly increased our runtime by a factor of 3.

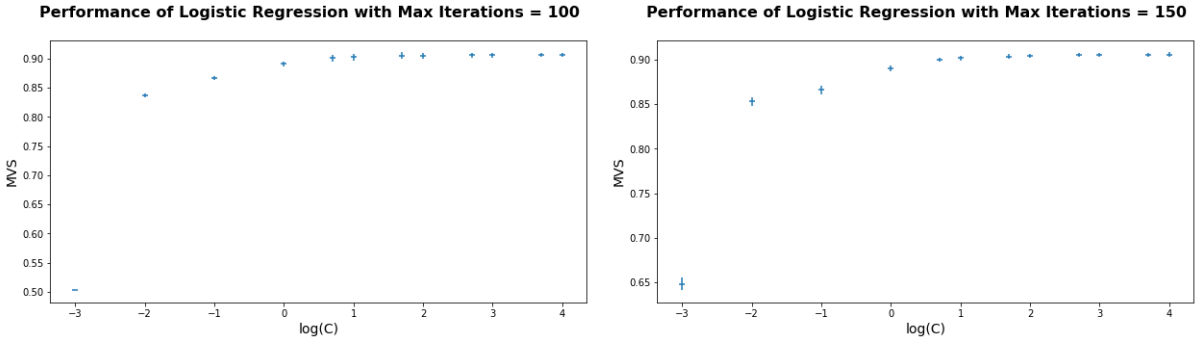

Figure 2: MVS vs C, for 100 and 150 max iterations. A slight increase in MVS is seen from log(C) = 1 to log(C) = 2, but all measurements agree within their uncertainites.

(74844 features and 25000 examples), we fit both models and predicted using a training/validation split of 80/20% (identical split between the custom and packaged model). The prediction scores were identical, demonstrating our custom model was implemented properly. As expected, the packaged `sklearn` Bernoulli Naïve Bayes is computationally faster to fit, as it has been optimized. With the initial implementations of our custom Bernoulli Naïve Bayes, our model was significantly slower – but with some optimization and using sparse matrix functionality of `numpy`, we were able to dramatically improve speed and approach that of the packaged model. In the final tests, it took 2.74 seconds to train our custom model, and 0.07 seconds to train the `sklearn` implementation (Intel i7, 1.90 GHz, 8 processors). This runtime could be improved by further, for example by better utilizing sparse matrix manipulations. However, as the custom model was able to run in a realistic time on the full dataset, these improvements were unnecessary.

5.2 Model Comparison and Feature Selection

The results of our trials are shown in Figure 1. Logistic regression was shown to outperform decision trees in both accuracy and run time. In simple bag of words unigram trials with default parameters, logistic regression had a maximum accuracy of $85.34 \pm 0.60\%$, while decision trees had a maximum accuracy of $70.02 \pm 0.66\%$. This large difference in performance was also mirrored in runtime, where decision trees took 202 seconds compared to logstic regression at 79 seconds. As such, all other experiments used logistic regression as the classifier, with this previous logistic regression trial as the control.

After choosing the classifier, we performed experiments with feature selection. In order of ascending effect

Figure 3: Effect of Max Iterations on MVS. All standard errors overlap, suggesting no statistically significant difference between trials.

on MVS score, the features tested were the use of both unigrams and bigrams, the use of binary occurrence instead of counts of words, and the use of TF-IDF. The use of both unigrams and bigrams increased our MVS from $85.34 \pm 0.60\%$ to $85.77 \pm 0.53\%$. Also, transforming our count vector to a vector of binary occurrences increased our model's MVS to $87.17 \pm 0.48\%$, when only considering unigrams. Moreover, TF-IDF transformation of our features improved our model's MVS to $88.65 \pm 0.19\%$ when considering a count of unigrams. A model including all of these parameters, a TF-IDF transformed binary occurrence vector of both unigrams and bigrams, gave a 3.31% increase in performance, an improvement of roughly 5.5 standard errors.

This final model was then used to conduct multiple experiments with varying C values. As shown in figure 2, larger C values increase MVS up until a value of C = 10 for max iterations = 100 or 150. However, the number of max iterations did not affect our results. With a C-value of 7500, our model performed similarly across 5 different max iteration values, as shown in figure 3.

5.3 Kaggle Performance

Our final model is a logistic regression model using binary occurances of TF-IDF normalized unigrams and bigrams with an inverse regularization strength (C-value) of 7500 with a maximum iteration number of 150. This gives a test score of 91.253 % on the Kaggle test set.

6 Discussion and Conclusion

In this project, we used the Bernoulli Naïve Bayes, logistic regression, and decision tree models with different feature extraction pipelines to predict the sentiment of IMDb reviews. From the results discussed above, we can conclude that the logistic regression classifier with a TF-IDF transformed binary occurrence vector of both unigrams and bigrams results in the maximum accuracy. The Bernoulli Naïve Bayes classifier also provides good accuracy, whereas the decision tree classifier was found to be relatively inaccurate.

To improve the results, future work could involve testing more classifier models, such as support vector machines or deep-neural networks – or testing more complex linguistic features. For example, we could test features based on complex verb tense patterns or punctuation style, which may provide slightly improved accuracy scores and provide interesting insight and interpretability. Moreover, as IMDb allows for numerical ratings, this work could be extended to score prediction. However, the inherent variance between critics could make this an untenable project. Future projects could also branch out from this specific application, for example the development of multilingual sentiment analysis models. Furthermore, the sentiment of a reviewer could be classified in a multi-class manner rather than in a binary-class manner which include more complex sentiments such as “neutral”, “happy” and “angry”.

References

[1] Daniel Hnyk, *Creating Your Own Estimator in SciKit-Learn* http://danielhnyk.cz/creating-your-own-estimator-scikit-learn/

[2] L. Tian et al; *Polarity and Intensity: the Two Aspects of Sentiment Analysis* arXiv:1807.01466v1

[3] Pedregosa et al., *Scikit-learn: Machine Learning in Python*, *JMLR 12*, pp. 2825-2830, 2011.

[4] *Comparing randomized search and grid search for hyperparameter estimation* https://scikit-learn.org/stable/auto_examples/model_selection/plot_randomized_search.html

[5] Hadi Pouransari, Saman Ghili *Deep learning for sentiment analysis of movie reviews* Standford University, CS224

[6] Walaa Medhata, Ahmed Hassanb, Hoda Korashy *Sentiment analysis algorithms and applications: A survey* Ain Shams Engineering Journal (2014) 5, 1093–1113

[7] Palak Baid, Apoorva Gupta, Neelam Chaplot *Sentiment Analysis of Movie Reviews using Machine Learning Techniques* International Journal of Computer Applications, Volume 179 – No.7, 2017

[8] Francois Chollet, J.J. Allaire *Classifying movie reviews: a binary classification example* , Github https://jjallaire.github.io/deep-learning-with-r-notebooks/notebooks/3.4-classifying-movie-reviews

[9] *Sentiment Analysis with Naive Bayes* https://github.com/bozhang0504/IMDB-Sentiment-Analysis

[10] *Text Classification for Sentiment Analysis* https://www.samyzaf.com/ML/imdb/imdb.html

[11] Yoon Kim, *Convolutional Neural Networks for Sentence Classification* Conference on Empirical Methods in Natural Language Processing, 1746–1751, 2014

Appendix A Statement of Contributions

Benjamin MacLellan - Implementation of custom Naïve Bayes
Mahad Khan - Testing, Script and Document Proof-reading
John Flores - Pipeline Function Creation and Testing, Readme Writing

Appendix B Features Description

Consider the string "twinkle twinkle little star twinkle twinkle"

Bag of Words A vector of counts of different words. The feature vector is the number of times each word appears in the document – where each feature vector entry x_j corresponds to a specific word. In this example the feature vector is,

$$x = [4, 1, 1]$$

(4 "twinkle", 1 "little", 1 "star")

n-grams An n-gram is a string of n words, usually n consecutive words in the text. This is an extension of the bag-of-words approach, where we now consider consecutive strings of words. Using unigrams and bigrams, for example, the feature vector is

$$x = [4, 1, 1, 2, 1, 1, 1]$$

(4 "twinkle", 1 "little", 1 "star", 2 "twinkle twinkle", 1 "twinkle little", 1 "little star", 1 "star twinkle")

Binary Occurrence - A vector of counts which is 1 if the term appears in the document, 0 if not. This can be viewed as a similar approach as the bag-of-words, but instead of integer values for x_j, it is restricted to binary values. From a feature vector in bag-of-words, the binary occurrence feature vector becomes $x_j^{\text{bin}} = 0$ iff $x_j^{\text{bow}} = 0$ and $x_j^{\text{bin}} = 1$ iff $x_j^{\text{bow}} > 0$, where bin = binary occurance and bow = bag of words.

TF-IDF A transformation of the count vector following:

$$\text{TFIDF} = \log \frac{\text{Docs in Corpus}}{\text{Docs with term t} + 1}$$

This means that rare words are weighted greater than common words.