COMP 551 - Applied Machine Learning
Mini Project 2 Report

Matthew Lesko-Krleza
260692352

Michael Ding
260690732

Wuyang Zheng
260830900

February 23, 2019

Contents

1 Abstract

In this mini project, we were tasked with training multiple classification models, including a Bernoulli Naive Bayes classifier implemented from scratch, to perform sentiment analysis on IMDB movie reviews and with comparing their accuracy with one another. Furthermore, we were also tasked in experimenting with different text extraction and transformation features. Finally, we used our best performing model to submit sentiment analysis predictions to a private Kaggle competition organized for our course. You'll find that we experimented with logistic regression, decision tree, and linear SVM classification models. We extracted binary occurrences of words, and transformed text word counts to TF*IDF features($Term\ Frequency * Inverse\ Document\ Frequency$). You'll find that our best performing model is a linear soft SVM model tuned with various hyper parameters, using TF*IDF features. Our best performing model can achieve a classification accuracy of roughly 91.493% on the test set. This report details the comparison and findings of our classification models, and text features.

Keywords: applied machine learning, linear classification, sentiment analysis, imdb movie reviews, SVM, decision trees, logistic regression, TF*IDF

2 Introduction

We had five major tasks: implement a Bernoulli Naive Bayes model from scratch, run experiments using at least two different classifiers from SciKit-Learn, try at least two different text feature extraction pipelines, develop a model validation pipeline, and submit sentiment analysis predictions on a test set from an IMDB movie reviews data set to a private Kaggle competition. The data set is provided by the course instructor through the Kaggle competition website and consists of English movie reviews from IMDB in text format. The movie reviews can be for any movie. The training and test sets each consist of 25000 data points. The data set is already cleaned for us, i.e. there are no missing values in the training or test sets. Some of our most important findings are introduced below. We found that using a soft SVM provides more accurate classifications than the logistic regression, decision tree, and Bernoulli Naive Bayes models given our data. When testing our SVM model with only a word count feature and normalizer, we had an accuracy of 89% on the validation set. We found that using a TF*IDF feature and a normalizer allowed us to increase our SVM accuracy by roughly 1% on the validation set, whereas using a binary occurrences feature decreased our SVM accuracy to 88%. We realized that using the Grid Search cross validation technique took more computation time than the Random Search cross validation technique but allowed for a better prediction accuracy on the validation set. Our most important finding is that it could be simple to achieve an accuracy of 90% on the test set but it can take significantly more effort and time to get an accuracy of 91% or upwards by performing numerous experiments with different parameters and tuning.

3 Related Work

This IMDB sentiment analysis dataset is provided by Stanford University where they have present their a vector space model that uses a mix of supervised and unsupervised learning techniques [1]. Their model has an accuracy of 88.89% when tested on the IMDB data set.

One work demonstrates that CCN and LSTM neural networks are useful for NLP and sentiment analysis tasks [2]. The state-of-the-art neural network model called ULMFiT achieves a 95.4%

accuracy score on the IMDB data set [3]. ULMFiT is a 3-layer transfer learning neural network. It was trained on the Wikitext 103 and the IMDB movie review data sets. We decided to start the project with a Naive Bayes model using the Opinion Lexicons[4] which is composed of 6000 positive and negative sentiment words.

4 Dataset and setup

Each review (data point) is within its own text file, and as for the training set, positive and negative reviews are located under the *pos* and *neg* directories respectively, thereby giving each review from the training set a label. Whereas for the testing set, each review is also within its own text file, but positive and negative reviews aren't differentiated from one another in any way like the training set. The whole text content of a review is within one line, and any new lines made by the user submitting the review are identified by a $< /br >$ tag. Tokenization and normalization are done for each classifier. The tokenization regular expression selects tokens of 2 or more alphanumeric characters (punctuation is completely ignored and always treated as a token separator). Unless specified, the default normalization technique for each classifier is 'l2' normalization.

5 Proposed Approach

5.1 Model Description

In this project, the models used are Bernoulli Naive Bayes classifier which is implemented from scratch, and Logistic Regression, Decision Tree, Linear Support Vector Machine(SVM) that are provided from SciKit-Learn packages [5].

Bernoulli Naive Bayes classifier is one kind of Naive Bayes classifier and this model is useful when features are binary-valued. The algorithm is naive because it assumes that the all of the features are conditionally independent and uses Bayes' Theorem. Hence the name Bernoulli Naive Bayes. The feature used is the binary occurrence of the opinion feature feature using sentiment lexicons from Bing Liu's "opinion lexicons" (6,000 positive/negative words). This algorithm for this project is implemented from scratch as per the class theory on Bernoulli Naive Bayes.

Logistic Regression algorithm is similar to the linear regression algorithm. Linear Regression is designed for predicting continuous values whereas Logistic Regression determines a linear hyperplane to classify data points. Logistic Regression uses the Linear Equation to predict continuous values and a Logistic function to squeeze the predicted values within the range 0 to 1, thereby classifying points. Our motivation behind using this model is that we implemented a linear regression model in the last project, and wanted to see if the same linear equation can still be effective for classification.

Decision Tree is a tree-like model. Leaves are classes, and inner nodes are binary classification rules. The learning process partitions the training data into separate classes using the features provided. The motivations to use it are that it is easily interpretable, and the mathematics are simple. However, it has the lowest accuracy score when compared to the logistic regression and linear SVM and is sensitive to the training data.

Support Vector Machine searches for a hyper-plane to separate different classes of data and the separating hyper-plane should equally split the maximum gaps between different classes in the data set. The margin distance is determined by its "support vectors". These are data points lying on either edge of the margin. The model attempts to maximize the margin to have maximum confidence in separating points close to the hyper-plane. Our motivations for using the algorithm is because of its flexibility with smaller data sets and the linearity of the data set. The algorithm is widely used in text classification and sentiment analysis because it is suited to train on data sets with less than 100K samples, and can use non-linear kernels for non-linear classification.

5.2 Feature Extraction

We extracted and tested our features in three steps. In the first step, we extracted and tested features to determine which model we should prefer from the set of models we decided to experiment with. Since text data can't be used directly to train models, we extracted the word count with different ngrams, and normalized it using SciKit-Learn's "CountVectorizer", and "Normalizer" APIs [5].

In the second step, we tested Binary Occurrences and TF*IDF text features to determine which of the two has a greater increase in accuracy score on the best scoring model from the first step.

We used SciKit-Learn's "CountVectorizer" for feature extraction, and its "TfidfTransformer" for transforming the extracted ngrams into a TF*IDF feature. Normalization was performed in the last step as to standardize our data. Finally, we implemented our own opinion lexicon feature feature using sentiment lexicons from Bing Liu's "opinion lexicons" (6,000 positive/negative words). However, our results showed that the opinion lexicon feature didn't perform well against the two other text features.

5.3 Algorithm Selection and Implementation

Before training the model, the training data is shuffled and split into training and validation partitions. 80% of the training data is kept for training, and 20% of the training data is left aside for validation. We used the "l2" regularization strategy by default but realized "l1" performed better in our best classifier. The most significant criteria for algorithm selection in this project is the accuracy score on the validation data set.

5.4 Model Optimization

Based on ours results we chose the TF*IDF feature as the one to use for training our best performing classifier. Since the models and feature extraction processes are based on SciKit-Learn's modules, changing the parameters of the functions of these modules is simple. We used a Grid Search Cross Validation technique to minimize overfitting and to tune our hyperparameters. For the parameter "tokenizer" in "CountVectorizer", the default is none, we implemented our own tokenizer functions which used "textblob tokenizer" and "stemming tokenizer" but they just increase accuracy slightly. However, we noticed that the two tokenizers "token.tokenize" and "nltk.word tokenize" improved our accuracy more significant. "token.tokenize" improved the accuracy most significantly so it was used in our best performing model. Our final parameters that are determined by Grid Search CV to improve the model most significantly are the following: "ngram range" set to "(1,2)", "binary" set to "True","strip accents" set to "unicode", "norm" set to "l1","smooth idf" set to "false", and "C" set to "10". Interestingly enough, we noticed that C is determined to be 10, as opposed to 1 or 5. This means that our SVM prefers to have a small margin but with little to no missclassifications. We also used a 3-Fold cross validation technique because we didn't notice a significant increase in accuracy when increasing the fold number to something like 10 or above. The trade-off in performance and compute time in a 10-Fold vs a 3-Fold made us choose the 3-Fold.

6 Results

6.1 Bernoulli Naive Bayes

We used the 6000 words from Opinion Lexicon as feature. 1 means presence in the text and 0 mean no occurrence. Bernoulli Naive Bayes gives a 85% accuracy as we submitted it as first submission on Kaggle competition with test sets. It also has an 82% accuracy on the validation set. It gives an acceptable accuracy and we decided to investigate on other potential base learning algorithm.

6.2 Base Learning Algorithm Comparison

After obtaining results from Bernoulli Naive Bayes, we implemented a pipeline with SkLearn's CountVectorizer for feature extraction and pre processed it with a Normalizer. We wanted to experiment with and compare three different models, and use the highest scoring one for further experiments. We trained Logicstic Regression, Decision Tree and SVM models. Table 1 displays a comparison in in precision, recall, and f1-score on the validation set.

	Precision(%)	Recall(%)	F1-Score(%)
Linear Regression	84.945	84.900	84.898
Decision Tree	71.159	71.160	71.159
SVM	88.468	88.440	88.439

Table 1: Different Learning Algorithm Comparison)

Table 1 shows that SVM scores highest with an accuracy of 88.468%. We selected the SVM model for further experimentation.

6.3 Different Feature Extraction Comparison

We wanted to compare the performance of different feature extraction methods mentioned in Section 5.2. We tested three different feature extraction pipelines, Binary Occurrence, Opinion Lexicon Count, and TF*IDF, with the Occurrence Count used as a baseline. Word OccurenceCount with TF*IDF gives a better result.

	Precision(%)	Recall(%)	F1-Score(%)	Run Time(sec)
SVM+OccurenceCount	88.468	88.440	88.439	10.12
SVM+BinaryOccurence	88.508	88.500	88.500	8.62
SVM+OpinionLexiconCount	82.064	81.980	81.971	4.39
SVM+OccurenceCount+TF*IDF	89.071	89.040	89.039	9.75

Table 2: Different Feature Model Comparison)

6.4 Model optimization result comparison

Table 3 displays the details about weighted average accuracy on validation data set and corresponding running time when using different tokenizers on TF*IDF. Run time is defined as the time it takes to train the model and predict on the validation set. At a quick glance, "textblob tokenizer", "nltk.word tokenize", improve the accuracy close to 90% but are significantly slower with training times of 205 and 43 seconds respectively. Whereas "token.tokenize" increases the accuracy to 90% and keeps the learning time fast at 4 seconds. So we used "token.tokenize" for our final model to take advantage of high accuracy and low run time.

Furthermore, using Grid Search CV for cross validation and hyperparameter tuning allowed us to reach an accuracy of 91.5% on the validation set. This best model is used predict the test data and the accuracy score on the leader board is of 91.493%. We also notice that there isn't any significant difference in the precision, recall, and f1-score metrics, therefore we can conclude that our classifier isn't biased towards making either false positives or false negatives.

	Precision(%)	Recall(%)	F1-Score(%)	Run Time(sec)
Linear SVM + OccurenceCount	88.468	88.440	88.439	10.12
SVM+OccurenceCount+TF*IDF	89.071	89.040	89.039	9.75
SVM+OccurenceCount+TF*IDF+textblob tokenizer	89.153	89.140	89.139	205.213
SVM+OccurenceCount+TF*IDF+stemming tokenizer	88.804	88.800	88.799	125.807
SVM+OccurenceCount+TF*IDF+nltk.word tokenize	89.827	89.820	89.820	43.778
SVM+OccurenceCount+TF*IDF+token.tokenize	89.881	89.880	89.879	4.118
SVM+OccurenceCount+TF*IDF+token.tokenize+ parameter combination	91.582	91.580	91.579	59.377

Table 3: The results in optimization procedures

7 Discussion and Conclusion

To conclude, SVM outperforms the other learning algorithms such as Decision Tree, Logistic Regression, and Bernoulli Naive Bayes. In the future, we would like to investigate on other interesting algorithms such as Deep Learning [2], and NBSVM[6] since their papers presented better results than what we got. Obtaining best parameters with GridSearchCV function from Sklearn was also a crucial step in training our model. We should pay attention when using external libraries. We have tried typo auto-correction from TextBlob and decided to not use it since it didn't significantly increase our accuracy score. Furthermore, in one instance in typo correction, the word "Ufortunately" was corrected to "fortunately", but the user could have very well meant "Unfortunately". Finally, we noticed it was trivial to train a model to reach 90% accuracy, but it took significantly more experiments and effort to achieve an increase of 1.5% in accuracy.

8 Statement of Contributions

Matthew worked on writing the abstract, introduction, and related work sections, coding solutions for importing data, coding pipelines for all classifiers, merging code from other team members, and experimenting with different models to achieve a 90% accuracy on the test set in the Kaggle competition. Michael worked on data loading, preprocessing text, Naive Bayes's implementation, research and implementation of Opinion Lexicon feature. Wuyang worked on training and tuning the highest scoring classifier, and achieved a 91.493% on the test set, he also wrote the proposed approach section.

References

[1] A. L. Maas, R. E. Daly, P. T. Pham, D. Huang, A. Y. Ng, and C. Potts, "Learning word vectors for sentiment analysis," in *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies*, (Portland, Oregon, USA), pp. 142–150, Association for Computational Linguistics, June 2011.

[2] L. Zhang, S. Wang, and B. Liu, "Deep learning for sentiment analysis: A survey," *Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery*, vol. 8, no. 4, p. e1253, 2018.

[3] J. Howard and S. Ruder, "Fine-tuned language models for text classification," *CoRR*, vol. abs/1801.06146, 2018.

[4] M. Hu and B. Liu, "Mining and summarizing customer reviews," *Proceedings of the 2004 ACM SIGKDD international conference on Knowledge discovery and data mining - KDD 04*, 2004.

[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] S. Wang and C. D. Manning, "Baselines and bigrams: Simple, good sentiment and topic classification," in *Proceedings of the 50th Annual Meeting of the Association for Computational Linguistics: Short Papers - Volume 2*, ACL '12, (Stroudsburg, PA, USA), pp. 90–94, Association for Computational Linguistics, 2012.