COMP 551 mini project 2

Abdulla Alseiari Kenji Marshall Mian Hamza

March 13, 2020

Abstract

In this project, we optimized classification models for the IMDB Movie Review dataset, and the 20 Newsgroups dataset. To represent this data, we used four different feature engineering techniques that experimented with using ngrams, and binary/term frequency encodings. We then trained six different models on these different sets of features to find the optimal feature-model pairings, and to find the optimized parameters for each model. This was done using extensive grid searches with five-fold cross validation. We further built on these models via three ensembling techniques: hard voting, soft voting, and stacking with a meta-learner. We found that feature engineering, model selection, and ensembling all had positive effects on test accuracy. In the end, we achieved an accuracy of 0.90560 on the movie reviews and 0.69610 for the news articles, giving a reproducible total test accuracy of 0.85710 across both datasets.

1 Introduction

The goal of this project was to classify the IMDB Movie Review and 20 Newsgroup datasets. IMDB is a binary classification problem between positive and negative reviews, while Newsgroups is a 20 label multiclass classification problem. To approach the datasets, we employed four different feature engineering techniques, experimenting with ngrams, and binary/term frequency encodings to generate bag of word vectors. Subsequently, we trained six models: Logistic Regression, Support Vector Machine, Decision Tree, Random Forest, Adaboost, and Multinomial Naive Bayes. For each model, we found the best feature engineering technique and the optimal model parameters through exhaustive grid searches. Finally, we also experimented with ensembling these optimized models in three different ways with hard voting, soft voting, or stacking with a meta-learner.

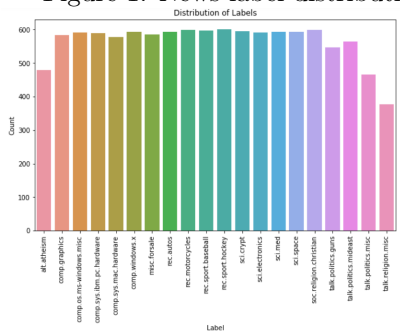
Our experiments showed how important it is to consider a plethora of different models and feature engineering techniques to find the optimal combination. Additionally, it showed how ensembling can provide meaningful gains in accuracy from models that have already been optimized extensively. The trained models reported a wide spread of results on the different datasets, with performances of optimized classifiers ranging from 0.74568 to 0.90560 on the movie reviews, and from 0.36073 to 0.69610 for the news articles. This gave an optimal accuracy of 0.85710 across all testing examples that is easily reproducible.

2 Related work

Text classification has many important applications such as in news labeling, customer feedback, and web searches. Since the rise of the internet, text classification models have gained a lot of attention. Researchers have started to develop automated methods to categorize text since manual categorization is a time-consuming task. Dalal, M. K., and Zaveri, M. A. proposed a generic scheme to deal with textual classification, the steps including "document pre-processing, feature extraction/selection, model selection, training [, then] testing the classifier"[1]. Since text classification problems usually have a large number of attributes "researchers have proposed feature reduction techniques like TF-IDF"[1]. In a previous literature review [1], there is no single Machine Learning model obtaining consistent higher scores in all types of text classification. However, the SVC model showed superiority in a binary class text classification as in the Dalal, M. K., and Zaveri, M. A. 2011 paper[1].

3 Dataset and setup

Figure 1: News label distribution



For the twenty newsgroups dataset, the data was fetched directly using Scikit-Learn[2]. As mentioned, this was a 20-label multiclass classification problem; the classes were relatively balanced with slightly less data observed under the "atheism" tag, and all articles under the "talk" subcategory (Figure 1). There were 11314 training examples and 7532 test examples. The IMDB movie reviews were loaded directly by walking through the operating system's file system – they were perfectly balanced between positive and negative. The dataset had 25000 train examples and 25000 test examples.

Both datasets were first stripped of extra whitespace and converted to lower case. The IMDB movie reviews were populated with HTML "
" tags, while the news articles had newline characters, and awkward textual content like e-mails peppered throughout; these were removed using regex. Subsequently, both datasets were stemmed with the NLTK PorterStemmer [3]. To generate features, Scikit-Learn's CountVectorizer and TfidfTransformer were used in sequence. To reduce features, the fifty most common English words were identified as stop words and words used in more than 90% of documents, or used less than 5 times overall were ignored [4]. For each dataset, four different sets of features were generated; the CountVectorizer parameters and the naming convention used to refer to each feature set is shown in Table 1. These methods still generated a lot of features – 19604 for the IMDB review training data and 12331 for the news training data.

ngram_range	binary	name
(1, 1)	False	"movie" or "news"
(1, 3)	False	e.g. "news_ngrams"
(1, 1)	True	e.g. "news_binary"
(1, 3)	True	e.g. "news_binary_ngrams"

Table 1: Proposed features and corresponding CountVectorizer parameters, dataset names

4 Proposed approach

We implemented 6 models: Logistic Regression, Decision Trees, Support Vector Machines, Adaboost, Random Forest, and Multinomial Naive Bayes. We also extended on these models via different ensembling techniques including hard voting, soft voting, and stacking.

- Logistic Regression makes predictions by learning weights to compute a weighted sum of all features, and passing that sum through a thresholded logistic function. To optimize it, we minimize the cross entropy loss, which is equivalent to maximizing the Bernoulli (binary case) or Categorical (multiclass case) likelihood of the labels given the data [5].
- Decision Trees split regions successively based on the value of a feature which maximally decreases a metric like entropy or gini index (i.e. this uses a greedy heuristic). This can continue indefinitely until the tree can classify the entire dataset perfectly, or when it's no longer worth splitting the data based on regularization parameters like max depth [6].
- Support Vector Machines generate a hyperplane to separate different classes with the largest margin possible, while also minimizing margin violations. The trade-off between these two objectives is defined by regularization parameters [6].
- Adaboost, short for adaptive boosting, uses a large number of weak learners together to create an ensembled strong learner. The weak learners (e.g. decision stumps) are trained successively and place more priority on correctly classifying examples that were previously misclassified [5].
- Random Forest classifiers are also ensemble models that group together a large number of decision trees. The decision trees are trained using bagging, where they all sample from the data with replacement to allow them

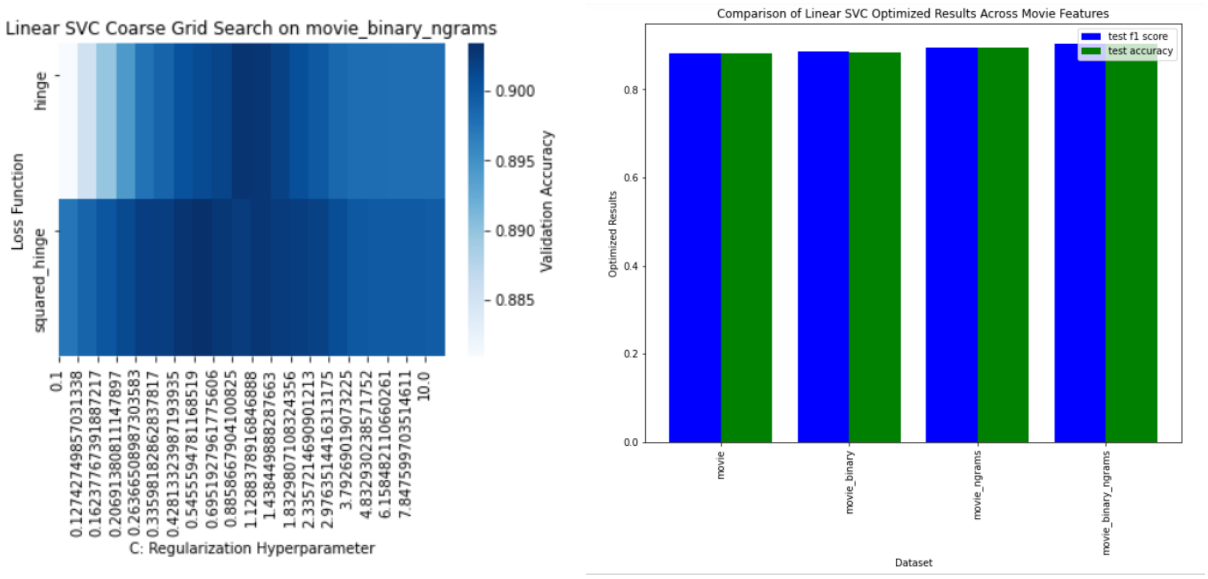


Figure 2: Left: Coarse grid search over loss functions and regularization with Scikit-Learn's LinearSVC and the movie_binary_ngrams data. Optimal parameters identified as C: 0.5455594781168519, loss: squared_hinge. Right: Comparison of test results for optimal LinearSVC implementations on each IMDB feature engineering method. As represented in Table 2, movie_binary_ngrams had the best performance both in accuracy and f1 score.

to learn from different subsets of the data. Additionally, to further decorrelate the decision trees, they also typically only sample a subset of features to make their decisions [5].

- Naive Bayes is the only generative classifier we used. This means that instead of maximizing likelihood, Naive Bayes learns the joint distribution of labels and data, which allows the usage of Bayes' rule to maximize the posterior probability of the label given the data. Since word frequency provides a discrete distribution, we used Multinomial Naive Bayes [5].

For each of these models used, we defined a grid over important parameters that governed aspects of the models like regularization, learning rate, loss functions, and more. These grids are shown in ModelEvaluation.ipynb. For each model-dataset pairing, Scikit-Learn's GridSearchCV with 5-fold cross-validation was used to search over the parameter space. Then, based on ideal validation accuracy, the best model-dataset pairing for each model was further optimized in a finer grid search (these are shown in FinalizingResults.ipynb). The first, more coarse search gave us a rough understanding of the range of optimal parameters for each of our models. The second, finer search was computed in the close neighborhood of the optimal parameters to further improve the models. We would then use these parameters to compute our test accuracy. Because we used GridSearchCV, this automatically incorporates a stratified k-fold cross validation technique to ensure each fold has balanced amounts of each label when defining training/validation splits. All models were implemented with Scikit-Learn estimators.

After this optimization process was complete, we also explored different ways to ensemble the best of these optimized models. First, we tried hard/soft majority voting with mlxtend's EnsembleVoteClassifier. This implementation was used over Scikit-Learn as it let us use pre-trained models, thus saving us the computational cost of refitting. To weight the votes of each model, their normalized validation accuracy was used. Because we'd already identified the optimal parameters for these models, cross-validation wasn't necessary, and we could make predictions directly on the test set to evaluate performance. All models naturally extended to hard voting, but Scikit-Learn's LinearSVC class does not output probabilities. Thus, we trained an SVC classifier with a linear kernel (and the same regularization parameters) instead. We also tried mlxtend's StackingCVClassifier, with Scikit-Learn's default Logistic Regression implementation as the meta-learner. This uses five-fold cross-validation to generate predictions from each ensembled model on the validation data. The Logistic Regression meta-learner then takes these predictions as input and trains with the actual labels. This allows it to assign weights to each model's predictions.

5 Results

From the process explained above, the results for the ideal model-dataset pairing are reported below for each model. An example of this process is given in Figure 2. In order to ensemble the models, the best performing models were incorporated into the vote. For IMDB, this was Logistic Regression, Naive Bayes and Linear SVC. For news, this was Logistic Regression, Naive Bayes, Linear SVC, and Random Forest. For the stacking ensemble, this used all the optimized models. In Table 2 we show the average training and validation accuracy for the fine grid search, as well as the test accuracy with the optimized model.

Model	Optimal Datasets	IMDB Train/Val/Test	News Train/Val/Test
Logistic Regression	movie_binary_ngrams news_binary_ngrams	0.99820/0.90232/0.90312	0.97371/0.73466/0.68016
Multinomial Naive Bayes	movie_binary_ngrams news	0.96580/0.88972/0.88168	0.90092/0.74668/0.69610
Decision Tree	movie_binary_ngrams news	0.79396/0.73508/0.74568	0.47658/0.37405/0.36073
Linear SVC	movie_binary_ngrams news_binary_ngrams	0.99656/0.90296/0.90400	0.96853/0.74942/0.69238
Random Forest	movie_binary_ngrams news_binary_ngrams	0.98516/0.80452/0.80880	0.70355/0.51750/0.48699
Adaboost	movie_binary_ngrams news_ngrams	0.86828/0.85160/0.85270	0.64928/0.42081/0.45154
Hard Voting Ensemble	movie_binary_ngrams news_binary_ngrams	0.99668/NA/0.90468	0.97030/NA/0.69503
Soft Voting Ensemble	movie_binary_ngrams news_binary_ngrams	0.98303/NA/0.90560	0.96129/NA/0.68919
Stacking Ensemble	movie_binary_ngrams news_binary_ngrams	0.97628/NA/0.90384	0.33162/NA/0.26899

Table 2: Best dataset and corresponding optimized accuracy results for each model

Overall, the best results for the movie datasets came from ensembling the models with soft voting wherein we reached 0.90560. For the news articles, Multinomial Naive Bayes had the best performance, with 0.69610. This allows for a total accuracy of 0.85710 across all the testing data. All models were trained with a random seed of 42 to ensure reproducibility; their optimal parameters and the process of ensemble training can be viewed in Ensembling.ipynb. The data processing is easily reproduced from DataProcessing.ipynb.

To gain more insight into the optimal models, their confusion matrices are shown in Figure 3. The soft voting ensemble misclassified positive and negative examples in roughly equal quantities. The Multinomial Naive Bayes model showed more interesting behavior in its confusion. It struggled with topics that might be considered less specific, like sci.electronics which was confused with many of the computer articles. These computer articles were also less discriminatory; comp.graphics, comp.os.ms-windows.misc or comp.sys.ibm.pc.hardware were often mixed up. Finally, the less well-represented classes (as seen in Figure 1) all had poor accuracy, such as with talk.politics.misc, and talk.religion.misc. This makes sense as Naive Bayes would have had less data to learn accurate distributions. Interestingly, the alt.atheism tag, another less common tag, was over-represented and was predicted more often than it should have been. An explanation could come from the fact that The optimal hyperparameters for the model had fit_prior set to False, meaning it used a uniform prior for each label. As such, the uniform prior would have assigned a higher probability to the atheism label than is warranted, causing it to predicted more often.

6 Discussion and Conclusion

Overall, this project demonstrated how important the choice of model is when approaching a machine learning problem. Training and optimizing a plethora of different models can be computationally expensive, but as indicated by our results this yields improved performance. The models we trained reported a wide spread of testing accuracy, and also showed how more complex models don't necessarily lead to improved performance. A model like Random Forest is great at capturing complex trends in data, but was actually outperformed by Multinomial Naive Bayes on both datasets.

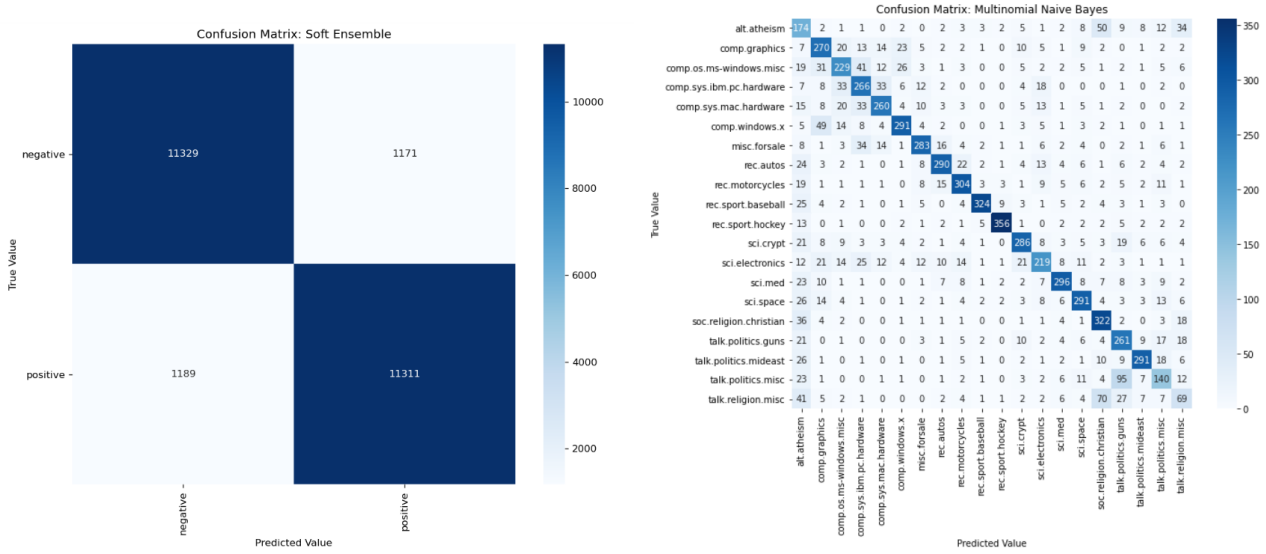


Figure 3: Left: confusion matrix for soft ensemble on movie_binary_ngrams. Right: confusion matrix for Multinomial Naive Bayes on news.

This also showed how crucial it is to work on feature engineering. The results you obtain are bounded by how effectively your data represents the problem. In almost every model, performance was improved by using ngrams or by using binary representations, showing how impactful feature engineering can be. Creating optimal classifiers requires in-depth analyses of the types of data and the roles that different classifiers play in classification.

Finally, this experiment also showcased the value of ensembling optimized models, since for both datasets, the ensembles gave some of the best results. Especially for datasets like these which have a huge amount of features and are prone to overfitting, ensembling can be a valuable tool in combating variance problems.

One limitation was the lack of time and computational resources to explore additional models and feature engineering techniques. In models like Random Forest and Decision Trees, these have a huge number of parameters to optimize over. Given more resources, a more comprehensive grid search could have been conducted to improve performance. Additionally, adding more models like Gradient Boosting, Multilayer Perceptrons, and different Neural Network architectures could have improved our ensembles, or offered improved individual performance. Alternate feature engineering techniques like using word embeddings would have also been interesting extensions; for example, encoding movie reviews as 100 learned word vectors (from libraries like FastText) might better represent the content of the review.

In conclusion, through extensive grid searches across different models and feature engineering techniques, performance on the IMDB Movie Review and 20 Newsgroup datasets was optimized. Four different feature engineering techniques, six different base models, and three different ensembling methods were all analyzed. A final accuracy of 0.90560 on the IMDB Movie Reviews, and 0.69610 on the 20 Newsgroup dataset was achieved. This gives a total accuracy of 0.85710.

7 Statement of Contributions

Kenji Marshall did the data processing and feature engineering, and contributed to the model optimization. Abdulla Alseiari and Mian Hamza developed and optimized most of the models. Kenji Marshall developed the ensembled models. Everyone contributed equally to the writing of the report.

References

[1] Mita K Dalal and Mukesh A Zaveri. Automatic text classification: a technical review. *International Journal of Computer Applications*, 28(2):37–40, 2011.

[2] 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*, 12:2825–2830, 2011.

[3] Steven Bird, Ewan Klein, and Edward Loper. *Natural Language Processing with Python*. O'Reilly Media, 2009.

[4] Inc. Rype. 100 most common english words (learn 85% of english in 100 days). URL `https://www.rypeapp.com/most-common-english-words/`.

[5] Trevor Hastie, Robert Tibshirani, and Jerome Friedman. *The Elements of Statistical Learning*. Springer Series in Statistics. Springer New York Inc., New York, NY, USA, 2001.

[6] Christopher M. Bishop. *Pattern Recognition and Machine Learning (Information Science and Statistics)*. Springer, 1 edition, 2007. ISBN 0387310738. URL