Abstract

The object of this project was to classify Reddit comments input words into their respective subreddits. In order to do this, our task was to extract features from the comments and implement various classification models from sci-kit learn such as a Support Vector Machine (SVM with the SGDClassifier), Logistic Regression, Ridge Classifier, Complement Naive Bayes and a Gaussian Naive Bayes. We also evaluated the effect of employing sci-kit learn's ensemble methods, namely VotingClassifier (stacking) and BaggingClassifier. Finally, we implemented a Bernoulli Naive Bayes which uses binary occurrence matrix instead of real values. The dataset that these models were tested on has 30000 datapoints, each fell into one of 20 possible subreddits. Our testing shows Complement Naïve Bayes as the best base model, and a stacking ensemble of all the models slightly beat our best base model.

Introduction

Reddit is a social networking site in which users can act in a read-only relationship without an account as well as read-write form where users have anonymous names and contribute to communities. Communities are broken down by topic and these operate under the pseudonym "subreddits" where users can contribute to a specific topic. These subreddits are rarely altered by the administrators and are community-run. Ultimately, the goal was to evaluate various models and preprocessing techniques in order to optimize text classification by specific subreddits.

The idea behind logistic regression for a binary classification is to designate a binary value Y as preferred, and then estimating the P(Y) as follows

$$P(Y) = \frac{1}{1 + e^{w^T x}}$$

X represents the vector containing the data and W is the vector containing the weight. The outcome of this function suggests that Y is true if the result is > 0.5. The goal is to minimize the mean squared error across all the data points. In order to do so, we optimize for the mean squared error and iteratively re-run the algorithm using the new weights at each iteration.

$$\vec{w} := \sum_j \alpha_j c_j \vec{d}_j, \;\; \alpha_j \geq 0,$$

Furthermore, a Support Vector Machine (SVM) was implemented. The basic idea here is that we look at the extremes of a dataset and create a decision boundary, which is also referred to as a "hyperplane". The formula is as follows:

The ~dj is such that αj is > 0 is the support vectors, since they wrap the decision boundary ~w which represents the hyperplane. The goal is to have this "wrap" which is represented as the support vectors be as large as possible. Ultimately, however, the accuracy of the decision boundary is extremely important to consider since a sub-optimal decision boundary could lead to misclassification. Once this is determined, support vectors are drawn on either side of the decision boundary and intersect the aforementioned most extreme points of the classes. The "extremities" are typically denoted as "D+" for the closest positive point and "D-" for the closest negative point on either side of the decision boundary. These support vectors surround the decision boundary as a sort of fail safe during classification. This was implementable because the classes were linearly separable.

Related Work

In order to further understand the performance and key factors of each text classification method, further research was performed. An example of this is, "Text Classification Using Support Vector Machines"[1] which outlines the role of the hyperplane and support vectors as a function of extreme data points and suggests incremental data mining as a proposed solution. Furthermore, research was done in regards to sentiment as an input and how it could skew results. In the below research, the sentiment problem was not considered for this implementation, however, "Thumbs up? Sentiment Classification using Machine Learning Techniques"[2], suggests that text classification could be skewed by not including sentiment. Sentiment could play a role in text classification since the same word could have multiple meanings and therefore fall into different classes, also known as subreddits, depending on the sentiment.

Dataset and Setup

Our task was learning a classification model based on the 70000 sized training set. They are evenly distributed into 20 categories/classes. We cleaned and processed the data into a sparse matrix of word features. We removed URLs from each comment, which we suspected would create noise, and lemmatized all the words[3] (e.g. the words loves, loved, loving were all presented as love). We constructed features using sci-kit learn's TFIDF Vectorizer to re-weigh features to transform the raw data to sparse matrix. This allowed us to appropriate weigh words based on the information they carry based on their rarity in our training corpus. Furthermore, we included n-grams up to size 3 so that our models could also consider words that have different meanings when occurring next to other words. Including n-grams caused our feature set to grow exponentially, so to help improve speed and prevent overfitting, we used the SelectKBest variable ranking tool to reduce dimensionality to the 100000 best features based on a chi^2 test. For our own implementation of Bernoulli Naïve Bayes, we applied the same preprocessing steps, except with a Count Vectorizer to output a binary sparse matrix appropriate for the model.

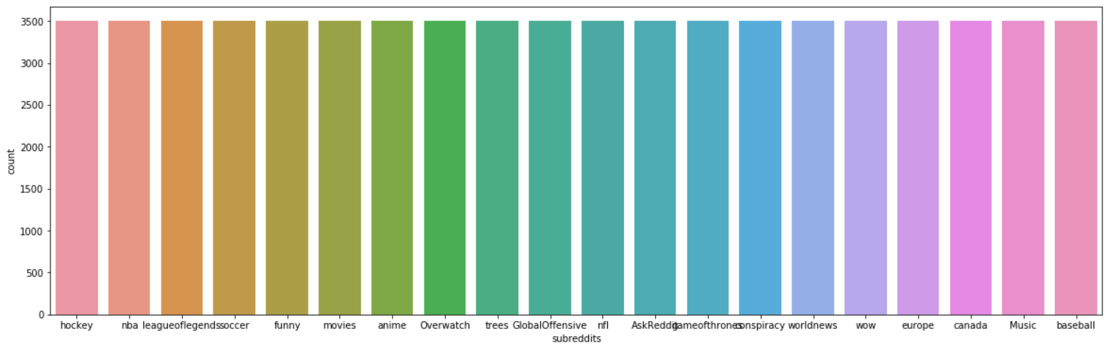

Figure 1: Subreddits and their distributions in the training data

[1] Sarkar et. Al "Text Classification using Support Vector Machine." *International Journal of Engineering Science Invention*, 2015. http://www.ijesi.org/papers/Vol(4)11/G411033037.pdf

[2] Pang, Bo, Lillian Lee, and Shivakumar Vaithyanathan. "Thumbs up?" *Proceedings of the ACL-02 Conference on Empirical Methods in Natural Language Processing - EMNLP 02*, 2002. https://doi.org/10.3115/1118693.1118704.

[3] Simon. "Lemmatize Whole Sentences with Python and Nltk's WordNetLemmatizer." Simon's blog, July 2, 2018. https://simonhessner.de/lemmatize-whole-sentences-with-python-and-nltks-wordnetlemmatizer/?source=post_page-----c1bfff963258----------------------.

Proposed Approach

We derived our final score from the following classifiers: Logistic Regression, Complement Naïve Bayes, Multinomial Naïve Bayes, SGDClassifier and Ridge Classifier. We evaluated bagging, boosting and stacking ensemble methods on each of our base models. Since we were working on a relatively large sample sets with hundreds of thousands features, models that train or predict at a greatly slower speed were not efficient. Therefore we abandoned the implementation of Decision Tree and Multi-Layer Perceptron. Then we decided not to derive the task from a single model. Consequently, we drove the stack ensemble method : set up a meta-model to output predictions based on the different weak models result. Those base estimators were relatively fast and performed good on cross validation tests. The meta model we came up with at the end reached the highest accuracy on the k-fold cross validation sets among base classifiers. In addition, the one-versus-rest schemes did a better job in the logistic regression model, and we emphasized on weighing features, so we switched the solver to 'liblinear' with l2 regularization.

Results

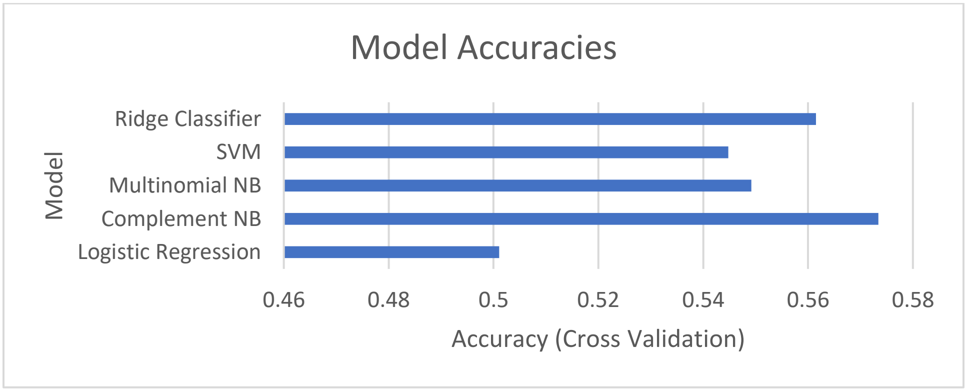

Figure 2: Model cross validation accuracies

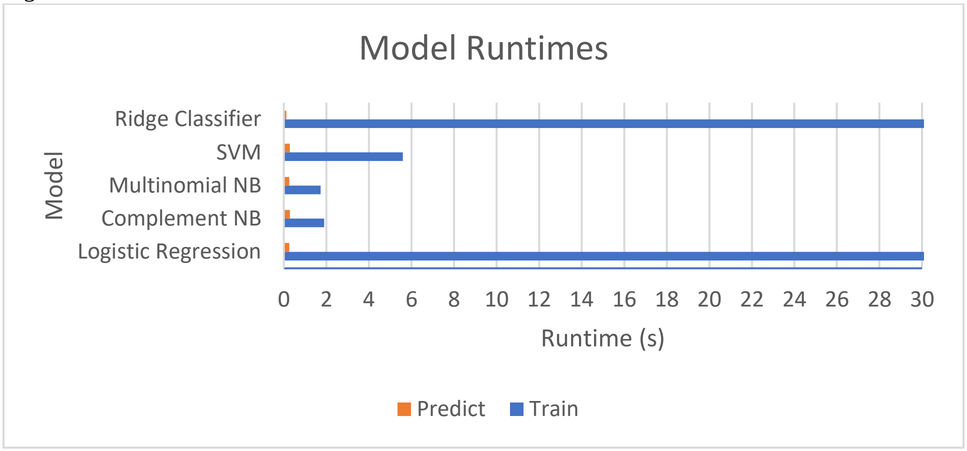

Figure 3: Model runtimes in seconds

The model that resulted in the best prediction accuracy was Complement Naïve Bayes, that gave us an accuracy of 56.177% on the test set.

We were surprised to see that more complex base models such as Logistic Regression and SVM, even after fine-tuning of hyperparameters, did not perform better than any of our Naïve Bayes models (Figure 2). Therefore, model complexity is generally not a good estimator for performance, and models should be chosen with knowledge of the task.

While performing bagging with sklearn's BaggingClassifier, we saw a consistent increase in cross validation accuracy as we increased the number of estimators in the ensemble, reporting a 61% cross validation accuracy when used on Complement Naïve Bayes. However, when finally used to make test set predictions, our suspicions of overfitting were confirmed when an even lower accuracy of 51% was reported. Thus, bagging should not be relied on for large accuracy gains, but rather for marginal optimizations.

Stacking with VotingClassifier proved to be a good option that allowed us to improve accuracy without driving up computational cost. Because all our base models could be trained relatively quickly, we were able to easily explore the best subset of models to be employed in a majority vote classifier. A combination of all base models proved to perform the best in cross validation resulting in an average accuracy of 61.31%, however, potentially due to overfitting, it only beat the base Complement Naïve Bayes model slightly at a test set accuracy of 56.377%

Theoretically, AdaBoost would increase model accuracy by training multiple weak learners that adapt to the errors in previous iterations. However, during our implementation the computational cost of training such learners was too great for our task. For our models, a very large amount of weak learners were required to increase accuracy sufficiently and we finally decided to stop evaluating with AdaBoost in the interest of time.

In comparing the runtimes of our models, we can see that Naïve Bayes models trained the quickest, and as expected with eager models, they were all able to compute predictions quickly (Figure 3). It should also be noted that our from-scratch implementation of Bernoulli Naïve Bayes prediction time was much greater, reaching multiple minutes, as we did not leverage vector operations when computing feature likelihoods, causing a linear increase in runtime on the number of features in the dataset.

Conclusion

After various experiments with model selection and ensemble methods, we found that the best base model for Reddit comment classification was Complement Naïve Bayes. We were able to beat base models' accuracies with the use of bagging and stacking methods, however our models did suffer from overfitting. Future studies could further investigate optimal model combinations for stacking, or more thoroughly explore the relationship between number of estimators in a bagging classifier and test error.

Statement of Contributions

Tian Bai

- Data preprocessing
- Sklearn ensemble and base method implementation

Elie Climan

- Sklearn base model implementation
- Research into similar works

Alex Wang

- Sklearn base model implementation
- Gathering and comparing model performance data
- Bernoulli Naïve Bayes implementation