COMP 551 Project 2 Report

Group 17
Doreen He: 260761484, Vitaly Kondulukov: 260799450, Tristan Shoemaker: 260640998

October 21, 2019

Abstract

Text analysis and classification is an important class of problem in machine learning. Supervised text classification is an automated process of classification of text into predefined categories. We have explored this with several different machine learning algorithms: Multinomial Naive Bayesian classification (NB), Logistic Regression (LR), Random Forest classification (RF) and Support Vector Machines (SVM). In this report, we applied these methods to the classification of forum comments from `reddit.com` into specific subforums. Additionally, we have used an ensemble model as a combination of selected models was entered into a Kaggle competition. We have found that the ensemble model is most accurate, with an accuracy of 59.7 %, tested with 2-fold cross-validation. Of the single models, Multinomial Naive Bayesian was most accurate, with an accuracy of 59.0 %, tested with 2-fold cross-validation.

Introduction

`Reddit.com` is a forum and content aggregator. Our goal is to classify a reddit comment as coming from one of 20 selected subreddits (subforum communities). The training dataset is a collection of raw comments from the website with a maximum length of 10,000 characters, and may include ASCII characters, links, emojis and formatting characters, as well as which subreddit the comment came from. We are testing a variety of models and text pre-processing steps in order to maximize the classification accuracy, and also determine the effectiveness of different model types on one classification task. All models and processing algorithms except for NB were based on implementations in `scikit-learn` [1].

Related Work

Jee Ian Tam performed a similar subreddit classification task in 2017 [2]. Twenty subreddits with a large number of comments in the top fifty subreddits were chosen as the classification target, different from our choice of subreddits. They used Recurrent Convolutional Neural Networks (CNN) and Attention-Pooled CNN, finding that Recurrent CNN with GRU (Gated Recurrent Units) gave the best accuracy with an F1 accuracy score of 0.53. We are using "simpler" models, but have achieved a greater accuracy of 59.7 %.

Dataset and Setup

The reddit comment dataset includes a comment id, the comment text and the subreddit from where the comment came. There are 20 subreddits/categories: hockey, nba, leagueoflegends, soccer, funny, movies, anime, Overwatch, trees, GlobalOffensive, nfl, AskReddit, gameofthrones, conspiracy, worldnews, wow, europe, canada, Music, baseball. The dataset is very balanced with each category having 3500 sample comments, for a total of 70000 samples.

Proposed Approach

Preparation and Feature Selection

Feature engineering and selection are two central tasks in data preparation for text classification. In this work, constructing suitable features from given features according to each model leads to an improved predictive performance.

We followed a few common practices to process, tokenize and vectorize the raw text data by their TF-IDF scores. In order to capture a wide range of features from the dataset, we built word level, 1 to 3 gram word level and character level TF-IDF vectorizers. With large amount of features generated from the text, a large number of irrelevant features increases the training time exponentially and increase the risk of overfitting. To minimize the overfitting issue and maximize our test data accuracy, we tried the following feature reduction approaches:

- Removing stopwords (commonly occuring words)
- L2 regularization
- Removing words appearing in more than 60 % of comments
- Removing words appearing less than 2 times among all comments
- Lower casing all words
- Lemmatization
- Chi-squared feature extraction

We found out that the simple TF-IDF vectorizer works very well during learning and produces higher accuracy overall compared to a count vectorizer. However, although we expected lemmatization might reduce some nonuniformity thus reduce more noise, our experiments indicate that the influence of word lemmatization is negative rather than positive.

On the other hand, stopword removal improves the classification accuracy in all models. To further scale down the amount of irrelevant features, one can apply a wide range of feature reduction methods, including mutual information, F-test, chi-squared test, etc. In our case, chi-squared is used due to its relatively inexpensive computation and fast performance. By doing so, we significantly reduced the dimension of classified comments and compressed the classification time. Moreover, many machine learning algorithms perform better or converge faster when features are on a relatively similar scale and close to normally distributed. Some algorithms that we choose to use are in this category, including logistic regression and support vector machines. Therefore, we also performed standardizing/normalization to help features arrive in a more digestible form for these algorithms.

Model Selection

There are many different choices of learning models. We experiment with five different models in this classification task, four of which are base models, and the last a stacked ensemble model. Naive Bayes is a probabilistic classifier which makes strong assumptions about the normal distribution and independence among predictors, but is known to be well suited to text classification. Logistic regression is well understood and general robust model which should provide a baseline of classification. Support Vector Machines is a representation of the examples as points in the space, mapped so that the examples of the separate categories are divided by a clear gap that is as wide as possible. The random forest model is a type of ensemble models, in particular a bagging model. It is part of the decision tree family. We tested the effectiveness of these types of classification models on the task and results are discussed below.

Model Evaluation and Hyper-parameter Tuning

All of our models are trained on a 70% training set and tested on 30% validation set for hyper-parameters tuning. Since the `reddit_train.csv` and `reddit_test.csv` have size ratio 7:3, we think it would be suitable to keep the ratio close to train-test ratio, so the accuracy of the each model is estimated on 2-fold cross validation. A number of parameters is fine tuned for each model to get a best fit classification. In particular, max tree depth and number of estimators in random forest, learning rate and max number

of iteration in logistic regression, Laplace smoothing α parameter in Multinomial Naive Bayes, and L2 penalty parameter C and loss funtions in Linear Support Vector Classification are adjusted according to each model's performance on the task.

Results

Random Forest Classifier

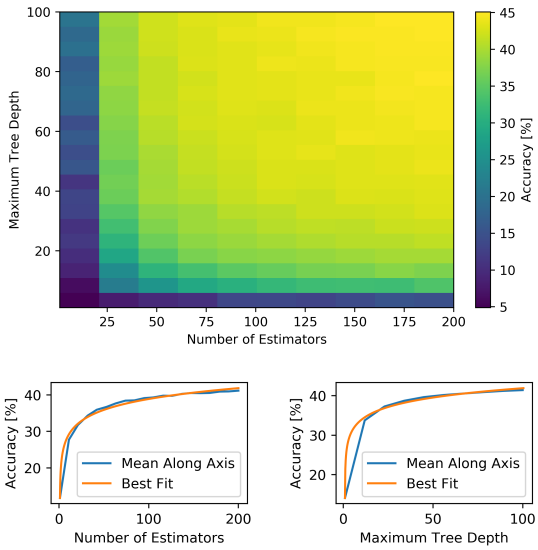

Figure 1: Grid search results for random forest model, showing the effects of the maximum tree depth and number of estimators (trees) parameters. Subplots show fits along one axis of a log function of form $a \log(bx + c)$, where a, b and c are unbounded parameters.

The `RandomForestClassifier` implemented in SKLearn has two major hyperparameters: the number of estimators, in this case trees, and the maximum tree depth. Looking to explore this parameter space we did a grid search which is presented in Figure 1. As expected, increasing the depth or number of trees both increase the model accuracy, however this also rapdily increases the computation time. Two plots of the mean value along each of the axes show however that the gained accuracy is roughly logarithmic for each parameter. This allows us to select maximum depth and number of estimator values that maximize the accuracy for a given training time. However, we found that decision tree based classifiers (decision trees, random forest, extra random trees) had generally low accuracy of 49 % and high training times compared to NB or LR for any hyperparameter setting.

Logistic Regression

The `LogisticRegressionClassifier` was implemented in SKLearn with multiple tweaked hyperparemeters, but the parameters giving the highest accuracy were the C parameter for regularization and the `fit_intercept` parameter. The default state of such model was when the solver was set to `'liblinear'`, `multi_class` was set to one versus the rest and regularization was set to l2. By tweaking the C parameter that controls the regularization strength and by setting `fit_intercept` to false, the highest average 5-fold accuracy was observed with C=1.0 of 55.5 %. This accuracy was observed with other parameters being set to default except the latter two.

Naive Bayes Classifier

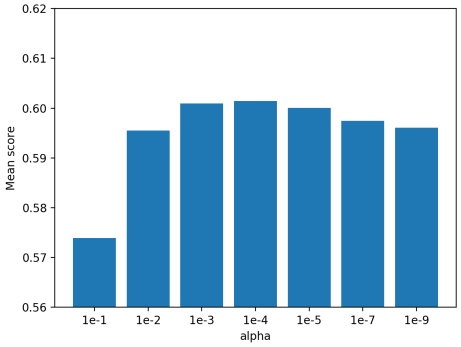

Figure 2: Optimal parameter search result for the multinomialNB model, showing the prediction accuracy with respect to the Laplace smoothing α parameter.

The `NaiveBayesClassifier`, specifically multinomial Naive Bayes from SKLearn, is the most accurate single model among all the classification models that we experimented with an average accuracy of 59.0 % with $\alpha = 10^{-4}$. It is not only accurate, but also extremely fast due to its low computation cost, which makes it very efficient working on our large dataset.

For Naive Bayes to work, every probability estimate has to be non-zero; consequently, Laplace smoothing, also known as Lidstone smoothing, must be applied to incorporate a tiny probability correction in all features. In SkLearn `multinomialNB` implementation, the α parameter for Laplace smoothing can be adjusted. In general, a small alpha is required for the true probability to not be affected. However, as shown in Figure 2, initially a decrease in alpha does lead to higher accuracy; however, the model performance peaks at $\alpha = 10^{-4}$ and decreases as alpha approaching to zero. Moreover, in addition to SKLearn Naive Bayes classifier, we had to implement our own. Our implementation does not run as fast as SKLearn's however the accuracy is very comparable. With 5-fold cross validation, on the full training set vectorized into binary features, we got a 55.37 % accuracy score. SKLearn's vanilla Bernoulli Naive Bayes implementation scored 56 % on the same pre-processed data set. The time required to train and evaluate our model is 118 seconds compared to SKLearn 0.26 seconds. The difference comes from the utilization of python for-loops in our NB. The fit function is actually comparable to the library's time, which is 0.104 seconds. But when we try to predict a lot of data examples at the same time, the python loop in our implementation goes through each validation example and calculates a scoring value for each class. On the other hand, SKLearn uses a very clever NumPy array implementation. NumPy arrays utilize parallelism and C++ lightning loops, which makes the scoring much faster than in our model. The Naive Bayes classifier run time comparisons were done on an Intel Core i5-7500.

Support Vector Machine

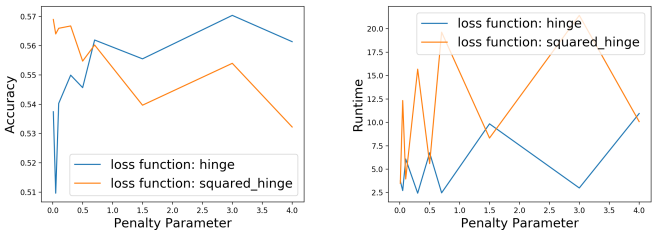

Figure 3: The upper figure shows the grid search result for the `LinearSVC` model accuracy with respect to the choice of loss function and regularization parameter; The lower figure plots the runtime relationship between hinge and square hinge loss functions.

The `SupportVectorMachine` is a set of supervised learning methods that are effective in high dimensional spaces. It uses a subset of training points in the decision function, and therefore is highly memory efficient. SkLearn provides various models from support vector machine family that are capable of performing multi-class classifications, namely `SVC`, `NuSVC` and `LinearSVC`. Among those models, we selected `LinearSVC` for our task due to its flexibility in the choice of penalties and loss functions. The natural question that arises is how the choice of loss functions and scale of regularization parameter affects the model performance. Our model is built with an L2 regularization penalty. We again run a grid search to find the influence of loss functions and regularization penalty parameters. It is illustrated in Figure 3 that the squared hinge function performs well with a small parameter, whereas the hinge function produces similar accuracy when the penalty parameter is large. In our case, we optimally adjusted the parameter setting to a penalty of 0.3 with the squared hinge loss function. In terms of the run time, squared hinge loss function generally takes longer than hinge loss function, and both loss functions take slightly more time as the penalty parameter increases, however the benefit of using the loss function outweighs the time penalty. Additionally, it is interesting to note that the runtimes for two loss functions repetitively converges and diverges over various intervals on penalty parameter. Moreover, to explore more potential of linear SVC, we also experiment on `SGDClassifier` (SGD), which implements regularized

linear SVC models with stochastic gradient descent learning: the gradient of the loss is estimated each sample at a time and the model is updated along the way with a decreasing learning rate. With the same parameter setting, using 0.3 penalty parameter and squared hinge loss function, SGDClassifier gives an average accuracy of 56.1 %, which is slightly less than LinearSVC model.

Kaggle Model: Weighted Voting Classifier

Finally, we construct an ensemble voting method, and combine all the results of all the predictions based on previous mentioned models. Since we have tested all the selected base models alone, it is evident that some models are more qualified than others in terms of prediction accuracy. Weighing the decisions of those models more heavily may further improve the overall performance. Evidently, Naive Bayes is the best performing single model; therefore, it's accuracy is set to be the base line and it's prediction is given the most weight. The random forest model is discarded due to lack of accuracy. The voting weights distribution is given in Table **??**.

	NB	LR	SGD	SVM
Weight	4.8	1.0	2.0	2.0

Table 1: Weighting in the KM.

On 2-cross validation, the ensemble model predict with 59.4% accuracy, which is 0.4% higher than the best single base classifier, namely the multinomial Naive Bayes model. With this model testing on the entire training set and predict on Kaggle test set, it is able to achieve 59.7% accuracy. The final training time, evaluation time and accuracy of the discussed models are presented in Table 2.

	NB	LR	RF	SVM	SDGC	KM
TT [sec]	0.222	12.1	15.5	27.9	16.2	27
ET [sec]	0.01	0.01	0.6	0.3	0.5	0.3
Acc [%]	59.0	55.5	43.8	57.0	56.1	59.7

Table 2: Comparison of training time (TT), evaluation time (ET) and 2-fold cross-validation accuracy (Acc) between the four evaluated models. All models were run with parameters to maximize their accuracy.

Discussion and Conclusion

Based on our study on various models and combination of models we have learned several key points. Firstly, machine learning classifiers are application dependent and inconsistent in performance. Classifiers must be selected to match the problem, and more complex/advanced classifiers do not necessarily perform better as evidenced by the success of Multinomial NB. Additionally, The same classifier with a slight change in the hyper-parameters can lead to significant variation in classification results. Techniques such as grid searches are invaluable in optimizing classifier accuracy, and hyper-parameters must be re-tuned if the feature set is significantly altered.

We have also observed that feature selection is key to improving accuracy. Both adding features to provide more information to the classifiers, but also feature reduction. Feature reduction as detailed in the "Proposed Approach" section is important both for increasing the speed of classification and improving the accuracy. Increasing the classification speed allows for faster optimization of the hyper-parameters and more time to experiment with different classifiers, which also results in improved accuracy. This is even more important as the size of the dataset increases.

Finally, using a ensemble of models allows for each model's strengths to be combined. Combining our models resulted in a small but important increase in our classification accuracy that allowed us to place significantly higher in the Kaggle competition. However before combining, individual models must be thoroughly tested to optimize hyper-parameters and understand their strengths and weaknesses. The combination method must also be well chosen. We found success with a weighted voting classifier that allowed us to prioritize the the models that we know did well, primarily NB.

Statements of Contribution

- Vitaly Kondulukov: Worked on Naive Bayes implementation and Logistic Regression exploration.
- Tristan Shoemaker: Worked on random forest model, parameter optimization, figure 1 and part of text.
- Doreen He: Worked on feature preparation and selection. Explored different feature engineering methods and classification models. Worked on the final model and part of text.

References

[1] F Pedregosa et al. "Scikit-learn: Machine Learning in {P}ython". In: *Journal of Machine Learning Research* 12 (2011), pp. 2825–2830.

[2] Jee Ian Tam. "Classifying Reddit Comments by Subreddit". In: 2017. URL: https://www.semanticscholar.org/paper/Classifying-Reddit-comments-by-subreddit-Tam/552d9a60fd4e31273d2c1b90ff4de38bda2cb174