COMP 551 Applied Machine Learning Miniproject 2 Report

Zhi Wen, Yunfei Cheng, Xiaobin Shang

Abstract

The task explored in this project is multi-class Reddit comment classification. Specifically, there are twenty subreddits that each Reddit comment in the dataset is possibly extracted from, and the task is to predict which subreddit a given comment is from. A variety of approaches for building classifiers were experimented. We particularly explored three types of classifiers: two-stage classifiers that embed comments and classify basing on those embeddings, end-to-end classifiers that embed comments inherently using their own vectorizers trained jointly, and ensemble classifiers that utilized other classifiers' outputs. We managed to build a classifier that performs slightly better than the TA baseline, scoring 57.51% in terms of accuracy.

1 Introduction

In this miniproject we developed multiple classifiers that classify a comment from Reddit into one of twenty possible subreddits the comment was extracted from. The dataset we trained and tested on was balanced. The approaches we experimented can be roughly divided into three classes: those that first generate vector representations for comments and then classify using those vectors as inputs, those that directly take textual comments as inputs, and the ensembles of multiple classifiers.

The first type of approach involves two stages, namely embedding comments and classifying, which we assumed to have roughly equal importance. We experimented with a number of embedding methods, including scikit-learn's TF-Idf vectorizer, CountVectorizer, Fasttext[1] and doc2vec[2]. For classifiers, we tested a range of classifiers including Bernoulli Naive Bayes implemented from scratch, multinomial Naive Bayes, logistic regression, random forest and Support Vector Machine (SVM).

The second kind of approach, which was more end-to-end, included popular deep learning models such as BERT[3]. Because of the huge number of parameters and the complex structures of these models, they require large corpora to train, such as the Wikipedia corpora. Fortunately, most of the models have open-sourced their codes and provided pre-trained models as well. These types of models have their own vectorizers which were jointly trained during pre-training.

We built two types of ensemble classifiers with different approaches. The first approach was stacking individually trained classifiers and training a meta-classfier that takes each of their classification results as a input feature. The second one, a rather boosting-like approach, was training a classifier on all 20 classes. After observing which classes performed significantly worse, a subsidiary binary classifier was constructed for each of the under-performing classes to predict whether a comment belongs to that class. Finally, subsidiary classifiers were stacked on top of the main classifier in a similar manner as the first approach.

The most surprising finding was how difficult it was to reach the 57.33% TA baseline, especially considering the scope and variety of our experimentations. For instance, while it is intuitive to assume that classifiers built on state-of-the-art deep learning models like BERT should achieve a higher accuracy than the TA baseline, it was in fact at par with the TA's multinomial Naive Bayes model at best even after tuning multiple hyper-parameters. This shows that multi-class classification problem with small dataset is a nightmare even to state-of-the-art pre-trained models.

2 Related Work

Much of the NLP research community's attention has been focused on word and document embeddings, which can be further utilized for downstream classification tasks. There are many off-the-shelf word embedding techniques, such as word2vec[4] and Fasttext[1]. To generate sentence embeddings from word embeddings, one common practice is averaging all embedding of words in a given sentence, which is the method we adopted. In addition to word embeddings, one particular sentence embedding technique we explored was doc2vec[2]. In contrast to word embeddings, document embeddings, or in our case sentence embeddings, has two advantages: that they inherently preserve the order of words in sentences, and that they have fixed dimensions for sentences of varied lengths.

Among all pre-trained general-purpose language models, BERT is a very powerful one pre-trained on Wikipedia[3]. After being published, BERT and its variants have quickly proved their powerfulness by achieving state-of-the-art in a wide range of NLP tasks, such as sentence classification and question answering. As a language model, BERT could be easily fine-tuned on other datasets for specific downstream tasks by adding a few layers on the top.

3 Dataset and setup

The original dataset consists of a training set and a test set, with about 70,000 samples and 30,000 samples respectively. The dataset is balanced in that there are roughly equal number of samples from each category. The training set is further split into train-train, train-valid, train-test sets with the ratio of 0.8: 0.1: 0.1 for internal testing before submitting to kaggle. The same preprocessing procedure was applied to all datasets to ensure fair comparisons among different classifiers and consistency in training and testing phases. In specific, the preprocessing procedure includes converting to lowercase, removing stopwords, removing punctuations and lemmatization, in the given order. We also experimented removing hyperlinks, but it had a negative impact on classification results, which we speculated to be because hyperlinks, though different from natural texts, also contain phrases that can give hints on which subreddit a comment belongs to. In addition, we mapped each textual 'subreddit' to a unique integer which is used in training and testing in order for classifiers to easily access.

For better performance, we adopted the pre-training concept which has proved to be powerful in other NLP tasks. We downloaded an additional dataset that contains 11,360,100 reddit comments as of December 2017 and their meta-data, including subreddits[1]. This additional dataset was used to generate Tf-IDF embeddings and to train classifiers with more samples. Surprisingly, despite the huge size of the dataset and the fact that it is also a reddit comment dataset, including it in generating embeddings and training classifiers did not result in improvement in performance. We suspect the cause is that samples in this dataset have a different distribution from samples in the original dataset. Also, because of the huge amount of additional computational burden brought by including this dataset, we were unable to test all classifiers with this dataset.

4 Proposed approach

4.1 Two-stage approaches: comment embedding and training classifiers

Tf-IDF, which stands for Term Frequency–Inverse Document Frequency, is a widely used technique for vectorizing sentences and documents. Although the concept of this method is simple, we found through trial and error, that the choice of hyperparameters, particularly the maximal word frequency and the minimal word count, had a profound impact on the quality of embeddings and in turn on classifiers' performance. Besides Tf-IDF, we also explored other off-the-shelf pre-trained word embeddings, which could then be used as input to any classifier. The intuition behind this approach is that these word embeddings or word embedding techniques better capture words' semantics than Tf-IDF and thus provides more information for classifiers. One particular embedding is [5]. This embedding algorithm claims to exploit both local and global context for generating word embedding. We tried to embed comment by averaging embeddings of each word in that comment, but classifiers using this type of embedding performed extremely poorly, giving around 11% accuracy over all classes. We were unable to identify the cause of this poor performance, so we stopped using this embedding after preliminary experiment. We also experimented with Fasttext, which is a word embedding technique proposed by Facebook[1]. Similarly, using Fasttext did not result in any gain in accuracy. In addition, we explored using doc2vec to embed sentences directly and then use the embeddings as inputs to classifiers[2]. Compared to the naive way of generating embeddings of comments by averaging embeddings of words in comments, embedding sentences using doc2vec resulted in a significant improvement in performance. Embedding each comment

[1]https://files.pushshift.io/reddit/comments/

with a vector of 200 dimensions, classifiers from sklearn package such as multinomial Naive Bayes and logistic regression all achieved around 41% accuracy, without tuning hyperparameters. However, one drawback of doc2vec is the significantly greater amount of computational resources and time needed to train the embedding model, compared to word embedding techniques. In practice it took a few hours to train for 200 iterations a model that generate 200-dimensional embeddings on the entire dataset using 12 cpus. This, combined with the fact that classifiers using doc2vec embeddings stumble at 41% accuracy, made us decided to use Tf-IDF as embedding in the end.

A Bernoulli Naïve Bayes model was implemented from scratch. Bernoulli Naïve Bayes model is a commonly used approach for text classification based on the naive Bayes assumption[6]. For this classifier, we used the binary count vectorizer from the scikit learn library to generate binary vectors for each comment. In order to test the model's performance while finish training and validation in a timely manner, we set the limits of the maximal number of features to 1000, 2000 and 5000. We found empirically that accuracy varied significantly with respect to the number of features and the size of the training data.

This model was very slow in training and testing. Our initial implementation would take half an hour to train the model on 1000 samples with 2000 features. The training time for the same setup was reduced to less than 2 minutes after exploiting numpy-based matrix operations. However, it still took about 3 hours for the model to be trained on the entire internal training set with 2000 features, and 6 hours with 5000 features.

4.2 End-to-end approaches: BERT

Apart from two-stage approaches, we also utilized a state-of-the-art language model, BERT, for our task. This approach is a reasonable attempt beacause BERT, as a generic-purpose language model, should in principle be able to adapt to specific NLP tasks, since it is believed to have captured bidirectional rules of language and semantics of words during pre-training. And it is reasonable to assume it can perform well in our task, since it was trained on a much larger corpus than the dataset in our project and therefore may better capture the essence of the English language. We investigated the practicality and efficacy of fine-tuning BERT on this project's dataset. For simpler 2 implementation, we used the PyTorch-based Transformers package[5] instead of the original Tensorflow-based implementation. We used five GPUs to facilitate computation. We adapted exemplary scripts from Transformers and built interface for pre-trained BERT model to access our datasets. For BERT, important hyper-parameters in fine-tuning include the number of epochs, learning rate, and learning rate decay. We examined how different combinations of such hyper-parameters can impact the model's performance on our internal validation set. Particularly, BERT performed on par with the TA baseline after tuning hyperparameters. However, due to BERT's limitation of input sequence length, and the fact that there are a number of comments in the corpus that have a few thousands words which is beyond the limitation, BERT can not generate its predictions on the entire test set to submit to kaggle competition.

4.3 Ensemble approaches: AdaBoost and stacking

We also tried a number of ensemble methods, hoping they can perform better than the classifiers they are based on. A comprehensive summary of combinations of classifiers and corresponding meta-classifer can be found in Table 1.

In particular, we explored one kind of stacking. Specifically, we first trained a total of 20 binary Multinomial Naive Bayes classifiers, each for a subreddit category. Then we used these classifiers to compute log-likelihood for samples, resulting in a 20-dimensional vector for each sample, which was then used as the input representation of that sample to the meta-classifier. We explored a number of models for meta-classifier, including logistic regression, SVM and multilayer perceptron. We found empirically that the choice of model structure had little impact on performance, and thus for computational considerations we used logistic regression as the meta-classifier.

5 Results

Highest accuracy achieved by each type of classifier are recorded in Table 1. Abbreviations and their explanations are in Table 2. Inputs to classifiers are TF-Idf embeddings if not specified. It is surprising to find that on average stacking models, or in general ensemble methods, performed similarly as, if not worse than, using individual classifiers. One possible explanation for this is the small size of dataset limits the amount of information a classifier can learn through training, and thus the increase in complexity of a model's structure might have more negative impact than positive. As mentioned in Section 4.3, BERT is unable to generate its predictions on the test set for kaggle, since there are many sentences that have more than 512 words.

Table 1: Highest accuracy of each type of classifiers

Model	**Accuracy (%)**
MNB	**58**
Raw text, BERT	57
Stacking: MLP(20×MNB)	55
Stacking: LR(20×MNB)	54
LR	54
SVM	53
Stacking: MLP(20×MNB), with AC	51
CV, BNB	48
Stacking: Voting(MNB+LR+SVM+RF+Ada)	46
Stacking: GB(MNB+LR+SVM+RF+Ada)	46
RF	46

In particular, Table 3 shows the accuracy achieved with Bernoulli Naive Bayes classifier implemented from scratch and corresponding number of features used. As expected, performance in terms of accuracy improves when more features are used as input.

Table 2: Terms and explanation

Term	**Explanation**
MNB	Multinomial Naïve Bayes
LR	Logistic Regression
MLP	Multilayer Perceptron
AC	Additional Corpus
CV	Count Vectorizer
BNB	Bernoulli Naïve Bayes
RF	Random Forest
Ada	AdaBoost
Voting	Voting Classifier
GB	GradientBoosting

Table 3: Different number of features for Bernoulli Naive Bayes classifier and corresponding accuracy

Number of Features	**Accuracy (%)**
1000	33
2000	40
5000	**48**

Similarly, accuracy achieved with BERT and corresponding combination of hyper-parameters are shown in Table 4. There is no distinctive correlation between accuracy and any of the hyperparameters, yet in general accuracy improves with more epochs and larger learning rate. Also, introducing decay to adjust learning rate in fine-tuning appears to be helpful.

Our best performing model was built with a rigorous search of optimal hyperparameters, both of TF-Idf vectorizer and of multinomial Naive Bayes classifier. In the meantime, we also carefully experimented the difference of including and excluding hyperlinks in comments. The best performing model was built with hyperparameter $\alpha = 0.22$ while keeping hyperlinks in comments. This confirms our hypothesis that hyperlinks also contain information helpful for classification.

6 Discussion and Conclusion

In this project, we constructed and analyzed multiple classifiers for the multi-class Reddit comments classification problem with various embedding techniques and types of classifiers. More specifically, we explored the two-stage approaches, end-to-end approaches, and ensemble approaches. In addition to off-the-shelf classifiers or pre-trained models, we implemented a Bernoulli Naïve Bayes model from scratch and optimized it. Surprisingly, after extensive experiments on various classifiers and comparison of their performance, Multinomial Naive Bayes Classifier, a relatively simple model, performed the best when

Table 4: Different combinations of hyper-parameters of BERT and corresponding accuracy

Learning Rate	**Decay**	**# of Epochs**	**Accuracy(%)**
5.00E-05	0.00E+00	6	**57.33**
3.00E-05	1.00E-07	9	56.99
5.00E-05	0.00E+00	9	56.88
5.00E-05	1.00E-08	9	56.88
5.00E-05	1.00E-07	9	56.87
2.00E-05	1.00E-07	9	56.73
3.00E-05	1.00E-08	12	56.56
3.00E-05	1.00E-07	12	56.56
5.00E-05	0.00E+00	3	56.5
3.00E-05	0.00E+00	9	56.37
3.00E-05	0.00E+00	6	55.63
2.00E-05	0.00E+00	9	55.39
3.00E-05	0.00E+00	3	54.3
2.00E-05	0.00E+00	6	53.54
2.00E-05	0.00E+00	3	49.18

combined with our data preprocessing. We speculate that the incompetence of more advanced models in our problem setting is due to the multi-class nature and the small amount of data available.

7 Statement of Contributions

Zhi Wen is responsible for data pre-processing, finding additional dataset, exploring end-to-end approach with BERT, the stacking ensemble approach, and much of report writing.

Yunfei Cheng is responsible for experimenting two-stage methods, especially tuning hyper-parameters, and the boosting ensemble approach. He also contributed to report writing.

Xiaobin Shang is responsible for implementing Bernoulli Naive Bayes model and its optimization. He also contributed to report writing.

References

[1] P. Bojanowski, E. Grave, A. Joulin, and T. Mikolov, “Enriching Word Vectors with Subword Information,” *Trans. Assoc. Comput. Linguist.*, vol. 5, pp. 135–146, dec 2017.

[2] Q. Le and T. Mikolov, “Distributed representations of sentences and documents,” in *31st Int. Conf. Mach. Learn. ICML 2014*, vol. 4, pp. 2931–2939, International Machine Learning Society (IMLS), 2014.

[3] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, “BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding,” oct 2018.

[4] T. Mikolov, I. Sutskever, K. Chen, G. Corrado, and J. Dean, “Distributed representations ofwords and phrases and their compositionality,” in *Adv. Neural Inf. Process. Syst.*, Neural information processing systems foundation, 2013.

[5] E. H. Huang, R. Socher, C. D. Manning, and A. Y. Ng, “Improving Word Representations via Global Context and Multiple Word Prototypes,” tech. rep., 2012.

[6] A. Mccallum and K. Nigam, “A Comparison of Event Models for Naive Bayes Text Classification,” in *AAAI*, 1998.