Comparative Analysis of Machine Learning Classifiers for Reddit Comment Classification

Alexander Peplowski (20148127) Harmanpreet Singh (20164950)

Abstract

Text classification is a key aspect of several natural language applications in the modern world. In this project, we investigate various machine learning techniques to classify forum comments into different categories. Initially, we implement a Naïve Bayes Classifier with Laplace smoothing using a bag of word features. Additionally, we examine multiple other classifiers including Linear Support Vector Classifier, Logistic Regression, and Multi-Layer Perceptron (MLP). We also explored feature extraction approaches such as term-frequency inverse document frequency (tf-idf) and self-trained word2vec embeddings.

1. Introduction

1.1. Problem Description

There is a large amount of unstructured textual information available in today's online world. Multi-class text classification approaches can alleviate the problem of partitioning unlabelled textual information into pre-defined classes. The objective of this paper is to explore the machine learning classification methods to classify posts from popular social media forum Reddit. We begin by evaluating the perfomance of the Naive Bayes Classifier for this task before moving on to other techniques. The specific classification task will be to assign Reddit forum comments to the correct sub-forum (subreddit). The data set consists of 70,000 forum comments extracted from one of 20 subforums.

1.2. Key Findings

We discovered that the multi-class classification accuracies depend on a combination of factors, including the preprocessing steps applied and the choice of machine learning algorithm.

After finding the optimal hyperparameters for each algorithm, we obtain the best accuracy using a Voting Classifier implementing soft voting aggregating 3 models: The Stochastic Gradient Descent Classifier (SGD) with modified Huber loss, the Multinomial Naive Bayes (MNB) Classifier with an alpha of 0.25, and a MultiLayer Perceptron (MLP) Classifier with hidden layer size of 256.

Additionally, best performance was obtained when the forum text comments were vectorized into tf-idf features after splitting the text comments into words, removing special characters, and removing stop words. An accuracy of 58.09% was obtained with a 4-fold cross validation on ensembled soft voting model.

2. Feature Design

The selection of text features is a fundamental and important step for information retrieval and text classification. In developing our model, we tested several distinct methods of feature extraction to represent text data into a vector. Subsequently, these features were provided as input to a number of supervised learning classification algorithms for evaluation.

2.1. Preprocessing

Preprocessing is an essential part of any natural language processing pipeline. We started by pre-processing the raw data by transforming the sentences into lowercase and stripping special characters. Afterward, we removed stop words using the nltk[4] English stopword database. Additionally, we also tried applying stemming and lemmatization on our dataset to reduce each word to its root or lemma. Interestingly, our model performances degraded when we applied them with our models. So we decided not to drop this particular pre-processing step.

2.2. Bag of Word Features

On this processed dataset, we investigated different feature extraction approaches such as a bag of words, tf-idf, and word embeddings. First, we explored the bag of words (CountVectorizer from sklearn) text vectorizer. CountVectorizer creates a very high dimensional sparse array. After running a few experiments with the bag of words features, we found it to be inefficient because of its skewness to high-frequency words.

2.3. Tf-idf Features

Following that we employed a tf-idf vectorizer to convert text into vectors. Tf-idf word vectorizer and word-

embeddings were found to be more robust than CountVectorizer. We analyzed different parameters from Sklearn TfidfVectorizer and discovered that applying sublinear term frequency scaling and removing accents from text gave better classification scores.

2.4. Word2Vec Embeddings

Recent works have shown promising results with word-embeddings on various text classification tasks. Word embeddings [5, 6] can capture better contextual information than tf-idf as later doesn't consider neighboring words while generating features. So, we trained word2vec model using the gensim [8] implementation of word2vec across a range of window sizes and vector dimensions. Word2vec embeddings are combined by averaging the trained word vectors from each sentence to effectively represent a sentence vector, ignoring words not found in the word2vec model. Later the sentence vector was passed as input to train text classification models. We tested Neural Networks, Long Short Term Memory (LSTM) networks, and Convolution Neural Networks (CNN) models to train classification models with these word2vec vectors. The best accuracy among all these deep learning models with word2vec embeddings was around 48%.

We also experimented with weighted word vector averaging schemes, where the weights were derived from tf-idf matrix of weights learned across all classes on the entire corpus of dataset. The goal of these weights is to increase the importance of the vectors corresponding to words that are more relevant to that particular subreddit while down weighting words relative to their global frequency across all subreddits in the corpus. The accuracy from this method was approximately 42% which was poor compared to other models.

2.5. Fasttext Embeddings

Similar to word2vec embeddings, fasttext embeddings [1, 3] also work on the distributional hypothesis of data to efficiently learn text representations. Fastext embeddings are more robust to out-of-vocabulary words unlike word2vec. We improved the performance of text classification model by using word bigrams, instead of just unigrams.This is especially important for classification tasks like ours where the word order is important. We used one-vs-all classifiers with softmax loss and adaptive learning rate to optimize our model accuracies. Further, fasttext automatic hyperparamter optimization was explored to find the best hyperparameters for our dataset to build efficient models. Our best model with fasttext embeddings gave around 52% accuracy.

3. Algorithms

We investigated several machine learning based classification models for this task. We will present them individually below:

3.1. Naive Bayes

We employed two variants of Naive Bayes models: Multinomial Naive Bayes and Complement Naive Bayes. The Naive Bayes classifiers have worked quite well on our task of text classification of multinomially distributed data. Moreover, these methods are extremely fast compared to more sophisticated methods.

The Multinomial Naive Bayes was the first algorithm selected. The smoothing hyperparameter alpha was the only parameter that was tuned. The smoothing priors account for features not present in the learning samples and prevent zero probabilities in further computations. We obtained the best results with tf-idf features and alpha = 0.25, consistently yielding an accuracy around 57%.

Later we examined Complement Naive Bayes, which is an adaptation of the standard MNB algorithm that is particularly suited for imbalanced data sets. Specifically, CNB uses statistics from the complement of each class to compute the models weights. CNB produced slightly better results than MNB with a smoothing prior value of 1.0.

3.2. Multi-Layer Perceptron Classifer

The Multi-Layer Perceptron (MLP) algorithm is a dense network of nodes. The size and depth of the layers, the regularization coefficient, and the maximum number of training epochs were the hyperparameters selected for tuning.

The classifier was trained using a hidden layer of 256 units and only one epoch of training. It was found that the MLP classifier overfits the training data quickly. The best results were obtained with optimized tf-idf features when trained for only one epoch with very little regularization (i.e. low weights of 0.0001 and no dropout), hence avoiding overfitting. The average accuracy after hyperparameter tuning is this model was around 56%.

3.3. Stochastic Gradient Descent (SGD) Classifier with Modified Huber Loss

Stochastic Gradient Descent (SGD) is a simple yet very efficient approach for discriminative learning of linear classifiers under convex loss functions such as (linear) Support Vector Machines and Logistic Regression. SGD implements regularized linear models with stochastic gradient descent (SGD) learning.

We used the SGD classifier with modified huber loss from scikit learn over the LinearSVM classifier since it provides probability estimation, while the latter does not (probability prediction is useful when building a voting classifier,

to be discussed later). Additionally, it was preferred over logistic regression since it was found to provide better accuracy.

The regularization multiplier, alpha, was the only hyperparameter that was tuned.

3.4. Voting classifier

The voting classifier is a method for combining dissimilar algorithms to improve model performance. The idea behind the voting classifier is that dissimilar algorithms are likely to make dissimilar classification mistakes. Because of this, when their classification confidence scores (predicted class probabilities) are combined, they will hopefully correct the mistakes by majority vote of the other classifier.

The previously mentioned classifiers (Naive Bayes, Multi-Layer Perception, SGD Classifer) were combined by using their predicted probabilities of the occurrence of each class. This technique is known as 'soft' voting. All algorithms were given equal voting weight. The mean probability across the three classifiers was used as the basis for predicting the final class.

An alternative to 'soft' voting is 'hard' voting, where the final predicted value of the class is used to vote, rather than the predicted probability. However, this method provided worse results since it operates using less information from each classifier.

4. Methodology

We tested our model on various machine learning algorithms imported from the Scikit Learn software package [7] ranging from Naive Bayes to ensembling methods and deep learning. We deployed a Random Grid Search to find the optimal hyperparameters for each algorithm. The performance of each algorithm during hypterparameter tuning was evaluated using 4-fold cross-validation on the training set.

The settings of the scikit learn text vectorizer implementation were tuned by evaluating the impact of changing a parameter on the performance of the Naive Bayes Classifier. Finally, tf-idf vectors of size 88,702 were passed as input to the classification pipeline for MNB, SGD, MLP and Voting Classifier.

It was more difficult to tune the hyperparameters for the MLP algorithm since the training is computationally expensive when using the sklearn library. The training time is long since the sklearn library only uses CPU operations to do its calculations. Therefore, the equivalent implementation was tested using Keras [2], where it was possible to quickly try out different architectures on a GPU. We utilized Google Colab to try out differnet model pileines as they provided us with free GPU resource for experimentation. MLP models were optimized using the log-loss function with size of minibatches of 64.

Further, we employed L2 regularizer penalty to the modified huber loss to shrink SGD model parameters to avoid overfitting and decrease the model capacity.

5. Results

Results: Present a detailed analysis of your results, including graphs and tables as appropriate. This analysis should be broader than just the Kaggle result: include a short comparison of the most important hyperparameters and all methods (at least 3) you implemented.

We summarize the accuracies of selected models along with the features selected in Table 1. All of the individual classifiers were evaluated using 4-fold cross validation on the training set.

We present our comparative analysis of algorithms with mainly two feature extraction methods. The first method was tf-idf features. In this case, we represent the words in high dimension sparse vectors based on its count and document frequency. The second technique involved generating reduced dimension sentence embeddings by averaging word embeddings generated from word2vec model.

In the tf-idf models, all three classifiers, MNB, SGD and MLP, consistently gave accuracy above 55%. With heuristic and trials we found that the sublinear frequency scaling and performing other character normalization during the preprocessing step boosted our classification accuracies. Utilizing higher order n-grams resulted in diminishing marginal returns on model accuracy.

In the second method of feature extraction, vector representations of text was used, based on the word2vec approach. The best classification accuracy of machine learning algorithms with these features was around 39.4% with MNB classifier and averaging weighted word vectors. However, deep learning methods such as Neural Networks and LSTM gave us comparatively better scores of approximately 48%. We expectated that word2vec features based classification methods should perform better than the tf-idf features based methods. But the results showed otherwise. The potential reason for this could be that our self trained word2vec embeddings were not very enriching because being trained on small dataset of 70k subreddits. One way to rectify this would be to use pre-trained word2vec or GloVe embeddings trained on huge text datasets.

All of our algorithms performed quite bad for few co-related classes. As seen in confusion matrix in figure 1, the mis-classification was high among each other in classes like Canada, Worldnews and Europe because all of them fall under the category of Region. Similar trend was found in classes Funny and AskReddit. We believe if we utilize ensembled models which perform better in these co-related classes would result in increased model accuracy.

Classifier	Accuracy (%)
MNB + tf-idf features	57.3
MNB + word2vec features	39.4
MLP + tf-idf features	56.7
MLP + word2vec features	48.4
SGD + modified huber loss + tf-idf	55.5
SGD + modified huber loss + word2vec	22.1
Soft voting (MNB + MLP + SGD) + tf-idf	58.6

Table 1. Classifier comparisons

6. Discussion and Conclusion

In this work, we discover that the classification accuracies depend on a lot of factors, particularly the dataset and its corresponding preprocessing strategies. The Reddit dataset provided to us only had class label information. But there might be inconsistencies in the training dataset because all examples might not represent good quality information. Incorporating metadata information from the Reddit posts, and potentially weighting the training data to train more heavily on highly up-voted, potentially more topically relevant posts could improve performance.

Individually, Naive Bayes, MLP and SGD, all had decent performances on our Reddit multi-class text classification dataset. However, combining these strong models with equal weights to form a soft voting classifier outperformed each one of them. Hence, this technique is useful to balance out the weaknesses of the individual classifier's weaknesses. Meta classifier is, therefore, both accurate and stable, taking advantage of base models' merits.

As a future step, we could explore Bidirectional LSTMs with attention and pre-trained word embeddings and try soft ensembling of well-calibrated classifiers. Beyond the analysis done in this report, there are few other state-of-the-art text classification complex models such as Bidirectional Encoder Representations from Transformers (BERT) and XL-Net that came out recently and have shown some promising results on text classification and language modelling tasks. However, we didn't explore these approaches during this competition due to computation and memory constraints. As future work, we could aim to examine their performance on text classification.

7. Statement of Contributions

Contributions were evenly distributed between the authors. Both authors worked together in refining the problem statement and methodology. Later, each author worked using their own code base, and as improvements were discovered in their model, the ideas were shared with the other author, which finally lead to a successful result. Both authors worked as a group to write the final report.

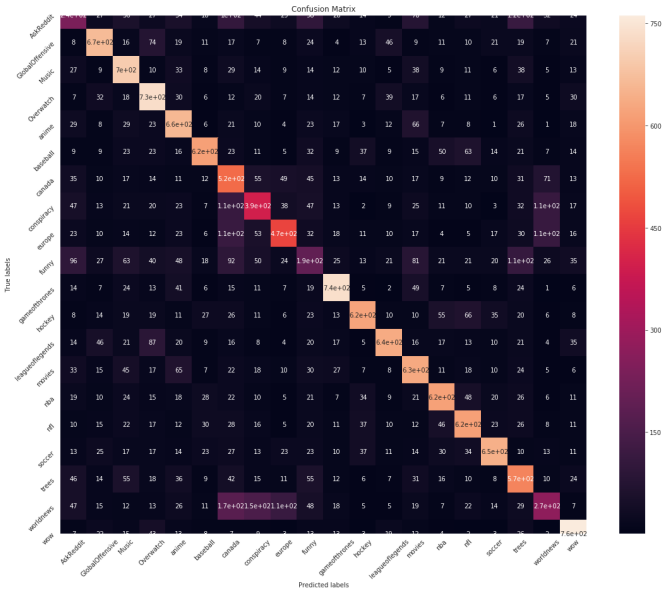

Figure 1. Confusion matrix with MNB Classifier

We hereby state that all the work presented in this report is that of the authors.

References

[1] Piotr Bojanowski, Edouard Grave, Armand Joulin, and Tomas Mikolov. Enriching word vectors with subword information. *arXiv preprint arXiv:1607.04606*, 2016.

[2] François Chollet et al. Keras. `https://keras.io`, 2015.

[3] Armand Joulin, Edouard Grave, Piotr Bojanowski, and Tomas Mikolov. Bag of tricks for efficient text classification. *arXiv preprint arXiv:1607.01759*, 2016.

[4] Edward Loper and Steven Bird. Nltk: The natural language toolkit. In *In Proceedings of the ACL Workshop on Effective Tools and Methodologies for Teaching Natural Language Processing and Computational Linguistics. Philadelphia: Association for Computational Linguistics*, 2002.

[5] Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. Efficient estimation of word representations in vector space. *arXiv preprint arXiv:1301.3781*, 2013.

[6] Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg S Corrado, and Jeff Dean. Distributed representations of words and phrases and their compositionality. In *Advances in neural information processing systems*, pages 3111–3119, 2013.

[7] 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.

[8] Radim Řehůřek and Petr Sojka. Software Framework for Topic Modelling with Large Corpora. In *Proceedings of the LREC 2010 Workshop on New Challenges for NLP Frameworks*, pages 45–50, Valletta, Malta, May 2010. ELRA. `http://is.muni.cz/publication/884893/en`.