Comparative Analysis of Machine Learning Classifiers on Reddit Posts

Team PunjaBERT: Akshay S. Rana (20152453) [*1] **Himanshu Arora (20152429)** [*1]

1. INTRODUCTION

Text classification is one of the most common real-world problems solved by machine learning. In this paper, we try to learn the most accurate model to classify user posts on a popular user-discussion website called Reddit. We experiment with several classification algorithms, ensemble learning and feature engineering methods on a dataset of 70,000 posts belonging to 20 categories. We find that TfIdf vectors after lemmatization and stopword removal trained on a soft voting ensemble of a Complement Naive Bayes classifier, a Multinomial Naive Bayes classifier, and a Multilayer Perceptron achieves the best accuracy of 60.15% on the Kaggle public test dataset.

1.1. Problem

Reddit is a popular social news aggregation and discussion website on which user-posts are organized by subject into user-created boards called subreddits which cover a variety of topics including music, anime, movies, NBA, etc. With over 330 million users and around 138,000 active subreddits, being able to accurately classify user-posts into these categories is the key to organize the vast amount of information and efficiently run the website. Formally, this is called the problem of text classification. It aims at identifying to which of a set of categories a new text document belongs, on the basis of a training set of data containing text documents whose category memberships is known. Specifically, it is a multi-class classification as each document can be classified into one of more than two categories. We aim to learn patterns across these categories which differentiate them the most.

1.2. Dataset

We tackle the above problem by training machine learning classifiers on a dataset of 70,000 text documents of user-posts which can belong to one of 20 different subreddits (categories). The posts are equally distributed among all the categories, i.e. each category has 3500 user-posts. The posts may have frequent grammatical mistakes as the platform is informal which makes the problem even harder. Also, each post can significantly vary in length. For instance, the mean number of words per post across all the categories is 52 whereas the corresponding median values is 30 words, indicating the presence of outlier posts. Specifically, the category Music containing posts with an average length of 92 words. To avoid the issue of overfitting, we use a testing dataset of 30,000 posts and evaluate the generalization performance of our models.

[*]*Equal contribution* [1]*Professional Masters in Machine Learning, MILA, University of Montreal, Quebec, Canada.*

This report is in addition to the Kaggle Competition for Text Classification on Reddit posts in IFT6390 Fall 2019

2. FEATURE DESIGN

2.1. Preprocessing

Since human language is inherently intricate, preprocessing the data is a significant part of any text classification pipeline as a simple machine learning algorithm cannot understand the complexity of raw language text.

2.1.1. Tokenization

As a first and very natural step, we tokenize the posts into words since it is much easier for our algorithms to work with individual words than with whole sentences.

2.1.2. Stopwords removal

Next, we remove all the words which occur frequently in all of the classes. This is because we are looking for the features which differentiate the classes the most and these words, formally called as Stop Words, do not contain any such information. On an average, 49% of words in every post in the dataset are stop words, with a standard deviation of 0.01.

2.1.3. Stemming and lemmatization

To reduce the complexity of the data, we perform an additional preprocessing step of trying to reduce each word to its root or lemma, called stemming and lemmatization respectively. Stemming is much simpler as compared to lemmatization as lemmatization depends on correctly identifying the intended part-of-speech and meaning of a word in a sentence, as well as within the larger context surrounding that sentence. We choose to perform lemmatization as it is known to perform better than stemming in practice.

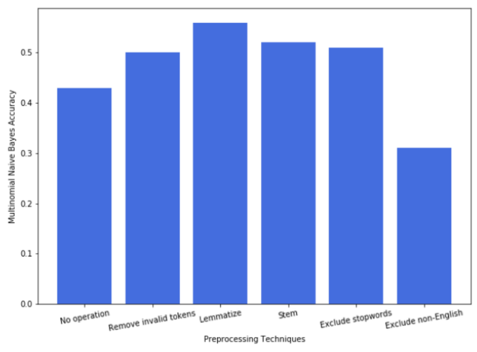

Figure 1. Effect of preprocessing techniques on Naive Bayes

2.2. Feature Transformation

Even after thorough preprocessing of the data, textual data always needs to be converted to it's numerical representation for machine learning algorithms to work. This can be done in several ways. We experiment with the most popular ones of them to find out which one works the best on our dataset. These methods are described below and their performance is discussed along with the classification algorithms in the next section.

2.2.1. VOCABULARY

The simplest way to convert text data to numbers is to build a vocabulary of all the unique words and assign each word in a sentence with its index in the vocabulary.

2.2.2. BAG-OF-WORDS MODEL

To make it simpler for our models to learn the language, we assume that the order of the words in a sentence does not affects the class predictions. This is called a Bag-of-words model wherein each document vector contains the count of each word in the vocabulary.

2.2.3. TF-IDF

To obtain the optimal performance on our classification task, we also performed an even more powerful way of representing text called TF-IDF which is short for Term frequency - Inverse document frequency. This statistic reflects how important a word is to a document in a collection. It's value increases proportionally to the number of times a word appears in the document and is offset by the number of documents in the corpus that contain the word, which helps to adjust for the fact that some words appear more frequently in general.

2.3. Word Embeddings

Word embeddings are meaningful numerical representations of words. They encode the meaning of a word in numbers

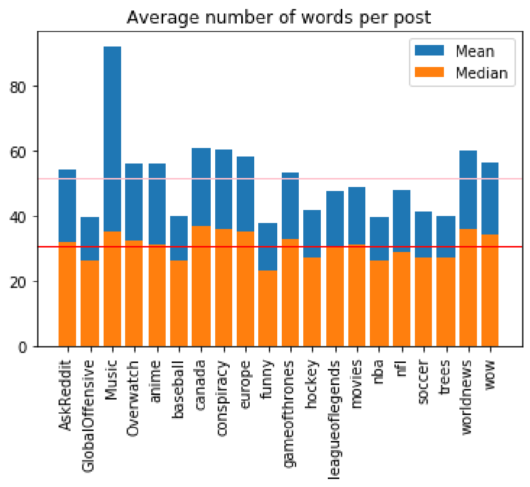

Figure 2. Average number of words per post for each category

allowing machine learning models to better understand the language yielding high accuracy. These are broadly categorized into custom and pretrained word embeddings.

2.3.1. CUSTOM

Custom embeddings are learned from our dataset. These are useful for specialized tasks where the meaning of the words are dependent on the context of the problem.

2.3.2. PRETRAINED

Pretrained embeddings are learned or calculated using large, general-purpose datasets. There are many ways to learn them but we can skip the learning part and directly use them for the words in our dataset.

For our experimentation, we tried to use all of them to find the optimal classifier including custom embeddings, Word2Vec, GloVe, ELMo, and BERT embedding.

3. ALGORITHMS

We experiment with several different classification algorithms to get the best prediction accuracies on our dataset.

3.1. NAIVE BAYES CLASSIFIER

We start our search for the best classification algorithm with the most commonly used for text, the Naive Bayes classifier (NBC). NBC is a probabilistic classifier known for its high performance, high scalability, and ease of training. Text data is generally high-dimensional due to number of different words it can have. Even though it makes strong assumptions about the independence of words, it seems to work really well in practice. We implemented the NBC from scratch as part of the Milestone 1 of the competition. Running this algorithm using Bag of Words achieved an accuracy of 0.56

3.1.1. Laplace Smoothing

Another common technique to further improve the accuracy of a NBC is to smooth the text data using Laplace Smoothing, also known as Additive Smoothing. Additive smoothing allows the assignment of non-zero probabilities to words which do not occur in the sample. Recent studies have proven that additive smoothing is more effective than other probability smoothing methods in several retrieval tasks. We incorporated Laplace Smoothing in our implementation of the NBC and applied it on the stemmed and lemmatized training set. Using grid search, we found out that the best value of the hyperparam α to be 0.25.

3.1.2. Log-probabilities

Since NBC works over products of probability values, the values can get extremely small. This can lead to practical issues due to the way floating-point numbers are represented in the memory. We tried to fix this potential problem by taking logarithm of the probability values and work over sum of log-values instead of product of small probability improving the accuracy of our NBC from 0.556 to 0.572

3.2. LOGISTIC REGRESSION

Logistic regression is a statistical model that uses a logistic function to model a categorical dependent variable. We train a logistic regression model on our training set of user-posts (albeit with less hope) and as expected, achieved a sub-optimal accuracy of only 50%. Logistic regression is a very basic machine learning model and it cannot capture non-linearity within the features of our data. A more advanced algorithm that can capture such dependencies is described in the next section.

3.3. SUPPORT VECTOR MACHINE

Since there can be many hyperplanes separating the classes in our data, one reasonable choice as the best hyperplane is the one that represents the largest separation, or margin, between any two classes. A support vector machine classifier chooses a hyperplane so that the distance from it to the nearest data point on each side is maximized. We could also apply the kernel trick to learn a non-linear hyperplane but since our data is already very high-dimensional we went ahead with a linear classifier.

3.4. MULTI-LAYER PERCEPTRON

A multilayer perceptron is a simple and shallow feedforward neural network optimized using backpropagation. It is a very powerful algorithm as it can model functions of arbitrary complexity. We trained a multi-layer perceptron with a single hidden layer of 250 units and achieved an accuracy of 58%.

3.5. DEEP LEARNING

Although out-of-scope at the time of the competition, we tried neural networks including specialized neural networks for text data such as 1D convolutional networks and Long-Short Term Memory.

4. METHODOLOGY

Apart from the core machine learning classification algorithms, we use several advanced techniques to find the optimal classifier including k-fold cross validation, regularization, hyper-parameter tuning, and ensembling methods.

4.1. K FOLD

K-fold cross validation is a validation technique that is used to improve the generalization performance of a model. We first divide our dataset into K subsamples of equal size, train on k-1 subsamples and validate on the remaining subsample. We repeat this k times and average the accuracy measures. K values of 5 or 10 are typically preferred in practice as they provide test error rate estimates that do not suffer from high bais or variance. Considering the size of our dataset, we choose a value of $k = 5$ for our experiments. This means that we divide our training dataset of 70,000 samples into 5 subsamples of 14,000 random samples each. We train 5 models, one for each of these subsamples and train on the remaining subsample. The accuracy of our models is the average of these accuracies.

4.2. REGULARIZATION

Throughout our experiments with several machine learning classification algorithms, we found the models to be overfitting a lot and thus regularization was a key component in finding a classifier with maximum generalized performance. We have used L2 regularization in the Multi Layer perceptron. We also observed that the deep learning approaches tends to overfit after 2 epochs and techniques like dropout were successfull on generalising the model. We used regularization extensively throughout our models as discussed in the next section.

4.3. HYPER PARAMETER TUNING

We tweaked many hyper params in our algorithms like in the SVM, the optimization problem of finding the plane contains a hyperparameter C which controls the number of errors to allow while finding the maximum-margin hyperplane and this acts as a regularizer. We used grid search to find the optimal value of C to be 0.2 with an accuracy of 57%. We did random search for the best laplace smoothing parameter for Naive Bayes and doing 0.25 to be the best.

4.4. ENSEMBLING

After trying to train individual models, we tried to utilize the potential of ensembling different models. Ensemble methods use multiple learning algorithms to obtain better predictive performance than could be obtained from any of the constituent learning algorithms alone. Evaluating the prediction of an ensemble typically requires more computation than evaluating the prediction of a single model, so ensembles may be thought of as a way to compensate for poor learning algorithms by performing a lot of extra computation. Fast algorithms such as decision trees are commonly used in ensemble methods (for example, random forests), although slower algorithms can benefit from ensemble techniques as well. Empirically, ensembles tend to yield better results when there is a significant diversity among the models. We tried three different ensemble methods of different classifiers as described below.

4.4.1. BAGGING

Bagging, also known as Bootstrap Aggregating, uses the Bootstrap algorithm to sample different datasets (with replacement) from the original dataset. A model of our choice is trained on each of these sampled datasets and the output is decided by voting among the models. This reduces variance in the model thus avoids overfitting, increasing generalization performance.

We trained two bagging ensembles of support vector machine classifiers and Naive Bayes classifiers respectively. We bootstrapped both the dataset and the features of the data to add more flexibility to our models. The Naive Bayes classifiers ensemble performed much better (56% accuracy) than Support Vector Machine classifiers ensemble (48% accuracy).

4.4.2. RANDOM FORESTS

It is the most commonly used bagging method in machine learning. Random forests use a modified decision tree learning algorithm that selects, at each candidate split in the learning process, a random subset of features in the data. It is a way of averaging multiple deep decision trees, trained on different parts of the same training set, with the goal of reducing the variance.This comes at the expense of a small increase in the bias and some loss of interpretability, but generally greatly boosts the performance in the final model. We trained a Random Forest each with 10, 50, and 100 decision trees and surprisingly, achieved a very low accuracy of 49%.

4.4.3. BOOSTING

Boosting is an ensembling method which, unlike bagging, primarily reduces the bias in our model instead of variance. Most boosting algorithms consist of iteratively learning weak classifiers with respect to a distribution and adding them to a final strong classifier. When they are added, they are typically weighted in some way that is usually related to the weak learners' accuracy. After a weak learner is added, the data weights are readjusted, known as ”re-weighting”. Misclassified input data gain a higher weight and examples that are classified correctly lose weight.Thus, future weak learners focus more on the examples that previous weak learners misclassified. We tried two of the most popular boosting methods and describe our results below.

4.4.4. ADAPTIVE BOOSTING

AdaBoost, also known as Adaptive Boosting is a very powerful boostnig method. It is adaptive in the sense that subsequent weak learners are tweaked in favor of those instances misclassified by previous classifiers. AdaBoost (with decision trees as the weak learners) is often referred to as the best out-of-the-box classifier.

4.4.5. GRADIENT BOOSTING

Gradient Boosting is another popular boosting method. It is slightly different from the adaptive boosting in the way the weak learners are trained. Rather than changing the sampling distribution based on the performance of a weak learner, it trains the subsequent weak learners on the errors of the prior learners. A more common form of gradient boosting XGBoost, also known as Xtreme Gradient Boosting, essentially works on the same principle but it is more regulaized which reduces overfitting and increases the generalization performance.

We trained both AdaBoost and XGBoost, achieving an accuracy of 0.438 and 0.491.

5. RESULTS

Table 1 summarizes the performances of all our approaches, where BoF denotes the Bag of words and TfIdf denotes the Term-frequency inverse document frequency. As evident, the TfIdf performs better with 80k features trained on a voting classifier of Complement Naive Bayes, Multinomial Naive Bayes and a Multi layer Perceptron with 250 hidden units. It can be seen that TfIdf consistly performs better than the Count Vectorizer given any classifier.

Weighted majority algorithms are used to construct a compound algorithm from a pool of prediction algorithms. The weighting can be done in either hard or soft way. Hard voting uses predicted class labels for majority rule voting. Whereas, soft voting predicts the class label based on the maximum of the sums of the predicted probabilities. We experiment with multiple hard and soft weighted classifiers.

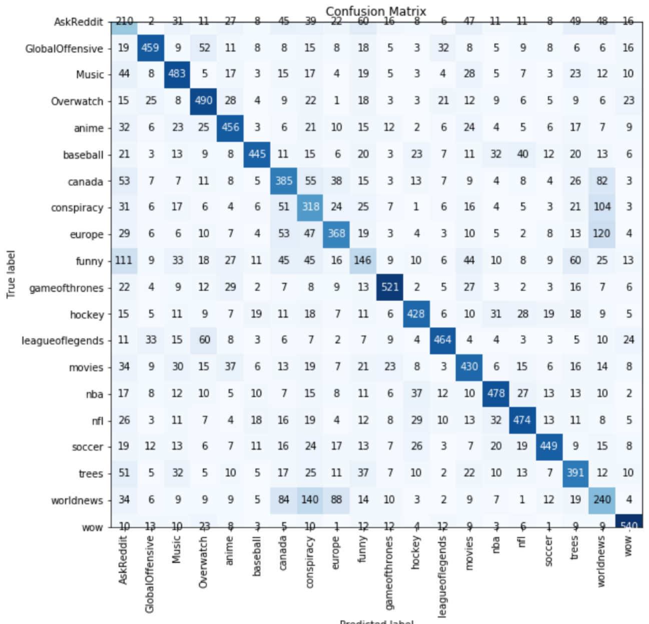

Figure 3. Confusion matrix for all classes on validation set

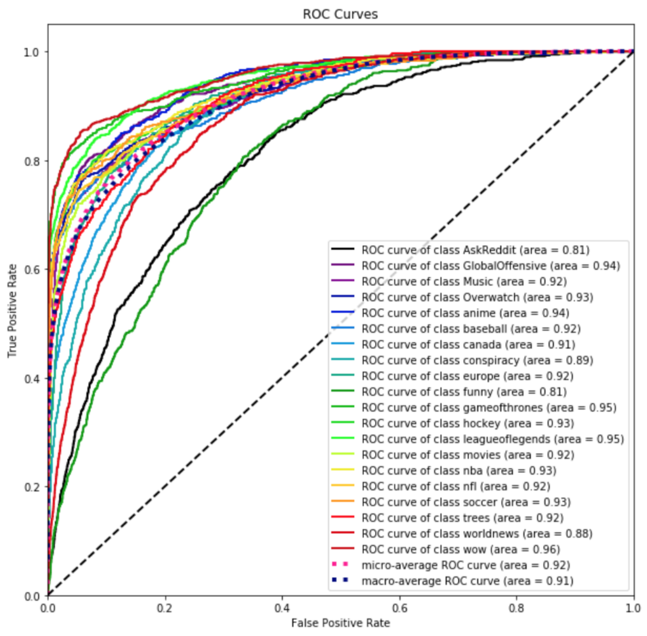

Figure 4. ROC curve for all classes on validation set

For each learner in every ensemble, we use the best hyper-parameters identified in the section above. Our hard-voting classifier consisting of a multinomial Naive Bayes classifier, a logistic regression classifier, a multilayer perceptron, and a support vector machine classifier achieved an accuracy of 58.5%. We also trained several soft-voting classifiers and the best one was an ensemble of a Complement Naive Bayes classifier, a multinomial Naive Bayes classifier a multilayer perceptron with an accuracy of 60%.

Sequence respecting approaches should perform better at text classification, hence we used embeddings which takes into account the context of the document. Although, it has advantages over the bag of words approach but the Deep Learning architectures for it requires huge training size and computation resources. We tried few architectures with the limited time and resources we had. The results seem to match all the machine learning algorithms in just one epoch. Upon careful hyper-param tuning, it can surpass accuracies of our best machine learning ensemble of tfIdf vectors.

We made use of word2vec vectors trained on skipgram model which are capable of encapsulating the context of the words, and trained a dense neural network on top of it. We also trained a 1D convolution layer on tfidf and word2vec and it seemed to perform quite well as well. Another approach of pretrained embeddings BERT (Bidirectional Transformers for Language Understanding) was used alongwith LSTM and Bidirectional LSTMs. Table 2 summarizes all the scores from these models.

Models	Validation	Public Data
MNB + BoF	0.556	0.55
MNB + Lemma + BoF	0.561	0.55
MNB-uni + Lemma + TfIdf	0.575	0.57
MNB-bi + Lemma + TfIdf	0.563	0.56
LR + BoF	0.496	0.48
LR + Lemma + TfIdf	0.527	0.51
SVM + BoF	0.512	0.52
SVM-uni + TfIdf	0.545	0.53
SVM-bi + TfIdf	0.538	0.52
MLP(250) + TfIdf-20k	0.567	0.57
MLP(250) + TfIdf-50k	0.573	0.58
Decision Tree + TfIdf	0.456	-
Random Forest +TfIdf	0.335	-
AdaBoost + TfIdf	0.438	-
SVM+MNB	0.583	0.58
SVM+MNB+MLP	0.591	0.59
CNM+MNB+MLP	**0.597**	**0.60**

Table 1. Validation scores across all models and features

Models	Embedding	Validation Set	Public Data
DNN	**Word2Vec**	**0.58**	**0.59**
CNN	Word2Vec	0.57	0.58
RNN	BERT	0.58	0.58
LSTM	BERT	0.58	0.58

Table 2. Validation scores across all models and features

As seen in table 3, classes with unique words seem to per-

Classes	Precision	Recall	F1 Score	ROC
baseball	0.77	0.62	0.69	0.92
soccer	0.76	0.64	0.69	0.93
wow	0.77	0.77	0.77	0.96
gameofthrones	0.77	0.74	0.75	0.95
leagueoflegends	0.75	0.68	0.71	0.95
askReddit	0.32	0.34	0.33	0.80
worldnews	0.32	0.34	0.33	0.88
funny	0.29	0.22	0.77	0.81
conspiracy	0.36	0.48	0.41	0.89
canada	0.48	0.52	0.50	0.91

Table 1. Precision Recall scores for top 5 and bottom 5 classes

form very well compared to the generic classes where contextual information is really important. It is difficult for a bag of words model to learn the intricacies of the information in classes like *funny, worldnews, askReddit, etc.*. Even the Bi-gram and tri-gram approaches does seem to have a little effect on these classes, but it makes it difficult for the other classes and thus decreases the average accuracy.

The confusion matrix shows that most of the misclassification happens on *funny-askReddit* and *conspiracy-worldnews*. After careful analysis of the training data, it was found that it is really difficult for the humans too to make a difference between these correlating classes. However on the contrary, classes like *gameofthrones, leagueoflegends* have words which make them unique and easy to classify.

6. DISCUSSION

We observe that the Naive Bayes classifier consistently performed better than other machine learning classifiers and boosted the performance of ensemble classifiers. The reason NBC worked well on this dataset, and in general over text, is that it scales well with the number of features. Text data is very high-dimensional as the number of features scale linearly with the vocabulary of the data. Since NBC is a really simple probabilistic classifier that works on the Bayes theorem, it remains relatively unaffected by the number of features. Whereas, iterative learning based classifiers such as logistic regression, support vector machine classifier, and multilayer perceptron may lead to overfitting if the number of features are more than the number of training samples.

In deep learning algorithms, the sequence learning algorithms such as the LSTM performed similar to the NBC even with an architecture having less capacity and minimal training. These algorithms learn the dependence between the words as a sequence in the sentence which the NBC fails to capture. Bi-directional context-based algorithms such as BERT can significantly improve the accuracy but we couldn't experiment much with them due to limited computational resources and time.

7. STATEMENT OF CONTRIBUTION

1. Defining the Problem: Both
2. Developing the methodology: Both
3. Coding the solution:
   Akshay: Coded Machine learning and feature transformation techniques.
   Himanshu: Coded deep learning experiments.
4. Performing the data analysis
   Akshay: Worked on exploratory data analysis
   Himanshu: Worked on Embeddings for deep learning methods.
5. Writing the report
   Akshay: Wrote tables, graphs, results and conclusions.
   Himanshu: Wrote everything else.
   We also verified each other's written parts.

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

8. REFERENCES

1. *https://en.wikipedia.org/wiki/Ensemble_learning*
2. *https://en.wikipedia.org/wiki/Additive_smoothing*
3. *https://towardsdatascience.com/deep-transfer-learning-for-natural-language-processing-text-classification-with-universal-1a2c69e5baa9*
4. *https://towardsdatascience.com/machine-learning-nlp-text-classification-using-scikit-learn-python-and-nltk-c52b92a7c73a*