ECSE 551 Fall 2020 Mini-project 2 Report

Fei Peng
260712440
fei.peng@mail.mcgill.ca

Yukai Zhang
260710915
yukai.zhang@mail.mcgill.ca

Yi Zhu
260716006
yi.zhu6@mail.mcgill.ca

Abstract

Bernoulli Naïve Bayes classification has become one of the most popular approaches in machine learning for document/text classification [1]. In this mini-project, we investigated and evaluated the performance of Bernoulli Naïve Bayes classifier by implementing it from scratch, and used it to analyze text from the website Reddit. Before feeding the dataset to the classifier, it was vectorized to change its text-based nature into numerical features. Different types of vectorization techniques were explored, and the best one was selected, which reached the highest accuracy in the k-fold validation, for reporting overall performance. Furthermore, experiments were conducted on the dataset using eight other additional classifiers from the SciKit learn package. Among them, the model that achieved the highest accuracy in k-fold validation was chosen for classifying the Kaggle test data. According to our testing result, tweaking the hyperparameters of the models does not have any significant impact on the result, however, choosing the correct type of vectorizer and classifier has the dominant effect on improving accuracy. Details will be presented in the following parts of this report.

1 Introduction

The boost in the popularity of participatory web and social networking sites has brought about many machine learning opportunities and challenges [2], for instance, content classification. As a significant portion of social media data is in natural language format, the main objective of this mini-project is to apply machine learning methods on classification of text data. The dataset was obtained from Reddit, a popular website where users post and comment on content in different themed communities (i.e., subreddit), with eight classification categories: rpg, anime, datascience, hardware, cars, gamernews, gamedev, and computers.

Before feeding any data into the classifiers, it has to be vectorized to change its text-based nature into numerical values. We implemented five different vectorizers, namely count vectorizer (CV), count vectorizer with stopwords (CVSW), TF-IDF vectorizer (TF-IDFV), count vectorizer with stemming (CVS), and count vectorizer with lemmatization (CVL). Except for Bernoulli Naïve Bayes classifier which requires binary input (TF-IDF vectorizer is therefore not feasible), all other classifiers were fed with z-score normalized input to improve accuracy.

In the first part of this project, we implemented a Bernoulli Naïve Bayes classifier from scratch and investigated its performance using k-fold cross validation. Bernoulli Naïve Bayes algorithm is a less costly generative model which proposes Naïve Bayes assumption (assume features of sample x are conditionally independent given class y) for simplification. Compared to discriminative learning (e.g., Logistic Regression) which directly estimate $P(y|x)$, generative learning models $P(x|y)$ and $P(y)$ separately and use Bayes' rule to estimate $P(y|x)$ which allows more flexibility. Moreover, to avoid the occurrence of zero probability, we also implemented Laplace smoothing in the fit method.

The second part is to perform experiments using other classifiers from SciKit learn package, and measure their performance in terms of speed and accuracy. The classifiers we investigated are: Logistic Regression (LR), Multinomial Naïve Bayes (MNB), Support Vector Machine (SVM), Random Forest (RF), Decision Tree (DT), AdaBoost (AB), K-Neighbors (KN), and Multi-layer Perceptron classifier (MLP) [3, 4]. In addition, we explored the behaviour of Hard/Soft Voting Classifiers and the Stacking Classifier (SC) [3, 4] to further improve the accuracy. According to our experiments, while tweaking the hyperparameters of the classifiers could slightly increase the accuracy (at the cost of dramatically increasing training time), choosing the proper vectorizer and classifier helped improve the performance significantly.

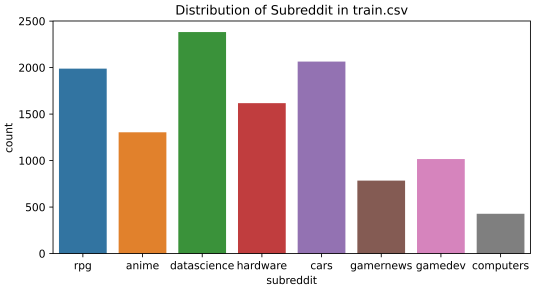

Figure 1: Class/Subreddit Distribution of Train Data

Table 1: Number of Features and Processing Time of Five Different Types of Vectorization Methods

	CV	CVSW	TF-IDFV	CVS	CVL
Number of Features	41033	40729	41033	20765	24777
Time (s)	0.96842	0.91925	1.0158	29.1	144.17

2 Datasets

Before performing any investigation on machine learning models, it is important that the characteristics of the dataset is properly analyzed and understood. Aside from that, data preprocessing lays the groundwork for the following data analysis, as its product is the final training set [5]. The datasets we used are *train.csv* which contains 11582 samples and their corresponding class/subreddit, and *test.csv* that includes 2898 test samples with their ID and features from which we predicted their class/subreddit (and submitted to Kaggle competition). In this project, the following data preprocessing had been performed.

2.1 Entropy Analysis

The classification of the Reddit dataset is an example of multi-class classification. The distribution of each class/subreddit is shown in Figure 1. The uncertainty in prediction can be quantified by the entropy of each dataset with equation 1, and the calculated data entropy is 2.847 which stands for a relatively even distribution and could be preferable for some classification techniques.

$$H(X) = -\sum_{x \epsilon X} p(x) log_2 p(x) \tag{1}$$

2.2 Vectorization

The features of the raw Reddit dataset is in text format, thus it is necessary that they are vectorized and transformed into numerical values before feeding the data into any classifier [4]. The pure count vectorizer simply counts the occurrence of words, while count vectorizer with stopwords eliminates the typical English stopwords before counting. TF-IDF vectorizer calculates the term frequency times inverse document frequency. Stemming and lemmatization recover the canonical form of the words. The numbers of features after vectorization are presented in Table 1, as well as the processing time for performing each vectorization method. It is indicated that stemming and lemmatization decrease feature number by nearly half which could help increase the training speeding for training classifiers in the future, however, the processing time of themselves is approximately ten to a hundred times that of without stemming or lemmatization. They are even less preferable as they failed to improve classification accuracy either.
On the other hand, we also considered obtaining binary vector representation for the text input (by setting $binary = True$ for vectorization). In this case, the result would rely on the presence or absence of each feature, instead of the number of its occurrences. This argument must be set to $True$ for Bernoulli Naïve Bayes classifier which requires the features to be binary-valued [3]. It was also tested for other classifiers to see whether it would help increase accuracy, and the result will be discussed in the following parts of this report.

2.3 Normalization

Normalizing the data into z-score can centralize the value distribution of the data while keeping its precision. However, for text-based datasets like Reddit, this preprocessing approach might not provide a satisfying result, as the meaning of the numerical representation of features is the number of occurrences of each word in each comment. This value is usually either 0 or 1.

2.4 Maximum Features

The $max\_features$ argument of the vectorizer functions is provided by sklearn to build a vocabulary that only considers the top $max\_features$ ordered by term frequency across the corpus [3, 4]. For the preprocessing input data of Bernoulli Naïve Bayes classifier, we set $max\_features = 5000$ as our implementation may not be optimized for speed. By setting the maximum features, the time consumed by fit and predict functions was significantly shortened, as the number of features was one eighth of the original dataset.

3 Proposed Approach

In this section, we are going to discuss the features selection and methods implementation in terms of accuracy, robustness/consistency, and speed. However, due to time constraint of this project and page limitation of this report, we are only going to discuss the methods with outstanding performances.

3.1 Training/Validation Splits

As mentioned in the previous section, the training datasets contains 11582 samples and testing datasets contains 2898 samples. The ratio of training to testing samples is 3.99655 (4:1). Therefore we chose to perform 5-fold validation (training:testing = 4:1) on all the chosen classifiers to compare and select the optimum model for competition. In this way, we could ensure consistency between the validation and testing process, and increase the possibility to reach higher accuracy on testing data.

3.2 Laplace Smoothing

Originally we would use equation 2 to calculate $\theta_{j,k}$ matrix (the probability that feature j occurs given class k). However, it could cause zero division when a certain class does not occur in the dataset ($P(y = k) = 0$). By applying Laplace smoothing, we could prevent zero division by adding the numerator by 1 and denominator by 2.

$$\theta_{j,k} = P(x_j = 1|y = k) = \frac{P((x_j = 1 \cap y = k)}{P(y = k)} \tag{2}$$

3.3 Algorithm Selection

At the beginning of this project, we started with building a Bernoulli Naïve Bayes classifier (BNB) from scratch, which is one of the linear classification methods. Like all generative learning approaches, Bernoulli Naïve Bayes separately model $P(x|y)$ and $P(y)$ and use Bayes' rule to estimate $P(y|x)$ where x represents features and y is the corresponding classes. Moreover, compared to linear discriminant analysis (LDA) which is another generative learning method, it proposes Naïve Bayes assumption to reduce calculation cost. Therefore, assume there are overall k classes and j features, it could directly calculate the probability of each class (i.e., k) and the probability of each feature given each class (i.e., $\theta_{j,k}$). Laplace smoothing could be applied when calculating $\theta_{j,k}$ to prevent zero division. Compared to Multinomial Naïve Bayes where the input data are typically represented as word vector counts [3], Bernoulli Naïve Bayes only allows binary-valued input, therefore count vectorizer (with $binary = True$) without normalization should be used to preprocess the input data. Finally, it would use these probabilities to make predictions for the testing data.
Aside from Bernoulli Naïve Bayes, all other classifiers accept non-binary features, thus we have the flexibility to choose the most suitable vectorizer-classifier pair. Next, we continued to investigate the behaviour of linear models by performing experiments on Logistic Regression classifier and Linear Support Vector Machine (SVM) provided by sklearn [4]. In contrast to generative learning, Logistic Regression, a discriminative learning method, directly estimate $P(y|x)$ by first obtaining the weight (w) of each feature with gradient descent and then apply a logistic/sigmoid function to predict classes. Linear SVM predicts classes by separating samples with hyperplanes (i.e., linear classifiers) and optimizing them by maximizing the margin of each classifier. These linear models, especially SVM, have promising performance when the number of samples is relatively small when deep learning is not feasible, and their accuracy could be increased by applying regularization, which will be discussed in the next subsection.
Furthermore, we focused on improving the accuracy by exploring more complex methods. The idea is to combine the simple learning algorithms into a Neural Network or Ensemble methods [6]. Multi-layer Perceptron (MLP) is a Neural Network classifier provided by sklearn [4], which uses a network (with hidden layers) of stacked perceptron-like elements to represent non-linearly separate functions, where perceptrons are simple linear classifiers. Stacking Classifier (SC) is one of the stacking Ensemble methods [4], which consists in stacking the output of different base learners and using a final classifier/meta model to compute the final prediction (usually Logistic Regression). The features of the final classifier would be the combination of output from base learners. This method combines the strength of each individual estimator to produce the optimum output.
To select the best model, we mainly focused on choosing the one that provides highest accuracy, but also made certain trade-offs between accuracy and speed. We found that by tweaking regularization terms and hyperparameters by a certain amount, we could bring some improvements to accuracy. In addition, using the best models as base learners, we could train a Stacking Classifier to achieve better accuracy. We also considered other classifiers such as K-Neighbors, Decision Trees, and other Ensemble classifiers like Random Forest (Bagging), AdaBoost (Boosting), and Hard/Soft Voting Classifier (Stacking), but they either were not as satisfying enough or fails to increase accuracy by sacrificing speed, thus we would not go deeper into them.

3.4 Regularization and Hyperparameters

In linear models such as Logistic Regression and Linear SVM, there is a regularization parameter (i.e., C) when defining the classifier. It is the inverse of regularization strength where smaller values specify stronger regularization. For the MLP classifier, the regularization is controlled by parameter alpha which is the L2 penalty [3]. The detail effect and result of regularization will be discussed in the next section. As for hyperparameter, we increased the value of max_iter (i.e., maximum iteration) from default (set by sklearn and varies for different classifiers) to 1000, to make sure the algorithm converges. Moreover, for the MLP classifier, the hidden layer size and learning rate also influences the accuracy and speed which should be considered.

Table 2: Accuracy and Processing Time for Five Classifiers

	BNB	LR	Linear SVM	MLP	SC
Maximum Accuracy	0.81316	0.89613	0.89026	0.90822	0.91029
Processing Time (s)	168.842	116.797	321.477	1379.423	19795.276

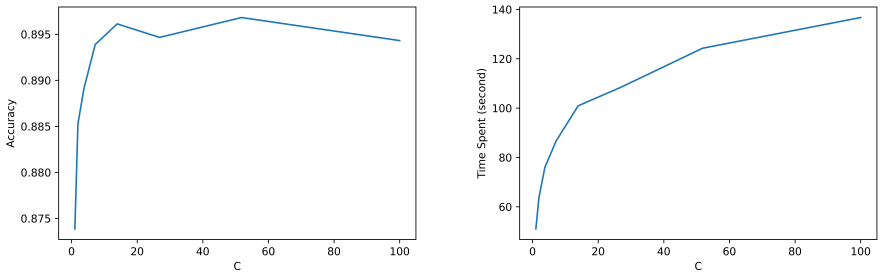

Figure 2: Mean Validation Accuracy (left) and Processing Time (right) with respect to regularization term of Logistic Regression

4 Results

The results we obtained in this project are summarized in Table 2, where the processing time is for 5-fold validation and represented in seconds. Due to page limitation, please refer to Appendix A for more details. The result used for final submission on Kaggle was obtained using Stacking Classifier since it has the highest cross validation accuracy, and the test set leaderboard accuracy we achieved is 0.93095.
All combinations of vectorizer type and classification methods were evaluated, and details are illustrated in tables of Appendix A. The optimum vectorizer for preprocessing the input of concerned sklearn classifiers is TF-IDF vectorizer, one of the explanation is that TF-IDF vectorizer determine if a term is important/indicative of a document by not only the number of occurance in the document but also whether it is a relative rare word overall.

4.1 Bernoulli Naïve Baye

This classifier was implemented from scratch. The accuracy and speed are shown in Table 2. The maximum accuracy was achieved when the $max\_features$ is 5000, binary is set to $True$, and $stop\_words$ equals to $ENGLISH\_STOP\_WORDS$ for count vectorizer during data preprocessing. Laplace smoothing is also applied to prevent zero division. To have a more thorough understanding of its performance, we compared the accuracy and speed with that provided by sklearn. The maximum achieved accuracy is 0.83448, with processing time equals 4.717 seconds, while the exact same vectorizer was used. Note that the sklearn model was able to cooperate with the TF-IDF vectorizer, but this is not the case for our model. The sklearn model does not require binary input because there is a $binarize$ parameter which handles non-binary input. Overall, the accuracy of our model is no worse than that from sklearn, however, ours requires around 40 times processing time.

4.2 Logistic Regression:

As shown in Figure2, the accuracy could be increased by applying less regularization in Logistic Regression. It was explained earlier that C is the inverse of regularization strength, and the default value is 1.0. The accuracy reaches its maximum when C is around 40.0, and the processing time remains within a reasonable range. The accuracy and speed are shown in Table 2.

4.3 Linear SVM

As indicated in Figure 3, the accuracy could as well be increased by applying less regularization in Linear SVM. The accuracy reaches its maximum when C is around 1.0, and this is also when the processing time is the lowest among all models. The accuracy and speed are shown in Table 2.

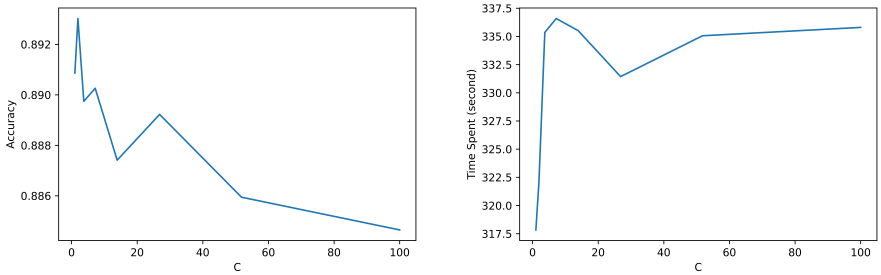

Figure 3: Mean Validation Accuracy (left) and Processing Time (right) with respect to regularization term of Linear SVM

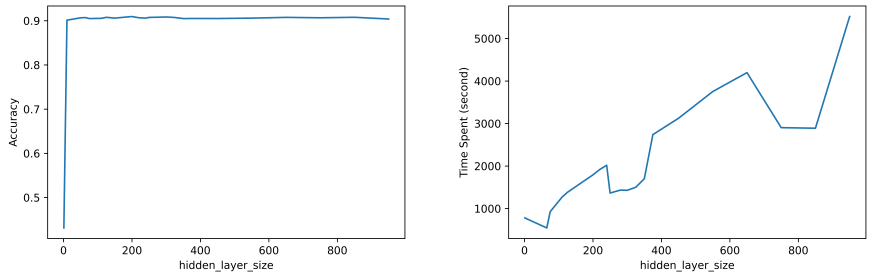

Figure 4: Mean Validation Accuracy (left) and Processing Time (right) with respect to hidden layer size of MLP Classifier

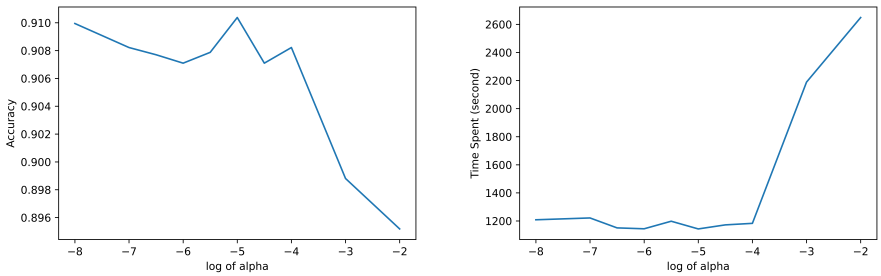

Figure 5: Mean Validation Accuracy (left) and Processing Time (right) with respect to regularization term of MLP Classifier

4.4 MLP Classifier

For this classifier, we set the $learning\_rate\_init = 0.0001$ and $learning\_rate =$'$adaptive$' (keeps the learning rate constant to '$learning\_rate\_init$' as long as training loss keeps decreasing), which helps increase the accuracy slightly over that of default setting. Moreover, we investigated the effect of changing hidden layer size, and the result is illustrated in Figure 4. As shown in the plots, the optimum choice would be 100 hidden layers in terms of both accuracy and speed. In addition, the effect of regularization was examined, and the result is indicated in Figure 5, the classifier reaches maximum accuracy when $alpha$ equals 10^{-5}. The accuracy and processing time are shown in Table 2.

4.5 Stacking Classifier

Compared to the Voting Classifier which uses voting as its final classifier, Stacking Classifier feeds the output of base learners into another machine learning method as its final classifier. In our implementation, the three best models from Logistic Regression, Linear SVM, and MLP Classifier are selected as base learners, which have already shown promising performance, and the final classifier is another Logistic Regression model. The accuracy and speed are shown in Table 2. The Stacking method improves accuracy by combining the advantage of each base learner, in return, it sacrifices more than 90% of the training speed.

5 Discussion and Conclusion

In conclusion, the Bernoulli Naïve Bayes classifier implemented in this project performed as expected on the Reddit dataset. It achieved an accuracy of 81.3%, with processing time equals 168.842 seconds. However, compared to the Bernoulli Naïve Bayes classifier in sklearn, our model is significantly slower, and not able to accept non-binary input, therefore cannot coorperate with TF-IDF vectorizer or any other non-binary vectorizer. The future investigation could be improving the speed of our algorithm in terms of data structure by changing the remaining nested loops with numpy matrix multiplication. Another direction would be dealing with non-binary data by binarizing the input before the fit function.

On the other hand, we achieved a leaderboard accuracy of 0.93095 using the Stacking Classifier in sklearn, with Logistic Regression, Linear SVM, and MLP Classifier as three base learners and another Logistic Regression as final classifier. We found that linear models as well as their combinations have relatively better performance than other (non-linear) machine learning models we investigated. In the future, we may take a closer look at the cause of this phenomenon, and find out a better model to improve accuracy.

6 Statement of Contributions

Fei Peng implemented the Bernoulli Naïve Bayes classifier from scratch, as well as various types of vectorizers for data preprocessing. Yukai Zhang was in charge of researching feasible classifier implementations in sklearn, measuring validation accuracies of different classifiers using k-fold validation, and selecting the best model for kaggle test. Yi Zhu was responsible for examining the classifiers in terms of the training speed, and the effect of choosing different hyperparameters on speed and accuracy.

References

[1] G. Singh, B. Kumar, L. Gaur, and A. Tyagi, "Comparison between Multinomial and Bernoulli Naïve Bayes for Text Classification," in *2019 International Conference on Automation, Computational and Technology Management (ICACTM),* 24-26 April 2019 2019, pp. 593-596, doi: 10.1109/ICACTM.2019.8776800.

[2] L. Tang and H. Liu, "Leveraging social media networks for classification," *Data Mining and Knowledge Discovery,* vol. 23, no. 3, pp. 447-478, 2011/11/01 2011, doi: 10.1007/s10618-010-0210-x.

[3] F. Pedregosa et al.,"Scikit-learn: Machine Learning in Python," *Journal of Machine Learning Research,* vol. 12, pp. 2825–2830, 2011.

[4] L. Buitinck et al., "API design for machine learning software: Experiences from the scikit-learn project," *API Design for Machine Learning Software: Experiences from the Scikit-learn Project,* 09/01 2013.

[5] S. Kotsiantis, D. Kanellopoulos, and P. Pintelas, "Data preprocessing for supervised learning," *International Journal of Computer Science,* vol. 1, no. 2, pp. 111-117, 2006.

[6] T. G. Dietterich, "Ensemble Methods in Machine Learning," Berlin, Heidelberg, 2000: Springer Berlin Heidelberg, in *Multiple Classifier Systems,* pp. 1-15.

Appendix

(Please see following attached pages for Appendix A: Results and Appendix B: Code Implementations)