Implementation of models to analyze text from the Reddit website

ECSE 551 - Machine Learning for Engineers

Norman Kong (260923365)

March 2023

Contents

1 Abstract

This project aims at implementing a Bernoulli Naive Bayes model from scratch, comparing its performance to one of sklearn's built in models and participating in the Kaggle competition. The classification task at hand was to classify Reddit posts/comments based on the subreddit they originated from. Several textual preprocessing steps were required: the removal of stopwords (based on stopword lists on the Internet and punctuation), and the stemming of the textual data into tokens. The Bernoulli Naive Bayes model averaged 77.58% accuracy after 10-fold cross validation implemented with sklearn's KFold class. We compare the Naive Bayes Bernoulli model to a logistic regression model with stochastic gradient descent, included in the sklearn library. This averaged 92.6% accuracy over the same 10 fold cross validation and so this was used for the final testing set at the Kaggle competition. At the preliminary Kaggle competition, this model scored an accuracy of 95.18%. We see that this model generalizes well to new data. Future work should explore different hyper-parameters for the SGD model, as well as the performance of completely different models on the same task.

2 Introduction

One important and popular application of machine learning models is in the task of text classification. Briefly, this is a machine learning task where machine learning models are applied to textual data to assign a category (or label) to text samples. This project will explore a simple text classification task on data from the Reddit website. Reddit is a popular social media forum where users create posts in subreddits (online communities based on specific topics). My aim with this project is to identify the subreddit where each post originally came from. The dataset is gathered from posts and comments of four subreddits: `Trump`, `Obama`, `Musk`, `Ford`. Our data was provided by Prof. Armanfard. This data is split into a train and a test dataset allowing us to respectively train and test our models. The implementation of our models will then be assessed using a k-fold cross validation class provided by sklearn and our best model will then be entered into the Kaggle competition in the hope of getting the best performance on unknown data.

3 Datasets

Two text datasets were provided to us in the form of csv files: a train dataset that we will use to train our models and a test dataset that we will use to test our models. Each sample is composed of a text (a post or comment) and a label, which is one of 4 classes (which corresponds to the subreddit the text came from).

3.1 Preprocessing

As with any machine learning pipeline, the data must first be preprocessed; that is, converted to a format that is more convenient for our models. We first split the text into small bits of text that are generally separated by spaces (tokenize the text). We must also convert this data into a format that is easier to work with: word vectors. Word vectors contain n features, where n is the number of words in the dictionary that you've chosen (this is a set of words that you have chosen to represent in your data). For each feature, the vector contains a number indicating either how many counts of that word is present in that sample, or a binary value to indicate the presence of that word in that sample. An example of a word vector is shown in Figure 1.

This is done by using sklearn's `CountVectorizer` class. I also experimented with creating my own analyzer for the class consisting of the following. The `stem_sentence()` method converts the strings to lowercase, splits a sentence into tokens (essentially into text that was originally

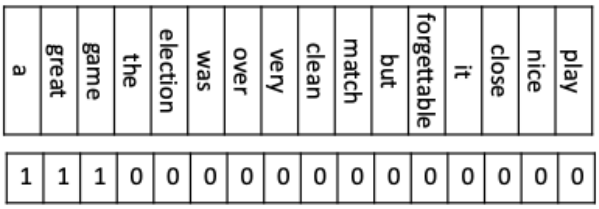

Figure 1: This example assumes a dictionary of 16 words, indicated in the first row. Hence, each sample is represented by a vector of dimension 16. Figure credits to Prof. Armanfard.

separated by spaces). It also stems the words (that is, shorten the word to its root based on regex). The `preprocess_text()` returns the text after using `stem_sentence()`, taking out stopwords and taking out numbers. Since numbers might actually be useful tokens for prediction; I chose to go with sklearn's default analyzer for the experiments. I first included this functionality because the data seemed to contain noise-like numbers as tokens. These preprocessing methods were mostly exploratory.

Removing stopwords is a common preprocessing method for textual data. Stopwords are words that are removed from data because they are assumed to be not meaningful. We created a custom set of stopwords by combining `nltk`'s (a Python library for language data) standard english stopwords, a stopwords list found online here and punctuation. Another important preprocessing step is to binarize the values for each feature; more specifically, we change all values greater than 0 to 1 since the Bernoulli model is not concerned with token counts but rather only whether or not the token is present in that sample. Finally, we map each category to an integer (0 to 3).

3.2 Train dataset

The train dataset comprises of 719 Reddit comments made out of 73 611 words and organized in 719 rows. Every sample is comprised of a 'body' attribute which corresponds to the text for that sample, and its label, stored in a 'subreddit' attribute. The distribution of this data is as follows: the `Trump`, `Obama`, `Musk` and `Ford` categories contain 179, 179, 180, 180 samples respectively. This is close to a uniform distribution and so this dataset exhibits high entropy.

3.3 Test dataset

The test dataset consists of 279 reddit comments (samples) made out of 28 856 words and stored in 279 rows. Since this is the test dataset to evaluate the performance of our model, the labels are omitted.

4 Proposed Approach

4.1 K-fold cross validation

As per the requirements of the project, we implement K-Fold cross validation for K=10. The implementation is done via the `KFold()` class in the sklearn library. This class separates the data into K sets, split between train and validation sets. We loop over these sets and for each iteration, we train our model with that iteration's training set and compute its accuracy score with that iteration's validation set. To evaluate the model's performance we then compute the average accuracy score across all folds. The strongest model is judged to be the model with the best mean accuracy. Documentation for this class can be found here.

4.2 Naive Bayes Bernoulli

The K-fold cross validation step above yields test and validation datasets. We then pass this data to the `CountVectorizer` class which converts the text to a matrix of token counts (in other words, this class vectorizes the text). This processed data is passed to our `BernoulliNaiveBayes` class. We initialize any instance of this class with the `alpha` parameter. This is required for Laplace smoothing and defaults to 1. There are the `usual` (usual as in similar to sklearn's structure) fit and predict methods. The bulk of the algorithm is in the fit method. We compute the appropriate probabilities based on counts in the training data and Bayes' theorem. This model is largely based on the following equation:

$$P(y \mid x) \propto P(y)\Pi_{1\leq k\leq n}P(x_k \mid y) \tag{1}$$

where n is the number of features per sample. Note that in computing $P(y \mid x)$, we omit the denominator from Bayes' theorem since the probability of Y given X is proportional to the probability of Y times the probability of X given Y (where X and Y correspond to samples and labels, respectively). For the implementation of the model, we compute log likelihoods rather than likelihoods in order to avoid underflow (from repeated multiplications of small numbers) and to simplify calculations (e.g. converting multiplication operations to addition, using log rules). The algorithm pseudocode which helped guide this implementation is found by the textbook Introduction to Information Retrieval [**IIR**], found here. Material from lecture also aided with the implementation.

4.3 SGD Classifier

Since the Bernoulli Naive Bayes model is simple (when compared to other ML models), we decided to compare the performance of our Naive Bayes algorithm to another relatively simple model. I chose sklearn's logistic regression model since it was well studied in MP1 and is of similar complexity to the Bernoulli Naive Bayes model. Briefly, it is a model that models log-odds ratio as a linear function of input features. However, since the task is to categorize rich text data for a Kaggle competition, I chose to use a logistic regression model trained with stochastic gradient descent (SGD). This is implemented in the `SGDClassifier` class with hyperparameters: `loss='log_loss', penalty='l2', alpha=1e−3, random_state=42, tol=1e−3`. Further details about the class can be found here.

5 Results

5.1 Naive Bayes Bernoulli

Using 10-fold cross validation on the training set, we obtain an average accuracy of 77.3%. Table 1 displays accuracy scores for each folds.

Fold	fold 1	fold 2	fold 3	fold 4	fold 5	fold 6	fold 7	fold 8	fold 9	fold 10
Accuracy (%)	73.6	79.2	84.7	75.0	75.0	75.0	83.3	68.1	77.5	81.7

Table 1: Accuracy scores across 10 folds during the training and validation phase.

The training and prediction process had a runtime of 12m31s. Since we found that the SGD model performed better on its 10-fold cross validation, we did not use the Naive Bayes model for the Kaggle competition.

5.2 SGD Classifier

Using 10-fold cross validation on the training set, we obtain an average accuracy of 92.6%. Table 2 displays accuracy scores for each folds.

Fold	fold 1	fold 2	fold 3	fold 4	fold 5	fold 6	fold 7	fold 8	fold 9	fold 10
Accuracy (%)	91.7	94.4	91.7	90.3	91.7	95.8	94.4	88.8	93.0	94.4

Table 2: Accuracy scores across 10 folds during the training and validation phase.

The train and validation phase had a runtime of 13s. Since this classifier performed better than the Naive Bayes model for 10-fold cross validation, this model was used for the Kaggle competition. At submission, it scored an accuracy of 95.18%, tested on roughly 30% of the testing data.

6 Discussion and Conclusion

Our experiments showed that the SGD Classifier model outperformed the Bernoulli Naive Bayes model on 10-fold cross validation, with these models averaging 92.6% and 77.57% accuracy respectively. A possible explanation for is that the Bernoulli model makes a strong simplifying assumption; it assumes that token counts are not relevant and only considers the presence of tokens within samples. It is possible this assumption is too strong and removes too much information for the model's classification. The SGD Classifier was used for the Kaggle competition and scored 95.18% on the preliminary test set. Future work should explore the performance of the SGD model on various combinations of hyper-parameters, as well as other models such as Decision Trees and SVMs.

7 Statement of Contributions

This work was completed by Norman Kong.