Project 2: Reddit Text Classification with Naive Bayes and Scikit-Learn

Miguel Alfaro
McGill University
miguel.alfarozapata@mail.mcgill.ca

Tolu Olutayo
McGill University
toluwaleke.olutayo@mail.mcgill.ca

Jillian Cardinell
McGill University
jillian.cardinell@mail.mcgill.ca

Abstract

In this project, we experiment with a series of supervised learning models to predict the subreddit category labels of text from a dataset of Reddit posts. We experiment with different modes of Naive Bayes algorithm, Random Forests, Logistic Regression, and Support Vector Machines. We also experiment with various feature extraction methods aiming to optimize performance of these algorithms on the Reddit dataset. We evaluate and compare all models using 10-fold cross validation accuracy on an available training set of 11 582 Reddit posts. The model with the highest 10-fold accuracy was submitted to the Reddit Classification competition on Kaggle. The model found to have the highest accuracy was Logistic Regression trained with a maximum of 500 iterations,tf-idf vectorizer with a document frequency minimum of 0.0004, the Scikit-Learn English stop word set, and our novel vocabulary selection method paired with χ^2 selection of the top 5000 features. This model presented a 0.8929 10-fold accuracy on the provided training set. This model also obtained a 0.8815 accuracy on 30% of the final testing data, as reported by Kaggle.

1 Introduction

Text classification is an important problem in modern machine learning. Common examples of text classification tasks include classifying emails as spam based on their text content, classifying movie reviews as positive or negative, or deciding the main topic of some text article [1].

In this project, we work to classify Reddit posts based on the subreddit it was posted to. The training set provided consists of 11 582 Reddit posts from 8 different subreddits. We experiment with several classifiers and many different modes of each classifier to determine which one is most fit for the task. We focus on variants of Bernoulli Naive Bayes, Random Forests, Logistic Regression, and Support Vector Machines (SVMs). To tailor each model to the Reddit dataset, we apply a series of feature selection methods, balancing methods, class prior modes, and other data-based modifications to the models. All models are evaluated and compared using 10-fold cross-validation accuracy on the entire training set. The model with the highest 10-fold cross-validation accuracy will be used on the final test data and submitted to the Reddit Kaggle Competition.

2 Data

The Reddit training dataset contains 11 582 training samples from 8 different subreddits which act as categorical labels. The subreddit labels include anime, cars, computers, datascience, gamedev, gamernews, hardware, and rpg. Figure 1 shows the distribution of the training samples amongst the subreddits. There is an imbalance in the data, with the majority of the samples being from the datascience subreddit (20.6% of the dataset), and a small portion of data belonging to the computers subreddit (3.7% of the dataset). Cars and rpg are the other predominant subreddits comprising 17.8% and 17.2% of the dataset respectively. The average length in words of all the Reddit posts is 105.9 words. Note that this data was found to contain 200 duplicate samples. These duplicates were removed before training in order to prevent additional biasing.

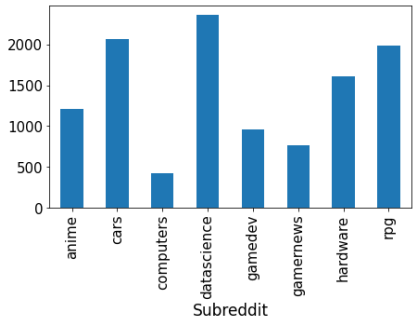

Figure 1: Class distribution of the Reddit test data

3 Methods

3.1 Naive Bayes

Naive Bayes is a simple Bayesian probabilistic algorithm that assumes features are conditionally independent given the corresponding labels. Naive Bayes predicts by maximizing the log-likelihood function defined as $L(D) = \prod_{i=1:n} P(y_i|x_i)$, where y_i is the label and x_i is the data [2].

Previous experiments with Naive Bayes variants have found that small tweaks to the basic model can improve performance. In experiments with short documents and class imbalances, the class prior probabilities in the formula have been found to decrease performance [3]. For our study, our documents are fairly short and suffer from class imbalance, therefore we can potentially improve performance by assuming a uniform class prior. The dictionary used, or feature set used, is also important for performance. Prior to Naive Bayes classification, feature selection is performed by analyzing the relationship between a certain feature (word) and the output class. Correlation statistics can be used to find words (or features) that are most related to the class label [3]. The χ^2 statistic, mutual information (MI), and ANOVA F-value method can all be used to select a refined dictionary [3,4].

In this experiment we test a series of combination models, using χ^2, MI, and ANOVA-F with both the traditional Naive Bayes algorithm and the uniform prior Naive Bayes.

3.2 Random Forest

Random forests (RF) are an ensemble method that employs a series of decision trees. N many bootstrap samples are made and N many trees are made from each set. A random subset of features is used at each node in every tree, and none of the trees are pruned. RFs make a final prediction from the results of many trees [5].

SK-Learn provides a default RF model, *sklearn.ensemble.RandomForestClassifier*, and many of the base parameters settings are considered optimal by the Sk-Learn team; however, some variables require extra attention [4]. We use Scikit-Learn's RF model with some parameter modifications. RFs should consider the number of total features in the dataset when setting the number of feature subsets to use at each node. The default method in Scikit-Learn is the square root of the total number of features [4]. However, it is recommended that the number of features used at each node is further experimented with [6]. For the RF experiments, we look at 5000 features selected using ANOVA-F, so the recommended value for the number of features at each node is approximately 71. To explore the accuracy more, we will test +/- 10 around this recommended value. Another important hyperparameter to consider is the number of trees in the forest. A previous extensive study used several datasets and a wide assortment of forests with a different number of trees. They found that the optimal range they reported was 64-128 trees [7]. For this paper, we test the extremes of this range. The criteria for splitting at the nodes is also an important consideration. Information gain is the metric to determine the splitting criteria; however, there are two main ways of calculating information gain.The default setting is the gini index, which can be more computationally efficient as it does not include a log in the calculation; however, the entropy metric has a wider range so it may allow for more precise decision making [8]. In this study we experiment with both.

3.3 Logistic Regression

Logistic Regression (LR) is a binary prediction model. It linearly models the log-odds ratio of the probabilities as $a = ln\frac{P(y=1|x)}{P(y=0|x)} = \mathbf{w}^T\mathbf{x}$, then predicts the class of new data by taking the sigmoid function of the linear model [9]. Scikit-Learn provides a model for LR, *sklearn.linear_model.LogisticRegression*, and the researchers suggest extending LR to multiclass problems using the one-vs-all approach [4,10]. One-vs-all makes a LR binary classifier for every

class, where one class is treated as "1" and all other classes are treated as "0" [11]. We also use L2 regularization as recommended by Scikit-learn developers [10].

We assume that the test data will have a similar class-distribution as the training data provided since the more frequently occurring classes are likely more popular subreddits. For this reason, we do not perform any weight adjustments to balance the classifier because the biases introduced from the class imbalances will presumably create desirable classifier performance.

Other than balancing, LR does not have a mode for feature management based on the data, so for this model, we focus on feature selection experiments. We experiment with a variety of feature selector methods including stop word sets, TF-IDF, χ^2 and different parameters of these selection methods [12]. TF-IDF is a scoring method that orders terms by their term frequency (TF) multiplied by their document frequency (IDF). Term frequency is defined as the term appearance divided by the document length, and IDF is defined as $log \frac{\#of\ documents\ in\ corpus}{Documents\ containing\ the\ term+1}$. We also present a novel combination vocabulary selection algorithm. We propose to study the prediction results of some base LR classifiers and identify the class pairs that are misclassified the most. These poorly classified pairs are then isolated into separate datasets and features are selected from them using a TF-IDF vectorizer with χ^2 feature selector. These features found from the isolated datasets are then added to the feature set found on the whole dataset, and duplicates are deleted. This is done to provide more distinguishing information on the classes that are often confused. We term this method vocabulary selection.

3.4 Support Vector Machine

SVMs attempt to separate two classes of data with a hyperplane in the feature space. SVMs form a hyperplane by identifying the closest points (support vectors) to the hyperplane and maximizing the distance between the two (the margin) [13]. In hard SVMs, the support vectors must also remain outside of some defined margin [14]. In cases where the data cannot be separated in the basic feature space, kernel methods can be employed. Kernel functions map the data on a higher dimensional non-linear space to allow the data to be more easily separated by a hyperplane and margin. The radial basis function (RBF) kernel is a favoured kernel because it has infinite expansions and therefore has great potential to separate linearly non-separable data [15].

SVMs are inherently binary; however, there are several methods to adjust the SVM model to work with multiclass data. The two most common methods are one-vs-all or one-vs-one decision functions. As mentioned before, one-vs-all makes a binary classifier for every class, thus we have N many SVMs for N many classes. One-vs-one makes a binary classifier for each pair of classes so we have $\frac{N(N-1)}{2}$ [11]. The preferred type is a debated topic, however the one-vs-one type SVMs do train faster and tend to work better with problems with many classes [16]. For this study we experiment with both one-vs-one and one-vs-all.

Although previously we did not employ class balancing efforts on the LR classifier to create valuable biases, an uneven distribution in classes can generally decrease the performance of SVMs. Therefore, in these experiments with SVMs, the balanced mode is used. This adjusts the "C" parameter of the SVM classifier, which decides the misclassification impact on the decision boundary, in other words, the strictness of the margin. The balanced mode sets the C vector such that the C value for each class is inversely proportional to the frequency of that class, meaning it is more strict for classes with fewer samples. Since this mode is recommended by Scikit-Learn in imbalanced problems, we use it in all our SVM experiments [17]. We use the Scikit-learn module for hard SVMs, *sklearn.svm.SVC* [4]. Unless explicitly mentioned in this section, the hyperparameter settings defined by Scikit-Learn for SVC are kept as default.

4 Results

All models are trained and validated on the entire training set of Reddit data provided. Each model is trained and evaluated using 10-fold cross-validation accuracy. The data is randomly shuffled for cross-validation experimentation. All models use the same basic vectorizing function, with specific feature extraction methods used as explained in the Methods section.

Table 1 shows the results of the experiments using various Naive Bayes classifiers. These results show that, contrary to our hypothesis, the uniform prior decreased performance overall. Additionally, ANOVA-F was the best method of feature extraction, yielding the highest 10-fold accuracy of 0.8769. The worst performing feature extraction method only used the 5000 most frequently occurring words, without actually analyzing the relationship between the word and the label. This relatively low performance makes sense since the features are not selected with consideration of the class label.

Table 1: 10-fold cross-validation accuracy on the Reddit data set from variants of the Bernoulli Naive Bayes classifier.

Model		**10-Fold Cross-Validation Accuracy**
Prior	**Feature Selection**	
Classic Prior	**Most frequent**	0.8537
	ANOVA-F	**0.8769**
	χ^2	0.8755
	MI	0.8548
Uniform Prior	**Most frequent**	0.8445
	ANOVA-F	0.8717
	χ^2	0.8723
	MI	0.8530

Table 2: 10-fold cross-validation accuracy on the Reddit data set from variants of the Random Forest classifier.

Gini		
Maximum Features	**Trees**	**10-Fold Cross-Validation Accuracy**
61	**64**	0.8187
	128	**0.8214**
71	**64**	0.8154
	128	0.8174
81	**64**	0.8117
	128	0.8158
Entropy		
61	**64**	0.8050
	128	0.8058
71	**64**	0.7998
	128	0.8052
81	**64**	0.7985
	128	0.8021

The results of the various RF classifiers are shown in Table 2. This series of experiments found that with this data set, a higher number of trees improved performance. All trials with 128 trees performed higher than their 64 tree equivalents. The number of maximum features used at each node had the opposite trend. Trials with a higher number of maximum features at each node performed worse than those with only 61 features. RFs using gini-index as the criteria performed better than their entropy counterparts. The gini-index does not use logarithm of probability, unlike entropy, an directly sums the square of the probability [8]. The direct use of the probabilities could have lead to the improved results seen in the gini experiments. Overall the best model used gini, 61 features, 128 trees, and obtained an accuracy of 0.8214.

Table 3: 10-fold cross-validation accuracy using Logistic Regression models with different feature extractors.

Feature Extraction Mode		**10-Fold Cross-Validation Accuracy**
TF-IDF settings	**Correlation Statistic**	
Maximum features = 5000	None	0.8816
Maximum features = 10000	χ^2, 5000 features	0.8862
	ANOVA-F, 5000 features	0.8851
Document frequency minimum = 0.0004	χ^2, 2500	0.8720
	χ^2, 5000	0.8863
	χ^2, 7500	0.8876
Document frequency minimum = 0.0004, English stop word set	χ^2, 7500	0.8872
Document frequency minimum = 0.0004, English stop word set, Vocabulary selection	χ^2, 5000	**0.8929**

Table 3 shows the results of varying feature selection methods for the Logistic Regression Model. All trials use the TF-IDF vectorizer. The stop word list from sklearn was used to remove commonly used words and a minimum

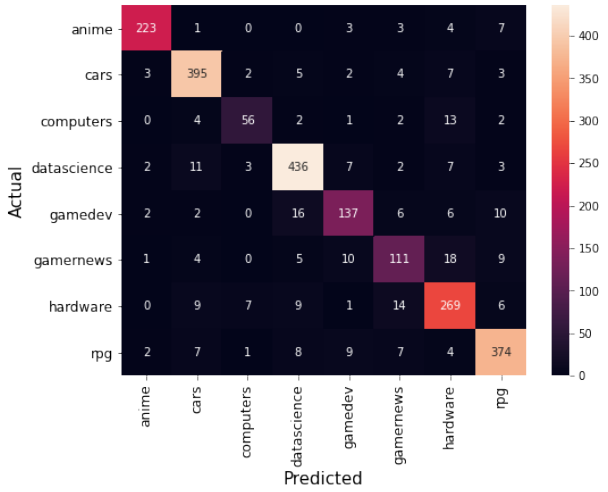

Figure 2: Confusion Matrix over hold-out data (20% of the training set) using the first prediction.

document frequency of 0.0004 was used to ignore infrequent terms and typographical errors. Using a maximum set of 10000 words with the TF-IDF vectorizer, the χ^2 feature selector performed better than the ANOVA-F feature selector. The accuracies of three different vocabulary sizes were compared. As the performance of the model did not increase significantly from 5000 to 7500 words, we keep the size of vocabulary to 5000 words.

To perform the vocabulary selection described in the Methods section, a confusion matrix was generated over a validation data set using TF-IDF document frequency minimum=0.0004, 7500 words selected usingχ^2, and stop word list. Figure 2 shows that most miss-classification occurred between the classes: computers, hardware, gamer news, and game development. A focused vocabulary from these classes was extracted and selected using TF-DF and χ^2. This vocabulary was added to the one from the first part (TF-IDF+χ^2=5000). This vocabulary method combined with document frequency cut off and stop word set produced the highest accuracy of 0.8929.

Table 4: 10-fold cross-validation accuracy on the Reddit data set from different decision function modes of SVM.

Model Type	**10-Fold Cross Validation Accuracy**
one-v-one	0.8517
one-v-all	**0.8524**

Table 4 shows that in this particular study, decision function shape has a very minute impact on 10-fold accuracy. Both one-v-one and one-v-all models perform fairly well, and only differ by 0.0007 with one-v-all being slightly superior. This supports the results reported in other literature on the uncertain information regarding a preferred decision function shape. Either one of these models is likely to perform well on the final test data.

5 Discussion and Conclusion

Our experiments explored different models, feature selection, and optimization methods for the Reddit text classification problem. The set of models studied in this paper included Naive Bayes, RFs, LRs, and SVMS. In general, we found that of these models LRs performed the best, consistently reaching an accuracy of at least 0.8720. RFs presented the lowest overall accuracies, maxing out at only 0.8214. The model that we recommend based on our experiments is the logistic regression model combined with a minimum document frequency of 0.0004, the English stop word set, and the vocabulary creation method defined in this work. This algorithm returned a 0.8929 cross validation accuracy on the training set, and an accuracy score of 0.8815 on 30% of the testing data as reported by the Reddit Classification competition on Kaggle.

6 Statement of Contributions

Tolu developed the read-in and output pipelines, and wrote the Naive Bayes code. Tolu wrote the README file and formatted the final version of the code for submission. Jillian and Miguel researched tailoring methods for the Scikit-Learn models. Miguel wrote the code for the models, and created the vocabulary. Jillian and Miguel performed data analysis. Jillian wrote the report. Miguel and Tolu edited the report.

References

[1] D. Jurafsky, Class Lecture, Topic: “Text Classification and Naive Bayes”, CS124, Stanford University, Stanford, California, USA, Jan. 2020.
https://web.stanford.edu/class/cs124/lec/naivebayes.pdf [2] N. Armanfard, Class Lecture, Topic: "Lecture 8 —Linear Classifiers cont.", ECSE 551, Electrical and Computer Engineering Department, McGill University, Montreal, Canada, Sept. 2020
[3] K. M. Schneider, “Techniques for improving the performance of naive bayes for text classification,” *Lect. Notes Comput. Sci.*, vol. 3406, no. i, pp. 682–693, 2005.
[4] F. Pedregosa et al., “Scikit-learn: Machine Learning in Python,” *J. Mach. Learn. Res.*, vol. 12, pp. 2825–2830, 2011.
[5] N. Armanfard, Class Lecture, Topic: "Lecture 16 —Ensemble ", ECSE 551, Electrical and Computer Engineering Department, McGill University, Montreal, Canada, Nov. 2020
[6] L. Brieman and A. Cutler, “Random forests - classification manual,” 2007. [Online]. Available: https://www.stat.berkeley.edu/ breiman/RandomForests/cc_manual.htm.
[7]T. Oshiro, P. Perez and J. Baranauskas, "How Many Trees in a Random Forest?", *Machine Learning and Data Mining in Pattern Recognition*, pp. 154-168, 2012. Available: 10.1007/978-3-642-31537-4_13.
[8]A. Hershy, “Gini Index vs Information Entropy,” Medium, 10-Jul-2019. [Online]. Available: https://towardsdatascience.com/gini-index-vs-information-entropy-7a7e4fed3fcb.
[9] N. Armanfard, Class Lecture, Topic: "Lecture 6 — Linear Classification", ECSE 551, Electrical and Computer Engineering Department, McGill University, Montreal, Canada, Sept. 2020
[10] Scikit-Learn Developers. sklearn.linear_model.LogisticRegression v 0.23.2. [Online]. Available https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.LogisticRegression.html
[11] H. Li, F. Qi, and S. Wang, “A comparison of model selection methods for multi-class support vector machines,” in *Lecture Notes in Computer Science*, 2005, vol. 3483, no. IV, pp. 1140–1148.
[12] N. Armanfard, Class Lecture, Topic: "Lecture 12 — Feature Construction, Dimension Reduction", ECSE 551, Electrical and Computer Engineering Department, McGill University, Montreal, Canada, Oct. 2020
[13] N. Armanfard, Class Lecture, Topic: "Lecture 14 —Instance-Based Learning (cont.), Support Vector Machines ", ECSE 551, Electrical and Computer Engineering Department, McGill University, Montreal, Canada, Oct. 2020
[14] N. Armanfard, Class Lecture, Topic: "Lecture 15 — Support Vector Machines (cont.) ", ECSE 551, Electrical and Computer Engineering Department, McGill University, Montreal, Canada, Oct. 2020
[15] A. Shashua, “Introduction to Machine Learning: Class Notes 67577,” arXiv Prepr., pp. 1–105, 2009.
[16]J. Milgram, M. Cheriet, and R. Sabourin, “‘One Against One’ or ‘One Against All’: Which One is Better for Handwriting Recognition with SVMs?,” *Tenth Int. Work. Front. Handwrit. Recognit.*, pp. 1–6, 2006.
[17] Scikit-Learn Developers. 1.4. Support Vector Machines v 0.23.2. [Online]. Available https://scikit-learn.org/stable/modules/svm.htmlsvm-classification

Appendix A

```
from google.colab import drive
drive.mount('/content/gdrive')

# imports
import numpy as np
import pandas as pd
from sklearn.feature_extraction.text import CountVectorizer #can try using tf-idf to see if
    improved accuracy
from sklearn.pipeline import Pipeline
from sklearn.feature_selection import SelectKBest, chi2, f_classif
from sklearn.feature_extraction import text
from sklearn import preprocessing #encode labels as integers from 0-7
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import accuracy_score
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score
import spacy
##
import nltk
nltk.download('punkt')
nltk.download('wordnet')
nltk.download('stopwords')
from nltk.stem import PorterStemmer
```

```
from nltk import word_tokenize
from nltk.stem import WordNetLemmatizer
from nltk.corpus import wordnet
from sklearn.feature_extraction import text
from nltk.corpus import stopwords

#change filepath
pd_train_data = pd.read_csv('/content/gdrive/My Drive/ECSE 551 Project 2/train.csv')
pd_test_data = pd.read_csv('/content/gdrive/My Drive/ECSE 551 Project 2/test.csv')
#pd_train_data.head()


#uncomment this block to run the logistic regression model
'''
'''Create Vocabulary'''
pf = pd_train_data.drop_duplicates(subset='body')
#print(pd_train_data['body'][9745]==pd_train_data['body'][9785])
#print(pf.shape)
#train_data = pd_train_data['body'].tolist()
#train_labels = pd_train_data['subreddit'].tolist()
train_data = pf['body'].tolist()
train_labels = pf['subreddit'].tolist()
sklearn_stopwords = text.ENGLISH_STOP_WORDS

vectorizer0 = TfidfVectorizer(min_df=.0004,stop_words=sklearn_stopwords,analyzer='word')
X0 = vectorizer0.fit_transform(train_data)
features=vectorizer0.get_feature_names()
train_data20 = X0.toarray()
train_data0 = SelectKBest(chi2,k=5000).fit(train_data20,train_labels)

new_features = []
for index in train_data0.get_support(indices=True):
       new_features.append(features[index])
print(len(new_features))
rpg_data=[]
rpg_label=[]
anime_data=[]
datascience_data=[]
datascience_label=[]
hardware_data=[]
hardware_label=[]
cars_data=[]
gamernews_data=[]
gamernews_label=[]
gamedev_data=[]
gamedev_label=[]
computers_data=[]
computers_label=[]
'''separate by classes'''
for lineNum in range(len(train_data)):
    if (train_labels[lineNum]=='rpg'):
        rpg_data.append(train_data[lineNum])
        rpg_label.append(train_labels[lineNum])
    #elif(train_labels[lineNum]=='anime'):
    #    anime_data.append(train_data[lineNum])
    elif(train_labels[lineNum]=='datascience'):
        datascience_data.append(train_data[lineNum])
        datascience_label.append(train_labels[lineNum])
    elif(train_labels[lineNum]=='hardware'):
        hardware_data.append(train_data[lineNum])
        hardware_label.append(train_labels[lineNum])
    #elif(train_labels[lineNum]=='cars'):
    #    cars_data.append(train_data[lineNum])
    elif(train_labels[lineNum]=='gamernews'):
        gamernews_data.append(train_data[lineNum])
```

```
        gamernews_label.append(train_labels[lineNum])
    elif(train_labels[lineNum]=='gamedev'):
        gamedev_data.append(train_data[lineNum])
        gamedev_label.append(train_labels[lineNum])
    elif(train_labels[lineNum]=='computers'):
        computers_data.append(train_data[lineNum])
        computers_label.append(train_labels[lineNum])
GDvsDS=list.copy(gamedev_data)
GDvsDSlabel=list.copy(gamedev_label)
for lineNum in range(len(datascience_data)):
    GDvsDS.append(datascience_data[lineNum])
    GDvsDSlabel.append(datascience_label[lineNum])

GNvsHW=list.copy(gamernews_data)
GNvsHWlabel=list.copy(gamernews_label)
for lineNum in range(len(hardware_data)):
    GNvsHW.append(hardware_data[lineNum])
    GNvsHWlabel.append(hardware_label[lineNum])

GNvsRPG=list.copy(gamernews_data)
GNvsRPGlabel=list.copy(gamernews_label)
for lineNum in range(len(gamedev_data)):
    GNvsHW.append(rpg_data[lineNum])
    GNvsHWlabel.append(rpg_label[lineNum])

PCvsHW=list.copy(computers_data)
PCvsHWlabel=list.copy(computers_label)
for lineNum in range(len(hardware_data)):
    GNvsHW.append(hardware_data[lineNum])
    GNvsHWlabel.append(hardware_label[lineNum])

'''vectorize each class separately'''
def getVocabulary(dataset,datalabel):
    vect = TfidfVectorizer(max_df=.9, stop_words=sklearn_stopwords, analyzer='word')
    Data=vect.fit_transform(dataset)
    features=vect.get_feature_names()
    bestData = SelectKBest(chi2,k=1000).fit(Data.toarray(),datalabel)
    indexes=bestData.get_support(indices=True)
    nfeatures = []
    for index in indexes:
        nfeatures.append(features[index])
    return nfeatures

vocabulary=getVocabulary(GNvsHW,GNvsHWlabel)
for lineNum in range(len(vocabulary)):
  new_features.append(vocabulary[lineNum])
vocabulary=getVocabulary(GDvsDS,GDvsDSlabel)
for lineNum in range(len(vocabulary)):
    new_features.append(vocabulary[lineNum])
vocabulary=getVocabulary(GNvsHW,GNvsHWlabel)
for lineNum in range(len(vocabulary)):
    new_features.append(vocabulary[lineNum])
vocabulary=getVocabulary(PCvsHW,PCvsHWlabel)
for lineNum in range(len(vocabulary)):
    new_features.append(vocabulary[lineNum])
#for word in ['ps2','ps3','ps4','ps5','ps1','480p','pass']:
#  new_features.append(word)

words=set(new_features)'''

#comment this block out to run logistic regression model

#erase before final version
pf = pd_train_data.drop_duplicates(subset='body')
#print(pd_train_data['body'][9745]==pd_train_data['body'][9785])
```

```
#print(pf.shape)
#train_data = pd_train_data['body'].tolist()
#train_labels = pd_train_data['subreddit'].tolist()
train_data = pf['body'].tolist()
train_labels = pf['subreddit'].tolist()
sklearn_stopwords = text.ENGLISH_STOP_WORDS
vectorizer = CountVectorizer(strip_accents = ascii, max_features = 10000,binary=True,
                         stop_words='english',analyzer='word',ngram_range=(1, 1)) #should we put
                             stop words?
X = vectorizer.fit_transform(train_data)
train_data2 = X.toarray()
train_data = SelectKBest(f_classif,k=5000).fit_transform(train_data2,train_labels)
#to do the inverse of the result do not forget to use le object by using inverse_transform(y)
    method
le = preprocessing.LabelEncoder()
train_labels = le.fit_transform(train_labels)

'''#uncomment this block to run logistic regression model
#vectorize text data
vectorizer = TfidfVectorizer(vocabulary=words,analyzer='word',binary=True)
X = vectorizer.fit_transform(train_data)
train_data = X.toarray()

#to do the inverse of the result do not forget to use le object by using inverse_transform(y)
    method
le = preprocessing.LabelEncoder()
train_labels = le.fit_transform(train_labels)'''

#create hold out data for testing
X_train, X_test, y_train, y_test = train_test_split(train_data, train_labels, test_size=0.20,
    random_state=42)
print(X_train.shape,y_train.shape,X_test.shape,y_test.shape)

#create Naivebayes class
class Naivebayes:
   def fit(self,X,y):
           #group training data into distinct classes and store as a list\n",
           grouped = []
           for i in np.unique(y):
             temp = []
             for j,k in zip(X,y):
               if k==i:
                 temp.append(j)
             grouped.append(temp)

           #calculate prior probabilities of each class and store in a list
           #use logs to avoid floating point underflow because we will be multiplying-
           #a lot of small numbers together and we dont want them getting approximated to zero

           self.prior_prob = np.array([np.log(len(i)/len(y)) for i in grouped])
           print()

           #calculate conditional probabilities for each class
           #variables needed:
           #(1) how many documents does word occur for a given class,(2) total no of documents of
               class and-
           #(3) add laplace smoothing constant

           #(1)
           word_occurs = np.array([np.array(i).sum(axis=0) for i in grouped])+1

           #(2)
           no_of_doc = np.array([[np.array(i).shape[0] for i in grouped]])+2
```

```
            #compute conditional probabilities by dividing. Use vectors instead of writing a for
                loop
            self.cond_prob = word_occurs/no_of_doc.T

    def predict(self, X):
      #store result of log probabilities
      #select the max
      #assume X is binary encoded
      self.labs = []
      for i in X:
        result = []
        temp_yes = np.log(self.cond_prob)*i #zeros out non occuring words and calculates log of
            conditional probabilities
        temp_no = np.array(1-i)*np.log(1-self.cond_prob) #zeros out occuring words and calculates
            log of 1-conditional probabilities
        temp = temp_yes + temp_no
        for j in temp:
          result.append(np.array([j]).sum(axis=1))
        result = np.array(result).T+self.prior_prob
        self.labs.append(np.argmax(result))

    def accu_eval(self,y):
      self.acc = 0
      for i in range(len(y)):
        if self.labs[i]==y[i]:
          self.acc+=1
      print("The accuracy over the test data is",self.acc/len(y)*100,"%")
      return self.acc/len(y)

    def write_to_csv(self):
      #return csv file with labels
      self.labs = le.inverse_transform(self.labs)
      dt = {'subreddit':self.labs}
      df = pd.DataFrame(dt)
      df.to_csv('out.csv',columns=['subreddit'],index_label=['id'])

d = Naivebayes()
d.fit(X_train,y_train)
d.predict(X_test)
d.accu_eval(y_test)

test_data = pd.read_csv('/content/gdrive/My Drive/ECSE 551 Project 2/test.csv')
test_data = test_data['body'].tolist()
test_data = vectorizer.transform(test_data)
#x_new = SelectKBest(chi2,k=5000).fit(train_data2,train_labels)
#test_data = x_new.transform(test_data)
test_data = test_data.toarray()
print(test_data.shape)

td = Naivebayes()
td.fit(train_data, train_labels)
td.predict(test_data)

#change filepath
td.write_to_csv()
s = pd.read_csv('./out.csv')
s.head()

def write_to_csv_sklearn(labs, model):
  #return csv file with labels
  labs = le.inverse_transform(labs)
  dt = {'subreddit': labs}
  df = pd.DataFrame(dt)
  filename = model + '_out.csv'
  df.to_csv(filename, columns=['subreddit'],index_label=['id'])
```

```
#Logistic Regression
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score

model = LogisticRegression(max_iter = 500)
logreg = model.fit(X_train, y_train)

predictions = logreg.predict(X_test)
acc = accuracy_score(y_test, predictions)
print('Logistic Regression Accuracy:', acc)

test_predictions = logreg.predict(test_data)

write_to_csv_sklearn(test_predictions, 'logreg')

from sklearn.model_selection import KFold
from sklearn.metrics import accuracy_score


#this loop will run n_splits number of times
def K_Fold_Eval(n_splits, model, X_train, y_train):
  kf = KFold(n_splits, shuffle = True)
  accs = []

  for train_ind, val_ind in kf.split(X_train):
    kf_x_train, kf_x_val = X_train[train_ind], X_train[val_ind]
    kf_y_train, kf_y_val = y_train[train_ind], y_train[val_ind]

    fitted = model.fit(kf_x_train, kf_y_train)
    predictions = fitted.predict(kf_x_val)
    acc = accuracy_score(kf_y_val, predictions)
    print(acc)
    accs.append(acc)

  av_acc = sum(accs)/len(accs)

  return av_acc

def K_Fold_Eval_NB(n_splits, model, X_train, y_train):
  kf = KFold(n_splits = n_splits, shuffle = True)
  accs = []

  for train_ind, val_ind in kf.split(X_train):
    kf_x_train, kf_x_val = X_train[train_ind], X_train[val_ind]
    kf_y_train, kf_y_val = y_train[train_ind], y_train[val_ind]

    #fitted = model.fit(kf_x_train, kf_y_train)
    model.fit(kf_x_train, kf_y_train)
    model.predict(kf_x_val)
    acc = model.accu_eval(kf_y_val)
    accs.append(acc)

  av_acc = sum(accs)/len(accs)

  return av_acc


#av_acc_log = K_Fold_Eval(10, LogisticRegression(max_iter = 500), X_train, y_train)
#print('K-fold Accuracy for Logistic Regression', av_acc_log)



#Logistic Regression K fold
```

```
av_acc_log = K_Fold_Eval(10, LogisticRegression(class_weight= None, max_iter = 500), train_data,
    train_labels)
print('K-fold Accuracy for Logistic Regression', av_acc_log)

k_fold_NB = K_Fold_Eval_NB(10, Naivebayes(), train_data, train_labels)
print('K-fold Accuracy for Naivebayes', k_fold_NB)
```