A

Report on

COMP 4900 Mini-project 2

by

Group 2-21

Group members:

Zeye Gu

(Student No. 101036562)

Peng Li

(Student No. 101047123)

Chengyi Song

(Student No. 101033544)

(Carleton University)

2020-11-7

ABSTRACT

The goal of this project is to classify comments from Reddit into different subreddits, for example, some comments are from the hardware subreddit and some are from the gamernews subreddit. We built Bernoulli Naïve Bayes from scratch, and import Logistic Regression, Decision Tree, Stochastic Gradient Descent from sklearn. The main steps choose useful words firstly from the comments and then train the program with those words. We tried different methods to choose words to include TF-IDF, vectorize, lemmatizer, and feature selection, which gives different accuracies.

There several important findings for this project. First is that the accuracy of classification depends on the choice of words. Choosing useful words from comments provides much higher accuracy. Second is that the implementation of Logistic Regression provides better accuracy than Bernoulli Naïve Bayes. The last finding is that using shuffles to randomize the data in the k-folder also helps to improve the accuracy of classification.

1 Introduction and Motivation

The task of this project is to train the program to classify comments, which form Reddits, into eight different subreddits. There are two datasets provided for this project. The training dataset includes 11582 classified comments from eight subreddits, rpg, anime, datascience, hardware, cars, gamernews, gamedev, and computers. The test dataset includes 2897 comments without classification. To do the classification, we implemented Bernoulli Naïve Bayes from scratch as the first part. We used three different classifiers from SciKit learn package as the second part of the project. For the implementation of Bernoulli Naïve Bayes, we used vectorization, and lemmatization to select the first 5000 features. For the classifiers that we directly used from sklearn library, we used TF-IDF and feature selection to select useful words.

When we working on this project, we have three important findings. The first one is about the choice of words. As we tried different approaches, we found that different words chosen would lead to huge different accuracy. The second finding is that the best result comes from Stochastic Gradient Descent which is the best model to predict the test dataset. The third finding is that, shuffling the data when testing with KFold shows an apparent improvement in validation accuracy.

2 Dataset

2.1 Dataset Introduction

The training dataset has 11582 samples and they are gathered from posts and comments from 8 subreddits. The test dataset includes 2897 comments without classifications.

```
datascience    2382
cars           2064
rpg            1988
hardware       1617
anime          1304
gamedev        1016
gamernews       784
computers       427
Name: subreddit, dtype: int64
```

Figure 1: The number of samples in each subreddits

2.2 Dataset preprocessing

The following paragraph is one of the rpg sample.

Call of Cthulhu should work absolutely fine. It has full rules support for games set in the present day and nothing about a wilderness setting should cause any problems. You will probably have to come up with your own survival rules though since that's not going to be very developed in the core published rules. It should be very easy for you to do with CoC's hit point ability score and skill rules. They're very flexible. Delta Green would work too but I think you'd need to do a bit more "trimming" if you want to get away from DG's emphasis on government agents. CoC is a little more easily adaptable for general "normal people" characters.

The string contains both capital and small letters, punctuations,words in different forms space characters and insignificant words such as "for", "to", "of" etc.

```
def check(word):
    word= word.lower()
    if bool(re.search(r'\d', word)) or bool(re.match(r'[^\w]', word)):
        return False
    else:
        return True
```

Figure 2: The check function in data preprocessing

By using check(word), all the numbers, punctuations and space characters will be removed and the filted string will be procssed to WordNetLemmatizer which is imported from nltk.stem so that "rules" will be transformed to "rule". The modified samples will be stored in a nparray.

3 Proposed Approach

3.1 Bernoulli Naïve Bayes

3.1.1 Fitting the data

As we want to implement Bernoulli Naïve Bayes mehotd, we need to calculate the value of parameter theta_k first . By taking input of text(x) and category(y), we are fitting each specific category in theta_k, and the probability of every single word also save into that specific category. For one word, there are two theta values, theta_0 and theta_1. Theta_0 is the value if there is no such word in test data, and theta_1 is the value if there is such word in test data

3.1.2 Predict

By taking input of test data, and predict each text in each category probability. There is a loop used to calculate the likelihood of eight subreddits. The method we used to calculate is the formula of Bernoulli Naïve Bayes,. After finishing the loop, we would get eight predict array which saves in our likelihood array, choose the highest value in each column to get the final prediction

3.2 Test with different classifers from sklearn

.

3.2.1 Build Python pipeline

Python pipeline is used to combine TF-IDF, feature selection, and classifer together to make the code proper in the code.

For TF-IDF, we set ngram_range=(1, 2),max_df=0.8, sublinear_tf=True, ngram_range to sperate samples so we have featrues. For example, if we apply ngram_range=(1, 2) on "I did it", the string will be spearate to "I", "did", "it", "I did ", and "did it". The parameter stop_words is not used here, however max_df = 0.7 is applied to automatically detect and filter stop words based on intra corpus document frequency of terms. For sublinear_tf=True, when caculating TF-IDF, it uses 1+log(tf) instead of tf.

For feature selection, we selected the top 20 percent of the featrues to keep. The feature score is caculated by ANOVA F-value for the provided sample.

Three different classifiers from sklearn have been tested and compared in the project. We applied KFold cross validation with K = 5 and shuffle = true, TF-IDF, and feature selection with the same hyper-parameters that were mentioned in the previous paragraphes.

3.2.2 Stochastic Gradient Descent

The estimator implements regularized linear models with stochastic gradient descent(SGD) learning. The test accuracy of SGD is the highest, and it is slightly higher than the test accuracy of Logistic regression.

Suppose there are n samples. In every iteration of gradient descent, it caculate n derivatives. However, in SGD, it randomly picked one or a subset of the sample(mini-batch) to caculate the derivative. It is more effective than gradient descent when estimate by using large dataset.

4 Results

4.1 Bernoulli Naïve Bayes

Fold/Accuracy	Train Accuracy	Validation Accuracy
1	0.8345249196974407	0.7757638529259451
2	0.8347321521085898	0.7897462454686691
3	0.8394115209283051	0.7917098445595855
4	0.8319519270617488	0.794818652849741
5	0.8360961458765023	0.778238341968912
6	0.8403439701616245	0.7756476683937824
Aveage	0.8361767726390351	0.7843207676944391

Table 1: Train accuray and Validation accuary Table for Bernoulli Naïve Bayes

By using 6 K-fold to training and validation the data, and the final average of them. As the output shows, we got the accuracy with the train set and test set. The accuracy of the training set similar to the test set, which means the prediction result is reliable. There is also no big difference between the accuracy of each fold. The accuracy is influenced by the choice of words, so with the method, we used for choosing words, the final average accuracy is about 80 percent.

4.2 Test with different classifers from sklearn

	Train Accuracy	Validation Accuracy
Logistic Regression	0.9307761804991503	0.8382828206028432
DecisionTree	0.9998273163882511	0.6638729060492283
SGD	0.9921429725441584	0.8897417376856351

Table 2: Train accuray and Validation accuary Table for applied classifiers

By using 5 K-fold cross validation and the same feature selection, tfidf setup, the Table 2 shows that SGD is the most appropriate model, and Decision Tree is the least appropriate model for this text classification dataset.

5 Discussion and Conclusion

After doing this project, we learned the efficient way to select the features which help to classify an object. For this project, the features are the words in the comments. We implemented different methods to select words, include stop words, TF-IDF, vectorization, lemmatization. The goal of feature selection of this project is to pass the words used very common or not related to the theme of comment, for example, words like "the" or "is". We want to reduce the size of features so that the classification based on features would be accurate. Stop words helps to find common words, but there would remain some words that are not common but not related to the theme. TF-IDF helps to select those words. The correct steps to select words are first vectorizing all the words. so that there would be no duplicated words. Then use lemmatization to find words with the same root, for example, apple and apples or ate and eat. Then use stop words to select out the common words. Finally, use TF-IDF to select out the words which not helpful for classification. The combination of those technique provides an efficient method of word selection. Except that, we used stochastic gradient descent to achieve our highest score of 88.84 percent on Kaggle. In the future, we need to find a way to transform words into the same form, for example, "buy" and "bought" are the same word.

6 Statement of Contributions

Zeye Gu:Responsible for experiements with Logistic Regression, Decision Tree, and Stochastic Gradient Descent. Processed report by using LaTex
Chengyi Song: Working on the implementation of Bernoulli Naïve Bayes, lemmatization, vectorization, the algorithm of choosing words, k-fold and part of the report.
Pen Li: Working on the implementation of Bernoulli Naïve Bayes, fit function, prediction function, k-fold, part of implementaion of logistic regression and part of the report.

REFERENCE

Lecture 2: Machine Learning 1 - Linear Classifiers, SGD — Stanford CS221: AI (Autumn 2019) https://www.youtube.com/watch?v=zrT2qETJilw&t=1358s&ab_channel=stanfordonline

Native Bayes

In [6]:

```
import numpy as np
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.feature_extraction.text import TfidfVectorizer
import pandas as pd
from sklearn.model_selection import KFold
import nltk
nltk.download('wordnet')
nltk.download('wordnet')
from nltk.stem import WordNetLemmatizer
from nltk import word_tokenize
import re

class NBModel:
    def __init__(self):
        self.theta_j_1=None
        self.prob=None
        self.num_feature=0
        self.theta_k = [None] * 8
        self.theta_j_k = [None] * 8
        self.class_prob = [None] * 8

    def preProcess(self,train,test):
        vectorizer = CountVectorizer(binary=True,max_features=5000)
        vectorizer.fit(train)
        trainVec = vectorizer.transform(train).toarray()
        testVec = vectorizer.transform(test).toarray()
        self.num_feature = len(vectorizer.get_feature_names())
        return trainVec, testVec
    def preProcess2(self,test):
        vectorizer = CountVectorizer(binary=True,max_features=5000)
        vectorizer.fit(test)
        testV = vectorizer.transform(test).toarray()
        self.num_feature = len(vectorizer.get_feature_names())
        return testV


    def fit(self,x,y,k):
        theta_1 = (y == k).sum() / float(y.shape[0])
        theta_0 = (y != k).sum() / float(y.shape[0])
        self.prob=np.log((theta_1/theta_0))
        self.theta_j_1 = np.zeros(self.num_feature)
        predict_label = np.zeros(x.shape[0])
        self.class_prob[k-1] = theta_1/theta_0
        self.theta_k[k-1] = theta_1

        p0Num = np.ones(x.shape[1])
        p1Num = np.ones(x.shape[1])

        for j in range(self.num_feature):
            p1Num[j]+=np.sum(x[y == k,j])

        # Laplace smoothing
        self.theta_j_1[:] = p1Num[:]/ float((y == k).sum()+2)
        self.theta_j_k[k-1] = self.theta_j_1[:]

    def predict(self,x):
        llh_k = [None] * 8
        llh=np.zeros(x.shape[0])
        predict=[None]*(x.shape[0])

        for k in range(0,8):
            llh[:] = np.sum(x[:, :] * np.log(self.theta_j_k[k][:]) + (1 - x[:, :]) * np.log(
                (1 - self.theta_j_k[k][:])),axis=1)
            llh[:]+=self.class_prob[k]
            llh_k[k] = llh[:]
            llh=np.zeros(x.shape[0])
        return np.argmax(llh_k, axis=0) + 1

def Accu_eval2(y, predict):
    count = 0
    for i in range(0, len(y)):
        if predict[i] ==y[i]:
            count+=1
    # print("The Accu is: ",accuracy)
    return count/len(y)

data = pd.read_csv('train.csv',dtype=str)
test = pd.read_csv('test.csv',dtype=str)
data = data.values
```

```
test = test.values
np.random.shuffle(data)
body = data[:, 0]
subReddit = data[:, 1]
testbody=test[:,1]
testIndex=test[:,0]

def check(word):
    word= word.lower()
    if bool(re.search(r'\d', word)) or bool(re.match(r'[^\w]', word)):
        return False
    else:
        return True

wnl = WordNetLemmatizer()
lemmatized = [None]*body.shape[0]
test_lemmatized = [None]*testbody.shape[0]
i = 0
for s in testbody:
    test_lemmatized[i] = " ".join([wnl.lemmatize(word) for word in word_tokenize(s) if (check(w
    i+=1
test_lemmatized=np.array(test_lemmatized)
i=0
for s in body:
    lemmatized[i] = " ".join([wnl.lemmatize(word) for word in word_tokenize(s) if (check(word))
    i+=1
def result(re,fo,k):
    for i in range(1,len(fo)):
        if(fo[i]==1):
            re[i]=k
    return re
lemmatized=np.array(lemmatized)
print("lemmatized Done!!!!")

nb=NBModel()

#k fold
kf = KFold(n_splits=6, shuffle=False)
def toint(x):
    i=0
    if x=='rpg':
        i=1
    if x=='anime':
        i=2
    if x=='datascience':
        i=3
    if x=="hardware":
        i=4
    if x=="cars":
        i=5
    if x=="gamernews":
        i=6
    if x=="gamedev":
        i=7
    if x=='computers':
        i=8
    return i
def toString(x):
    if x == 1:
        return 'rpg'
    elif x == 2:
        return 'anime'
    elif x == 3:
        return 'datascience'
    elif x == 4:
        return "hardware"
    elif x == 5:
        return "cars"
    elif x == 6:
        return "gamernews"
    elif x == 7:
        return "gamedev"
    elif x == 8:
        return 'computers'
    else:
        return 'None'
t=data

result = [None]*testIndex.shape[0]
for c in range(0,len(t[:,1])):
    t[c,1]=toint(t[c,1])

temp=t[:,1]
AvgAccuV=0
AvgAccuT=0
print("In different K(6) fold:")
for train_index, test_index in kf.split(lemmatized):
    trainx = lemmatized[train_index]
    trainy =temp[train_index]
    valix = lemmatized[test_index]
    valiy = temp[test_index]
    train, vali = nb.preProcess(trainx, valix)
```

```
    train, vali = nb.preProcess(trainX, valiX)
    for k in range(1,9):
        nb.fit(train,trainy,k)
    predictV = nb.predict(vali)
    predictT = nb.predict(train)
    AvgAccuV +=Accu_eval2(valiy,predictV)
    AvgAccuT += Accu_eval2(trainy,predictT)
    print("(Train set)The Accuracy for  is ", Accu_eval2(trainy,predictT))
    print("(Validation set)The Accuracy for  is ", Accu_eval2(valiy,predictV))


print("The Average of 6 fold")
print("(Train set)The Accuracy for  is ", AvgAccuT/6)
print("(Validation set)The Accuracy for  is ", AvgAccuV/6,"\n")
```

```
[nltk_data] Downloading package wordnet to /Users/guzeye/nltk_data...
[nltk_data]   Package wordnet is already up-to-date!
[nltk_data] Downloading package wordnet to /Users/guzeye/nltk_data...
[nltk_data]   Package wordnet is already up-to-date!
```

```
lemmatized Done!!!!
In different K(6) fold:
(Train set)The Accuracy for  is  0.8345249196974407
(Validation set)The Accuracy for  is  0.7757638529259451
(Train set)The Accuracy for  is  0.8347321521085898
(Validation set)The Accuracy for  is  0.7897462454686691
(Train set)The Accuracy for  is  0.8394115209283051
(Validation set)The Accuracy for  is  0.7917098445595855
(Train set)The Accuracy for  is  0.8319519270617488
(Validation set)The Accuracy for  is  0.794818652849741
(Train set)The Accuracy for  is  0.8360961458765023
(Validation set)The Accuracy for  is  0.778238341968912
(Train set)The Accuracy for  is  0.8403439701616245
(Validation set)The Accuracy for  is  0.7756476683937824
The Average of 6 fold
(Train set)The Accuracy for  is  0.8361767726390351
(Validation set)The Accuracy for  is  0.7843207676944391
```

Test with different classifier with kfold

In [7]:

```
import numpy as np
import pandas as pd
import nltk
#nltk.download("popular")
from nltk.stem import WordNetLemmatizer
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.feature_extraction.text import CountVectorizer
from nltk import word_tokenize
from sklearn.model_selection import KFold
from sklearn.pipeline import Pipeline
import gc
import re

from sklearn.linear_model import LogisticRegression
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.feature_selection import SelectPercentile, f_classif, SelectFromModel
from sklearn.linear_model import SGDClassifier
data = pd.read_csv('train.csv',dtype=str)
test = pd.read_csv('test.csv',dtype=str)
data = data.values
test = test.values
np.random.shuffle(data)
body = data[:, 0]
subReddit = data[:, 1]
testbody=test[:,1]
testIndex=test[:,0]

def check(word):
    word= word.lower()
    if bool(re.search(r'\d', word)) or bool(re.match(r'[^\w]', word)):
        return False
    else:
        return True

wnl = WordNetLemmatizer()
lemmatized = [None]*body.shape[0]
test_lemmatized = [None]*testbody.shape[0]
i = 0
for s in testbody:
    test_lemmatized[i] = " ".join([wnl.lemmatize(word) for word in word_tokenize(s) if (check(w
    i+=1
test_lemmatized=np.array(test_lemmatized)
i=0
for s in body:
    lemmatized[i] = " ".join([wnl.lemmatize(word) for word in word_tokenize(s) if (check(word))
    i+=1
lemmatized=np.array(lemmatized)
```

```
#ngram_range=(1, 2) "I love it" -> 'I' 'love' 'it' && 'I love' 'love it'
#max_df ignore the words that the document frequency higher then x when building the voab list
#sublinear_tf use 1+log(tf) instead of tf


text_classiferLR =Pipeline([('tfidf', TfidfVectorizer(ngram_range=(1, 2),max_df=0.9, sublinear_
                     ('feature_selection', SelectPercentile(f_classif, percentile=40)),
                     ('Logistic', LogisticRegression(max_iter=2000))])

text_classiferDecisionTree = Pipeline([('tfidf', TfidfVectorizer(ngram_range=(1, 2),max_df=0.9,
                     ('feature_selection', SelectPercentile(f_classif, percentile=40)),
                     ('DT', DecisionTreeClassifier())])

text_classiferSGD =Pipeline([('tfidf', TfidfVectorizer(ngram_range=(1, 2),max_df=0.9, sublinear
                     ('feature_selection', SelectPercentile(f_classif, percentile=40)),
                     ('SGD', SGDClassifier())])
# k fold with default parameter
kfCValidator = KFold(n_splits=5, shuffle=True,random_state=None)
def predictFactory(trainx,trainy,validationx,validationy,classifer,totalAccuaryV,totalAccuaryT)
    totalAccuaryV =0
    totalAccuaryT =0
    #fit
    classifer.fit(trainx,trainy)
    #predict
    predictV = classifer.predict(validationx)
    predictT = classifer.predict(trainx)
    #sum the accuray for each fold
    totalAccuaryV += np.mean(predictV == validationy)
    totalAccuaryT += np.mean(predictT == trainy)

    return totalAccuaryV, totalAccuaryT



totalAccuaryLRVali =0
totalAccuaryLRTrain =0
totalAccuaryDecisionTreeVali =0
totalAccuaryDecisionTreeTrain =0
totalAccuarySGDVali =0
totalAccuarySGDTrain =0
for train_yield, test_yield in kfCValidator.split(lemmatized):
    trainx = lemmatized[train_yield]
    trainy = subReddit[train_yield]
    validationx = lemmatized[test_yield]
    validationy = subReddit[test_yield]

    addLRV, addLRT = predictFactory(trainx,trainy,validationx,validationy,text_classiferLR,tota
    addDTV, addDTT = predictFactory(trainx,trainy,validationx,validationy,text_classiferDecisio
    addSGDV,addSGDT= predictFactory(trainx,trainy,validationx,validationy,text_classiferSGD,tot
    totalAccuaryLRVali += addLRV
    totalAccuaryLRTrain += addLRT
    totalAccuaryDecisionTreeVali +=addDTV
    totalAccuaryDecisionTreeTrain +=addDTT
    totalAccuarySGDVali +=addSGDV
    totalAccuarySGDTrain+=addSGDT


LR_validation = totalAccuaryLRVali / 5.0
LR_train = totalAccuaryLRTrain / 5.0

DC_validation = totalAccuaryDecisionTreeVali/5.0
DC_train = totalAccuaryDecisionTreeTrain/5.0

SGD_validation = totalAccuarySGDVali/5.0
SGD_Train = totalAccuarySGDTrain/5.0
print("LogisticRegression Overall vali Accu:",LR_validation )
print("LogisticRegression Overall train Accu", LR_train)

print("DecisionTree Overall vali Accu:",DC_validation )
print("DecisionTree Overall train Accu", DC_train)


print("SGD Overall vali Accu:",SGD_validation )
print("SGD Overall train Accu", SGD_Train)
```

```
LogisticRegression Overall vali Accu: 0.8364712498965743
LogisticRegression Overall train Accu 0.9553401866452556
DecisionTree Overall vali Accu: 0.6581783438920705
DecisionTree Overall train Accu 0.9998273163882511
SGD Overall vali Accu: 0.8868074299519284
SGD Overall train Accu 0.9975393004664326
```

Generate csv file for submission, tested by using test.csv

In [ ]:
```
import numpy as np
import pandas as pd
import nltk
#nltk.download("popular")
```

```
from nltk.stem import WordNetLemmatizer
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.feature_extraction.text import CountVectorizer
from nltk import word_tokenize
from sklearn.model_selection import KFold
from sklearn.pipeline import Pipeline
import gc
import re

from sklearn.linear_model import LogisticRegression
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.feature_selection import SelectPercentile, f_classif, SelectFromModel
from sklearn.linear_model import SGDClassifier
data = pd.read_csv('train.csv',dtype=str)
test = pd.read_csv('test.csv',dtype=str)
data = np.array(data)
test = np.array(test)

np.random.shuffle(data)
# body = data[:, 0]

body = (data[:, 0]).flatten()
subReddit = data[:, 1]
testbody = test[:, 1].flatten()
# testbody=test[:,1]
testIndex=test[:,0]



def check(word):
    word= word.lower()
    if bool(re.search(r'\d', word)) or bool(re.match(r'[^\w]', word)):
        return False
    else:
        return True

wnl = WordNetLemmatizer()
lemmatized = [None]*body.shape[0]
test_lemmatized = [None]*testbody.shape[0]
i = 0
for s in testbody:
    test_lemmatized[i] = " ".join([wnl.lemmatize(word) for word in word_tokenize(s) if (check(w
    i+=1
test_lemmatized=np.array(test_lemmatized)
i=0
for s in body:
    lemmatized[i] = " ".join([wnl.lemmatize(word) for word in word_tokenize(s) if (check(word))
    i+=1
lemmatized=np.array(lemmatized)



text_classifer=Pipeline([('tfidf', TfidfVectorizer(ngram_range=(1, 2),stop_words={'english'},ma
                     ('feature_selection', SelectPercentile(f_classif, percentile=20)),
                     ('clf', SGDClassifier())])


text_classifer.fit(lemmatized,subReddit)
predict_array = text_classifer.predict(test_lemmatized)
predict_array = np.reshape(predict_array, (len(predict_array), 1))
pd.DataFrame(predict_array).to_csv("result.csv")
```