

```

1  # -*- coding: utf-8 -*-
2  """Bernoulli NB Testing.ipynb
3
4  Automatically generated by Colaboratory.
5
6  Original file is located at
7      https://colab.research.google.com/drive/199uJ4b4t-1xxUSn2onGApXLUmpPofQDt
8
9  <center><h1>Mini Project 2 - Bernoulli Naïve Bayes</h1>
10 <h4>This file tests the performance of the Bernoulli Naïve Bayes implemented by
    ourselves.</h4></center>
11
12 <h3>Team Members:</h3>
13 <center>
14 Yi Zhu, 260716006<br>
15 Fei Peng, 260712440<br>
16 Yukai Zhang, 260710915
17 </center>
18 """
19
20 from google.colab import drive
21 drive.mount('/content/drive')
22
23 # make path = './' in-case you are running this locally
24 path = '/content/drive/My Drive/ECSE_551_F_2020/Mini_Project_02/'
25
26 import numpy as np
27 import pandas as pd
28 import matplotlib.pyplot as plt
29
30 from time import time
31 from sklearn.model_selection import train_test_split
32 from sklearn.preprocessing import Normalizer
33 from sklearn.feature_extraction.text import CountVectorizer, TfidfVectorizer
34 from sklearn.feature_extraction import text
35 from sklearn import metrics
36 from sklearn.model_selection import GridSearchCV, cross_val_score, KFold
37 from sklearn.pipeline import make_pipeline
38 from sklearn.preprocessing import LabelEncoder
39
40 !pip install nltk
41 import nltk
42 nltk.download('punkt')
43 nltk.download('wordnet')
44 nltk.download('averaged_perceptron_tagger')
45
46 from nltk.stem import PorterStemmer
47 from nltk import word_tokenize
48 from nltk import word_tokenize
49 from nltk.stem import WordNetLemmatizer
50 from nltk.corpus import wordnet
51
52 """# Import Data"""
53
54 reddit_dataset = pd.read_csv(path+"train.csv")
55 reddit_test = pd.read_csv(path+"test.csv")
56
57 X = reddit_dataset['body']
58 y = reddit_dataset['subreddit']
59
60 """# Define Vectorizer
61 ### (To vectorize the text-based data to numerical features)
62

```

```

63 1. CountVectorizer
64 1) Use "CountVectorizer" to transform text data to feature vectors.
65 2) Normalize your feature vectors
66 """
67
68 def count_vectorizer(X_train, X_test):
69     vectorizer = CountVectorizer(binary=True)
70     vectors_train = vectorizer.fit_transform(X_train)
71     vectors_test = vectorizer.transform(X_test)
72
73     return vectors_train, vectors_test
74
75 """2. CountVectorizer with stop word
76 1) Use "CountVectorizer" with stop word to transform text data to vector.
77 2) Normalize your feature vectors
78 """
79
80 def count_vec_with_sw(X_train, X_test, max_features):
81     stop_words = text.ENGLISH_STOP_WORDS
82     if max_features:
83         vectorizer = CountVectorizer(stop_words=stop_words, binary=True,
max_features=5000)
84     else:
85         vectorizer = CountVectorizer(stop_words=stop_words, binary=True)
86     vectors_train_stop = vectorizer.fit_transform(X_train)
87     vectors_test_stop = vectorizer.transform(X_test)
88
89     return vectors_train_stop, vectors_test_stop
90
91 """3. TF-IDF
92 1) use "TfidfVectorizer" to weight features based on your train set.
93 2) Normalize your feature vectors
94 """
95
96 def tfidf_vectorizer(X_train, X_test):
97     tf_idf_vectorizer = TfidfVectorizer(binary=True)
98     vectors_train_idf = tf_idf_vectorizer.fit_transform(X_train)
99     vectors_test_idf = tf_idf_vectorizer.transform(X_test)
100
101     return vectors_train_idf, vectors_test_idf
102
103 """4. CountVectorizer with stem tokenizer
104 1) Use "StemTokenizer" to transform text data to vector.
105 2) Normalize your feature vectors
106 """
107
108 class StemTokenizer:
109     def __init__(self):
110         self.wnl = PorterStemmer()
111     def __call__(self, doc):
112         return [self.wnl.stem(t) for t in word_tokenize(doc) if t.isalpha()]
113
114
115 def count_vec_stem(X_train, X_test):
116     vectorizer = CountVectorizer(tokenizer=StemTokenizer(), binary=True)
117     vectors_train_stem = vectorizer.fit_transform(X_train)
118     vectors_test_stem = vectorizer.transform(X_test)
119
120     return vectors_train_stem, vectors_test_stem
121
122 """5. CountVectorizer with lemma tokenizer
123 1) Use "LemmaTokenizer" to transform text data to vector.
124 2) Normalize your feature vectors

```

```

125 """
126
127 def get_wordnet_pos(word):
128     """Map POS tag to first character lemmatize() accepts"""
129     tag = nltk.pos_tag([word])[0][1][0].upper()
130     tag_dict = {"J": wordnet.ADJ,
131                 "N": wordnet.NOUN,
132                 "V": wordnet.VERB,
133                 "R": wordnet.ADV}
134     return tag_dict.get(tag, wordnet.NOUN)
135
136
137 class LemmaTokenizer:
138     def __init__(self):
139         self.wnl = WordNetLemmatizer()
140     def __call__(self, doc):
141         return [self.wnl.lemmatize(t,pos =get_wordnet_pos(t)) for t in
word_tokenize(doc) if t.isalpha()]
142
143
144 def count_vec_lemma(X_train, X_test):
145     vectorizer = CountVectorizer(tokenizer=LemmaTokenizer(), binary=True)
146     vectors_train_lemma = vectorizer.fit_transform(X_train)
147     vectors_test_lemma = vectorizer.transform(X_test)
148
149     return vectors_train_lemma, vectors_test_lemma
150
151 """# Bernoulli Naïve Bayes Classifier"""
152
153 # Bernoulli Naïve Bayes
154 class BernoulliNB:
155     '''
156         This is the Bernoulli Naïve Bayes class, containing fit, perdict and
157         accu_eval functions,
158         as well as many other useful functions.
159     '''
160
161     def __init__(self, laplace):
162         self.laplace = laplace # true for performing Laplace smoothing
163         self.le = LabelEncoder() # encoder for classes
164
165     def fit(self, X, y):
166         '''
167             This function takes the training data X and its corresponding
168             labels vector y as input,
169             and execute the model training.
170
171             X - features of traning data
172             y - class labels
173         '''
174         # Laplace smoothing paramerters
175         num = 0
176         den = 0
177         if self.laplace:
178             num += 1
179             den += 2
180
181         # encode the text-based class type to numerical values
182         le = self.le
183         le.fit(y)
184         y_label = le.transform(y)
185         n_k = len(le.classes_) # number of classes
186         n_j = X.shape[1] # number of features

```

```

185     N = len(y) # number of samples
186
187     theta_k = np.zeros(n_k) # probability of class k
188     theta_j_k = np.zeros((n_k, n_j)) # probability of feature j given
class k
189
190     # compute theta values
191     for k in range(n_k):
192         count_k = (y_label==k).sum()
193         theta_k[k] = count_k / N
194         for j in range(n_j):
195             theta_j_k[k][j] = (X[y_label==k, j].sum()+num) / (count_k+den)
196
197     # store the theta values to this instance
198     self.theta_k = theta_k
199     self.theta_j_k = theta_j_k
200     # print("Finished fitting...")
201
202     def predict(self, X):
203         '''
204         This function takes a set of data as input and outputs predicted
labels for the input points.
205         '''
206         le = self.le
207         theta_k = self.theta_k
208         theta_j_k = self.theta_j_k
209
210         # this part works the same as the pseudo-code provided in Lecture 12
211         # but matrix multiplication is much faster than nested loops
212         i_m = np.zeros_like(X) # identity matrix
213         # predict classes
214         y_pred = np.argmax(X.dot(np.log(theta_j_k).T)+(i_m-X).dot(np.log(1-
theta_j_k).T)+theta_k, axis=1)
215
216         # transform back to text-based values
217         y_pred = le.inverse_transform(y_pred)
218         return y_pred
219
220 """### 1. K-fold validation using CountVectorizer"""
221
222 tic = time()
223 accuracies = []
224 clf = BernoulliNB(laplace=True)
225 kf = KFold(n_splits=5, shuffle=True)
226 for train_index, test_index in kf.split(X):
227     vectors_train, vectors_test = count_vectorizer(X[train_index], X[
test_index])
228     clf.fit(vectors_train, y[train_index])
229     a_s = metrics.accuracy_score(y[test_index], clf.predict(vectors_test))
230     print(a_s)
231     accuracies.append(a_s)
232
233 print("\t- Bernoulli Naïve Bayes + CountVectorizer -\nAccuracy: {}%\nTime
Spent: {}s".format(np.mean(accuracies), time()-tic))
234
235 """### 2. K-fold validation using CountVectorizer with stop word, max_features
=5000"""
236
237 # with max_features=5000
238 tic = time()
239 accuracies = []
240 clf = BernoulliNB(laplace=True)
241 kf = KFold(n_splits=5, shuffle=True)

```

```

242 for train_index, test_index in kf.split(X):
243     vectors_train, vectors_test = count_vec_with_sw(X[train_index], X[
test_index], max_features=True)
244     clf.fit(vectors_train, y[train_index])
245     a_s = metrics.accuracy_score(y[test_index], clf.predict(vectors_test))
246     print(a_s)
247     accuracies.append(a_s)
248
249 print("\t- Bernoulli Naïve Bayes + CountVectorizer with stop word, Max
Features = 5000 -\nAccuracy: {}%\tTime Spent: {}s".format(np.mean(accuracies
), time()-tic))
250
251 """### 3. K-fold validation using CountVectorizer with stop word"""
252
253 tic = time()
254 accuracies = []
255 clf = BernoulliNB(laplace=True)
256 kf = KFold(n_splits=5, shuffle=True)
257 for train_index, test_index in kf.split(X):
258     vectors_train, vectors_test = count_vec_with_sw(X[train_index], X[
test_index])
259     clf.fit(vectors_train, y[train_index])
260     a_s = metrics.accuracy_score(y[test_index], clf.predict(vectors_test))
261     print(a_s)
262     accuracies.append(a_s)
263
264 print("\t- Bernoulli Naïve Bayes + CountVectorizer with stop word -\nAccuracy
: {}%\tTime Spent: {}s".format(np.mean(accuracies), time()-tic))
265
266 """### 4. K-fold validation using CountVectorizer with stem tokenizer"""
267
268 tic = time()
269 accuracies = []
270 clf = BernoulliNB(laplace=True)
271 kf = KFold(n_splits=5, shuffle=True)
272 for train_index, test_index in kf.split(X):
273     vectors_train, vectors_test = count_vec_stem(X[train_index], X[test_index
])
274     clf.fit(vectors_train, y[train_index])
275     a_s = metrics.accuracy_score(y[test_index], clf.predict(vectors_test))
276     print(a_s)
277     accuracies.append(a_s)
278
279 print("\t- Bernoulli Naïve Bayes + CountVectorizer with stem tokenizer -\n
Accuracy: {}%\tTime Spent: {}s".format(np.mean(accuracies), time()-tic))
280
281 """### 5. K-fold validation using CountVectorizer with lemma tokenizer"""
282
283 tic = time()
284 accuracies = []
285 clf = BernoulliNB(laplace=True)
286 kf = KFold(n_splits=5, shuffle=True)
287 for train_index, test_index in kf.split(X):
288     vectors_train, vectors_test = count_vec_lemma(X[train_index], X[test_index
])
289     clf.fit(vectors_train, y[train_index])
290     a_s = metrics.accuracy_score(y[test_index], clf.predict(vectors_test))
291     print(a_s)
292     accuracies.append(a_s)
293
294 print("\t- Bernoulli Naïve Bayes + CountVectorizer with lemma tokenizer -\n
Accuracy: {}%\tTime Spent: {}s".format(np.mean(accuracies), time()-tic))

```