

```

1  # -*- coding: utf-8 -*-
2  """Bernoulli Naïve Bayes.ipynb
3
4  Automatically generated by Colaboratory.
5
6  Original file is located at
7      https://colab.research.google.com/drive/19asN10XElCYuHFT9Yao8DAXamARVxFy
8
9  <center><h1>Mini Project 2 - Bernoulli Naïve Bayes</h1>
10 <h4>The hyperparameters and models used in this file are chosen based on the
    findings in the testing file.</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 sklearn.model_selection import train_test_split
31 from sklearn.preprocessing import Normalizer
32 from sklearn.feature_extraction.text import CountVectorizer, TfidfVectorizer
33 from sklearn.feature_extraction import text
34 from sklearn import metrics
35 from sklearn.model_selection import GridSearchCV, cross_val_score, KFold
36 from sklearn.pipeline import make_pipeline
37 from sklearn.preprocessing import LabelEncoder
38
39 !pip install nltk
40 import nltk
41 nltk.download('punkt')
42 nltk.download('wordnet')
43 nltk.download('averaged_perceptron_tagger')
44
45 from nltk.stem import PorterStemmer
46 from nltk import word_tokenize
47 from nltk import word_tokenize
48 from nltk.stem import WordNetLemmatizer
49 from nltk.corpus import wordnet
50
51 """# Import Data"""
52
53 reddit_dataset = pd.read_csv(path+"train.csv")
54 reddit_test = pd.read_csv(path+"test.csv")
55
56 X = reddit_dataset['body']
57 y = reddit_dataset['subreddit']
58
59 """# Define Vectorizer
60 ### (To vectorize the text-based data to numerical features)
61
62 1. CountVectorizer

```

```

63 1) Use "CountVectorizer" to transform text data to feature vectors.
64 2) Normalize your feature vectors
65 """
66
67 def count_vectorizer(X_train, X_test):
68     vectorizer = CountVectorizer(binary=True)
69     vectors_train = vectorizer.fit_transform(X_train)
70     vectors_test = vectorizer.transform(X_test)
71
72     return vectors_train, vectors_test
73
74 """2. CountVectorizer with stop word
75 1) Use "CountVectorizer" with stop word to transform text data to vector.
76 2) Normalize your feature vectors
77 """
78
79 def count_vec_with_sw(X_train, X_test):
80     stop_words = text.ENGLISH_STOP_WORDS
81     vectorizer = CountVectorizer(stop_words=stop_words, binary=True)
82     vectors_train_stop = vectorizer.fit_transform(X_train)
83     vectors_test_stop = vectorizer.transform(X_test)
84
85     return vectors_train_stop, vectors_test_stop
86
87 """3. TF-IDF
88 1) use "TfidfVectorizer" to weight features based on your train set.
89 2) Normalize your feature vectors
90 """
91
92 def tfidf_vectorizer(X_train, X_test):
93     tf_idf_vectorizer = TfidfVectorizer(binary=True)
94     vectors_train_idf = tf_idf_vectorizer.fit_transform(X_train)
95     vectors_test_idf = tf_idf_vectorizer.transform(X_test)
96
97     return vectors_train_idf, vectors_test_idf
98
99 """4. CountVectorizer with stem tokenizer
100 1) Use "StemTokenizer" to transform text data to vector.
101 2) Normalize your feature vectors
102 """
103
104 class StemTokenizer:
105     def __init__(self):
106         self.wnl = PorterStemmer()
107     def __call__(self, doc):
108         return [self.wnl.stem(t) for t in word_tokenize(doc) if t.isalpha()]
109
110
111 def count_vec_stem(X_train, X_test):
112     vectorizer = CountVectorizer(tokenizer=StemTokenizer(), binary=True)
113     vectors_train_stem = vectorizer.fit_transform(X_train)
114     vectors_test_stem = vectorizer.transform(X_test)
115
116     return vectors_train_stem, vectors_test_stem
117
118 """5. CountVectorizer with lemma tokenizer
119 1) Use "LemmaTokenizer" to transform text data to vector.
120 2) Normalize your feature vectors
121 """
122
123 def get_wordnet_pos(word):
124     """Map POS tag to first character lemmatize() accepts"""
125     tag = nltk.pos_tag([word])[0][1][0].upper()

```

```

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

```

```

185
186     # compute theta values
187     for k in range(n_k):
188         count_k = (y_label==k).sum()
189         theta_k[k] = count_k / N
190         for j in range(n_j):
191             theta_j_k[k][j] = (X[y_label==k, j].sum()+num) / (count_k+den)
192
193     # store the theta values to this instance
194     self.theta_k = theta_k
195     self.theta_j_k = theta_j_k
196     print("Finished fitting...")
197
198     def predict(self, X):
199         '''
200         This function takes a set of data as input and outputs predicted
201         labels for the input points.
202         '''
203         le = self.le
204         theta_k = self.theta_k
205         theta_j_k = self.theta_j_k
206
207         # this part works the same as the pseudo-code provided in Lecture 12
208         # but matrix multiplication is much faster than nested loops
209         i_m = np.zeros_like(X) # identity matrix
210         # predict classes
211         y_pred = np.argmax(X.dot(np.log(theta_j_k).T)+(i_m-X).dot(np.log(1-
212 theta_j_k).T)+theta_k, axis=1)
213
214         # transform back to text-based values
215         y_pred = le.inverse_transform(y_pred)
216         return y_pred
217
218 """### 1. K-fold validation using CountVectorizer"""
219
220 accuracies = []
221 clf = BernoulliNB(laplace=True)
222 kf = KFold(n_splits=5, shuffle=True)
223 for train_index, test_index in kf.split(X):
224     vectors_train, vectors_test = count_vectorizer(X[train_index], X[
225 test_index])
226     clf.fit(vectors_train, y[train_index])
227     a_s = metrics.accuracy_score(y[test_index], clf.predict(vectors_test))
228     print(a_s)
229     accuracies.append(a_s)
230
231 print(np.mean(accuracies))
232
233 """### 2. K-fold validation using CountVectorizer with stop word, max_features
234 =5000"""
235
236 # with max_features=5000
237 accuracies = []
238 clf = BernoulliNB(laplace=True)
239 kf = KFold(n_splits=5, shuffle=True)
240 for train_index, test_index in kf.split(X):
241     vectors_train, vectors_test = count_vec_with_sw(X[train_index], X[
242 test_index])
243     clf.fit(vectors_train, y[train_index])
244     a_s = metrics.accuracy_score(y[test_index], clf.predict(vectors_test))
245     print(a_s)
246     accuracies.append(a_s)
247
248 print(np.mean(accuracies))

```

```
243 print(np.mean(accuracies))
244
245 """### 3. K-fold validation using CountVectorizer with stop word"""
246
247 accuracies = []
248 clf = BernoulliNB(laplace=True)
249 kf = KFold(n_splits=5, shuffle=True)
250 for train_index, test_index in kf.split(X):
251     vectors_train, vectors_test = count_vec_with_sw(X[train_index], X[
        test_index])
252     clf.fit(vectors_train, y[train_index])
253     a_s = metrics.accuracy_score(y[test_index], clf.predict(vectors_test))
254     print(a_s)
255     accuracies.append(a_s)
256
257 print(np.mean(accuracies))
258
259 """### 4. K-fold validation using CountVectorizer with stem tokenizer"""
260
261 accuracies = []
262 clf = BernoulliNB(laplace=True)
263 kf = KFold(n_splits=5, shuffle=True)
264 for train_index, test_index in kf.split(X):
265     vectors_train, vectors_test = count_vec_stem(X[train_index], X[test_index
        ])
266     clf.fit(vectors_train, y[train_index])
267     a_s = metrics.accuracy_score(y[test_index], clf.predict(vectors_test))
268     print(a_s)
269     accuracies.append(a_s)
270
271 print(np.mean(accuracies))
272
273 """### 5. K-fold validation using CountVectorizer with lemma tokenizer"""
274
275 accuracies = []
276 clf = BernoulliNB(laplace=True)
277 kf = KFold(n_splits=5, shuffle=True)
278 for train_index, test_index in kf.split(X):
279     vectors_train, vectors_test = count_vec_lemma(X[train_index], X[test_index
        ])
280     clf.fit(vectors_train, y[train_index])
281     a_s = metrics.accuracy_score(y[test_index], clf.predict(vectors_test))
282     print(a_s)
283     accuracies.append(a_s)
284
285 print(np.mean(accuracies))
```