

```
1 # -*- coding: utf-8 -*-
2 """Data Preprocessing.ipynb
3
4 Automatically generated by Colaboratory.
5
6 Original file is located at
7     https://colab.research.google.com/drive/1bRmLzkFmGP7xgU0UI6iv_p6dW1hX6R6P
8
9 <center><h1>Mini Project 2 - Bernoulli Naïve Bayes</h1>
10 <h3>Data Preprocessing</h3>
11 <h4>This file performs some of the operations on Data Preprocessing and
Analysis.</h4></center>
12
13 <h3>Team Members:</h3>
14 <center>
15 Yi Zhu, 260716006<br>
16 Fei Peng, 260712440<br>
17 Yukai Zhang, 260710915
18 </center>
19
20 # Importations
21 """
22
23 from google.colab import drive
24 drive.mount('/content/drive')
25
26 # make path = './' in-case you are running this locally
27 path = '/content/drive/My Drive/ECSE_551_F_2020/Mini_Project_02/'
28
29 import numpy as np
30 import pandas as pd
31 import matplotlib.pyplot as plt
32 import seaborn as sns
33 import random
34 from scipy import stats
35 from google.colab import files
36 from time import time
37
38 from sklearn.model_selection import train_test_split
39 from sklearn.preprocessing import Normalizer
40 from sklearn.feature_extraction.text import CountVectorizer, TfidfVectorizer
41 from sklearn.feature_extraction import text
42 from sklearn import metrics
43 from sklearn.model_selection import GridSearchCV, cross_val_score, KFold
44 from sklearn.pipeline import make_pipeline
45
46 !pip install nltk
47 import nltk
48 nltk.download('punkt')
49 nltk.download('wordnet')
50 nltk.download('averaged_perceptron_tagger')
51
52 from nltk.stem import PorterStemmer
53 from nltk import word_tokenize
54 from nltk import word_tokenize
55 from nltk.stem import WordNetLemmatizer
56 from nltk.corpus import wordnet
57
58 from sklearn.linear_model import LogisticRegression
59 from sklearn.naive_bayes import MultinomialNB
60 from sklearn import svm
61 from sklearn.ensemble import RandomForestClassifier
62 from sklearn.tree import DecisionTreeClassifier
```

```

63 from sklearn.ensemble import AdaBoostClassifier
64 from sklearn.neighbors import KNeighborsClassifier
65 from sklearn.neural_network import MLPClassifier
66
67 """# Data Preprocessing"""
68
69 reddit_dataset = pd.read_csv(path+"train.csv")
70 reddit_test = pd.read_csv(path+"test.csv")
71
72 X = reddit_dataset['body']
73 y = reddit_dataset['subreddit']
74
75 class Data_Processing:
76     def __init__(self, data, name='New Data'):
77         self.data = data
78         self.name = name
79
80     def show_y_dist(self, ydata):
81         plt.figure(figsize=(8,4))
82         plt.subplot(111), sns.countplot(x='subreddit', data=ydata)
83         plt.title('Distribution of Subreddit in {}'.format(self.name))
84         plt.savefig("Distribution of Subreddit in {}.png".format(self.name),
85                     dpi = 1200)
86         files.download("Distribution of Subreddit in {}.png".format(self.name))
87         plt.show()
88
89 data_analysis = Data_Processing(reddit_dataset.values, 'train.csv')
90 data_analysis.show_y_dist(reddit_dataset)
91
92 # calculate the data entropy
93 from sklearn.preprocessing import LabelEncoder
94 le = LabelEncoder() # encoder for classes
95 le.fit(y)
96 n_k = len(le.classes_)
97 N = len(y)
98 theta_k = np.zeros(n_k) # probability of class k
99 # compute theta values
100 for k in range(n_k):
101     count_k = (y_label==k).sum()
102     theta_k[k] = count_k / N
103
104 from scipy.stats import entropy
105 print("Data entropy is", entropy(theta_k, base=2))
106
107 """# Define Vectorizer
108 (To vectorize the text-based data to numerical features)
109
110 1. CountVectorizer
111 1) Use "CountVectorizer" to transform text data to feature vectors.
112 2) Normalize your feature vectors
113 """
114
115 def count_vectorizer(X_train):
116     vectorizer = CountVectorizer()
117     vectors_train = vectorizer.fit_transform(X_train)
118     return vectors_train
119
120 """2. CountVectorizer with stop word
121 1) Use "CountVectorizer" with stop word to transform text data to vector.
122 2) Normalize your feature vectors
123 """

```

```

124
125 def count_vec_with_sw(X_train):
126     stop_words = text.ENGLISH_STOP_WORDS
127     vectorizer = CountVectorizer(stop_words=stop_words)
128     vectors_train_stop = vectorizer.fit_transform(X_train)
129     return vectors_train_stop
130
131 """3. TF-IDF
132 1) use "TfidfVectorizer" to weight features based on your train set.
133 2) Normalize your feature vectors
134 """
135
136 def tfidf_vectorizer(X_train):
137     tf_idf_vectorizer = TfidfVectorizer()
138     vectors_train_idf = tf_idf_vectorizer.fit_transform(X_train)
139     return vectors_train_idf
140
141 """4. CountVectorizer with stem tokenizer
142 1) Use "StemTokenizer" to transform text data to vector.
143 2) Normalize your feature vectors
144 """
145
146 class StemTokenizer:
147     def __init__(self):
148         self.wnl = PorterStemmer()
149     def __call__(self, doc):
150         return [self.wnl.stem(t) for t in word_tokenize(doc) if t.isalpha()]
151
152
153 def count_vec_stem(X_train):
154     vectorizer = CountVectorizer(tokenizer=StemTokenizer())
155     vectors_train_stem = vectorizer.fit_transform(X_train)
156     return vectors_train_stem
157
158 """5. CountVectorizer with lemma tokenizer
159 1) Use "LemmaTokenizer" to transform text data to vector.
160 2) Normalize your feature vectors
161 """
162
163 def get_wordnet_pos(word):
164     """Map POS tag to first character lemmatize() accepts"""
165     tag = nltk.pos_tag([word])[0][1][0].upper()
166     tag_dict = {"J": wordnet.ADJ,
167                 "N": wordnet.NOUN,
168                 "V": wordnet.VERB,
169                 "R": wordnet.ADV}
170     return tag_dict.get(tag, wordnet.NOUN)
171
172
173 class LemmaTokenizer:
174     def __init__(self):
175         self.wnl = WordNetLemmatizer()
176     def __call__(self, doc):
177         return [self.wnl.lemmatize(t, pos=get_wordnet_pos(t)) for t in
178             word_tokenize(doc) if t.isalpha()]
179
180 def count_vec_lemma(X_train):
181     vectorizer = CountVectorizer(tokenizer=LemmaTokenizer())
182     vectors_train_lemma = vectorizer.fit_transform(X_train)
183     return vectors_train_lemma
184
185 """# Measure the time required for each vectorizer to perform vectorization

```

```
186
187 ### 1. CountVectorizer
188 """
189
190 tic = time()
191 X_vec = count_vectorizer(X)
192 print("\t\t- Count Vectorizer - \nfeature number: ", X_vec.shape[1], "\t\tTime
    spent: ", time()-tic, "s.")
193 """
194 """### 2. CountVectorizer with stop word"""
195
196 tic = time()
197 X_vec = count_vec_with_sw(X)
198 print("\t\t- Count Vectorizer with stop word - \nfeature number: ", X_vec.
    shape[1], "\t\tTime spent: ", time()-tic, "s.")
199 """
200 """### 3. TF-IDF"""
201
202 tic = time()
203 X_vec = tfidf_vectorizer(X)
204 print("\t\t- TF-IDF Vectorizer - \nfeature number: ", X_vec.shape[1], "\t\t
    Time spent: ", time()-tic, "s.")
205 """
206 """### 4. CountVectorizer with stem tokenizer"""
207
208 tic = time()
209 X_vec = count_vec_stem(X)
210 print("\t\t- CountVectorizer with stem tokenizer - \nfeature number: ", X_vec.
    shape[1], "\t\tTime spent: ", time()-tic, "s.")
211 """
212 """### 5. CountVectorizer with lemma tokenizer"""
213
214 tic = time()
215 X_vec = count_vec_lemma(X)
216 print("\t\t- CountVectorizer with lemma tokenizer - \nfeature number: ", X_vec.
    shape[1], "\t\tTime spent: ", time()-tic, "s.")
```

```

1 # -*- coding: utf-8 -*-
2 """Additional Classifiers.ipynb
3
4 Automatically generated by Colaboratory.
5
6 Original file is located at
7     https://colab.research.google.com/drive/1jQsRTZihNlPC99pBzRYqvTbowAY-tqi1
8
9 <center><h1>Mini Project 2 - Bernoulli Naïve Bayes</h1>
10 <h4>This file records the accuracies of the combinations of 8 classifiers and 5
vectorizers.</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
38 !pip install nltk
39 import nltk
40 nltk.download('punkt')
41 nltk.download('wordnet')
42 nltk.download('averaged_perceptron_tagger')
43
44 from nltk.stem import PorterStemmer
45 from nltk import word_tokenize
46 from nltk import word_tokenize
47 from nltk.stem import WordNetLemmatizer
48 from nltk.corpus import wordnet
49
50 """Additional classifiers:
51 1. Logistic Regression
52 2. Multinomial Naïve Bayes
53 3. Support Vector Machine
54 4. Random Forest
55 5. Decision Tree
56 6. Ada Boost
57 7. k-Neighbors
58 8. Neural Network
59 """
60
61 from sklearn.linear_model import LogisticRegression
62 from sklearn.naive_bayes import MultinomialNB

```

```

63 from sklearn import svm
64 from sklearn.ensemble import RandomForestClassifier
65 from sklearn.tree import DecisionTreeClassifier
66 from sklearn.ensemble import AdaBoostClassifier
67 from sklearn.neighbors import KNeighborsClassifier
68 from sklearn.neural_network import MLPClassifier
69
70 reddit_dataset = pd.read_csv(path+"train.csv")
71 reddit_test = pd.read_csv(path+"test.csv")
72
73 X = reddit_dataset['body']
74 y = reddit_dataset['subreddit']
75
76 """# Define Vectorizer
77 ### (To vectorize the text-based data to numerical features)
78
79 1. CountVectorizer
80 1) Use "CountVectorizer" to transform text data to feature vectors.
81 2) Normalize your feature vectors
82 """
83
84 def count_vectorizer(X_train, X_test):
85     vectorizer = CountVectorizer()
86     vectors_train = vectorizer.fit_transform(X_train)
87     vectors_test = vectorizer.transform(X_test)
88
89     normalizer_train = Normalizer().fit(X=vectors_train)
90     vectors_train = normalizer_train.transform(vectors_train)
91     vectors_test = normalizer_train.transform(vectors_test)
92
93     return vectors_train, vectors_test
94
95 """2. CountVectorizer with stop word
96 1) Use "CountVectorizer" with stop word to transform text data to vector.
97 2) Normalize your feature vectors
98 """
99
100 def count_vec_with_sw(X_train, X_test):
101     stop_words = text.ENGLISH_STOP_WORDS
102     vectorizer = CountVectorizer(stop_words=stop_words)
103     vectors_train_stop = vectorizer.fit_transform(X_train)
104     vectors_test_stop = vectorizer.transform(X_test)
105
106     normalizer_train = Normalizer().fit(X=vectors_train_stop)
107     vectors_train_stop= normalizer_train.transform(vectors_train_stop)
108     vectors_test_stop = normalizer_train.transform(vectors_test_stop)
109
110     return vectors_train_stop, vectors_test_stop
111
112 """3. TF-IDF
113 1) use "TfidfVectorizer" to weight features based on your train set.
114 2) Normalize your feature vectors
115 """
116
117 def tfidf_vectorizer(X_train, X_test):
118     tf_idf_vectorizer = TfidfVectorizer()
119     vectors_train_idf = tf_idf_vectorizer.fit_transform(X_train)
120     vectors_test_idf = tf_idf_vectorizer.transform(X_test)
121
122     normalizer_train = Normalizer().fit(X=vectors_train_idf)
123     vectors_train_idf= normalizer_train.transform(vectors_train_idf)
124     vectors_test_idf = normalizer_train.transform(vectors_test_idf)
125

```

```

126     return vectors_train_idf, vectors_test_idf
127
128 """4. CountVectorizer with stem tokenizer
129 1) Use "StemTokenizer" to transform text data to vector.
130 2) Normalize your feature vectors
131 """
132
133 class StemTokenizer:
134     def __init__(self):
135         self.wnl = PorterStemmer()
136     def __call__(self, doc):
137         return [self.wnl.stem(t) for t in word_tokenize(doc) if t.isalpha()]
138
139
140 def count_vec_stem(X_train, X_test):
141     vectorizer = CountVectorizer(tokenizer=StemTokenizer())
142     vectors_train_stem = vectorizer.fit_transform(X_train)
143     vectors_test_stem = vectorizer.transform(X_test)
144
145     normalizer_train = Normalizer().fit(X=vectors_train_stem)
146     vectors_train_stem= normalizer_train.transform(vectors_train_stem)
147     vectors_test_stem = normalizer_train.transform(vectors_test_stem)
148
149     return vectors_train_stem, vectors_test_stem
150
151 """5. CountVectorizer with lemma tokenizer
152 1) Use "LemmaTokenizer" to transform text data to vector.
153 2) Normalize your feature vectors
154 """
155
156 def get_wordnet_pos(word):
157     """Map POS tag to first character lemmatize() accepts"""
158     tag = nltk.pos_tag([word])[0][1][0].upper()
159     tag_dict = {"J": wordnet.ADJ,
160                 "N": wordnet.NOUN,
161                 "V": wordnet.VERB,
162                 "R": wordnet.ADV}
163     return tag_dict.get(tag, wordnet.NOUN)
164
165
166 class LemmaTokenizer:
167     def __init__(self):
168         self.wnl = WordNetLemmatizer()
169     def __call__(self, doc):
170         return [self.wnl.lemmatize(t,pos =get_wordnet_pos(t)) for t in
171             word_tokenize(doc) if t.isalpha()]
172
173 def count_vec_lemma(X_train, X_test):
174     vectorizer = CountVectorizer(tokenizer=LemmaTokenizer())
175     vectors_train_lemma = vectorizer.fit_transform(X_train)
176     vectors_test_lemma = vectorizer.transform(X_test)
177
178     normalizer_train = Normalizer().fit(X=vectors_train_lemma)
179     vectors_train_lemma= normalizer_train.transform(vectors_train_lemma)
180     vectors_test_lemma = normalizer_train.transform(vectors_test_lemma)
181
182     return vectors_train_lemma, vectors_test_lemma
183
184 """# Measure Accuracies of different classifiers using K-fold Validation
185
186 ## 1. Logistic Regression
187

```

```

188 ### 1. CountVectorizer
189 """
190
191 accuracies = []
192 clf = LogisticRegression(C=1.0, max_iter=1000, random_state=0)
193 kf = KFold(n_splits=5, shuffle=True)
194 for train_index, test_index in kf.split(X):
195     vectors_train, vectors_test = count_vectorizer(X[train_index], X[
196         test_index])
197     clf.fit(vectors_train, y[train_index])
198     accuracies.append(metrics.accuracy_score(y[test_index], clf.predict(
199         vectors_test)))
200
201 print(np.mean(accuracies))
202
203 """## 2. CountVectorizer with stop word"""
204
205 accuracies = []
206 clf = LogisticRegression(C=1.0, max_iter=1000, random_state=0)
207 kf = KFold(n_splits=5, shuffle=True)
208 for train_index, test_index in kf.split(X):
209     vectors_train, vectors_test = count_vec_with_sw(X[train_index], X[
210         test_index])
211     clf.fit(vectors_train, y[train_index])
212     accuracies.append(metrics.accuracy_score(y[test_index], clf.predict(
213         vectors_test)))
214
215 print(np.mean(accuracies))
216
217 """## 3. TF-IDF"""
218
219 accuracies = []
220 clf = LogisticRegression(C=40.0, max_iter=1000, random_state=0)
221 kf = KFold(n_splits=5, shuffle=True)
222 for train_index, test_index in kf.split(X):
223     vectors_train, vectors_test = tfidf_vectorizer(X[train_index], X[
224         test_index])
225     clf.fit(vectors_train, y[train_index])
226     accuracies.append(metrics.accuracy_score(y[test_index], clf.predict(
227         vectors_test)))
228
229 print(np.mean(accuracies))
230
231 """accuracy vs. hyperparameter C"""
232
233 c_vecs = np.logspace(0, 2, 5)
234 acc = []
235 for c_vec in c_vecs:
236     accuracies = []
237     clf = LogisticRegression(C=c_vec, max_iter=1000, random_state=0)
238     kf = KFold(n_splits=5, shuffle=True)
239     for train_index, test_index in kf.split(X):
240         vectors_train, vectors_test = tfidf_vectorizer(X[train_index], X[
241             test_index])
242         clf.fit(vectors_train, y[train_index])
243         accuracies.append(metrics.accuracy_score(y[test_index], clf.predict(
244             vectors_test)))
245     acc.append(np.mean(accuracies))
246
247 plt.xlabel('C')
248 plt.ylabel('Accuracy')
249 plt.plot(c_vecs, acc)

```

```

243
244 """## 4. CountVectorizer with stem tokenizer"""
245
246 accuracies = []
247 clf = LogisticRegression(C=1.0, max_iter=1000, random_state=0)
248 kf = KFold(n_splits=5, shuffle=True)
249 for train_index, test_index in kf.split(X):
250     vectors_train, vectors_test = count_vec_stem(X[train_index], X[test_index])
251     clf.fit(vectors_train, y[train_index])
252     accuracies.append(metrics.accuracy_score(y[test_index], clf.predict(
253         vectors_test)))
254 print(np.mean(accuracies))
255
256 """## 5. CountVectorizer with lemma tokenizer"""
257
258 accuracies = []
259 clf = LogisticRegression(C=1.0, max_iter=1000, random_state=0)
260 kf = KFold(n_splits=5, shuffle=True)
261 for train_index, test_index in kf.split(X):
262     vectors_train, vectors_test = count_vec_lemma(X[train_index], X[test_index])
263     clf.fit(vectors_train, y[train_index])
264     accuracies.append(metrics.accuracy_score(y[test_index], clf.predict(
265         vectors_test)))
266 print(np.mean(accuracies))
267
268 """## 2. Multinomial Naïve Bayes
269
270 ### 1. CountVectorizer
271 """
272
273 accuracies = []
274 clf = MultinomialNB()
275 kf = KFold(n_splits=5, shuffle=True)
276 for train_index, test_index in kf.split(X):
277     vectors_train, vectors_test = count_vectorizer(X[train_index], X[
278         test_index])
279     clf.fit(vectors_train, y[train_index])
280     accuracies.append(metrics.accuracy_score(y[test_index], clf.predict(
281         vectors_test)))
282 print(np.mean(accuracies))
283
284 """## 2. CountVectorizer with stop word"""
285
286 accuracies = []
287 clf = MultinomialNB()
288 kf = KFold(n_splits=5, shuffle=True)
289 for train_index, test_index in kf.split(X):
290     vectors_train, vectors_test = count_vec_with_sw(X[train_index], X[
291         test_index])
292     clf.fit(vectors_train, y[train_index])
293     accuracies.append(metrics.accuracy_score(y[test_index], clf.predict(
294         vectors_test)))
295
296 """## 3. TF-IDF"""
297 accuracies = []

```

```

298 clf = MultinomialNB()
299 kf = KFold(n_splits=5, shuffle=True)
300 for train_index, test_index in kf.split(X):
301     vectors_train, vectors_test = tfidf_vectorizer(X[train_index], X[
302         test_index])
303     clf.fit(vectors_train, y[train_index])
304     accuracies.append(metrics.accuracy_score(y[test_index], clf.predict(
305         vectors_test)))
306
307 """## 4. CountVectorizer with stem tokenizer"""
308
309 accuracies = []
310 clf = MultinomialNB()
311 kf = KFold(n_splits=5, shuffle=True)
312 for train_index, test_index in kf.split(X):
313     vectors_train, vectors_test = count_vec_stem(X[train_index], X[test_index
314         ])
315     clf.fit(vectors_train, y[train_index])
316     accuracies.append(metrics.accuracy_score(y[test_index], clf.predict(
317         vectors_test)))
318
319 """## 5. CountVectorizer with lemma tokenizer"""
320
321 accuracies = []
322 clf = MultinomialNB()
323 kf = KFold(n_splits=5, shuffle=True)
324 for train_index, test_index in kf.split(X):
325     vectors_train, vectors_test = count_vec_lemma(X[train_index], X[test_index
326         ])
327     clf.fit(vectors_train, y[train_index])
328     accuracies.append(metrics.accuracy_score(y[test_index], clf.predict(
329         vectors_test)))
330
331 """## 3. Support Vector Machine
332
333 ## 1. CountVectorizer
334 """
335
336 accuracies = []
337 clf = svm.SVC(kernel='linear', gamma='auto', C=1)
338 kf = KFold(n_splits=5, shuffle=True)
339 for train_index, test_index in kf.split(X):
340     vectors_train, vectors_test = count_vectorizer(X[train_index], X[
341         test_index])
342     clf.fit(vectors_train, y[train_index])
343     accuracies.append(metrics.accuracy_score(y[test_index], clf.predict(
344         vectors_test)))
345
346 """## 2. CountVectorizer with stop word"""
347
348 accuracies = []
349 clf = svm.SVC(kernel='linear', gamma='auto', C=1)
350 kf = KFold(n_splits=5, shuffle=True)
351 for train_index, test_index in kf.split(X):
352     vectors_train, vectors_test = count_vec_with_sw(X[train_index], X[

```

```

352 test_index])
353     clf.fit(vectors_train, y[train_index])
354     accuracies.append(metrics.accuracy_score(y[test_index], clf.predict(
355         vectors_test)))
356 print(np.mean(accuracies))
357
358 """## 3. TF-IDF
359
360 Linear
361 """
362
363 accuracies = []
364 clf = svm.SVC(kernel='linear', gamma='auto', C=1)
365 kf = KFold(n_splits=5, shuffle=True)
366 for train_index, test_index in kf.split(X):
367     vectors_train, vectors_test = tfidf_vectorizer(X[train_index], X[
368         test_index])
369     clf.fit(vectors_train, y[train_index])
370     accuracies.append(metrics.accuracy_score(y[test_index], clf.predict(
371         vectors_test)))
372
373 print(np.mean(accuracies))
374
375 """Non-Linear"""
376
377 accuracies = []
378 clf = svm.SVC(gamma='auto', C=1)
379 kf = KFold(n_splits=5, shuffle=True)
380 for train_index, test_index in kf.split(X):
381     vectors_train, vectors_test = tfidf_vectorizer(X[train_index], X[
382         test_index])
383     clf.fit(vectors_train, y[train_index])
384     accuracies.append(metrics.accuracy_score(y[test_index], clf.predict(
385         vectors_test)))
386
387 print(np.mean(accuracies))
388
389 """## 4. CountVectorizer with stem tokenizer"""
390
391 accuracies = []
392 clf = svm.SVC(kernel='linear', gamma='auto', C=1)
393 kf = KFold(n_splits=5, shuffle=True)
394 for train_index, test_index in kf.split(X):
395     vectors_train, vectors_test = count_vec_stem(X[train_index], X[test_index]
396     ])
397     clf.fit(vectors_train, y[train_index])
398     accuracies.append(metrics.accuracy_score(y[test_index], clf.predict(
399         vectors_test)))
400
401 print(np.mean(accuracies))
402
403 """## 5. CountVectorizer with lemma tokenizer"""
404
405 accuracies = []
406 clf = svm.SVC(kernel='linear', gamma='auto', C=1)
407 kf = KFold(n_splits=5, shuffle=True)
408 for train_index, test_index in kf.split(X):
409     vectors_train, vectors_test = count_vec_lemma(X[train_index], X[test_index]
410     ])
411     clf.fit(vectors_train, y[train_index])
412     accuracies.append(metrics.accuracy_score(y[test_index], clf.predict(
413         vectors_test)))

```

```

406
407 print(np.mean(accuracies))
408
409 """## 4. Random Forest
410
411 ### 1. CountVectorizer
412 """
413
414 accuracies = []
415 clf = RandomForestClassifier(max_depth=2, random_state=0)
416 kf = KFold(n_splits=5, shuffle=True)
417 for train_index, test_index in kf.split(X):
418     vectors_train, vectors_test = count_vectorizer(X[train_index], X[
419         test_index])
420     clf.fit(vectors_train, y[train_index])
421     accuracies.append(metrics.accuracy_score(y[test_index], clf.predict(
422         vectors_test)))
423
424 print(np.mean(accuracies))
425
426 """## 2. CountVectorizer with stop word"""
427
428 accuracies = []
429 clf = RandomForestClassifier(max_depth=2, random_state=0)
430 kf = KFold(n_splits=5, shuffle=True)
431 for train_index, test_index in kf.split(X):
432     vectors_train, vectors_test = count_vec_with_sw(X[train_index], X[
433         test_index])
434     clf.fit(vectors_train, y[train_index])
435     accuracies.append(metrics.accuracy_score(y[test_index], clf.predict(
436         vectors_test)))
437
438 print(np.mean(accuracies))
439
440 """## 3. TF-IDF"""
441
442 accuracies = []
443 clf = RandomForestClassifier(max_depth=2, random_state=0)
444 kf = KFold(n_splits=5, shuffle=True)
445 for train_index, test_index in kf.split(X):
446     vectors_train, vectors_test = tfidf_vectorizer(X[train_index], X[
447         test_index])
448     clf.fit(vectors_train, y[train_index])
449     accuracies.append(metrics.accuracy_score(y[test_index], clf.predict(
450         vectors_test)))
451
452 print(np.mean(accuracies))
453
454 """## 4. CountVectorizer with stem tokenizer"""
455
456 accuracies = []
457 clf = RandomForestClassifier(max_depth=2, random_state=0)
458 kf = KFold(n_splits=5, shuffle=True)
459 for train_index, test_index in kf.split(X):
460     vectors_train, vectors_test = count_vec_stem(X[train_index], X[test_index
461         ])
462     clf.fit(vectors_train, y[train_index])
463     accuracies.append(metrics.accuracy_score(y[test_index], clf.predict(
464         vectors_test)))
465
466 print(np.mean(accuracies))
467
468 """## 5. CountVectorizer with lemma tokenizer"""

```

```

461
462 accuracies = []
463 clf = RandomForestClassifier(max_depth=2, random_state=0)
464 kf = KFold(n_splits=5, shuffle=True)
465 for train_index, test_index in kf.split(X):
466     vectors_train, vectors_test = count_vec_lemma(X[train_index], X[test_index])
467     clf.fit(vectors_train, y[train_index])
468     accuracies.append(metrics.accuracy_score(y[test_index], clf.predict(
469         vectors_test)))
470 print(np.mean(accuracies))
471
472 """## 5. Decision Tree
473
474 ### 1. CountVectorizer
475 """
476
477 accuracies = []
478 clf = DecisionTreeClassifier(random_state=0)
479 kf = KFold(n_splits=5, shuffle=True)
480 for train_index, test_index in kf.split(X):
481     vectors_train, vectors_test = count_vectorizer(X[train_index], X[
482         test_index])
483     clf.fit(vectors_train, y[train_index])
484     accuracies.append(metrics.accuracy_score(y[test_index], clf.predict(
485         vectors_test)))
486 print(np.mean(accuracies))
487 """
488 """## 2. CountVectorizer with stop word"""
489
490 accuracies = []
491 clf = DecisionTreeClassifier(random_state=0)
492 kf = KFold(n_splits=5, shuffle=True)
493 for train_index, test_index in kf.split(X):
494     vectors_train, vectors_test = count_vec_with_sw(X[train_index], X[
495         test_index])
496     clf.fit(vectors_train, y[train_index])
497     accuracies.append(metrics.accuracy_score(y[test_index], clf.predict(
498         vectors_test)))
499 """
500 """## 3. TF-IDF"""
501
502 accuracies = []
503 clf = DecisionTreeClassifier(random_state=0)
504 kf = KFold(n_splits=5, shuffle=True)
505 for train_index, test_index in kf.split(X):
506     vectors_train, vectors_test = tfidf_vectorizer(X[train_index], X[
507         test_index])
508     clf.fit(vectors_train, y[train_index])
509     accuracies.append(metrics.accuracy_score(y[test_index], clf.predict(
510         vectors_test)))
511 """
512 """## 4. CountVectorizer with stem tokenizer"""
513
514 accuracies = []
515 clf = DecisionTreeClassifier(random_state=0)
516 kf = KFold(n_splits=5, shuffle=True)

```

```

516 for train_index, test_index in kf.split(X):
517     vectors_train, vectors_test = count_vec_stem(X[train_index], X[test_index]
518     ])
518     clf.fit(vectors_train, y[train_index])
519     accuracies.append(metrics.accuracy_score(y[test_index], clf.predict(
519         vectors_test)))
520
521 print(np.mean(accuracies))
522
523 """## 5. CountVectorizer with lemma tokenizer"""
524
525 accuracies = []
526 clf = DecisionTreeClassifier(random_state=0)
527 kf = KFold(n_splits=5, shuffle=True)
528 for train_index, test_index in kf.split(X):
529     vectors_train, vectors_test = count_vec_lemma(X[train_index], X[test_index]
529     ])
530     clf.fit(vectors_train, y[train_index])
531     accuracies.append(metrics.accuracy_score(y[test_index], clf.predict(
531         vectors_test)))
532
533 print(np.mean(accuracies))
534
535 """## 6. Ada Boost
536
537 ### 1. CountVectorizer
538 """
539
540 accuracies = []
541 clf = AdaBoostClassifier(n_estimators=100, learning_rate=0.5, random_state=0)
542 kf = KFold(n_splits=5, shuffle=True)
543 for train_index, test_index in kf.split(X):
544     vectors_train, vectors_test = count_vectorizer(X[train_index], X[
544         test_index])
545     clf.fit(vectors_train, y[train_index])
546     accuracies.append(metrics.accuracy_score(y[test_index], clf.predict(
546         vectors_test)))
547
548 print(np.mean(accuracies))
549
550 """## 2. CountVectorizer with stop word"""
551
552 accuracies = []
553 clf = AdaBoostClassifier(n_estimators=100, learning_rate=0.5, random_state=0)
554 kf = KFold(n_splits=5, shuffle=True)
555 for train_index, test_index in kf.split(X):
556     vectors_train, vectors_test = count_vec_with_sw(X[train_index], X[
556         test_index])
557     clf.fit(vectors_train, y[train_index])
558     accuracies.append(metrics.accuracy_score(y[test_index], clf.predict(
558         vectors_test)))
559
560 print(np.mean(accuracies))
561
562 """## 3. TF-IDF"""
563
564 # Commented out IPython magic to ensure Python compatibility.
565 # find the best hyperparameters
566 clf = AdaBoostClassifier(random_state=0)
567 model = make_pipeline(clf)
568
569 param_grid = {'adaboostclassifier__n_estimators': [50, 100],
569     'adaboostclassifier__learning_rate': [0.1, 0.5, 1]}

```

File - E:\Study\ECSE551\Mini_Project_2\additional_classifiers.py

```

571 grid = GridSearchCV(model, param_grid)
572
573 X_train, X_test, y_train, y_test = train_test_split(X, y, train_size=0.8,
574 test_size=0.2, random_state=0)
575 # %time grid.fit(vectors_train, y_train)
576 print(grid.best_params_)
577
578 accuracies = []
579 clf = AdaBoostClassifier(n_estimators=100, learning_rate=0.5, random_state=0)
580 kf = KFold(n_splits=5, shuffle=True)
581 for train_index, test_index in kf.split(X):
582     vectors_train, vectors_test = tfidf_vectorizer(X[train_index], X[
583         test_index])
584     clf.fit(vectors_train, y[train_index])
585     accuracies.append(metrics.accuracy_score(y[test_index], clf.predict(
586         vectors_test)))
587
588 print(np.mean(accuracies))
589
590 """## 4. CountVectorizer with stem tokenizer"""
591
592 accuracies = []
593 clf = AdaBoostClassifier(n_estimators=100, learning_rate=0.5, random_state=0)
594 kf = KFold(n_splits=5, shuffle=True)
595 for train_index, test_index in kf.split(X):
596     vectors_train, vectors_test = count_vec_stem(X[train_index], X[test_index]
597     ])
598     clf.fit(vectors_train, y[train_index])
599     accuracies.append(metrics.accuracy_score(y[test_index], clf.predict(
600         vectors_test)))
601
602 print(np.mean(accuracies))
603
604 """## 5. CountVectorizer with lemma tokenizer"""
605
606 accuracies = []
607 clf = AdaBoostClassifier(n_estimators=100, learning_rate=0.5, random_state=0)
608 kf = KFold(n_splits=5, shuffle=True)
609 for train_index, test_index in kf.split(X):
610     vectors_train, vectors_test = count_vec_lemma(X[train_index], X[test_index]
611     ])
612     clf.fit(vectors_train, y[train_index])
613     accuracies.append(metrics.accuracy_score(y[test_index], clf.predict(
614         vectors_test)))
615
616 print(np.mean(accuracies))
617
618 """## 7. k-Neighbors
619
620 #### 1. CountVectorizer
621 """
622
623 accuracies = []
624 neigh = KNeighborsClassifier(n_neighbors=3)
625 kf = KFold(n_splits=5, shuffle=True)
626 for train_index, test_index in kf.split(X):
627     vectors_train, vectors_test = count_vectorizer(X[train_index], X[
628         test_index])
629     neigh.fit(vectors_train, y[train_index])
630     accuracies.append(metrics.accuracy_score(y[test_index], neigh.predict(
631         vectors_test)))
632
633 print(np.mean(accuracies))

```

```

625 print(np.mean(accuracies))
626
627 """### 2. CountVectorizer with stop word"""
628
629 accuracies = []
630 neigh = KNeighborsClassifier(n_neighbors=3)
631 kf = KFold(n_splits=5, shuffle=True)
632 for train_index, test_index in kf.split(X):
633     vectors_train, vectors_test = count_vec_with_sw(X[train_index], X[
634         test_index])
635     neigh.fit(vectors_train, y[train_index])
636     accuracies.append(metrics.accuracy_score(y[test_index], neigh.predict(
637         vectors_test)))
638
639 print(np.mean(accuracies))
640
641 """### 3. TF-IDF"""
642
643 accuracies = []
644 neigh = KNeighborsClassifier(n_neighbors=3)
645 kf = KFold(n_splits=5, shuffle=True)
646 for train_index, test_index in kf.split(X):
647     vectors_train, vectors_test = tfidf_vectorizer(X[train_index], X[
648         test_index])
649     neigh.fit(vectors_train, y[train_index])
650     accuracies.append(metrics.accuracy_score(y[test_index], neigh.predict(
651         vectors_test)))
652
653 """### 4. CountVectorizer with stem tokenizer"""
654
655 accuracies = []
656 neigh = KNeighborsClassifier(n_neighbors=3)
657 kf = KFold(n_splits=5, shuffle=True)
658 for train_index, test_index in kf.split(X):
659     vectors_train, vectors_test = count_vec_stem(X[train_index], X[test_index]
660     ])
661     neigh.fit(vectors_train, y[train_index])
662     accuracies.append(metrics.accuracy_score(y[test_index], neigh.predict(
663         vectors_test)))
664
665 """### 5. CountVectorizer with lemma tokenizer"""
666
667 accuracies = []
668 neigh = KNeighborsClassifier(n_neighbors=3)
669 kf = KFold(n_splits=5, shuffle=True)
670 for train_index, test_index in kf.split(X):
671     vectors_train, vectors_test = count_vec_lemma(X[train_index], X[test_index]
672     ])
673     neigh.fit(vectors_train, y[train_index])
674     accuracies.append(metrics.accuracy_score(y[test_index], neigh.predict(
675         vectors_test)))
676
677 """## 8. Neural Network
678 """
679

```

```

680 accuracies = []
681 kf = KFold(n_splits=5, shuffle=True)
682 for train_index, test_index in kf.split(X):
683     vectors_train, vectors_test = count_vectorizer(X[train_index], X[
684         test_index])
684     clf = MLPClassifier(random_state=0, max_iter=300).fit(vectors_train, y[
685         train_index])
685     accuracies.append(metrics.accuracy_score(y[test_index], clf.predict(
686         vectors_test)))
686
687 print(np.mean(accuracies))
688
689 """## 2. CountVectorizer with stop word"""
690
691 accuracies = []
692 kf = KFold(n_splits=5, shuffle=True)
693 for train_index, test_index in kf.split(X):
694     vectors_train, vectors_test = count_vec_with_sw(X[train_index], X[
695         test_index])
695     clf = MLPClassifier(random_state=0, max_iter=300).fit(vectors_train, y[
696         train_index])
696     accuracies.append(metrics.accuracy_score(y[test_index], clf.predict(
697         vectors_test)))
698
699 print(np.mean(accuracies))
700
701 """## 3. TF-IDF"""
702
703 accuracies = []
704 kf = KFold(n_splits=5, shuffle=True)
705 for train_index, test_index in kf.split(X):
706     vectors_train, vectors_test = tfidf_vectorizer(X[train_index], X[
707         test_index])
708     clf = MLPClassifier(random_state=0, max_iter=300).fit(vectors_train, y[
709         train_index])
710     accuracies.append(metrics.accuracy_score(y[test_index], clf.predict(
711         vectors_test)))
712
713 print(np.mean(accuracies))
714
715 """## 4. CountVectorizer with stem tokenizer"""
716
717 accuracies = []
718 kf = KFold(n_splits=5, shuffle=True)
719 for train_index, test_index in kf.split(X):
720     vectors_train, vectors_test = count_vec_stem(X[train_index], X[test_index]
721         )
722     clf = MLPClassifier(random_state=0, max_iter=300).fit(vectors_train, y[
723         train_index])
724     accuracies.append(metrics.accuracy_score(y[test_index], clf.predict(
725         vectors_test)))
726
727 print(np.mean(accuracies))
728
729 """## 5. CountVectorizer with lemma tokenizer"""
730
731 accuracies = []
732 kf = KFold(n_splits=5, shuffle=True)
733 for train_index, test_index in kf.split(X):
734     vectors_train, vectors_test = count_vec_lemma(X[train_index], X[test_index]
735         )
736     clf = MLPClassifier(random_state=0, max_iter=300).fit(vectors_train, y[
737         train_index])

```

```
729     accuracies.append(metrics.accuracy_score(y[test_index], clf.predict(  
    vectors_test)))  
730  
731 print(np.mean(accuracies))
```

```

1 # -*- coding: utf-8 -*-
2 """Additional Classifiers Testing.ipynb
3
4 Automatically generated by Colaboratory.
5
6 Original file is located at
7     https://colab.research.google.com/drive/16E3avMbgZz1YnYvo1uuc4HLzgHZiaDCQ
8
9 <center><h1>Mini Project 2 - Bernoulli Naïve Bayes</h1></center>
10 This file consists two parts:
11
12 In the first part, it measures the accuracies and time spent of the Logistic
Regression based on the output of 5 different classifiers. The effect of data
normalization is also measured.
13
14 In the second part, it measures the accuracies and time spent of the remaining
classifiers given that a TF-IDF vectorizer is used. In the end of the second
part, we also did some tests on the Bernoulli Naïve Bayes implemented by
sklearn upon all different vectorizers.
15
16 <h3>Team Members:</h3>
17 <center>
18 Yi Zhu, 260716006<br>
19 Fei Peng, 260712440<br>
20 Yukai Zhang, 260710915
21 </center>
22 """
23
24 from google.colab import drive
25 drive.mount('/content/drive')
26
27 # make path = './' in-case you are running this locally
28 path = '/content/drive/My Drive/ECSE_551_F_2020/Mini_Project_02/'
29
30 import numpy as np
31 import pandas as pd
32 import matplotlib.pyplot as plt
33
34 from time import time
35 from sklearn.model_selection import train_test_split
36 from sklearn.preprocessing import Normalizer
37 from sklearn.feature_extraction.text import CountVectorizer, TfidfVectorizer
38 from sklearn.feature_extraction import text
39 from sklearn import metrics
40 from sklearn.model_selection import GridSearchCV, cross_val_score, KFold
41 from sklearn.pipeline import make_pipeline
42
43 !pip install nltk
44 import nltk
45 nltk.download('punkt')
46 nltk.download('wordnet')
47 nltk.download('averaged_perceptron_tagger')
48
49 from nltk.stem import PorterStemmer
50 from nltk import word_tokenize
51 from nltk import word_tokenize
52 from nltk.stem import WordNetLemmatizer
53 from nltk.corpus import wordnet
54
55 """Additional classifiers:
56 1. Logistic Regression
57 2. Multinomial Naïve Bayes
58 3. Support Vector Machine

```

```

59 4. Random Forest
60 5. Decision Tree
61 6. Ada Boost
62 7. k-Neighbors
63 8. Neural Network
64 """
65
66 from sklearn.linear_model import LogisticRegression
67 from sklearn.naive_bayes import MultinomialNB
68 from sklearn import svm
69 from sklearn.ensemble import RandomForestClassifier
70 from sklearn.tree import DecisionTreeClassifier
71 from sklearn.ensemble import AdaBoostClassifier
72 from sklearn.neighbors import KNeighborsClassifier
73 from sklearn.neural_network import MLPClassifier
74
75 reddit_dataset = pd.read_csv(path+"train.csv")
76 reddit_test = pd.read_csv(path+"test.csv")
77
78 X = reddit_dataset['body']
79 y = reddit_dataset['subreddit']
80
81 """# Define Vectorizer
82 ### (To vectorize the text-based data to numerical features)
83
84 1. CountVectorizer
85 1) Use "CountVectorizer" to transform text data to feature vectors.
86 2) Normalize your feature vectors
87 """
88
89 def count_vectorizer(X_train, X_test, normalize=True):
90     vectorizer = CountVectorizer()
91     vectors_train = vectorizer.fit_transform(X_train)
92     vectors_test = vectorizer.transform(X_test)
93
94     if normalize:
95         normalizer_train = Normalizer().fit(X=vectors_train)
96         vectors_train = normalizer_train.transform(vectors_train)
97         vectors_test = normalizer_train.transform(vectors_test)
98
99     return vectors_train, vectors_test
100
101 """2. CountVectorizer with stop word
102 1) Use "CountVectorizer" with stop word to transform text data to vector.
103 2) Normalize your feature vectors
104 """
105
106 def count_vec_with_sw(X_train, X_test, normalize=True, features_5k=False):
107     stop_words = text.ENGLISH_STOP_WORDS
108     if features_5k:
109         vectorizer = CountVectorizer(stop_words=stop_words, max_features=5000)
110     else:
111         vectorizer = CountVectorizer(stop_words=stop_words)
112     vectors_train_stop = vectorizer.fit_transform(X_train)
113     vectors_test_stop = vectorizer.transform(X_test)
114
115     if normalize:
116         normalizer_train = Normalizer().fit(X=vectors_train_stop)
117         vectors_train_stop= normalizer_train.transform(vectors_train_stop)
118         vectors_test_stop = normalizer_train.transform(vectors_test_stop)
119
120     return vectors_train_stop, vectors_test_stop
121

```

```

122 """3. TF-IDF
123 1) use "TfidfVectorizer" to weight features based on your train set.
124 2) Normalize your feature vectors
125 """
126
127 def tfidf_vectorizer(X_train, X_test, normalize=True):
128     tf_idf_vectorizer = TfidfVectorizer()
129     vectors_train_idf = tf_idf_vectorizer.fit_transform(X_train)
130     vectors_test_idf = tf_idf_vectorizer.transform(X_test)
131
132     if normalize:
133         normalizer_train = Normalizer().fit(X=vectors_train_idf)
134         vectors_train_idf= normalizer_train.transform(vectors_train_idf)
135         vectors_test_idf = normalizer_train.transform(vectors_test_idf)
136
137     return vectors_train_idf, vectors_test_idf
138
139 """4. CountVectorizer with stem tokenizer
140 1) Use "StemTokenizer" to transform text data to vector.
141 2) Normalize your feature vectors
142 """
143
144 class StemTokenizer:
145     def __init__(self):
146         self.wnl =PorterStemmer()
147     def __call__(self, doc):
148         return [self.wnl.stem(t) for t in word_tokenize(doc) if t.isalpha()]
149
150
151 def count_vec_stem(X_train, X_test, normalize=True):
152     vectorizer = CountVectorizer(tokenizer=StemTokenizer())
153     vectors_train_stem = vectorizer.fit_transform(X_train)
154     vectors_test_stem = vectorizer.transform(X_test)
155
156     if normalize:
157         normalizer_train = Normalizer().fit(X=vectors_train_stem)
158         vectors_train_stem= normalizer_train.transform(vectors_train_stem)
159         vectors_test_stem = normalizer_train.transform(vectors_test_stem)
160
161     return vectors_train_stem, vectors_test_stem
162
163 """5. CountVectorizer with lemma tokenizer
164 1) Use "LemmaTokenizer" to transform text data to vector.
165 2) Normalize your feature vectors
166 """
167
168 def get_wordnet_pos(word):
169     """Map POS tag to first character lemmatize() accepts"""
170     tag = nltk.pos_tag([word])[0][1][0].upper()
171     tag_dict = {"J": wordnet.ADJ,
172                 "N": wordnet.NOUN,
173                 "V": wordnet.VERB,
174                 "R": wordnet.ADV}
175     return tag_dict.get(tag, wordnet.NOUN)
176
177
178 class LemmaTokenizer:
179     def __init__(self):
180         self.wnl = WordNetLemmatizer()
181     def __call__(self, doc):
182         return [self.wnl.lemmatize(t,pos =get_wordnet_pos(t)) for t in
183             word_tokenize(doc) if t.isalpha()]
184

```

```

184
185 def count_vec_lemma(X_train, X_test, normalize=True):
186     vectorizer = CountVectorizer(tokenizer=LemmaTokenizer())
187     vectors_train_lemma = vectorizer.fit_transform(X_train)
188     vectors_test_lemma = vectorizer.transform(X_test)
189
190     if normalize:
191         normalizer_train = Normalizer().fit(X=vectors_train_lemma)
192         vectors_train_lemma= normalizer_train.transform(vectors_train_lemma)
193         vectors_test_lemma = normalizer_train.transform(vectors_test_lemma)
194
195     return vectors_train_lemma, vectors_test_lemma
196
197 """# Measure Accuracies and Time Spent of different classifiers using K-fold
198 Validation
199 ## 1. Logistic Regression
200
201 ### 1. CountVectorizer
202 """
203
204 tic = time()
205 accuracies = []
206 clf = LogisticRegression(C=1.0, max_iter=1000, random_state=0)
207 kf = KFold(n_splits=5, shuffle=True)
208 for train_index, test_index in kf.split(X):
209     vectors_train, vectors_test = count_vectorizer(X[train_index], X[
210         test_index])
211     clf.fit(vectors_train, y[train_index])
212     accuracies.append(metrics.accuracy_score(y[test_index], clf.predict(
213         vectors_test)))
214
215 print("\t- Logistic Regression + CountVectorizer + Normalize -\nAccuracy: {}%\n\tTime Spent: {}s".format(np.mean(accuracies), time()-tic))
216
217 tic = time()
218 accuracies = []
219 clf = LogisticRegression(C=1.0, max_iter=1000, random_state=0)
220 kf = KFold(n_splits=5, shuffle=True)
221 for train_index, test_index in kf.split(X):
222     vectors_train, vectors_test = count_vectorizer(X[train_index], X[
223         test_index], False)
224     clf.fit(vectors_train, y[train_index])
225     accuracies.append(metrics.accuracy_score(y[test_index], clf.predict(
226         vectors_test)))
227
228 print("\t- Logistic Regression + CountVectorizer + Unnormalize -\nAccuracy
229 : {}%\tTime Spent: {}s".format(np.mean(accuracies), time()-tic))
230
231 """## 2. CountVectorizer with stop word"""
232
233 tic = time()
234 accuracies = []
235 clf = LogisticRegression(C=1.0, max_iter=1000, random_state=0)
236 kf = KFold(n_splits=5, shuffle=True)
237 for train_index, test_index in kf.split(X):
238     vectors_train, vectors_test = count_vec_with_sw(X[train_index], X[
239         test_index])
240     clf.fit(vectors_train, y[train_index])
241     accuracies.append(metrics.accuracy_score(y[test_index], clf.predict(
242         vectors_test)))
243
244 print("\t- Logistic Regression + CountVectorizer with stop word + Normalize -\n

```

```

237 nAccuracy: {}%\tTime Spent: {}s".format(np.mean(accuracies), time()-tic))
238
239 tic = time()
240 accuracies = []
241 clf = LogisticRegression(C=1.0, max_iter=1000, random_state=0)
242 kf = KFold(n_splits=5, shuffle=True)
243 for train_index, test_index in kf.split(X):
244     vectors_train, vectors_test = count_vec_with_sw(X[train_index], X[
245         test_index], False)
246     clf.fit(vectors_train, y[train_index])
247     accuracies.append(metrics.accuracy_score(y[test_index], clf.predict(
248         vectors_test)))
249
249 print("\t- Logistic Regression + CountVectorizer with stop word + Unnormalize
- \nAccuracy: {}%\tTime Spent: {}s".format(np.mean(accuracies), time()-tic))
250 """
251 """## 3. TF-IDF"""
252
253 tic = time()
254 accuracies = []
255 clf = LogisticRegression(C=40.0, max_iter=1000, random_state=0)
256 kf = KFold(n_splits=5, shuffle=True)
257 for train_index, test_index in kf.split(X):
258     vectors_train, vectors_test = tfidf_vectorizer(X[train_index], X[
259         test_index])
260     clf.fit(vectors_train, y[train_index])
261     accuracies.append(metrics.accuracy_score(y[test_index], clf.predict(
262         vectors_test)))
263
263 print("\t- Logistic Regression + TF-IDF Vectorizer + Normalize -\nAccuracy
: {}%\tTime Spent: {}s".format(np.mean(accuracies), time()-tic))
264
264 tic = time()
265 accuracies = []
266 clf = LogisticRegression(C=40.0, max_iter=1000, random_state=0)
267 kf = KFold(n_splits=5, shuffle=True)
268 for train_index, test_index in kf.split(X):
269     vectors_train, vectors_test = tfidf_vectorizer(X[train_index], X[
270         test_index], False)
271     clf.fit(vectors_train, y[train_index])
272     accuracies.append(metrics.accuracy_score(y[test_index], clf.predict(
273         vectors_test)))
274
274 print("\t- Logistic Regression + TF-IDF Vectorizer + Unnormalize -\nAccuracy
: {}%\tTime Spent: {}s".format(np.mean(accuracies), time()-tic))
275 """
275 """## 4. CountVectorizer with stem tokenizer"""
276
276 tic = time()
277 accuracies = []
278 clf = LogisticRegression(C=1.0, max_iter=1000, random_state=0)
279 kf = KFold(n_splits=5, shuffle=True)
280 for train_index, test_index in kf.split(X):
281     vectors_train, vectors_test = count_vec_stem(X[train_index], X[test_index
282         ])
283     clf.fit(vectors_train, y[train_index])
284     accuracies.append(metrics.accuracy_score(y[test_index], clf.predict(
285         vectors_test)))
285
285 print("\t- Logistic Regression + CountVectorizer with stem tokenizer +
Normalize -\nAccuracy: {}%\tTime Spent: {}s".format(np.mean(accuracies), time
        ()-tic))
286

```

```

287 tic = time()
288 accuracies = []
289 clf = LogisticRegression(C=1.0, max_iter=1000, random_state=0)
290 kf = KFold(n_splits=5, shuffle=True)
291 for train_index, test_index in kf.split(X):
292     vectors_train, vectors_test = count_vec_stem(X[train_index], X[test_index],
293         ], False)
294     clf.fit(vectors_train, y[train_index])
295     accuracies.append(metrics.accuracy_score(y[test_index], clf.predict(
296         vectors_test)))
297
298 print("\t- Logistic Regression + CountVectorizer with stem tokenizer +
299 Unnormalize -\nAccuracy: {}%\tTime Spent: {}s".format(np.mean(accuracies),
300 time()-tic))
301
302 """## 5. CountVectorizer with lemma tokenizer"""
303
304 tic = time()
305 accuracies = []
306 clf = LogisticRegression(C=1.0, max_iter=1000, random_state=0)
307 kf = KFold(n_splits=5, shuffle=True)
308 for train_index, test_index in kf.split(X):
309     vectors_train, vectors_test = count_vec_lemma(X[train_index], X[test_index]
310         ])
311     clf.fit(vectors_train, y[train_index])
312     accuracies.append(metrics.accuracy_score(y[test_index], clf.predict(
313         vectors_test)))
314
315 print("\t- Logistic Regression + CountVectorizer with lemma tokenizer +
316 Normalize -\nAccuracy: {}%\tTime Spent: {}s".format(np.mean(accuracies), time()
317 ()-tic))
318
319 tic = time()
320 accuracies = []
321 clf = LogisticRegression(C=1.0, max_iter=1000, random_state=0)
322 kf = KFold(n_splits=5, shuffle=True)
323 for train_index, test_index in kf.split(X):
324     vectors_train, vectors_test = count_vec_lemma(X[train_index], X[test_index],
325         ], False)
326     clf.fit(vectors_train, y[train_index])
327     accuracies.append(metrics.accuracy_score(y[test_index], clf.predict(
328         vectors_test)))
329
330 print("\t- Logistic Regression + CountVectorizer with lemma tokenizer +
331 Unnormalize -\nAccuracy: {}%\tTime Spent: {}s".format(np.mean(accuracies),
332 time()-tic))
333
334 """## 2. Multinomial Naïve Bayes"""
335
336 tic = time()
337 accuracies = []
338 clf = MultinomialNB()
339 kf = KFold(n_splits=5, shuffle=True)
340 for train_index, test_index in kf.split(X):
341     vectors_train, vectors_test = tfidf_vectorizer(X[train_index], X[
342         test_index])
343     clf.fit(vectors_train, y[train_index])
344     accuracies.append(metrics.accuracy_score(y[test_index], clf.predict(
345         vectors_test)))
346
347 print("\t- Multinomial Naïve Bayes + TF-IDF Vectorizer + Normalize -\nAccuracy
348 : {}%\tTime Spent: {}s".format(np.mean(accuracies), time()-tic))
349

```

```

335 """## 3. Support Vector Machine
336
337 Linear
338 """
339
340 tic = time()
341 accuracies = []
342 clf = svm.SVC(kernel='linear', gamma='auto', C=1)
343 kf = KFold(n_splits=5, shuffle=True)
344 for train_index, test_index in kf.split(X):
345     vectors_train, vectors_test = tfidf_vectorizer(X[train_index], X[
346         test_index])
347     clf.fit(vectors_train, y[train_index])
348     accuracies.append(metrics.accuracy_score(y[test_index], clf.predict(
349         vectors_test)))
350
351 print("\t- Linear Support Vector Machine + TF-IDF Vectorizer + Normalize -\n"
352       "Accuracy: {}%\tTime Spent: {}s".format(np.mean(accuracies), time()-tic))
353
354 """## 4. Random Forest"""
355
356 tic = time()
357 accuracies = []
358 clf = RandomForestClassifier(max_depth=2, random_state=0)
359 kf = KFold(n_splits=5, shuffle=True)
360 for train_index, test_index in kf.split(X):
361     vectors_train, vectors_test = tfidf_vectorizer(X[train_index], X[
362         test_index])
363     clf.fit(vectors_train, y[train_index])
364     accuracies.append(metrics.accuracy_score(y[test_index], clf.predict(
365         vectors_test)))
366
367 print("\t- Random Forest + TF-IDF Vectorizer + Normalize -\n"
368       "Accuracy: {}%\tTime Spent: {}s".format(np.mean(accuracies), time()-tic))
369
370 """## 5. Decision Tree"""
371
372 tic = time()
373 accuracies = []
374 clf = DecisionTreeClassifier(random_state=0)
375 kf = KFold(n_splits=5, shuffle=True)
376 for train_index, test_index in kf.split(X):
377     vectors_train, vectors_test = tfidf_vectorizer(X[train_index], X[
378         test_index])
379     clf.fit(vectors_train, y[train_index])
380     accuracies.append(metrics.accuracy_score(y[test_index], clf.predict(
381         vectors_test)))
382
383 print("\t- Decision Tree + TF-IDF Vectorizer + Normalize -\n"
384       "Accuracy: {}%\tTime Spent: {}s".format(np.mean(accuracies), time()-tic))
385
386 """## 6. Ada Boost"""
387
388 tic = time()
389 accuracies = []
390 clf = AdaBoostClassifier(n_estimators=100, learning_rate=0.5, random_state=0)
391 kf = KFold(n_splits=5, shuffle=True)
392 for train_index, test_index in kf.split(X):
393     vectors_train, vectors_test = tfidf_vectorizer(X[train_index], X[
394         test_index])
395     clf.fit(vectors_train, y[train_index])
396     accuracies.append(metrics.accuracy_score(y[test_index], clf.predict(
397         vectors_test)))

```

```

387
388 print("\t- Ada Boost + TF-IDF Vectorizer + Normalize -\nAccuracy: {}%\tTime
      Spent: {}s".format(np.mean(accuracies), time()-tic))
389
390 """## 7. k-Neighbors"""
391
392 tic = time()
393 accuracies = []
394 neigh = KNeighborsClassifier(n_neighbors=3)
395 kf = KFold(n_splits=5, shuffle=True)
396 for train_index, test_index in kf.split(X):
397     vectors_train, vectors_test = tfidf_vectorizer(X[train_index], X[
398         test_index])
399     neigh.fit(vectors_train, y[train_index])
400     accuracies.append(metrics.accuracy_score(y[test_index], neigh.predict(
401         vectors_test)))
402
403 print("\t- k-Neighbors + TF-IDF Vectorizer + Normalize -\nAccuracy: {}%\tTime
      Spent: {}s".format(np.mean(accuracies), time()-tic))
404
405 """## 8. Neural Network"""
406
407 tic = time()
408 accuracies = []
409 kf = KFold(n_splits=5, shuffle=True)
410 for train_index, test_index in kf.split(X):
411     vectors_train, vectors_test = tfidf_vectorizer(X[train_index], X[
412         test_index])
413     clf = MLPClassifier(random_state=0, max_iter=300).fit(vectors_train, y[
414         train_index])
415     accuracies.append(metrics.accuracy_score(y[test_index], clf.predict(
416         vectors_test)))
417
418 print("\t- Neural Network + TF-IDF Vectorizer + Normalize -\nAccuracy: {}%\t
      Time Spent: {}s".format(np.mean(accuracies), time()-tic))
419
420 """## 9. Bernoulli Naïve Bayes (Sklearn Version)
421 <h2>This part is only used to test and compare the performance of the
      Bernoulli Naïve Bayes implemented by ourselves.</h2>
422 """
423
424 from sklearn.naive_bayes import BernoulliNB
425
426 """## 1. CountVectorizer"""
427
428 tic = time()
429 accuracies = []
430 kf = KFold(n_splits=5, shuffle=True)
431 for train_index, test_index in kf.split(X):
432     vectors_train, vectors_test = count_vectorizer(X[train_index], X[
433         test_index])
434     clf = BernoulliNB().fit(vectors_train, y[train_index])
435     accuracies.append(metrics.accuracy_score(y[test_index], clf.predict(
436         vectors_test)))
437
438 print("\t- BernoulliNB + CountVectorizer + Normalize -\nAccuracy: {}%\tTime
      Spent: {}s".format(np.mean(accuracies), time()-tic))
439
440 """## 2. CountVectorizer with stop word"""
441
442 tic = time()
443 accuracies = []
444 kf = KFold(n_splits=5, shuffle=True)

```

```

438 for train_index, test_index in kf.split(X):
439     vectors_train, vectors_test = count_vec_with_sw(X[train_index], X[
        test_index])
440     clf = BernoulliNB().fit(vectors_train, y[train_index])
441     accuracies.append(metrics.accuracy_score(y[test_index], clf.predict(
        vectors_test)))
442
443 print("\t- BernoulliNB + CountVectorizer with stop word + Normalize -\n"
      "Accuracy: {} \tTime Spent: {}s".format(np.mean(accuracies), time()-tic))
444
445 """## 3. CountVectorizer with stop word, max_features = 5000"""
446
447 tic = time()
448 accuracies = []
449 kf = KFold(n_splits=5, shuffle=True)
450 for train_index, test_index in kf.split(X):
451     vectors_train, vectors_test = count_vec_with_sw(X[train_index], X[
        test_index], features_5k=True)
452     clf = BernoulliNB().fit(vectors_train, y[train_index])
453     accuracies.append(metrics.accuracy_score(y[test_index], clf.predict(
        vectors_test)))
454
455 print("\t- BernoulliNB + CountVectorizer with stop word, max_features = 5000
      + Normalize -\nAccuracy: {} \tTime Spent: {}s".format(np.mean(accuracies),
      time()-tic))
456
457 """## 4. TF-IDF"""
458
459 tic = time()
460 accuracies = []
461 kf = KFold(n_splits=5, shuffle=True)
462 for train_index, test_index in kf.split(X):
463     vectors_train, vectors_test = tfidf_vectorizer(X[train_index], X[
        test_index])
464     clf = BernoulliNB().fit(vectors_train, y[train_index])
465     accuracies.append(metrics.accuracy_score(y[test_index], clf.predict(
        vectors_test)))
466
467 print("\t- BernoulliNB + TF-IDF Vectorizer + Normalize -\nAccuracy: {} \tTime
      Spent: {}s".format(np.mean(accuracies), time()-tic))
468
469 """## 5. CountVectorizer with stem tokenizer"""
470
471 tic = time()
472 accuracies = []
473 kf = KFold(n_splits=5, shuffle=True)
474 for train_index, test_index in kf.split(X):
475     vectors_train, vectors_test = count_vec_stem(X[train_index], X[test_index]
        )
476     clf = BernoulliNB().fit(vectors_train, y[train_index])
477     accuracies.append(metrics.accuracy_score(y[test_index], clf.predict(
        vectors_test)))
478
479 print("\t- BernoulliNB + CountVectorizer with stem tokenizer + Normalize -\n"
      "Accuracy: {} \tTime Spent: {}s".format(np.mean(accuracies), time()-tic))
480
481 """## 6. CountVectorizer with lemma tokenizer"""
482
483 tic = time()
484 accuracies = []
485 kf = KFold(n_splits=5, shuffle=True)
486 for train_index, test_index in kf.split(X):
487     vectors_train, vectors_test = count_vec_lemma(X[train_index], X[test_index]
        )

```

```
487 ])
488     clf = BernoulliNB().fit(vectors_train, y[train_index])
489     accuracies.append(metrics.accuracy_score(y[test_index], clf.predict(
490         vectors_test)))
491 print("\t- BernoulliNB + CountVectorizer with lemma tokenizer + Normalize -\n"
492       "Accuracy: {} \tTime Spent: {}s".format(np.mean(accuracies), time()-tic))
```

```

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/19asN10XEleCYuHFT9Yao8DAXamARVxFy
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
138             word_tokenize(doc) if t.isalpha()]
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, predict 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
186         class k

```

```

185         # compute theta values
186         for k in range(n_K):
187             count_k = (y_label==k).sum()
188             theta_k[k] = count_k / N
189             for j in range(n_j):
190                 theta_j_k[k][j] = (X[y_label==k, j].sum()+num) / (count_k+den)
191
192         # store the theta values to this instance
193         self.theta_k = theta_k
194         self.theta_j_k = theta_j_k
195         print("Finished fitting...")
196
197     def predict(self, X):
198         """
199             This function takes a set of data as input and outputs predicted
200             labels for the input points.
201         """
202         le = self.le
203         theta_k = self.theta_k
204         theta_j_k = self.theta_j_k
205
206         # this part works the same as the pseudo-code provided in Lecture 12
207         # but matrix multiplication is much faster than nested loops
208         i_m = np.zeros_like(X) # identity matrix
209         # predict classes
210         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)
211
212         # transform back to text-based values
213         y_pred = le.inverse_transform(y_pred)
214         return y_pred
215
216 """## 1. K-fold validation using CountVectorizer"""
217
218 accuracies = []
219 clf = BernoulliNB(laplace=True)
220 kf = KFold(n_splits=5, shuffle=True)
221 for train_index, test_index in kf.split(X):
222     vectors_train, vectors_test = count_vectorizer(X[train_index], X[
test_index])
223     clf.fit(vectors_train, y[train_index])
224     a_s = metrics.accuracy_score(y[test_index], clf.predict(vectors_test))
225     print(a_s)
226     accuracies.append(a_s)
227
228 print(np.mean(accuracies))
229
230 """## 2. K-fold validation using CountVectorizer with stop word, max_features
=5000"""
231
232 # with max_features=5000
233 accuracies = []
234 clf = BernoulliNB(laplace=True)
235 kf = KFold(n_splits=5, shuffle=True)
236 for train_index, test_index in kf.split(X):
237     vectors_train, vectors_test = count_vec_with_sw(X[train_index], X[
test_index])
238     clf.fit(vectors_train, y[train_index])
239     a_s = metrics.accuracy_score(y[test_index], clf.predict(vectors_test))
240     print(a_s)
241     accuracies.append(a_s)
242

```

```
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[
252         test_index])
253     clf.fit(vectors_train, y[train_index])
254     a_s = metrics.accuracy_score(y[test_index], clf.predict(vectors_test))
255     print(a_s)
256     accuracies.append(a_s)
257
258 print(np.mean(accuracies))
259
260 """## 4. K-fold validation using CountVectorizer with stem tokenizer"""
261
262 accuracies = []
263 clf = BernoulliNB(laplace=True)
264 kf = KFold(n_splits=5, shuffle=True)
265 for train_index, test_index in kf.split(X):
266     vectors_train, vectors_test = count_vec_stem(X[train_index], X[test_index]
267         )
268     clf.fit(vectors_train, y[train_index])
269     a_s = metrics.accuracy_score(y[test_index], clf.predict(vectors_test))
270     print(a_s)
271     accuracies.append(a_s)
272
273 print(np.mean(accuracies))
274
275 """## 5. K-fold validation using CountVectorizer with lemma tokenizer"""
276
277 accuracies = []
278 clf = BernoulliNB(laplace=True)
279 kf = KFold(n_splits=5, shuffle=True)
280 for train_index, test_index in kf.split(X):
281     vectors_train, vectors_test = count_vec_lemma(X[train_index], X[test_index]
282         )
283     clf.fit(vectors_train, y[train_index])
284     a_s = metrics.accuracy_score(y[test_index], clf.predict(vectors_test))
285     print(a_s)
286     accuracies.append(a_s)
287
288 print(np.mean(accuracies))
```

```

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-1xxUSn2on6ApxLUPMPofQDt
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
11 ourselves.</h4></center>
12
13 <h3>Team Members:</h3>
14 <center>
15 Yi Zhu, 260716006<br>
16 Fei Peng, 260712440<br>
17 Yukai Zhang, 260710915
18 </center>
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,
84         max_features=5000)
85     else:
86         vectorizer = CountVectorizer(stop_words=stop_words, binary=True)
87     vectors_train_stop = vectorizer.fit_transform(X_train)
88     vectors_test_stop = vectorizer.transform(X_test)
89
90     return vectors_train_stop, vectors_test_stop
91
92 """3. TF-IDF
93 1) use "TfidfVectorizer" to weight features based on your train set.
94 2) Normalize your feature vectors
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].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
142             word_tokenize(doc) if t.isalpha()]
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, predict and
157         accu_eval functions,
158         as well as many other useful functions.
159     """
160     def __init__(self, laplace):
161         self.laplace = laplace # true for performing Laplace smoothing
162         self.le = LabelEncoder() # encoder for classes
163
164     def fit(self, X, y):
165         """
166             This function takes the training data X and its corresponding
167             labels vector y as input,
168             and execute the model training.
169
170             X - features of traning data
171             y - class labels
172         """
173         # Laplace smoothing paramerters
174         num = 0
175         den = 0
176         if self.laplace:
177             num += 1
178             den += 2
179
180         # encode the text-based class type to numerical values
181         le = self.le
182         le.fit(y)
183         y_label = le.transform(y)
184         n_k = len(le.classes_) # number of classes
185         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
189     class k
190
191     # compute theta values
192     for k in range(n_k):
193         count_k = (y_label==k).sum()
194         theta_k[k] = count_k / N
195         for j in range(n_j):
196             theta_j_k[k][j] = (X[y_label==k, j].sum()+num) / (count_k+den)
197
198     # store the theta values to this instance
199     self.theta_k = theta_k
200     self.theta_j_k = theta_j_k
201     # print("Finished fitting...")
202
202 def predict(self, X):
203     """
204         This function takes a set of data as input and outputs predicted
205         labels for the input points.
206     """
207     le = self.le
208     theta_k = self.theta_k
209     theta_j_k = self.theta_j_k
210
211     # this part works the same as the pseudo-code provided in Lecture 12
212     # but matrix multiplication is much faster than nested loops
213     i_m = np.zeros_like(X) # identity matrix
214     # predict classes
215     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)
216
217     # transform back to text-based values
218     y_pred = le.inverse_transform(y_pred)
219     return y_pred
220
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: {}%\tTime
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[
244         test_index], max_features=True)
245     clf.fit(vectors_train, y[train_index])
246     a_s = metrics.accuracy_score(y[test_index], clf.predict(vectors_test))
247     print(a_s)
248     accuracies.append(a_s)
249
250 print("\t- Bernoulli Naïve Bayes + CountVectorizer with stop word, Max
251 Features = 5000 -\nAccuracy: {}%\tTime Spent: {}s".format(np.mean(accuracies),
252 ), time()-tic))
253
254 """## 3. K-fold validation using CountVectorizer with stop word"""
255
256 tic = time()
257 accuracies = []
258 clf = BernoulliNB(laplace=True)
259 kf = KFold(n_splits=5, shuffle=True)
260 for train_index, test_index in kf.split(X):
261     vectors_train, vectors_test = count_vec_with_sw(X[train_index], X[
262         test_index])
263     clf.fit(vectors_train, y[train_index])
264     a_s = metrics.accuracy_score(y[test_index], clf.predict(vectors_test))
265     print(a_s)
266     accuracies.append(a_s)
267
268 print("\t- Bernoulli Naïve Bayes + CountVectorizer with stop word -\nAccuracy
269 : {}%\tTime Spent: {}s".format(np.mean(accuracies), time()-tic))
270
271 """## 4. K-fold validation using CountVectorizer with stem tokenizer"""
272
273 tic = time()
274 accuracies = []
275 clf = BernoulliNB(laplace=True)
276 kf = KFold(n_splits=5, shuffle=True)
277 for train_index, test_index in kf.split(X):
278     vectors_train, vectors_test = count_vec_stem(X[train_index], X[test_index]
279     ])
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("\t- Bernoulli Naïve Bayes + CountVectorizer with stem tokenizer -\n
286 Accuracy: {}%\tTime Spent: {}s".format(np.mean(accuracies), time()-tic))
287
288 """## 5. K-fold validation using CountVectorizer with lemma tokenizer"""
289
290 tic = time()
291 accuracies = []
292 clf = BernoulliNB(laplace=True)
293 kf = KFold(n_splits=5, shuffle=True)
294 for train_index, test_index in kf.split(X):
295     vectors_train, vectors_test = count_vec_lemma(X[train_index], X[test_index]
296     ])
297     clf.fit(vectors_train, y[train_index])
298     a_s = metrics.accuracy_score(y[test_index], clf.predict(vectors_test))
299     print(a_s)
300     accuracies.append(a_s)
301
302 print("\t- Bernoulli Naïve Bayes + CountVectorizer with lemma tokenizer -\n
303 Accuracy: {}%\tTime Spent: {}s".format(np.mean(accuracies), time()-tic))

```

```

1 # -*- coding: utf-8 -*-
2 """Stacking Classifier.ipynb
3
4 Automatically generated by Colaboratory.
5
6 Original file is located at
7     https://colab.research.google.com/drive/1OKCClcD9Qf2Z59egC0iTBDwc2xkamhsM
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
38 !pip install nltk
39 import nltk
40 nltk.download('punkt')
41 nltk.download('wordnet')
42 nltk.download('averaged_perceptron_tagger')
43
44 from nltk.stem import PorterStemmer
45 from nltk import word_tokenize
46 from nltk import word_tokenize
47 from nltk.stem import WordNetLemmatizer
48 from nltk.corpus import wordnet
49
50 """Additional classifiers:
51 1. Logistic Regression
52 2. Multinomial Naïve Bayes
53 3. Support Vector Machine
54 4. Random Forest
55 5. Decision Tree
56 6. Ada Boost
57 7. k-Neighbors
58 8. Neural Network
59 """
60
61 from sklearn.linear_model import LogisticRegression
62 from sklearn.naive_bayes import MultinomialNB

```

```

63 from sklearn import svm
64 from sklearn.ensemble import RandomForestClassifier
65 from sklearn.tree import DecisionTreeClassifier
66 from sklearn.ensemble import AdaBoostClassifier
67 from sklearn.neighbors import KNeighborsClassifier
68 from sklearn.neural_network import MLPClassifier
69
70 reddit_dataset = pd.read_csv(path+"train.csv")
71 reddit_test = pd.read_csv(path+"test.csv")
72
73 X = reddit_dataset['body']
74 y = reddit_dataset['subreddit']
75
76 """1. CountVectorizer
77 1) Use "CountVectorizer" to transform text data to feature vectors.
78 2) Normalize your feature vectors
79 """
80
81 def count_vectorizer(X_train, X_test):
82     vectorizer = CountVectorizer()
83     vectors_train = vectorizer.fit_transform(X_train)
84     vectors_test = vectorizer.transform(X_test)
85
86     normalizer_train = Normalizer().fit(X=vectors_train)
87     vectors_train = normalizer_train.transform(vectors_train)
88     vectors_test = normalizer_train.transform(vectors_test)
89
90     return vectors_train, vectors_test
91
92 """2. CountVectorizer with stop word
93 1) Use "CountVectorizer" with stop word to transform text data to vector.
94 2) Normalize your feature vectors
95 """
96
97 def count_vec_with_sw(X_train, X_test):
98     stop_words = text.ENGLISH_STOP_WORDS
99     vectorizer = CountVectorizer(stop_words=stop_words)
100    vectors_train_stop = vectorizer.fit_transform(X_train)
101    vectors_test_stop = vectorizer.transform(X_test)
102
103    normalizer_train = Normalizer().fit(X=vectors_train_stop)
104    vectors_train_stop= normalizer_train.transform(vectors_train_stop)
105    vectors_test_stop = normalizer_train.transform(vectors_test_stop)
106
107    return vectors_train_stop, vectors_test_stop
108
109 """3. TF-IDF
110 1) use "TfidfVectorizer" to weight features based on your train set.
111 2) Normalize your feature vectors
112 """
113
114 def tfidf_vectorizer(X_train, X_test, binary=False):
115     stop_words = text.ENGLISH_STOP_WORDS
116     tf_idf_vectorizer = TfidfVectorizer(binary=binary, stop_words=stop_words)
117     vectors_train_idf = tf_idf_vectorizer.fit_transform(X_train)
118     vectors_test_idf = tf_idf_vectorizer.transform(X_test)
119
120     normalizer_train = Normalizer().fit(X=vectors_train_idf)
121     vectors_train_idf= normalizer_train.transform(vectors_train_idf)
122     vectors_test_idf = normalizer_train.transform(vectors_test_idf)
123
124     return vectors_train_idf, vectors_test_idf
125

```

```

126 """4. CountVectorizer with stem tokenizer
127 1) Use "StemTokenizer" to transform text data to vector.
128 2) Normalize your feature vectors
129 """
130
131 class StemTokenizer:
132     def __init__(self):
133         self.wnl = PorterStemmer()
134     def __call__(self, doc):
135         return [self.wnl.stem(t) for t in word_tokenize(doc) if t.isalpha()]
136
137
138 def count_vec_stem(X_train, X_test):
139     vectorizer = CountVectorizer(tokenizer=StemTokenizer())
140     vectors_train_stem = vectorizer.fit_transform(X_train)
141     vectors_test_stem = vectorizer.transform(X_test)
142
143     normalizer_train = Normalizer().fit(X=vectors_train_stem)
144     vectors_train_stem= normalizer_train.transform(vectors_train_stem)
145     vectors_test_stem = normalizer_train.transform(vectors_test_stem)
146
147     return vectors_train_stem, vectors_test_stem
148
149 """5. CountVectorizer with lemma tokenizer
150 1) Use "LemmaTokenizer" to transform text data to vector.
151 2) Normalize your feature vectors
152 """
153
154 def get_wordnet_pos(word):
155     """Map POS tag to first character lemmatize() accepts"""
156     tag = nltk.pos_tag([word])[0][1][0].upper()
157     tag_dict = {"J": wordnet.ADJ,
158                 "N": wordnet.NOUN,
159                 "V": wordnet.VERB,
160                 "R": wordnet.ADV}
161     return tag_dict.get(tag, wordnet.NOUN)
162
163
164 class LemmaTokenizer:
165     def __init__(self):
166         self.wnl = WordNetLemmatizer()
167     def __call__(self, doc):
168         return [self.wnl.lemmatize(t,pos =get_wordnet_pos(t)) for t in
169             word_tokenize(doc) if t.isalpha()]
170
171
172 def count_vec_lemma(X_train, X_test):
173     vectorizer = CountVectorizer(tokenizer=LemmaTokenizer())
174     vectors_train_lemma = vectorizer.fit_transform(X_train)
175     vectors_test_lemma = vectorizer.transform(X_test)
176
177     normalizer_train = Normalizer().fit(X=vectors_train_lemma)
178     vectors_train_lemma= normalizer_train.transform(vectors_train_lemma)
179     vectors_test_lemma = normalizer_train.transform(vectors_test_lemma)
180
181     return vectors_train_lemma, vectors_test_lemma
182
183 """## 9. Stacking classifier"""
184 from time import time
185
186 from sklearn.ensemble import StackingClassifier
187 tic = time()

```

```

188 accuracies = []
189 kf = KFold(n_splits=5, shuffle=True)
190 for train_index, test_index in kf.split(X):
191     vectors_train, vectors_test = tfidf_vectorizer(X[train_index], X[
192         test_index], binary=True)
193     estimators = [
194         ('mlp', MLPClassifier(max_iter=1000, learning_rate="adaptive",
195             learning_rate_init=0.0001)),
196         ('svc', svm.SVC(kernel='linear', gamma='auto', C=1, probability=True
197             )),
198         ('lr', LogisticRegression(C=40.0, max_iter=1000))
199     ]
200     clf = StackingClassifier(
201         estimators=estimators, final_estimator=LogisticRegression()
202     )
203     clf.fit(vectors_train, y[train_index])
204     y_test = clf.predict(vectors_test)
205     accuracies.append(metrics.accuracy_score(y[test_index], y_test))
206     print(accuracies[-1])
207 print("Average accuracy of Stacking Classification = {}".format(np.mean(
208     accuracies)));
209 toc = time()
210 print("Time spent for Stacking Classification (with 5 fold validation) = {}".format(toc - tic))
211 """
212 ## 10. Voting Classifier"""
213
214 from sklearn.ensemble import VotingClassifier
215 tic = time()
216 accuracies = []
217 kf = KFold(n_splits=5, shuffle=True)
218 for train_index, test_index in kf.split(X):
219     vectors_train, vectors_test = tfidf_vectorizer(X[train_index], X[
220         test_index], binary=True)
221     clf1 = MLPClassifier(max_iter=1000, learning_rate="adaptive",
222         learning_rate_init=0.0001)
223     clf2 = LogisticRegression(C=40.0, max_iter=1000)
224     clf3 = svm.SVC(kernel='linear', gamma='auto', C=1, probability=True)
225     eclf = VotingClassifier(estimators=[('mlp', clf1), ('lr', clf2), ('svc',
226         clf3)],
227                             voting='soft', weights=[1,1,1])
228     eclf = eclf.fit(vectors_train, y[train_index])
229     y_test = eclf.predict(vectors_test)
230     accuracies.append(metrics.accuracy_score(y[test_index], y_test))
231     print(accuracies[-1])
232 print("Average accuracy of Volting Classification = {}".format(np.mean(
233     accuracies)));
234 toc = time()
235 print("Time spent for Volting Classification (with 5 fold validation) = {}".format(toc - tic))

```

```
1 # -*- coding: utf-8 -*-
2 """Export CSV.ipynb
3
4 Automatically generated by Colaboratory.
5
6 Original file is located at
7     https://colab.research.google.com/drive/1uNTFaEcHjLGUKGUojaKE8gCcied9FaLQ
8
9 <center><h1>Mini Project 2 - Bernoulli Naïve Bayes</h1>
10 <h4>This file is for file exporting.</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
38 !pip install nltk
39 import nltk
40 nltk.download('punkt')
41 nltk.download('wordnet')
42 nltk.download('averaged_perceptron_tagger')
43
44 from nltk.stem import PorterStemmer
45 from nltk import word_tokenize
46 from nltk import word_tokenize
47 from nltk.stem import WordNetLemmatizer
48 from nltk.corpus import wordnet
49
50 """Additional classifiers:
51 1. Logistic Regression
52 2. Multinomial Naïve Bayes
53 3. Support Vector Machine
54 4. Random Forest
55 5. Decision Tree
56 6. Ada Boost
57 7. k-Neighbors
58 8. Neural Network
59 """
60
61 from sklearn.linear_model import LogisticRegression
62 from sklearn.naive_bayes import MultinomialNB
63 from sklearn import svm
```

```

64 from sklearn.ensemble import RandomForestClassifier
65 from sklearn.tree import DecisionTreeClassifier
66 from sklearn.ensemble import AdaBoostClassifier
67 from sklearn.neighbors import KNeighborsClassifier
68 from sklearn.neural_network import MLPClassifier
69
70 reddit_dataset = pd.read_csv(path+"train.csv")
71 reddit_test = pd.read_csv(path+"test.csv")
72
73 X = reddit_dataset['body']
74 y = reddit_dataset['subreddit']
75
76 """# Define Vectorizer
77 ### (To vectorize the text-based data to numerical features)
78
79 1. CountVectorizer
80 1) Use "CountVectorizer" to transform text data to feature vectors.
81 2) Normalize your feature vectors
82 """
83
84 def count_vectorizer(X_train, X_test):
85     vectorizer = CountVectorizer()
86     vectors_train = vectorizer.fit_transform(X_train)
87     vectors_test = vectorizer.transform(X_test)
88
89     # z-score normalization
90     normalizer_train = Normalizer().fit(X=vectors_train)
91     vectors_train = normalizer_train.transform(vectors_train)
92     vectors_test = normalizer_train.transform(vectors_test)
93
94     return vectors_train, vectors_test
95
96 """2. CountVectorizer with stop word
97 1) Use "CountVectorizer" with stop word to transform text data to vector.
98 2) Normalize your feature vectors
99 """
100
101 def count_vec_with_sw(X_train, X_test):
102     stop_words = text.ENGLISH_STOP_WORDS
103     vectorizer = CountVectorizer(stop_words=stop_words)
104     vectors_train_stop = vectorizer.fit_transform(X_train)
105     vectors_test_stop = vectorizer.transform(X_test)
106
107     # z-score normalization
108     normalizer_train = Normalizer().fit(X=vectors_train_stop)
109     vectors_train_stop= normalizer_train.transform(vectors_train_stop)
110     vectors_test_stop = normalizer_train.transform(vectors_test_stop)
111
112     return vectors_train_stop, vectors_test_stop
113
114 """3. TF-IDF
115 1) use "TfidfVectorizer" to weight features based on your train set.
116 2) Normalize your feature vectors
117 """
118
119 def tfidf_vectorizer(X_train, X_test):
120     tf_idf_vectorizer = TfidfVectorizer()
121     vectors_train_idf = tf_idf_vectorizer.fit_transform(X_train)
122     vectors_test_idf = tf_idf_vectorizer.transform(X_test)
123
124     # z-score normalization
125     normalizer_train = Normalizer().fit(X=vectors_train_idf)
126     vectors_train_idf= normalizer_train.transform(vectors_train_idf)

```

```

127     vectors_test_idf = normalizer_train.transform(vectors_test_idf)
128
129     return vectors_train_idf, vectors_test_idf
130
131 """4. CountVectorizer with stem tokenizer
132 1) Use "StemTokenizer" to transform text data to vector.
133 2) Normalize your feature vectors
134 """
135
136 class StemTokenizer:
137     def __init__(self):
138         self.wnl = PorterStemmer()
139     def __call__(self, doc):
140         return [self.wnl.stem(t) for t in word_tokenize(doc) if t.isalpha()]
141
142
143 def count_vec_stem(X_train, X_test):
144     vectorizer = CountVectorizer(tokenizer=StemTokenizer())
145     vectors_train_stem = vectorizer.fit_transform(X_train)
146     vectors_test_stem = vectorizer.transform(X_test)
147
148     # z-score normalization
149     normalizer_train = Normalizer().fit(X=vectors_train_stem)
150     vectors_train_stem= normalizer_train.transform(vectors_train_stem)
151     vectors_test_stem = normalizer_train.transform(vectors_test_stem)
152
153     return vectors_train_stem, vectors_test_stem
154
155 """5. CountVectorizer with lemma tokenizer
156 1) Use "LemmaTokenizer" to transform text data to vector.
157 2) Normalize your feature vectors
158 """
159
160 def get_wordnet_pos(word):
161     """Map POS tag to first character lemmatize() accepts"""
162     tag = nltk.pos_tag([word])[0][1][0].upper()
163     tag_dict = {"J": wordnet.ADJ,
164                 "N": wordnet.NOUN,
165                 "V": wordnet.VERB,
166                 "R": wordnet.ADV}
167     return tag_dict.get(tag, wordnet.NOUN)
168
169
170 class LemmaTokenizer:
171     def __init__(self):
172         self.wnl = WordNetLemmatizer()
173     def __call__(self, doc):
174         return [self.wnl.lemmatize(t,pos =get_wordnet_pos(t)) for t in
175         word_tokenize(doc) if t.isalpha()]
176
177 def count_vec_lemma(X_train, X_test):
178     vectorizer = CountVectorizer(tokenizer=LemmaTokenizer())
179     vectors_train_lemma = vectorizer.fit_transform(X_train)
180     vectors_test_lemma = vectorizer.transform(X_test)
181
182     # z-score normalization
183     normalizer_train = Normalizer().fit(X=vectors_train_lemma)
184     vectors_train_lemma= normalizer_train.transform(vectors_train_lemma)
185     vectors_test_lemma = normalizer_train.transform(vectors_test_lemma)
186
187     return vectors_train_lemma, vectors_test_lemma
188

```

```
189 """# Export csv"""
190
191 # test set id
192 X_id = reddit_test['id']
193 # test set features
194 X_test = reddit_test['body']
195
196 # vectorize the training and testing data
197 vectors_train, vectors_test = tfidf_vectorizer(X, X_test)
198 # perform MLP classification
199 clf = MLPClassifier(random_state=0, max_iter=1000, learning_rate="adaptive",
200                      learning_rate_init=0.0001).fit(vectors_train, y)
201
202 # predict the result
203 y_test = clf.predict(vectors_test)
204
205 # put the result into a pandas dataframe
206 result = {'id': X_id, ' subreddit': y_test}
207 df = pd.DataFrame(data=result)
208
209 # export to csv
210 df.to_csv('result.csv', index=False)
211
212 # download csv
213 from google.colab import files
214 files.download('result.csv')
```