```python
# -*- coding: utf-8 -*-
"""Additional Classifiers Testing.ipynb

Automatically generated by Colaboratory.

Original file is located at
    https://colab.research.google.com/drive/16E3avMbgZz1YnYvo1uuc4HLzgHZiaDCQ

<center><h1>Mini Project 2 - Bernoulli Naïve Bayes</h1></center>
This file consists two parts:

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.

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.

<h3>Team Members:</h3>
<center>
Yi Zhu, 260716006<br>
Fei Peng, 260712440<br>
Yukai Zhang, 260710915
</center>
"""

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

# make path = './' in-case you are running this locally
path = '/content/drive/My Drive/ECSE_551_F_2020/Mini_Project_02/'

import numpy as np
import pandas as pd
import matplotlib.pyplot as plt

from time import time
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import Normalizer
from sklearn.feature_extraction.text import CountVectorizer, TfidfVectorizer
from sklearn.feature_extraction import text
from sklearn import metrics
from sklearn.model_selection import GridSearchCV, cross_val_score, KFold
from sklearn.pipeline import make_pipeline

!pip install nltk
import nltk
nltk.download('punkt')
nltk.download('wordnet')
nltk.download('averaged_perceptron_tagger')

from nltk.stem import PorterStemmer
from nltk import word_tokenize
from nltk import word_tokenize
from nltk.stem import WordNetLemmatizer
from nltk.corpus import wordnet

"""Additional classifiers:
1. Logistic Regression
2. Multinomial Naïve Bayes
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
    word_tokenize(doc) if t.isalpha()]
183
```

```
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
    Validation
198
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[
    test_index])
210     clf.fit(vectors_train, y[train_index])
211     accuracies.append(metrics.accuracy_score(y[test_index], clf.predict(
    vectors_test)))
212
213 print("\t- Logestic Regression + CountVectorizer + Normalize -\nAccuracy: {}%\
    tTime Spent: {}s".format(np.mean(accuracies), time()-tic))
214
215 tic = time()
216 accuracies = []
217 clf = LogisticRegression(C=1.0, max_iter=1000, random_state=0)
218 kf = KFold(n_splits=5, shuffle=True)
219 for train_index, test_index in kf.split(X):
220     vectors_train, vectors_test = count_vectorizer(X[train_index], X[
    test_index], False)
221     clf.fit(vectors_train, y[train_index])
222     accuracies.append(metrics.accuracy_score(y[test_index], clf.predict(
    vectors_test)))
223
224 print("\t- Logestic Regression + CountVectorizer + Unnormalize -\nAccuracy
    : {}%\tTime Spent: {}s".format(np.mean(accuracies), time()-tic))
225
226 """### 2. CountVectorizer with stop word"""
227
228 tic = time()
229 accuracies = []
230 clf = LogisticRegression(C=1.0, max_iter=1000, random_state=0)
231 kf = KFold(n_splits=5, shuffle=True)
232 for train_index, test_index in kf.split(X):
233     vectors_train, vectors_test = count_vec_with_sw(X[train_index], X[
    test_index])
234     clf.fit(vectors_train, y[train_index])
235     accuracies.append(metrics.accuracy_score(y[test_index], clf.predict(
    vectors_test)))
236
237 print("\t- Logestic Regression + CountVectorizer with stop word + Normalize -\
```

```
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[
    test_index], False)
245     clf.fit(vectors_train, y[train_index])
246     accuracies.append(metrics.accuracy_score(y[test_index], clf.predict(
    vectors_test)))
247
248 print("\t- Logestic Regression + CountVectorizer with stop word + Unnormalize
     -\nAccuracy: {}%\tTime Spent: {}s".format(np.mean(accuracies), time()-tic))
249
250 """### 3. TF-IDF"""
251
252 tic = time()
253 accuracies = []
254 clf = LogisticRegression(C=40.0, max_iter=1000, random_state=0)
255 kf = KFold(n_splits=5, shuffle=True)
256 for train_index, test_index in kf.split(X):
257     vectors_train, vectors_test = tfidf_vectorizer(X[train_index], X[
    test_index])
258     clf.fit(vectors_train, y[train_index])
259     accuracies.append(metrics.accuracy_score(y[test_index], clf.predict(
    vectors_test)))
260
261 print("\t- Logestic Regression + TF-IDF Vectorizer + Normalize -\nAccuracy
    : {}%\tTime Spent: {}s".format(np.mean(accuracies), time()-tic))
262
263 tic = time()
264 accuracies = []
265 clf = LogisticRegression(C=40.0, max_iter=1000, random_state=0)
266 kf = KFold(n_splits=5, shuffle=True)
267 for train_index, test_index in kf.split(X):
268     vectors_train, vectors_test = tfidf_vectorizer(X[train_index], X[
    test_index], False)
269     clf.fit(vectors_train, y[train_index])
270     accuracies.append(metrics.accuracy_score(y[test_index], clf.predict(
    vectors_test)))
271
272 print("\t- Logestic Regression + TF-IDF Vectorizer + Unnormalize -\nAccuracy
    : {}%\tTime Spent: {}s".format(np.mean(accuracies), time()-tic))
273
274 """### 4. CountVectorizer with stem tokenizer"""
275
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     clf.fit(vectors_train, y[train_index])
283     accuracies.append(metrics.accuracy_score(y[test_index], clf.predict(
    vectors_test)))
284
285 print("\t- Logestic 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
    ], False)
293     clf.fit(vectors_train, y[train_index])
294     accuracies.append(metrics.accuracy_score(y[test_index], clf.predict(
    vectors_test)))
295 
296 print("\t- Logestic Regression + CountVectorizer with stem tokenizer +
    Unnormalize -\nAccuracy: {}%\tTime Spent: {}s".format(np.mean(accuracies),
    time()-tic))
297 
298 """### 5. CountVectorizer with lemma tokenizer"""
299 
300 tic = time()
301 accuracies = []
302 clf = LogisticRegression(C=1.0, max_iter=1000, random_state=0)
303 kf = KFold(n_splits=5, shuffle=True)
304 for train_index, test_index in kf.split(X):
305     vectors_train, vectors_test = count_vec_lemma(X[train_index], X[test_index
    ])
306     clf.fit(vectors_train, y[train_index])
307     accuracies.append(metrics.accuracy_score(y[test_index], clf.predict(
    vectors_test)))
308 
309 print("\t- Logestic Regression + CountVectorizer with lemma tokenizer +
    Normalize -\nAccuracy: {}%\tTime Spent: {}s".format(np.mean(accuracies), time
    ()-tic))
310 
311 tic = time()
312 accuracies = []
313 clf = LogisticRegression(C=1.0, max_iter=1000, random_state=0)
314 kf = KFold(n_splits=5, shuffle=True)
315 for train_index, test_index in kf.split(X):
316     vectors_train, vectors_test = count_vec_lemma(X[train_index], X[test_index
    ], False)
317     clf.fit(vectors_train, y[train_index])
318     accuracies.append(metrics.accuracy_score(y[test_index], clf.predict(
    vectors_test)))
319 
320 print("\t- Logestic Regression + CountVectorizer with lemma tokenizer +
    Unnormalize -\nAccuracy: {}%\tTime Spent: {}s".format(np.mean(accuracies),
    time()-tic))
321 
322 """## 2. Multinomial Naïve Bayes"""
323 
324 tic = time()
325 accuracies = []
326 clf = MultinomialNB()
327 kf = KFold(n_splits=5, shuffle=True)
328 for train_index, test_index in kf.split(X):
329     vectors_train, vectors_test = tfidf_vectorizer(X[train_index], X[
    test_index])
330     clf.fit(vectors_train, y[train_index])
331     accuracies.append(metrics.accuracy_score(y[test_index], clf.predict(
    vectors_test)))
332 
333 print("\t- Multinomial Naïve Bayes + TF-IDF Vectorizer + Normalize -\nAccuracy
    : {}%\tTime Spent: {}s".format(np.mean(accuracies), time()-tic))
334 
```

```
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[
    test_index])
346     clf.fit(vectors_train, y[train_index])
347     accuracies.append(metrics.accuracy_score(y[test_index], clf.predict(
    vectors_test)))
348 
349 print("\t- Linear Support Vector Machine + TF-IDF Vectorizer + Normalize -\n
    Accuracy: {}%\tTime Spent: {}s".format(np.mean(accuracies), time()-tic))
350 
351 """## 4. Random Forest"""
352 
353 tic = time()
354 accuracies = []
355 clf = RandomForestClassifier(max_depth=2, random_state=0)
356 kf = KFold(n_splits=5, shuffle=True)
357 for train_index, test_index in kf.split(X):
358     vectors_train, vectors_test = tfidf_vectorizer(X[train_index], X[
    test_index])
359     clf.fit(vectors_train, y[train_index])
360     accuracies.append(metrics.accuracy_score(y[test_index], clf.predict(
    vectors_test)))
361 
362 print("\t- Random Forest + TF-IDF Vectorizer + Normalize -\nAccuracy: {}%\t
    Time Spent: {}s".format(np.mean(accuracies), time()-tic))
363 
364 """## 5. Decision Tree"""
365 
366 tic = time()
367 accuracies = []
368 clf = DecisionTreeClassifier(random_state=0)
369 kf = KFold(n_splits=5, shuffle=True)
370 for train_index, test_index in kf.split(X):
371     vectors_train, vectors_test = tfidf_vectorizer(X[train_index], X[
    test_index])
372     clf.fit(vectors_train, y[train_index])
373     accuracies.append(metrics.accuracy_score(y[test_index], clf.predict(
    vectors_test)))
374 
375 print("\t- Decision Tree + TF-IDF Vectorizer + Normalize -\nAccuracy: {}%\t
    Time Spent: {}s".format(np.mean(accuracies), time()-tic))
376 
377 """## 6. Ada Boost"""
378 
379 tic = time()
380 accuracies = []
381 clf = AdaBoostClassifier(n_estimators=100, learning_rate=0.5, random_state=0)
382 kf = KFold(n_splits=5, shuffle=True)
383 for train_index, test_index in kf.split(X):
384     vectors_train, vectors_test = tfidf_vectorizer(X[train_index], X[
    test_index])
385     clf.fit(vectors_train, y[train_index])
386     accuracies.append(metrics.accuracy_score(y[test_index], clf.predict(
    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[
    test_index])
398     neigh.fit(vectors_train, y[train_index])
399     accuracies.append(metrics.accuracy_score(y[test_index], neigh.predict(
    vectors_test)))
400
401 print("\t- k-Neighbors + TF-IDF Vectorizer + Normalize -\nAccuracy: {}%\tTime
    Spent: {}s".format(np.mean(accuracies), time()-tic))
402
403 """## 8. Neural Network"""
404
405 tic = time()
406 accuracies = []
407 kf = KFold(n_splits=5, shuffle=True)
408 for train_index, test_index in kf.split(X):
409     vectors_train, vectors_test = tfidf_vectorizer(X[train_index], X[
    test_index])
410     clf = MLPClassifier(random_state=0, max_iter=300).fit(vectors_train, y[
    train_index])
411     accuracies.append(metrics.accuracy_score(y[test_index], clf.predict(
    vectors_test)))
412
413 print("\t- Neural Network + TF-IDF Vectorizer + Normalize -\nAccuracy: {}%\t
    Time Spent: {}s".format(np.mean(accuracies), time()-tic))
414
415 """## 9. Bernoulli Naïve Bayes (Sklearn Version)
416 <h2>This part is only used to test and compare the performance of the
    Bernoulli Naïve Bayes implemented by ourselves.</h2>
417 """
418
419 from sklearn.naive_bayes import BernoulliNB
420
421 """### 1. CountVectorizer"""
422
423 tic = time()
424 accuracies = []
425 kf = KFold(n_splits=5, shuffle=True)
426 for train_index, test_index in kf.split(X):
427     vectors_train, vectors_test = count_vectorizer(X[train_index], X[
    test_index])
428     clf = BernoulliNB().fit(vectors_train, y[train_index])
429     accuracies.append(metrics.accuracy_score(y[test_index], clf.predict(
    vectors_test)))
430
431 print("\t- BernoulliNB + CountVectorizer + Normalize -\nAccuracy: {}\tTime
    Spent: {}s".format(np.mean(accuracies), time()-tic))
432
433 """### 2. CountVectorizer with stop word"""
434
435 tic = time()
436 accuracies = []
437 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(
    vectors_test)))
490
491 print("\t- BernoulliNB + CountVectorizer with lemma tokenizer + Normalize -\n
    Accuracy: {}\tTime Spent: {}s".format(np.mean(accuracies), time()-tic))
```