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

Automatically generated by Colaboratory.

Original file is located at
    https://colab.research.google.com/drive/1jQsRTZihNlPC99pBzRYqvTbowAY-tqi1

<center><h1>Mini Project 2 - Bernoulli Naïve Bayes</h1>
<h4>This file records the accuracies of the combinations of 8 classifiers and 5 vectorizers.</h4></center>

<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 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
4. Random Forest
5. Decision Tree
6. Ada Boost
7. k-Neighbors
8. Neural Network
"""

from sklearn.linear_model import LogisticRegression
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
    word_tokenize(doc) if t.isalpha()]
171
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[
    test_index])
196     clf.fit(vectors_train, y[train_index])
197     accuracies.append(metrics.accuracy_score(y[test_index], clf.predict(
    vectors_test)))
198 
199 print(np.mean(accuracies))
200 
201 """### 2. CountVectorizer with stop word"""
202 
203 accuracies = []
204 clf = LogisticRegression(C=1.0, max_iter=1000, random_state=0)
205 kf = KFold(n_splits=5, shuffle=True)
206 for train_index, test_index in kf.split(X):
207     vectors_train, vectors_test = count_vec_with_sw(X[train_index], X[
    test_index])
208     clf.fit(vectors_train, y[train_index])
209     accuracies.append(metrics.accuracy_score(y[test_index], clf.predict(
    vectors_test)))
210 
211 print(np.mean(accuracies))
212 
213 """### 3. TF-IDF"""
214 
215 accuracies = []
216 clf = LogisticRegression(C=40.0, max_iter=1000, random_state=0)
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[
    test_index])
220     clf.fit(vectors_train, y[train_index])
221     accuracies.append(metrics.accuracy_score(y[test_index], clf.predict(
    vectors_test)))
222 
223 print(np.mean(accuracies))
224 
225 """accuracy vs. hyperparameter C"""
226 
227 c_vecs = np.logspace(0, 2, 5)
228 acc = []
229 for c_vec in c_vecs:
230     accuracies = []
231     clf = LogisticRegression(C=c_vec, max_iter=1000, random_state=0)
232     kf = KFold(n_splits=5, shuffle=True)
233     for train_index, test_index in kf.split(X):
234         vectors_train, vectors_test = tfidf_vectorizer(X[train_index], X[
    test_index])
235         clf.fit(vectors_train, y[train_index])
236         accuracies.append(metrics.accuracy_score(y[test_index], clf.predict(
    vectors_test)))
237 
238     acc.append(np.mean(accuracies))
239 
240 plt.xlabel('C')
241 plt.ylabel('Accuracy')
242 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(
    vectors_test)))
253
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(
    vectors_test)))
265
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[
    test_index])
278     clf.fit(vectors_train, y[train_index])
279     accuracies.append(metrics.accuracy_score(y[test_index], clf.predict(
    vectors_test)))
280
281 print(np.mean(accuracies))
282
283 """### 2. CountVectorizer with stop word"""
284
285 accuracies = []
286 clf = MultinomialNB()
287 kf = KFold(n_splits=5, shuffle=True)
288 for train_index, test_index in kf.split(X):
289     vectors_train, vectors_test = count_vec_with_sw(X[train_index], X[
    test_index])
290     clf.fit(vectors_train, y[train_index])
291     accuracies.append(metrics.accuracy_score(y[test_index], clf.predict(
    vectors_test)))
292
293 print(np.mean(accuracies))
294
295 """### 3. TF-IDF"""
296
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[
    test_index])
302     clf.fit(vectors_train, y[train_index])
303     accuracies.append(metrics.accuracy_score(y[test_index], clf.predict(
    vectors_test)))
304
305 print(np.mean(accuracies))
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     clf.fit(vectors_train, y[train_index])
315     accuracies.append(metrics.accuracy_score(y[test_index], clf.predict(
    vectors_test)))
316
317 print(np.mean(accuracies))
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     clf.fit(vectors_train, y[train_index])
327     accuracies.append(metrics.accuracy_score(y[test_index], clf.predict(
    vectors_test)))
328
329 print(np.mean(accuracies))
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[
    test_index])
341     clf.fit(vectors_train, y[train_index])
342     accuracies.append(metrics.accuracy_score(y[test_index], clf.predict(
    vectors_test)))
343
344 print(np.mean(accuracies))
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(
    vectors_test)))
355 
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[
    test_index])
368     clf.fit(vectors_train, y[train_index])
369     accuracies.append(metrics.accuracy_score(y[test_index], clf.predict(
    vectors_test)))
370 
371 print(np.mean(accuracies))
372 
373 """Non-Linear"""
374 
375 accuracies = []
376 clf = svm.SVC(gamma='auto', C=1)
377 kf = KFold(n_splits=5, shuffle=True)
378 for train_index, test_index in kf.split(X):
379     vectors_train, vectors_test = tfidf_vectorizer(X[train_index], X[
    test_index])
380     clf.fit(vectors_train, y[train_index])
381     accuracies.append(metrics.accuracy_score(y[test_index], clf.predict(
    vectors_test)))
382 
383 print(np.mean(accuracies))
384 
385 """### 4. CountVectorizer with stem tokenizer"""
386 
387 accuracies = []
388 clf = svm.SVC(kernel='linear', gamma='auto', C=1)
389 kf = KFold(n_splits=5, shuffle=True)
390 for train_index, test_index in kf.split(X):
391     vectors_train, vectors_test = count_vec_stem(X[train_index], X[test_index
    ])
392     clf.fit(vectors_train, y[train_index])
393     accuracies.append(metrics.accuracy_score(y[test_index], clf.predict(
    vectors_test)))
394 
395 print(np.mean(accuracies))
396 
397 """### 5. CountVectorizer with lemma tokenizer"""
398 
399 accuracies = []
400 clf = svm.SVC(kernel='linear', gamma='auto', C=1)
401 kf = KFold(n_splits=5, shuffle=True)
402 for train_index, test_index in kf.split(X):
403     vectors_train, vectors_test = count_vec_lemma(X[train_index], X[test_index
    ])
404     clf.fit(vectors_train, y[train_index])
405     accuracies.append(metrics.accuracy_score(y[test_index], clf.predict(
    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[
    test_index])
419     clf.fit(vectors_train, y[train_index])
420     accuracies.append(metrics.accuracy_score(y[test_index], clf.predict(
    vectors_test)))
421
422 print(np.mean(accuracies))
423
424 """### 2. CountVectorizer with stop word"""
425
426 accuracies = []
427 clf = RandomForestClassifier(max_depth=2, random_state=0)
428 kf = KFold(n_splits=5, shuffle=True)
429 for train_index, test_index in kf.split(X):
430     vectors_train, vectors_test = count_vec_with_sw(X[train_index], X[
    test_index])
431     clf.fit(vectors_train, y[train_index])
432     accuracies.append(metrics.accuracy_score(y[test_index], clf.predict(
    vectors_test)))
433
434 print(np.mean(accuracies))
435
436 """### 3. TF-IDF"""
437
438 accuracies = []
439 clf = RandomForestClassifier(max_depth=2, random_state=0)
440 kf = KFold(n_splits=5, shuffle=True)
441 for train_index, test_index in kf.split(X):
442     vectors_train, vectors_test = tfidf_vectorizer(X[train_index], X[
    test_index])
443     clf.fit(vectors_train, y[train_index])
444     accuracies.append(metrics.accuracy_score(y[test_index], clf.predict(
    vectors_test)))
445
446 print(np.mean(accuracies))
447
448 """### 4. CountVectorizer with stem tokenizer"""
449
450 accuracies = []
451 clf = RandomForestClassifier(max_depth=2, random_state=0)
452 kf = KFold(n_splits=5, shuffle=True)
453 for train_index, test_index in kf.split(X):
454     vectors_train, vectors_test = count_vec_stem(X[train_index], X[test_index
    ])
455     clf.fit(vectors_train, y[train_index])
456     accuracies.append(metrics.accuracy_score(y[test_index], clf.predict(
    vectors_test)))
457
458 print(np.mean(accuracies))
459
460 """### 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(
    vectors_test)))
469
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[
    test_index])
482     clf.fit(vectors_train, y[train_index])
483     accuracies.append(metrics.accuracy_score(y[test_index], clf.predict(
    vectors_test)))
484
485 print(np.mean(accuracies))
486
487 """### 2. CountVectorizer with stop word"""
488
489 accuracies = []
490 clf = DecisionTreeClassifier(random_state=0)
491 kf = KFold(n_splits=5, shuffle=True)
492 for train_index, test_index in kf.split(X):
493     vectors_train, vectors_test = count_vec_with_sw(X[train_index], X[
    test_index])
494     clf.fit(vectors_train, y[train_index])
495     accuracies.append(metrics.accuracy_score(y[test_index], clf.predict(
    vectors_test)))
496
497 print(np.mean(accuracies))
498
499 """### 3. TF-IDF"""
500
501 accuracies = []
502 clf = DecisionTreeClassifier(random_state=0)
503 kf = KFold(n_splits=5, shuffle=True)
504 for train_index, test_index in kf.split(X):
505     vectors_train, vectors_test = tfidf_vectorizer(X[train_index], X[
    test_index])
506     clf.fit(vectors_train, y[train_index])
507     accuracies.append(metrics.accuracy_score(y[test_index], clf.predict(
    vectors_test)))
508
509 print(np.mean(accuracies))
510
511 """### 4. CountVectorizer with stem tokenizer"""
512
513 accuracies = []
514 clf = DecisionTreeClassifier(random_state=0)
515 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     clf.fit(vectors_train, y[train_index])
519     accuracies.append(metrics.accuracy_score(y[test_index], clf.predict(
    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
    ])
530     clf.fit(vectors_train, y[train_index])
531     accuracies.append(metrics.accuracy_score(y[test_index], clf.predict(
    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[
    test_index])
545     clf.fit(vectors_train, y[train_index])
546     accuracies.append(metrics.accuracy_score(y[test_index], clf.predict(
    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[
    test_index])
557     clf.fit(vectors_train, y[train_index])
558     accuracies.append(metrics.accuracy_score(y[test_index], clf.predict(
    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],
570               'adaboostclassifier__learning_rate': [0.1, 0.5, 1]}
```

```
grid = GridSearchCV(model, param_grid)

X_train, X_test, y_train, y_test = train_test_split(X, y, train_size=0.8, test_size=0.2, random_state=0)
vectors_train, vectors_test = tfidf_vectorizer(X_train, X_test)
# %time grid.fit(vectors_train, y_train)
print(grid.best_params_)

accuracies = []
clf = AdaBoostClassifier(n_estimators=100, learning_rate=0.5, random_state=0)
kf = KFold(n_splits=5, shuffle=True)
for train_index, test_index in kf.split(X):
    vectors_train, vectors_test = tfidf_vectorizer(X[train_index], X[test_index])
    clf.fit(vectors_train, y[train_index])
    accuracies.append(metrics.accuracy_score(y[test_index], clf.predict(vectors_test)))

print(np.mean(accuracies))

"""### 4. CountVectorizer with stem tokenizer"""

accuracies = []
clf = AdaBoostClassifier(n_estimators=100, learning_rate=0.5, random_state=0)
kf = KFold(n_splits=5, shuffle=True)
for train_index, test_index in kf.split(X):
    vectors_train, vectors_test = count_vec_stem(X[train_index], X[test_index])
    clf.fit(vectors_train, y[train_index])
    accuracies.append(metrics.accuracy_score(y[test_index], clf.predict(vectors_test)))

print(np.mean(accuracies))

"""### 5. CountVectorizer with lemma tokenizer"""

accuracies = []
clf = AdaBoostClassifier(n_estimators=100, learning_rate=0.5, random_state=0)
kf = KFold(n_splits=5, shuffle=True)
for train_index, test_index in kf.split(X):
    vectors_train, vectors_test = count_vec_lemma(X[train_index], X[test_index])
    clf.fit(vectors_train, y[train_index])
    accuracies.append(metrics.accuracy_score(y[test_index], clf.predict(vectors_test)))

print(np.mean(accuracies))

"""## 7. k-Neighbors

### 1. CountVectorizer
"""

accuracies = []
neigh = KNeighborsClassifier(n_neighbors=3)
kf = KFold(n_splits=5, shuffle=True)
for train_index, test_index in kf.split(X):
    vectors_train, vectors_test = count_vectorizer(X[train_index], X[test_index])
    neigh.fit(vectors_train, y[train_index])
    accuracies.append(metrics.accuracy_score(y[test_index], neigh.predict(vectors_test)))

```

```
print(np.mean(accuracies))

"""### 2. CountVectorizer with stop word"""

accuracies = []
neigh = KNeighborsClassifier(n_neighbors=3)
kf = KFold(n_splits=5, shuffle=True)
for train_index, test_index in kf.split(X):
    vectors_train, vectors_test = count_vec_with_sw(X[train_index], X[
test_index])
    neigh.fit(vectors_train, y[train_index])
    accuracies.append(metrics.accuracy_score(y[test_index], neigh.predict(
vectors_test)))

print(np.mean(accuracies))

"""### 3. TF-IDF"""

accuracies = []
neigh = KNeighborsClassifier(n_neighbors=3)
kf = KFold(n_splits=5, shuffle=True)
for train_index, test_index in kf.split(X):
    vectors_train, vectors_test = tfidf_vectorizer(X[train_index], X[
test_index])
    neigh.fit(vectors_train, y[train_index])
    accuracies.append(metrics.accuracy_score(y[test_index], neigh.predict(
vectors_test)))

print(np.mean(accuracies))

"""### 4. CountVectorizer with stem tokenizer"""

accuracies = []
neigh = KNeighborsClassifier(n_neighbors=3)
kf = KFold(n_splits=5, shuffle=True)
for train_index, test_index in kf.split(X):
    vectors_train, vectors_test = count_vec_stem(X[train_index], X[test_index
])
    neigh.fit(vectors_train, y[train_index])
    accuracies.append(metrics.accuracy_score(y[test_index], neigh.predict(
vectors_test)))

print(np.mean(accuracies))

"""### 5. CountVectorizer with lemma tokenizer"""

accuracies = []
neigh = KNeighborsClassifier(n_neighbors=3)
kf = KFold(n_splits=5, shuffle=True)
for train_index, test_index in kf.split(X):
    vectors_train, vectors_test = count_vec_lemma(X[train_index], X[test_index
])
    neigh.fit(vectors_train, y[train_index])
    accuracies.append(metrics.accuracy_score(y[test_index], neigh.predict(
vectors_test)))

print(np.mean(accuracies))

"""## 8. Neural Network

### 1. CountVectorizer
"""

```

```
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[
    test_index])
684     clf = MLPClassifier(random_state=0, max_iter=300).fit(vectors_train, y[
    train_index])
685     accuracies.append(metrics.accuracy_score(y[test_index], clf.predict(
    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[
    test_index])
695     clf = MLPClassifier(random_state=0, max_iter=300).fit(vectors_train, y[
    train_index])
696     accuracies.append(metrics.accuracy_score(y[test_index], clf.predict(
    vectors_test)))
697 
698 print(np.mean(accuracies))
699 
700 """### 3. TF-IDF"""
701 
702 accuracies = []
703 kf = KFold(n_splits=5, shuffle=True)
704 for train_index, test_index in kf.split(X):
705     vectors_train, vectors_test = tfidf_vectorizer(X[train_index], X[
    test_index])
706     clf = MLPClassifier(random_state=0, max_iter=300).fit(vectors_train, y[
    train_index])
707     accuracies.append(metrics.accuracy_score(y[test_index], clf.predict(
    vectors_test)))
708 
709 print(np.mean(accuracies))
710 
711 """### 4. CountVectorizer with stem tokenizer"""
712 
713 accuracies = []
714 kf = KFold(n_splits=5, shuffle=True)
715 for train_index, test_index in kf.split(X):
716     vectors_train, vectors_test = count_vec_stem(X[train_index], X[test_index
    ])
717     clf = MLPClassifier(random_state=0, max_iter=300).fit(vectors_train, y[
    train_index])
718     accuracies.append(metrics.accuracy_score(y[test_index], clf.predict(
    vectors_test)))
719 
720 print(np.mean(accuracies))
721 
722 """### 5. CountVectorizer with lemma tokenizer"""
723 
724 accuracies = []
725 kf = KFold(n_splits=5, shuffle=True)
726 for train_index, test_index in kf.split(X):
727     vectors_train, vectors_test = count_vec_lemma(X[train_index], X[test_index
    ])
728     clf = MLPClassifier(random_state=0, max_iter=300).fit(vectors_train, y[
    train_index])
```

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