```
# -*- coding: utf-8 -*-
"""Stacking Classifier.ipynb

Automatically generated by Colaboratory.

Original file is located at
    https://colab.research.google.com/drive/1OKCClcD9Qf2Z59egC0iTBDwc2xkamhsM

<center><h1>Mini Project 2 - Bernoulli Naïve Bayes</h1>
<h4>The hyperparameters and models used in this file are chosen based on the findings in the testing file.</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 """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
    word_tokenize(doc) if t.isalpha()]
169
170
171 def count_vec_lemma(X_train, X_test):
172     vectorizer = CountVectorizer(tokenizer=LemmaTokenizer())
173     vectors_train_lemma = vectorizer.fit_transform(X_train)
174     vectors_test_lemma = vectorizer.transform(X_test)
175
176     normalizer_train = Normalizer().fit(X=vectors_train_lemma)
177     vectors_train_lemma= normalizer_train.transform(vectors_train_lemma)
178     vectors_test_lemma = normalizer_train.transform(vectors_test_lemma)
179
180     return vectors_train_lemma, vectors_test_lemma
181
182 """## 9. Stacking classifier"""
183
184 from time import time
185
186 from sklearn.ensemble import StackingClassifier
187 tic = time()
```

```
accuracies = []
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], binary=True)
    estimators = [
        ('mlp', MLPClassifier(max_iter=1000, learning_rate="adaptive", learning_rate_init=0.0001)),
        ('svc', svm.SVC(kernel='linear', gamma='auto', C=1, probability=True)),
        ('lr', LogisticRegression(C=40.0, max_iter=1000))
    ]
    clf = StackingClassifier(
        estimators=estimators, final_estimator=LogisticRegression()
    )
    clf.fit(vectors_train, y[train_index])
    y_test = clf.predict(vectors_test)
    accuracies.append(metrics.accuracy_score(y[test_index], y_test))
    print(accuracies[-1])
print("Average accuracy of Stacking Classification = {}".format(np.mean(accuracies)));
toc = time()
print("Time spent for Stacking Classification (with 5 fold validation) = {}".format(toc - tic))

"""## 10. Voting Classifier"""

from sklearn.ensemble import VotingClassifier
tic = time()
accuracies = []
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], binary=True)
    clf1 = MLPClassifier(max_iter=1000, learning_rate="adaptive", learning_rate_init=0.0001)
    clf2 = LogisticRegression(C=40.0, max_iter=1000)
    clf3 = svm.SVC(kernel='linear', gamma='auto', C=1, probability=True)
    eclf = VotingClassifier(estimators=[('mlp', clf1), ('lr', clf2), ('svc', clf3)],
                            voting='soft', weights=[1,1,1])
    eclf = eclf.fit(vectors_train, y[train_index])
    y_test = eclf.predict(vectors_test)
    accuracies.append(metrics.accuracy_score(y[test_index], y_test))
    print(accuracies[-1])
print("Average accuracy of Volting Classification = {}".format(np.mean(accuracies)));
toc = time()
print("Time spent for Volting Classification (with 5 fold validation) = {}".format(toc - tic))
```