```
# -*- coding: utf-8 -*-
"""Data Preprocessing.ipynb

Automatically generated by Colaboratory.

Original file is located at
    https://colab.research.google.com/drive/1bRmLzkFmGP7xgUOUI6iv_p6dW1hX6R6P

<center><h1>Mini Project 2 - Bernoulli Naïve Bayes</h1>
<h3>Data Preprocessing</h3>
<h4>This file performs some of the operations on Data Preprocessing and Analysis.</h4></center>

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

# Importations
"""

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
import seaborn as sns
import random
from scipy import stats
from google.colab import files
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

from sklearn.linear_model import LogisticRegression
from sklearn.naive_bayes import MultinomialNB
from sklearn import svm
from sklearn.ensemble import RandomForestClassifier
from sklearn.tree import DecisionTreeClassifier
```

```
from sklearn.ensemble import AdaBoostClassifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn.neural_network import MLPClassifier

"""# Data Preprocessing"""

reddit_dataset = pd.read_csv(path+"train.csv")
reddit_test = pd.read_csv(path+"test.csv")

X = reddit_dataset['body']
y = reddit_dataset['subreddit']

class Data_Processing:
    def __init__(self, data, name='New Data'):
        self.data = data
        self.name = name

    def show_y_dist(self, ydata):
        plt.figure(figsize=(8,4))
        plt.subplot(111), sns.countplot(x='subreddit', data=ydata)
        plt.title('Distribution of Subreddit in {}'.format(self.name))
        plt.savefig("Distribution of Subreddit in {}.png".format(self.name), dpi = 1200)
        files.download("Distribution of Subreddit in {}.png".format(self.name))
        plt.show()

data_analysis = Data_Processing(reddit_dataset.values, 'train.csv')
data_analysis.show_y_dist(reddit_dataset)

# calculate the data entropy
from sklearn.preprocessing import LabelEncoder
le = LabelEncoder()  # encoder for classes
le.fit(y)
y_label = le.transform(y)
n_k = len(le.classes_)
N = len(y)
theta_k = np.zeros(n_k)  # probability of class k
# compute theta values
for k in range(n_k):
    count_k = (y_label==k).sum()
    theta_k[k] = count_k / N

from scipy.stats import entropy
print("Data entropy is", entropy(theta_k, base=2))

"""# Define Vectorizer
(To vectorize the text-based data to numerical features)

1. CountVectorizer
1) Use "CountVectorizer" to transform text data to feature vectors.
2) Normalize your feature vectors
"""

def count_vectorizer(X_train):
    vectorizer = CountVectorizer()
    vectors_train = vectorizer.fit_transform(X_train)
    return vectors_train

"""2. CountVectorizer with stop word
1) Use "CountVectorizer" with stop word to transform text data to vector.
2) Normalize your feature vectors
"""
```

```
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
    word_tokenize(doc) if t.isalpha()]
178
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.")
```