NavieBayes.md

Pythoh Code of Naive Bayes

Rae YIngqi Zhang

```
from sklearn.feature_extraction.text import CountVectorizer, TfidfVectorizer
from nltk.stem.porter import PorterStemmer
import re
from sklearn.feature_extraction.text import CountVectorizer
from wordcloud import WordCloud, STOPWORDS
import matplotlib.pylab as plt
from sklearn.model_selection import train_test_split
from sklearn.metrics import confusion_matrix
import seaborn as sns
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.naive_bayes import MultinomialNB
```

Load text data and make a word count dataframe using CountVectorizer

```
DF_dairy = pd.read_csv("pubAg/content_dairy.csv")
DF_corn = pd.read_csv("pubAg/content_corn.csv")
DF_farms = pd.read_csv("pubAg/content_farms.csv")
DF_wheat = pd.read_csv("pubAg/content_wheat.csv")
DF_organic_farms = pd.read_csv("pubAg/content_organic_farms.csv")
DF_dairy.insert(0, "Label", "Dairy")
DF_corn.insert(0, "Label", "Corn")
DF_farms.insert(0, "Label", "Farms")
DF_wheat.insert(0, "Label", "Wheat")
DF_organic_farms.insert(0, "Label", "OrganicFarms")


CSV_DF = pd.concat([DF_dairy, DF_corn, DF_farms, DF_wheat, DF_organic_farms], ignore_index=True)
labels = CSV_DF["Label"]
```

CLEAN the content

```
AbstractLIST = []
for items in CSV_DF["abstract"]:
    line = str(items)
    # print(Title)
    line = re.sub(r'[,.;@#?!&$()%^*\-\']+', ' ', str(line), flags=re.IGNORECASE)
    line = re.sub(' +', ' ', str(line), flags=re.IGNORECASE)
    line = re.sub(r'\"', ' ', str(line), flags=re.IGNORECASE)
    print(line)
    # and replace it with a single space
    ## NOTE: Using the "^" on the inside of the [] means
    ## we want to look for any chars NOT a-z or A-Z and replace
    ## them with blank. This removes chars that should not be there.
    line = re.sub(r'[^a-zA-Z]', " ", str(line), flags=re.VERBOSE)
    line = line.replace(',', '')
    line = ' '.join(line.split())
    line = re.sub("\n|\r", "", line)
    # remove words of a given length.....
```

```
    line = ' '.join([wd for wd in line.split() if len(wd) > 3])
    AbstractLIST.append(line)

print("The abstract list is:\n")
print(AbstractLIST)
```

Remove all words that match the topics.

```
topic = ["Dairy", "Corn", "Farms", "Wheat", "Organic Farms"]
NewAbstractLIST = []

for element in AbstractLIST:
    print(element)
    element=str(element)
    print(type(element))
    ## make into list
    AllWords = element.split(" ")
    print(AllWords)

    ## Now remove words that are in your topics
    NewWordsList = []
    for word in AllWords:
        print(word)
        word = word.lower()
        if word in topic:
            print(word)
        else:
            NewWordsList.append(word)

    ##turn back to string
    NewWords = " ".join(NewWordsList)
    ## Place into NewHeadlineLIST
    NewAbstractLIST.append(NewWords)


STEMMER = PorterStemmer()
```

Use NLTK's PorterStemmer in a function

```
def MY_STEMMER(str_input):   #I like dogs a lot111 !!"
    words = re.sub(r"[^A-Za-z\-]", " ", str_input).lower().split()   # I, like, dogs, a
    words = [STEMMER.stem(w) for w in words]
    return words
```

Instantiate CV

```
MyVect_STEM=CountVectorizer(
                        analyzer = 'word',
                        stop_words='english',
                        ##stop_words=["and", "or", "but"],
                        #token_pattern='(?u)[a-zA-Z]+',
                        #token_pattern=pattern,
                        tokenizer=MY_STEMMER,
                        #strip_accents = 'unicode',
                        lowercase = True
                        )


MyVect_STEM_Bern=CountVectorizer(analyzer = 'word',
```

```
                        stop_words='english',
                        ##stop_words=["and", "or", "but"],
                        #token_pattern='(?u)[a-zA-Z]+',
                        #token_pattern=pattern,
                        tokenizer=MY_STEMMER,
                        #strip_accents = 'unicode',
                        lowercase = True,
                        binary=True    #  0 if the word is not in the doc  and a 1 if it is
                        )



MyVect_IFIDF=TfidfVectorizer(analyzer = 'word',
                        stop_words='english',
                        lowercase = True,
                        #binary=True
                        )

MyVect_IFIDF_STEM=TfidfVectorizer(analyzer = 'word',
                        stop_words='english',
                        tokenizer=MY_STEMMER,
                        #strip_accents = 'unicode',
                        lowercase = True,
                        #binary=True
                        )



MyCV_content1 = MyVect_STEM.fit_transform(NewAbstractLIST)
MyCV_content2 = MyVect_STEM_Bern.fit_transform(NewAbstractLIST)
MyCV_content3 = MyVect_IFIDF.fit_transform(NewAbstractLIST)
MyCV_content4 = MyVect_IFIDF_STEM.fit_transform(NewAbstractLIST)
```

Convert DTM to a DF

```
print(MyVect_STEM.vocabulary_)
vocab_dict = MyVect_STEM.vocabulary_
dict_key = vocab_dict.keys()
print("The vocab is: ", dict_key, "\n\n")
```

Use pandas to create data frames

```
Word_count = pd.DataFrame(MyCV_content1.toarray(), columns=dict_key)
Word_count_labeled1 = Word_count
Word_count_labeled1.insert(0, "Label", labels)
# Word_count.to_csv('pubAg_5content_word_count_NB.csv', index=False)

vocab_dict = MyVect_STEM_Bern.vocabulary_
dict_key = vocab_dict.keys()
Word_count = pd.DataFrame(MyCV_content2.toarray(), columns=dict_key)
Word_count_labeled2 = Word_count
Word_count_labeled2.insert(0, "Label", labels)
# Word_count.to_csv('pubAg_5content_word_count_NB.csv', index=False)

vocab_dict = MyVect_IFIDF.vocabulary_
dict_key = vocab_dict.keys()
Word_count = pd.DataFrame(MyCV_content3.toarray(), columns=dict_key)
Word_count_labeled3 = Word_count
Word_count_labeled3.insert(0, "Label", labels)
# Word_count.to_csv('pubAg_5content_word_count_NB.csv', index=False)
```

```
vocab_dict = MyVect_IFIDF_STEM.vocabulary_
dict_key = vocab_dict.keys()
Word_count = pd.DataFrame(MyCV_content4.toarray(), columns=dict_key)
Word_count_labeled4 = Word_count
Word_count_labeled4.insert(0, "Label", labels)
# Word_count.to_csv('pubAg_5content_word_count_NB.csv', index=False)
```

Replace the NaN with 0

```
FinalDF_STEM = Word_count_labeled1.fillna(0)
FinalDF_STEM_Bern = Word_count_labeled2.fillna(0)
FinalDF_TFIDF = Word_count_labeled3.fillna(0)
FinalDF_TFIDF_STEM = Word_count_labeled4.fillna(0)
```

Remove number columns

Create a function that removes columns that are/contain nums

```
def RemoveNums(SomeDF):
    # print(SomeDF)
    print("Running Remove Numbers function....\n")
    temp = SomeDF
    MyList = []
    for col in temp.columns:
        # print(col)
        # Logical1=col.isdigit()  ## is a num
        Logical2 = str.isalpha(col)  ## this checks for anything
        ## that is not a letter
        if (Logical2 == False):  # or Logical2==True):
            # print(col)
            MyList.append(str(col))
            # print(MyList)
    temp.drop(MyList, axis=1, inplace=True)
    # print(temp)
    # return temp

    return temp
```

Call the function

```
FinalDF_STEM = RemoveNums(FinalDF_STEM)
FinalDF_STEM_Bern = RemoveNums(FinalDF_STEM_Bern)
FinalDF_TFIDF = RemoveNums(FinalDF_TFIDF)
FinalDF_TFIDF_STEM = RemoveNums(FinalDF_TFIDF_STEM)
```

Save the files into .csv

```
FinalDF_STEM.to_csv("NB&SVM/FinalDF_STEM.csv")
FinalDF_STEM_Bern.to_csv("NB&SVM/FinalDF_STEM_Bern.csv")
FinalDF_TFIDF.to_csv("NB&SVM/FinalDF_TFIDF.csv")
FinalDF_TFIDF_STEM.to_csv("NB&SVM/FinalDF_TFIDF_STEM.csv")
```

Before we start our modeling, let's visualize and

explore.

##It might be very interesting to see the word clouds

for each of the topics.

```
List_of_WC = []
for mytopic in topic:
    tempdf = Final_Word_count_labeled[Final_Word_count_labeled['Labels'] == mytopic]
    print(tempdf)

    tempdf = tempdf.sum(axis=0, numeric_only=True)
    # print(tempdf)

    # Make var name
    NextVarName = str("wc" + str(mytopic))
    # print( NextVarName)
    ## Create and store in a list the wordcloud OBJECTS
    NextVarName = WordCloud(width=1000, height=600, background_color='#fbeed1',
                            min_word_length=4,
                            max_words=10000).generate_from_frequencies(tempdf)

    ## Here, this list holds all three wordclouds I am building
    List_of_WC.append(NextVarName)

print(List_of_WC)
```

Create the wordclouds

```
fig = plt.figure(figsize=(30, 30))
# figure, axes = plt.subplots(nrows=2, ncols=2)
NumTopics = len(topic)
for i in range(NumTopics):
    print(i)
    ax = fig.add_subplot(NumTopics, 1, i + 1)
    plt.imshow(List_of_WC[i], interpolation='bilinear')
    plt.axis("off")
    plt.savefig("NewClouds.pdf")
```

Naive Bayes

Step1

Create Training and Testing Data

```
## Remove columns that contain "-"
# cols = [c for c in FinalDF_STEM.columns if "-" in c[:] ]
# FinalDF_STEM=FinalDF_STEM.drop(cols, axis = 1)

## Create the testing set
## If train set is large enough, take a random sample.
import random as rd
#rd.seed(1234)

TrainDF1, TestDF1 = train_test_split(FinalDF_STEM, test_size=0.3)
```

```
TrainDF2, TestDF2 = train_test_split(FinalDF_TFIDF, test_size=0.3)
TrainDF3, TestDF3 = train_test_split(FinalDF_TFIDF_STEM, test_size=0.3)
TrainDF4, TestDF4 = train_test_split(FinalDF_STEM_Bern, test_size=0.4)
```

separate the label

```
Test1Labels=TestDF1["Label"]
Test2Labels=TestDF2["Label"]
Test3Labels=TestDF3["Label"]
Test4Labels=TestDF4["Label"]
```

remove labels

```
TestDF1 = TestDF1.drop(["Label"], axis=1)
TestDF2 = TestDF2.drop(["Label"], axis=1)
TestDF3 = TestDF3.drop(["Label"], axis=1)
TestDF4 = TestDF4.drop(["Label"], axis=1)
```

```
Train1Labels=TrainDF1["Label"]
Train2Labels=TrainDF2["Label"]
Train3Labels=TrainDF3["Label"]
Train4Labels=TrainDF4["Label"]

TrainDF1 = TrainDF1.drop(["Label"], axis=1)
TrainDF2 = TrainDF2.drop(["Label"], axis=1)
TrainDF3 = TrainDF3.drop(["Label"], axis=1)
TrainDF4 = TrainDF4.drop(["Label"], axis=1)
```

Step2

Create a function to generate prediction, and heatmap of confusion matrix

```
def get_NBprediction_confusionmatrix_and_heatmap(mapname, TrainDF, TrainLabels, TestDF, TestLabels):
    ModelNB = MultinomialNB()
    NB = ModelNB.fit(TrainDF, TrainLabels)
    Prediction = ModelNB.predict(TestDF)
    print(np.round(ModelNB.predict_proba(TestDF), 2))

    Prediction = ModelNB.predict(TestDF)
    print(np.round(ModelNB.predict_proba(TestDF), 2))
    print("\nThe prediction from NB is:")
    print(Prediction)
    print("\nThe actual labels are:")
    print(TestLabels)

    cnf_matrix = confusion_matrix(TestLabels, Prediction)
    print("\nThe confusion matrix is:")
    print(cnf_matrix)

    matrix_df = pd.DataFrame(cnf_matrix)
    matrix_df.index = topic
    matrix_df.columns = topic
    print(matrix_df)

    sns.set(rc={'axes.facecolor':'#fbeed1', 'figure.facecolor':'#fbeed1'})
    fig = sns.heatmap(matrix_df, cmap="YlGnBu")
    plt.title(mapname, fontname="ITC Officina Sans", fontweight="bold", color="#726abb")
```

```
    plt.xlabel("Prediction Labels", fontname="ITC Officina Sans", fontweight="bold", color="#726abb")
    plt.ylabel("Known Labels", fontname="ITC Officina Sans", fontweight="bold", color="#726abb")
    plt.xticks(fontname="ITC Officina Sans", color="#726abb")
    plt.yticks(fontname="ITC Officina Sans", color="#726abb")


get_NBprediction_confusionmatrix_and_heatmap('Confusion Matrix Heatmap 1 Naive Bayes with Stemmer', TrainDF1, Train1La
get_NBprediction_confusionmatrix_and_heatmap('Confusion Matrix Heatmap 2 Naive Bayes TFIDF', TrainDF2, Train2Labels, T
get_NBprediction_confusionmatrix_and_heatmap('Confusion Matrix Heatmap 3 Naive Bayes TFIDF with Stemmer', TrainDF3, Tr
get_NBprediction_confusionmatrix_and_heatmap('Confusion Matrix Heatmap 4 Naive Bayes with Stemmer Bernoulli', TrainDF4
```

Bernoulli

```
from sklearn.naive_bayes import BernoulliNB
BernModel = BernoulliNB()
BernModel.fit(TrainDF4, Train4Labels)
BernoulliNB(alpha=1.0, binarize=0.0, class_prior=None, fit_prior=True)
print("\nBernoulli prediction:\n")
Prediction=BernModel.predict(TestDF4)
print("\nActual:")
print(Test4Labels)
print("\The prediction\n")

bn_matrix = confusion_matrix(Test4Labels, Prediction)
print("\nThe confusion matrix is:")
print(bn_matrix)

matrix_df = pd.DataFrame(bn_matrix)
matrix_df.index = topic
matrix_df.columns = topic
print(matrix_df)

sns.set(rc={'axes.facecolor': '#fbeed1', 'figure.facecolor': '#fbeed1'})
fig = sns.heatmap(matrix_df, cmap="YlGnBu")
plt.title("Confusion Matrix Heatmap 4 Naive Bayes with Stemmer Bernoulli Model", fontname="ITC Officina Sans", fontwei
plt.xlabel("Prediction Labels", fontname="ITC Officina Sans", fontweight="bold", color="#726abb")
plt.ylabel("Known Labels", fontname="ITC Officina Sans", fontweight="bold", color="#726abb")
plt.xticks(fontname="ITC Officina Sans", color="#726abb")
plt.yticks(fontname="ITC Officina Sans", color="#726abb")
```