Week6: Digital Humanities

Group: Jennifer, Vedasri

In [1]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import time
import re
```

Q1. , it is faster to store the stopwords into a list or a dictionary

Time taken to replace dictionary key to value :0.721041202545166's; Time taken to remove the List of stowords:1.6510944366455078's

As below observation we observed that removing list of stopwords taking more time then replacing dictionary of key to values

Q2. Apply the naïve Bayes to solve the spam detection problem.

In [2]:
```
train = pd.read_csv('corpus/sms_spam.train.csv', delimiter=",")
print("Traing shape: ",train.shape)

test = pd.read_csv('corpus/sms_spam.test.csv', delimiter=",")
print("Test shape  : ",test.shape)
test.head(5)
```

```
Traing shape:  (5199, 2)
Test shape  :  (361, 2)
```

Out[2]:

	type	text
0	ham	happened here while you were adventuring
1	ham	Ask g or iouri, I've told the story like ten t...
2	ham	Sorry, I'll call later
3	spam	Great News! Call FREEFONE 08006344447 to claim...
4	ham	Dont know supports srt i thnk. I think ps3 can...

Class-wise training data

In [3]:
```
train['type'].value_counts(normalize=True)
```

Out[3]:
```
ham     0.866513
spam    0.133487
Name: type, dtype: float64
```

Class-wise training data

In [4]:
```
test['type'].value_counts(normalize=True)
```

Out[4]:
```
ham     0.853186
spam    0.146814
Name: type, dtype: float64
```

Cleaning the data

In [5]:
```
train_punctuation = train.copy()
train_punctuation['text'] = train['text'].str.replace('\W', ' ')

train_lower = train_punctuation.copy()
train_lower['text'] = train_punctuation['text'].str.lower()
train_lower.head(5)
```

```
c:\users\veda\appdata\local\programs\python\python37\lib\site-packages\ipykerne
l_launcher.py:2: FutureWarning: The default value of regex will change from Tru
e to False in a future version.
```

Out[5]:

	type	text
0	ham	hope you are having a good week just checking in
1	ham	k give back my thanks
2	ham	am also doing in cbe only but have to pay
3	spam	complimentary 4 star ibiza holiday or 10 000 ...
4	spam	okmail dear dave this is your final notice to...

In [6]:
```
test_punctuation = test.copy()
test_punctuation['text'] = test['text'].str.replace('\W', ' ')

test_lower = test_punctuation.copy()
test_lower['text'] = test_punctuation['text'].str.lower()
test_lower.head(5)
```

```
c:\users\veda\appdata\local\programs\python\python37\lib\site-packages\ipykerne
l_launcher.py:2: FutureWarning: The default value of regex will change from Tru
e to False in a future version.
```

Out[6]:

	type	text
0	ham	happened here while you were adventuring
1	ham	ask g or iouri i ve told the story like ten t...
2	ham	sorry i ll call later
3	spam	great news call freefone 08006344447 to claim...
4	ham	dont know supports srt i thnk i think ps3 can...

Calculating time taken to replace the dictionary keys to values

In [7]:
```
contractions = {
"ain't": "aim not",
"aren't": "are not",
"can't": "cannot",
"can't've": "cannot have",
"'cause": "because",
"could've": "could have",
"couldn't": "could not",
"couldn't've": "could not have",
"didn't": "did not",
"doesn't": "does not",
"don't": "do not",
"hadn't": "had not",
"hadn't've": "had not have",
"hasn't": "has not",
"haven't": "have not",
"he'd": "he had",
"he'd've": "he would have",
"he'll": "he will",
"he'll've": "he will have",
"he's": "he is",
"how'd": "how did",
"how'd'y": "how do you",
"how'll": "how will",
"how's": "how is",
"i'd": "I had",
"i'd've": "I would have",
"i'll": "I will",
"i'll've": "I will have",
"i'm": "I am",
"i've": "I have",
"isn't": "is not",
"it'd": "it would",
"it'd've": "it would have",
"it'll": "it will",
"it'll've": "it will have",
"it's": "it is",
"let's": "let us",
"ma'am": "madam",
"mayn't": "may not",
"might've": "might have",
"mightn't": "might not",
"mightn't've": "might not have",
"must've": "must have",
"mustn't": "must not",
"mustn't've": "must not have",
"needn't": "need not",
"needn't've": "need not have",
"o'clock": "of the clock",
"oughtn't": "ought not",
"oughtn't've": "ought not have",
"shan't": "shall not",
"sha'n't": "shall not",
"shan't've": "shall not have",
"she'd": "she would",
"she'd've": "she would have",
```

```
"she'll": "she will",
"she'll've": "she will have",
"she's": " she is",
"should've": "should have",
"shouldn't": "should not",
"shouldn't've": "should not have",
"so've": "so have",
"so's": "so as ",
"that'd": "that had",
"that'd've": "that would have",
"that's": "that is",
"there'd": "there would",
"there'd've": "there would have",
"there's": "there is",
"they'd": "they would",
"they'd've": "they would have",
"they'll": "they will",
"they'll've": "they will have",
"they're": "they are",
"they've": "they have",
"to've": "to have",
"wasn't": "was not",
"we'd": " we would",
"we'd've": "we would have",
"we'll": "we will",
"we'll've": "we will have",
"we're": "we are",
"we've": "we have",
"weren't": "were not",
"what'll": "what will",
"what'll've": "what will have",
"what're": "what are",
"what's": "what is",
"what've": "what have",
"when's": "when is",
"when've": "when have",
"where'd": "where did",
"where's": "where is",
"where've": "where have",
"who'll": "who will",
"who'll've": "who will have",
"who's": " who is",
"who've": "who have",
"why's": " why is",
"why've": "why have",
"will've": "will have",
"won't": "will not",
"won't've": "will not have",
"would've": "would have",
"wouldn't": "would not",
"wouldn't've": "would not have",
"y'all": "you all",
"y'all'd": "you all would",
"y'all'd've": "you all would have",
"y'all're": "you all are",
"y'all've": "you all have",
"you'd": "you had",
```

```
"you'd've": "you would have",
"you'll": "you will",
"you'll've": "you will have",
"you're": "you are",
"you've": "you have"
}

#print(contractions.get("you have", "you have"))

train_lower_updated = train_lower.copy()

start = time.time()
contractions_array = []
for i, line in enumerate(train_lower['text']):
    tokens_without_contractions = [contractions.get(word, word) for word in line.

    train_lower_updated['text'][i] = (" ").join(tokens_without_contractions)
end = time.time()
total_dict = end - start
print("Time taken to replace dictionary key to value :{}'s".format(total_dict))

train_lower_updated.head(5)
```

```
Time taken to replace dictionary key to value :0.6940398216247559's
```

Out[7]:

	type	text
0	ham	hope you are having a good week just checking in
1	ham	k give back my thanks
2	ham	am also doing in cbe only but have to pay
3	spam	complimentary 4 star ibiza holiday or 10 000 ...
4	spam	okmail dear dave this is your final notice to...

Calculating top 30 most occurance in ham and 30 spam features.

In [8]:
```
# converting to lower case

train_lower_only_word_tokens = train_lower.copy()

for i,s in enumerate(train_lower['text']):
    only_word_tokens = re.findall("[a-z]+", s, re.I)
    train_lower_only_word_tokens['text'][i] =  (" ").join(only_word_tokens)

train_lower_only_word_tokens.head(5)
```

Out[8]:

	type	text
0	ham	hope you are having a good week just checking in
1	ham	k give back my thanks
2	ham	am also doing in cbe only but have to pay
3	spam	complimentary star ibiza holiday or cash needs...
4	spam	okmail dear dave this is your final notice to ...

Calculating time taken to remove the words from file of list of string stop_words

In [9]:
```
# removing stop_words
import time

stop_words_1 = np.loadtxt('corpus/stopwords.txt', dtype='str')
train_remove_stop_words = train_lower_only_word_tokens.copy()

start = time.time()
for index,sms in enumerate(train_lower_only_word_tokens['text']):
    token_without_sw = [word for word in sms.split(" ") if word not in stop_words
    train_remove_stop_words['text'][index] = (" ").join(token_without_sw)
end = time.time()
total = end - start
print("Time taken to remove the List of stowords:{}'s".format(total))

train_remove_stop_words.head(5)
```

```
Time taken to remove the List of stowords:1.6200926303863525's
```

Out[9]:

	type	text
0	ham	hope good week checking
1	ham	k give back thanks
2	ham	also cbe pay
3	spam	complimentary star ibiza holiday cash needs ur...
4	spam	okmail dear dave final notice collect tenerife...

In [10]:
```
# removing stop words

stop_words_2 = np.loadtxt('corpus/StopwordSMART.txt', dtype='str')
train_remove_stop_words_2 = train_remove_stop_words.copy()

for index, sms in enumerate(train_remove_stop_words['text']):
    token_without_sw = [word for word in sms.split(" ") if  word not in stop_word
    train_remove_stop_words_2['text'][index] = (" ").join(token_without_sw)
#print(stop_words_2)
train_remove_stop_words_2.head(5)
```

Out[10]:

	type	text
0	ham	hope good week checking
1	ham	give back
2	ham	cbe pay
3	spam	complimentary star ibiza holiday cash urgent c...
4	spam	okmail dear dave final notice collect tenerife...

In [11]:
```
ham_tokens = train_remove_stop_words_2.loc[train_remove_stop_words_2['type'] == '
print("Total no. of hams in training set: ",len(ham_tokens))
spam_tokens = train_remove_stop_words_2.loc[train_remove_stop_words_2['type'] ==
print("Total no. of Spams in training set:",len(spam_tokens))
```

```
Total no. of hams in training set:  4505
Total no. of Spams in training set: 694
```

In [12]:
```
# Calculating top 30 ham words

from collections import Counter
words_all = []

for i, words in enumerate(ham_tokens['text']):
    total_words = words.split(" ")
    for w in total_words:
        words_all.append(w)

words_dict = Counter(words_all)
dict_sorted = {k: v for k, v in sorted(words_dict.items(), key=lambda item: item[

#print(dict_sorted)
words_ham_30 = list(dict_sorted.keys())[:30]
print(words_ham_30)
```

```
['ur', 'call', 'good', 'day', 'love', 'time', 'home', 'lor', 'da', 'dont', 'tod
ay', 'back', 'send', 'pls', 'night', 'hey', 'wat', 'dear', 'happy', 'hope', '',
'great', 'give', 'work', 'yeah', 'make', 'im', 'morning', 'phone', 'tomorrow']
```

In [13]:
```
# Calculating top 30 ham words

words_all = []

for i, words in enumerate(spam_tokens['text']):
    total_words = words.split(" ")
    for w in total_words:
        words_all.append(w)

words_dict = Counter(words_all)
dict_sorted = {k: v for k, v in sorted(words_dict.items(), key=lambda item: item[
words_spam_30 = list(dict_sorted.keys())[:30]
print(words_spam_30)
```

```
['call', 'free', 'txt', 'ur', 'stop', 'mobile', 'text', 'claim', 'reply', 'ww
w', 'prize', 'uk', 'send', 'cash', 'win', 'nokia', 'urgent', 'contact', 'msg',
'tone', 'week', 'service', 'box', 'guaranteed', 'customer', 'ppm', 'mins', 'pho
ne', 'cs', 'chat']
```

In [14]:
```
#list(set(words_spam_30).intersection(words_ham_30))
```

Train Naive bays classfier on top 30 ham and top 30 ham words

In [15]:
```
# Creating the vocabulary

train_lower['text'] = train_lower['text'].str.split()

vocabulary = []
for sms in train_lower['text']:
    for word in sms:
        vocabulary.append(word)

vocabulary = list(set(vocabulary))
len(vocabulary)
```

Out[15]: 8384

In [16]:
```
# Creating word counts per test

top_60_features = words_spam_30+ words_ham_30
word_counts_per_sms = {unique_word: [0] * len(train_lower['text']) for unique_wor

#print(train_lower['text'])

for index, sms in enumerate(train_lower['text']):
    for word in sms:
        if word in top_60_features:
            word_counts_per_sms[word][index] += 1
```

In [17]:
```
# Transformation of training set

word_counts = pd.DataFrame(word_counts_per_sms)
word_counts.head()
```

Out[17]:

	call	free	txt	ur	stop	mobile	text	claim	reply	www	...	hope		great	give	work	yeal
0	0	0	0	0	0	0	0	0	0	0	...	1	0	0	0	0	(
1	0	0	0	0	0	0	0	0	0	0	...	0	0	0	1	0	(
2	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	(
3	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	(
4	1	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	(

5 rows × 56 columns

In [18]:
```
# Concatinating Label, text to word-counts

training_set_clean = pd.concat([train_lower['type'], word_counts], axis=1)
training_set_clean['Words'] = train_lower['text']
training_set_clean.head(5)
```

Out[18]:

	type	call	free	txt	ur	stop	mobile	text	claim	reply	...		great	give	work	yeah	mak
0	ham	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	
1	ham	0	0	0	0	0	0	0	0	0	...	0	0	1	0	0	
2	ham	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	
3	spam	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	
4	spam	1	0	0	0	0	0	0	0	0	...	0	0	0	0	0	

5 rows × 58 columns

In [21]:
```
# Isolating spam and ham messages first

spam_messages = training_set_clean[training_set_clean['type'] == 'spam']
ham_messages = training_set_clean[training_set_clean['type'] == 'ham']
spam_messages.head(5)

# P(Spam) and P(Ham)
p_spam = len(spam_messages) / len(training_set_clean)
p_ham = len(ham_messages) / len(training_set_clean)

# N_Spam
n_words_per_spam_message = spam_messages['Words'].apply(len)
n_spam = n_words_per_spam_message.sum()

# N_Ham
n_words_per_ham_message = ham_messages['Words'].apply(len)
n_ham = n_words_per_ham_message.sum()

# N_Vocabulary
n_vocabulary = len(top_60_features)

# Laplace smoothing
alpha = 1
```

In [22]:
```
# Initiate parameters
parameters_spam = {unique_word:0 for unique_word in top_60_features}
parameters_ham = {unique_word:0 for unique_word in top_60_features}

# Calculate parameters
for word in top_60_features:
    n_word_given_spam = spam_messages[word].sum()

    #print( word, n_word_given_spam )
    p_word_given_spam = (n_word_given_spam + alpha) / (n_spam + alpha*n_vocabular
    parameters_spam[word] = p_word_given_spam

    n_word_given_ham = ham_messages[word].sum()
    p_word_given_ham = (n_word_given_ham + alpha) / (n_ham + alpha*n_vocabulary)
    parameters_ham[word] = p_word_given_ham
```

In [23]:
```
def classify(message):
    '''
    message: a string
    '''

    message = re.sub('\W', ' ', message)
    message = message.lower().split()

    p_spam_given_message = np.log(p_spam)
    p_ham_given_message = np.log(p_ham)

    for word in message:
        if word in parameters_spam:
             p_spam_given_message += np.log( parameters_spam[word])

        if word in parameters_ham:
             p_ham_given_message += np.log(parameters_ham[word])

    print('P(Spam|message):', p_spam_given_message)
    print('P(Ham|message):', p_ham_given_message)

    if p_ham_given_message > p_spam_given_message:
        print('Label: Ham')

    elif p_ham_given_message < p_spam_given_message:
        print('Label: Spam')
    else:
        print('Equal proabilities, have a human classify this!')
```

In [24]:
```
def classify_test_set(message):
    '''
    message: a string
    '''

    message = re.sub('\W', ' ', message)
    message = message.lower().split()

    p_spam_given_message = np.log(p_spam)
    p_ham_given_message = np.log(p_ham)

    for word in message:
        if word in parameters_spam:
            p_spam_given_message += np.log( parameters_spam[word])

        if word in parameters_ham:
            p_ham_given_message += np.log( parameters_ham[word])

    if p_ham_given_message > p_spam_given_message:
        return 'ham'
    elif p_spam_given_message > p_ham_given_message:
        return 'spam'
    else:
        return 'needs human classification'
```

In [25]:
```
test_lower['predicted'] = test_lower['text'].apply(classify_test_set)
test_lower.head()
```

Out[25]:

	type	text	predicted
0	ham	happened here while you were adventuring	ham
1	ham	ask g or iouri i ve told the story like ten t...	ham
2	ham	sorry i ll call later	ham
3	spam	great news call freefone 08006344447 to claim...	spam
4	ham	dont know supports srt i thnk i think ps3 can...	ham

Evaluation:

As below we have achieve the accuracy of 95.29% we have removed all punctuations, numbers and spaces. Trained naive bays classifier with top 30 most frequently occurance of ham words and 30 most frequently occurance of spam words.

In [26]:
```
correct = 0
total = test_lower.shape[0]

for row in test_lower.iterrows():
    row = row[1]
    if row['type'] == row['predicted']:
        correct += 1

print('Correct   : {}/{}'.format(correct, total))
print('Incorrect : {}/{}'.format(total-correct, total))
print('Accuracy  : {:.2f}%'.format(100*correct/total))
```

```
Correct   : 344/361
Incorrect : 17/361
Accuracy  : 95.29%
```

Apply the naïve Bayes to solve the authorship attribution problem related to the Federalist Papers (federalist-papersNew2.csv) with the twelve disputed papers.

In [27]:
```
import scipy.special
import itertools

from sklearn import preprocessing
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score, classification_report, confusion_matr
from sklearn.naive_bayes import GaussianNB
```

In [28]:
```
clf= GaussianNB()
le = preprocessing.LabelEncoder()
```

In [29]:
```
# Selecting all the words of Interest.

df = pd.read_csv('corpus/federalist-papersNew2.csv', index_col=0)
words_of_interest = ['upon', 'to', 'would','while','up']
df[words_of_interest].head()
```

Out[29]:

	upon	to	would	while	up
1	6	72	2	0	0
2	1	53	5	1	0
3	0	56	2	0	0
4	0	51	17	0	0
5	0	45	37	0	0

In [30]:
```
# separating the disputed essays

fed = disputed_essays = df[df['AUTHOR'] == 'Hamilton OR Madison'].index
assert len(disputed_essays) == 12  # there are twelve disputed essays
# numbers widely used to identify the essays
assert set(disputed_essays) == {49, 50, 51, 52, 53, 54, 55, 56, 57, 58, 62, 63}
```

In [31]:
```
fed = df_known = df.loc[df['AUTHOR'].isin(('Hamilton', 'Madison'))]
print(df_known['AUTHOR'].value_counts())
fed
```

```
Hamilton    51
Madison     14
Name: AUTHOR, dtype: int64
```

Out[31]:

	000	1	10	100	104	105	109	11	114	115	...	young	your	yourself	yourselves	zaleucu
1	0	2	0	0	0	0	0	0	0	0	...	0	10	0	0	
6	0	2	2	0	0	0	0	2	0	0	...	0	0	0	0	
7	0	1	0	0	0	0	0	0	0	0	...	0	0	0	0	
8	0	2	0	0	0	0	0	0	0	0	...	0	0	0	0	
9	0	1	0	0	0	0	0	0	0	0	...	0	0	0	0	
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	
81	0	2	0	0	0	0	0	0	0	0	...	0	0	0	0	
82	0	1	0	0	0	0	0	0	0	0	...	0	0	0	0	
83	0	2	0	0	0	0	0	0	0	0	...	0	0	0	0	
84	0	4	0	0	0	0	0	0	0	0	...	0	0	0	0	
85	0	2	0	0	0	0	0	0	0	0	...	0	4	0	0	

65 rows × 11501 columns

In [32]:
```
# Convert the 'AUTHOR' to numerical categories using label encoder

known_pap=fed[['upon','would','to','while','up','AUTHOR']]

known_pap['Author_Group']=le.fit_transform(known_pap['AUTHOR'])
known_pap=known_pap.drop('AUTHOR', axis=1)
known_pap
```

```
c:\users\veda\appdata\local\programs\python\python37\lib\site-packages\ipykerne
l_launcher.py:5: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/sta
ble/user_guide/indexing.html#returning-a-view-versus-a-copy (https://pandas.pyd
ata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-c
opy)
  """
```

Out[32]:

	upon	would	to	while	up	Author_Group
1	6	2	72	0	0	0
6	4	6	58	0	0	0
7	11	51	82	0	1	0
8	3	27	80	0	3	0
9	4	8	71	1	0	0
...	...	...	...	...	...	...
81	13	21	163	2	1	0
82	4	11	83	0	0	0
83	20	48	219	4	0	0
84	13	18	140	1	1	0
85	12	6	115	0	1	0

65 rows × 6 columns

In [33]:
```
disputed_essays = df[df['AUTHOR'] == 'Hamilton OR Madison']
disputed_essays.head()
```

Out[33]:

	000	1	10	100	104	105	109	11	114	115	...	young	your	yourself	yourselves	zaleucu
49	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	
50	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	
51	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	
52	0	0	0	0	0	0	0	0	0	0	...	1	0	0	0	
53	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	

5 rows × 11501 columns

In [34]:
```
#Splitting the test dataset for only the 'words_of_interest'

disp_essays=disputed_essays[['upon','would','to','while','up','AUTHOR']]
disp_essays
```

Out[34]:

	upon	would	to	while	up	AUTHOR
49	0	22	58	0	0	Hamilton OR Madison
50	1	11	28	0	0	Hamilton OR Madison
51	0	9	50	0	0	Hamilton OR Madison
52	0	8	72	0	0	Hamilton OR Madison
53	0	6	73	0	0	Hamilton OR Madison
54	2	6	61	0	0	Hamilton OR Madison
55	0	10	78	0	1	Hamilton OR Madison
56	0	4	39	0	0	Hamilton OR Madison
57	0	6	74	0	0	Hamilton OR Madison
58	0	12	61	0	0	Hamilton OR Madison
62	0	5	82	0	0	Hamilton OR Madison
63	0	11	88	0	2	Hamilton OR Madison

In [35]:
```
disp_essay_test=disp_essays.drop('AUTHOR',axis=1)


disp_essay_test.head()
```

Out[35]:

	upon	would	to	while	up
49	0	22	58	0	0
50	1	11	28	0	0
51	0	9	50	0	0
52	0	8	72	0	0
53	0	6	73	0	0

Naive Bayes Model Algorithm:

1. Calculate prior for all catergories of the Output(Hamilton or Madison)

```
--prior(Hamilton) = sum(essays by Hamilton)/ sum(all essays)
--prior(Madison) = sum(essays by Hamilton)/ sum(all essays)
```

2. Calculate the conditional probability for each category for each attribute for all the test samples

```
-- alpha=1 - using Laplace smoothing
-- Prob(word|Hamilton)= sum(word occurence)+ alpha/ (sum (all word
occurence)+ no.of attributes considered)
```

3. Sum the log of conditional probability and prior for each catergory

```
--posterior = log (prior)+ Sum(log(Prob of word|Hamilton)
```

4. Obtain the argmax for the posterior:

```
 -- and append to the y_pred- the chosen category for the selected t
est example.
```

5. Repeat steps for each test sample.

In [36]:
```
def prior(df,Y):
    classes= df[Y].unique()
    prior=[]
    for category in classes:
        prior.append(len(df[df[Y]==category])/len(df))
        #print(prior)
    return prior
def conditional_probability(df,feat_name,feat_val,Y,label):
    feat=list(df.columns)
    df=df[df[Y]==label]
    alpha=1
    num= df[feat_name].sum()+1
    dem=np.sum(df.sum(),axis=0)+feat_val
   # print('prob= ',num/dem)

    return np.log(num/dem)

def naive_bayes(df,X_test,Category):
    features= list(df.columns)[:-1]
    classes=df[Category].unique()
    priors=[]
    priors=prior(df,Category)
    #print(priors)
    Y_pred=[]

    for x in range(len(X_test)):
       # print('value x=  ',x)

        classes=list(df[Category].unique())
        posteriors=[]

        for i in range((len(classes))):
            posterior=0

            cond_prob=0
            for j in range(len(features)):
                count=X_test.iloc[x].loc[features[j]]
               # print(cond_prob,'features :=',features[j], x,count)
                cond_prob+=count*conditional_probability(df,features[j],len(featu
            posterior=np.log(priors[i])+cond_prob

            posteriors.append(posterior)
            #print('posteriors',posteriors)
        Y_pred.append(np.argmax(posteriors))
    return np.array(Y_pred)


df=known_pap
X_test=disp_essay_test
naive_bayes(df,X_test,Category='Author_Group')
```

Out[36]:
```
array([0, 0, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1], dtype=int64)
```

Using the above mentioned algorithm we are unable to obtain 100% accuracy, but 8 out of 11 essays written by Madison.

Using Naive bayes classifier in Scikit:

We can observe that all the disputed essays are written by Madison =1

In [37]:
```
X_train=known_pap[['upon','would','to','while','up']]
Y_train=known_pap['Author_Group']
X_test=disp_essay_test
clf.fit(X_train,Y_train)
y_pred=clf.predict(X_test)
y_pred
```

Out[37]:
```
array([1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1])
```

In [ ]: