Import Modules

In [55]:
```
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split as tts
##
from sklearn.base import BaseEstimator, TransformerMixin
from sklearn.pipeline import FeatureUnion, Pipeline
import nltk
from nltk import tokenize
from collections import Counter
from nltk.corpus import stopwords as sw
##
from sklearn.neighbors import KNeighborsClassifier as knc
from sklearn.ensemble import  RandomForestClassifier as rfc
from sklearn.naive_bayes import GaussianNB as gnb
from sklearn.naive_bayes import ComplementNB as cnb
##
from sklearn.metrics import confusion_matrix as cm
```

Get the Data

After getting the data and using the tokenizer, I decided to save the data in a csv for future use so I do not have to do that waiting again.

For importing original data

In [30]:
```
#############   TRAINING SET  ###################################
#Bring in the X - Values
infile = open('data_train.txt', 'r')
X = {'X' : [x for x in infile]}
infile.close()

#Bring in the Y - Values
infile = open('labels_train_original.txt', 'r')
replace = {'News': 0,
           'Opinion': 1,
           'Classifieds': 2,
           'Features': 3}
Y = {'Y' : [replace[y.rstrip('\n')] for y in infile]}
infile.close()

#Combine them
data = pd.concat([pd.DataFrame(Y), pd.DataFrame(X)], axis = 1)

#############   VALIDATE SET  ###################################
infile = open('data_valid.txt', 'r')
X_test_final = pd.DataFrame({'X' : [x for x in infile]})
infile.close()

infile = open('labels_valid_original.txt', 'r')
Y_test_final = pd.DataFrame({'Y' : [replace[y.rstrip('\n')] for y in infile]})
infile.close()

data_test = pd.concat([Y_test_final, X_test_final], axis = 1)
```

For importing edited data

Note: if importing the already edited data, then there is no reason to go through the transformation pipeline created next

In [4]:
```
#############   TRAINING SET  ####################################
#data = pd.read_csv('data_train_unique_tag.csv')
data = pd.read_csv('data_train_unique_s_tag.csv')
print(data.head(), '\n')

#############   VALIDATE SET  ####################################
#data_test = pd.read_csv('data_valid_unique_tag.csv')
data = pd.read_csv('data_valid_unique_s_tag.csv')
print(data_test.head())
```

```
   Y                                                  X  length  to the  NNS
\
0  0  the sign in front of the steepled church read ...    4282       0  0.2
1  2  lindsey larsen a soprano and samuel ramey the ...    1933       0  0.0
2  1  to the editor sylvia ann hewlett 's book creat...     745       1  0.0
3  0  illinois tool works inc glenview ill a maker o...    1188       0  0.0
4  1  to the editor robert schaeffer op ed feb 19 ex...     859       1  0.0

   VBP   NN  VBZ   VB  VBD  NNP
0  0.5  0.8  0.5  0.0  0.0  0.0
1  0.0  1.0  1.0  0.0  0.0  0.0
2  0.0  1.0  0.0  1.0  0.0  0.0
3  0.0  1.0  0.0  0.0  0.0  0.0
4  1.0  1.0  0.0  0.0  0.0  0.0

   Y                                                  X  length  to the  NNS
\
0  1  to the editor re restructuring for security by...     705       1  0.0
1  1  to the editor in small town gay america op ed ...     778       1  0.0
2  1  don king the boxing promoter has stated that m...    4732       0  0.0
3  1  to the editor bill keller god and george w bus...     794       1  0.0
4  0  andres rios stood in front of il monello and r...    2300       0  0.0

   VBP   NN  VBZ   VB  VBD  NNP
0  0.0  1.0  0.0  0.0  0.0  0.0
1  1.0  1.0  0.0  0.0  0.0  0.0
2  0.0  1.0  0.0  0.0  0.0  0.0
3  1.0  1.0  0.0  0.0  0.0  0.0
4  0.0  1.0  0.0  0.0  0.0  0.0
```

Data Transformation Pipeline

I did three custom transformations for the data 1) got the length of the article 2) 1 or 0 depending on whether the article started with 0 3) got the frequency of noun and verb types for the unique words in an article minus the stop words

classes

In [5]:
```
class length( BaseEstimator, TransformerMixin ):
    """
    This will return the length of the Article
    """
    #Class Constructor
    def __init__( self, get_length = True ):
        self._get_length = get_length

    #Return self nothing else to do here
    def fit( self, X, y = None ):
        return self

    #Method that describes what we need this transformer to do
    def transform( self, X, y = None ):
        if self._get_length:
            X.loc[:,'length'] = X['X'].apply(lambda x: len(x))

        print('Done Getting length\n')
        return X
```

In [6]:
```
class starts( BaseEstimator, TransformerMixin ):
    """
    This will create a column with a 1 if starts with 'to the'
    and 0 if it does not. (opinion articles)
    """
    #Class Constructor
    def __init__( self, to_the = True ):
        self._to_the = to_the

    #Return self nothing else to do here
    def fit( self, X, y = None ):
        return self

    #Method that describes what we need this transformer to do
    def transform( self, X, y = None ):
        if self._to_the:
            X.loc[:,'to the'] = X['X'].apply(lambda x: 1 if x[:6] == 'to the'
else 0)

        print('Done Getting "to the"\n')
        return X
```

In [29]:
```
class NLTK ( BaseEstimator, TransformerMixin ):
    """
    This class takes the articles and looks at the word types of all the uniqu
e words in the
    article. It then looks specifically at the verbs and nouns and creates a f
requency of each
    word type for each article
    """
    #Class Constructor
    def __init__( self, stopwords = None, keys = None ):
        self._stopwords  = stopwords or set(sw.words('english'))
        self._keys = keys

    #Return self nothing else to do here
    def fit( self, X, y = None ):
        return self

    #Method that converts tagged dictionary to frequency of nouns and verbs
    def transform( self, X, y = None ):

        def tokenize_and_tag(X):
            #Get the tokens - tokenize, drop stop words, drop repeat words (mo
stly to reduce time)
            tokens = tokenize.word_tokenize(X)
            key_words = list(set([words for words in tokens if words not in se
lf._stopwords]))
            #key_words = list(set([words for words in key_words if words.isalp
ha()]))
            #Get the tags of each word and count of each of the tags
            for words in key_words:
                tags = nltk.pos_tag(words)
                counts = dict(Counter(tag for word, tag in tags if tag.startsw
ith('N') or tag.startswith('V')))
            return counts
        #Run the definition created for all values in X
        print('Start Tokenizing and Tagging')
        counts = X['X'].apply(tokenize_and_tag)
        print('Done Tokenizing and Tagging\n')

        #for training set, self._keys == None, after, the keys will be
        #set so that the test set has same column names
        print('keys: {}'.format(self._keys))
        if self._keys == None:
            keys = []
            #Go through the articles and collect the uniqe tags in each articl
e
            for i in range(len(X)):
                interest = list(counts[i].keys())
                keys += [x for x in interest if x not in keys]
            self._keys = keys
            print('Got Keys:\n{}\n'.format(self._keys))

        #Create a dictionary to collect frequencies for easy transfer to Panda
s
        nvs = {}
        for i in self._keys:
```

```
            nvs[i] = []
        #Getting the frequencies of each of the Columns in Data
        for i in counts.index:
            #sum of verbs (startswith('V')) and nouns (startswith('N'))
            vs = sum([counts[i][k] for k in counts[i].keys() if k.startswith(
'V')])
            ns = sum([counts[i][k] for k in counts[i].keys() if k.startswith(
'N')])
            for j in nvs.keys():
                #go through column names and if that article has that tag, get
frequency
                if j in counts[i].keys():
                    t = counts[i][j]
                    nvs[j] += [t / vs if j.startswith('V') else t / ns]
                #if not, set frequency = 0
                else:
                    nvs[j] += [0]
        print('Got frequencies')

        return pd.concat([X,pd.DataFrame(nvs)], axis = 1)
```

Pipeline

In [31]:
```
#pipeline - create and transform
pipeline = Pipeline(steps = [('length', length()),
                             ('starts', starts()),
                             ('NLTK', NLTK())])

#Try without eliminating stop words
pipeline.set_params(NLTK__stopwords = [])

data = pipeline.transform(data) # Had to set data = pipeline because
                                # not editing dataframe in NLTK transform
```

```
Done Getting length

Done Getting "to the"

Start Tokenizing and Tagging
Done Tokenizing and Tagging

keys: None
Got Keys:
['NN', 'VBZ', 'VBP', 'VB', 'NNS', 'VBD', 'NNP']

Got frequencies
```

In [32]:
```
data.describe()
```

Out[32]:

	Y	length	to the	NN	VBZ	VBP	
count	2000.000000	2000.000000	2000.000000	2000.000000	2000.000000	2000.000000	2000.0000
mean	1.497500	2278.365000	0.253500	0.938102	0.083750	0.320542	0.1828
std	1.114726	2013.124167	0.435123	0.192258	0.268343	0.452130	0.3757
min	0.000000	47.000000	0.000000	0.000000	0.000000	0.000000	0.0000
25%	0.000000	605.750000	0.000000	1.000000	0.000000	0.000000	0.0000
50%	2.000000	1209.500000	0.000000	1.000000	0.000000	0.000000	0.0000
75%	2.000000	4029.250000	1.000000	1.000000	0.000000	1.000000	0.0000
max	3.000000	7231.000000	1.000000	1.000000	1.000000	1.000000	1.0000

In [33]:
```
#Write training set data to a file for future import
data.to_csv('data_train_unique_s_tag.csv', index = False)
```

Run Different Estimators

Try different base estimators to see which is best. I did this by not even looking at the validate set and split the original Training set to its own Train and test set. I then ran the following Methods for Classifying the data in their base model: 1) kNN 2) Random Forrest 3) Naive Bayes (Gaussian) 4) Naive Bayes (Complement) In the original fits I did three steps: Fit, Get Score, Store

Test Train Split and Score Tracker

In [34]:
```
#Split test and train of training set
X_train, X_test, y_train, y_test = tts(data.loc[:,'length':], data['Y'], strat
ify = data['Y'])
X_train = pd.DataFrame(X_train).reset_index(drop = True)
X_test = pd.DataFrame(X_test).reset_index(drop = True)
y_train = y_train.reset_index(drop = True)
y_test = y_test.reset_index(drop = True)

#Score Tracker
scores = { 'test' : {},
           'train' : {}
    }
```

kNN

In [35]:
```
#kNN (Base) - fit, get scores, store
neigh = knc()
neigh.fit(X_train, y_train)

n_train_score = neigh.score(X_train, y_train)
n_test_score = neigh.score(X_test, y_test)

scores['test']['knn'] = n_test_score
scores['train']['knn'] = n_train_score
```

Random Forrest

In [39]:
```
#RF (Base) - fit, get scores, store
rfc = rfc()
rfc.fit(X_train, y_train)

rfc_train_score = rfc.score(X_train, y_train)
rfc_test_score = rfc.score(X_test, y_test)

scores['test']['rfc'] = rfc_test_score
scores['train']['rfc'] = rfc_train_score
```

```
C:\Users\Dustin\Anaconda3\lib\site-packages\sklearn\ensemble\forest.py:245: F
utureWarning: The default value of n_estimators will change from 10 in versio
n 0.20 to 100 in 0.22.
  "10 in version 0.20 to 100 in 0.22.", FutureWarning)
```

Naive Bayes (Gaussian)

In [40]:
```
#Naive Bayes (Gaussian) - fit, get scores, store
nbg = gnb()
nbg.fit(X_train, y_train)

nbg_train_score = nbg.score(X_train, y_train)
nbg_test_score = nbg.score(X_test, y_test)

scores['test']['nbg'] = nbg_test_score
scores['train']['nbg'] = nbg_train_score
```

Naive Bayes (Complement)

In [41]:
```
#Naive Bayes (Complement) - fit, get scores, store
nbc = cnb()
nbc.fit(X_train, y_train)

nbc_train_score = nbc.score(X_train, y_train)
nbc_test_score = nbc.score(X_test, y_test)

scores['test']['nbc'] = nbc_test_score
scores['train']['nbc'] = nbc_train_score
```

Print Scores

In [42]:
```
print(pd.DataFrame(scores))
```

```
      test     train
knn  0.362  0.570000
nbc  0.318  0.381333
nbg  0.440  0.460667
rfc  0.432  0.933333
```

At this point, I feel there are a couple of options: 1) try and play with a kNN or Random Forrest Model 2) Just go with the best test Score Time has me starting with 2 and then will do 1 if able.

Compare Training and Validation Data (step 2)

This is done by transforming the test data the same training data was transformed. Then fitting the chosen algorithm to the full training set and then making prediction with the test set. Finally look at the confusion matrix for comparison.

In [43]:
```
#Transform Validate data
data_test = pipeline.transform(data_test)

#Split into X and Y
X_test_final = data_test.loc[:, 'length':]
Y_test_final = data_test['Y']
```

```
Done Getting length

Done Getting "to the"

Start Tokenizing and Tagging
Done Tokenizing and Tagging

keys: ['NN', 'VBZ', 'VBP', 'VB', 'NNS', 'VBD', 'NNP']
Got frequencies
```

In [44]:
```
data_test.to_csv('data_valid_unique_s_tag.csv', index = False)
data_test.describe()
```

Out[44]:

	Y	length	to the	NN	VBZ	VBP	
count	2000.000000	2000.000000	2000.000000	2000.000000	2000.000000	2000.000000	2000.0000
mean	1.490000	2232.154500	0.254000	0.940164	0.084750	0.326542	0.1907
std	1.128511	1979.238558	0.435406	0.188066	0.268288	0.455184	0.3838
min	0.000000	56.000000	0.000000	0.000000	0.000000	0.000000	0.0000
25%	0.000000	611.000000	0.000000	1.000000	0.000000	0.000000	0.0000
50%	1.000000	1132.000000	0.000000	1.000000	0.000000	0.000000	0.0000
75%	3.000000	4025.500000	1.000000	1.000000	0.000000	1.000000	0.0000
max	3.000000	7132.000000	1.000000	1.000000	1.000000	1.000000	1.0000

In [45]:
```
#training set
X_train_final = data.loc[:,'length':]
y_train_final = data['Y']
```

In [46]:
```
#fit the desired Algorithm
nbg = gnb()
nbg.fit(X_train_final, y_train_final)

#Predict
Y_test_predict = nbg.predict(X_test_final)
```

In [51]:
```
#Confusion Matrix - Standard Confusion, Percent Confusion
confusion = cm(Y_test_final, Y_test_predict)
fun = lambda x: x/sum(x)
cm_perc = np.apply_along_axis(fun, 1, confusion)

#Get the global Accuracy
global_acc = sum(cm_perc.diagonal())/4

print('{}\n\n{}\n\nGlobal Accuracy of {:.2f}'.format(confusion, cm_perc, globa
l_acc))
```

```
[[  2  34 476   0]
 [  7 391 109   0]
 [  1   7 461   1]
 [  2  83 426   0]]

[[0.00390625 0.06640625 0.9296875  0.        ]
 [0.01380671 0.77120316 0.21499014 0.        ]
 [0.00212766 0.01489362 0.98085106 0.00212766]
 [0.00391389 0.16242661 0.83365949 0.        ]]

Global Accuracy of 0.44
```

It did not classify anything as an article or feature. Not very helpful.

Random Forrest parameters (step 1)

In [63]:
```
#Keep track of the scores
rf_scores1 = {
    'test' : {},
    'train' : {}
}
```

In [64]:
```
#Play with min samples split
min_samples_split = [11, 12, 13, 14]

for i in min_samples_split:
    from sklearn.ensemble import  RandomForestClassifier as rfc
    #fit
    rfc = rfc(min_samples_split = i).fit(X_train, y_train)
    #Scores
    rfc_train_score = rfc.score(X_train, y_train)
    rfc_test_score = rfc.score(X_test, y_test)
    #Store
    rf_scores1['test']['{}'.format(i)] = rfc_test_score
    rf_scores1['train']['{}'.format(i)] = rfc_train_score
```

```
C:\Users\Dustin\Anaconda3\lib\site-packages\sklearn\ensemble\forest.py:245: F
utureWarning: The default value of n_estimators will change from 10 in versio
n 0.20 to 100 in 0.22.
  "10 in version 0.20 to 100 in 0.22.", FutureWarning)
C:\Users\Dustin\Anaconda3\lib\site-packages\sklearn\ensemble\forest.py:245: F
utureWarning: The default value of n_estimators will change from 10 in versio
n 0.20 to 100 in 0.22.
  "10 in version 0.20 to 100 in 0.22.", FutureWarning)
C:\Users\Dustin\Anaconda3\lib\site-packages\sklearn\ensemble\forest.py:245: F
utureWarning: The default value of n_estimators will change from 10 in versio
n 0.20 to 100 in 0.22.
  "10 in version 0.20 to 100 in 0.22.", FutureWarning)
C:\Users\Dustin\Anaconda3\lib\site-packages\sklearn\ensemble\forest.py:245: F
utureWarning: The default value of n_estimators will change from 10 in versio
n 0.20 to 100 in 0.22.
  "10 in version 0.20 to 100 in 0.22.", FutureWarning)
```

In [65]:
```
print(pd.DataFrame(rf_scores1))
```

```
     test     train
11  0.472  0.706000
12  0.468  0.682667
13  0.492  0.682667
14  0.460  0.676000
```

In [73]:
```
rf_scores2 = {
    'test' : {},
    'train' : {}
}
```

In [74]:
```
#Play with min samples leaf
min_samples_leaf = [15, 17, 19, 21, 23]

for i in min_samples_leaf:
    from sklearn.ensemble import  RandomForestClassifier as rfc
    #fit
    rfc = rfc(min_samples_split = 13, min_samples_leaf = i).fit(X_train, y_tra
in)
    #Scores
    rfc_train_score = rfc.score(X_train, y_train)
    rfc_test_score = rfc.score(X_test, y_test)
    #Store
    rf_scores2['test']['{}'.format(i)] = rfc_test_score
    rf_scores2['train']['{}'.format(i)] = rfc_train_score
```

```
C:\Users\Dustin\Anaconda3\lib\site-packages\sklearn\ensemble\forest.py:245: F
utureWarning: The default value of n_estimators will change from 10 in versio
n 0.20 to 100 in 0.22.
  "10 in version 0.20 to 100 in 0.22.", FutureWarning)
C:\Users\Dustin\Anaconda3\lib\site-packages\sklearn\ensemble\forest.py:245: F
utureWarning: The default value of n_estimators will change from 10 in versio
n 0.20 to 100 in 0.22.
  "10 in version 0.20 to 100 in 0.22.", FutureWarning)
C:\Users\Dustin\Anaconda3\lib\site-packages\sklearn\ensemble\forest.py:245: F
utureWarning: The default value of n_estimators will change from 10 in versio
n 0.20 to 100 in 0.22.
  "10 in version 0.20 to 100 in 0.22.", FutureWarning)
C:\Users\Dustin\Anaconda3\lib\site-packages\sklearn\ensemble\forest.py:245: F
utureWarning: The default value of n_estimators will change from 10 in versio
n 0.20 to 100 in 0.22.
  "10 in version 0.20 to 100 in 0.22.", FutureWarning)
C:\Users\Dustin\Anaconda3\lib\site-packages\sklearn\ensemble\forest.py:245: F
utureWarning: The default value of n_estimators will change from 10 in versio
n 0.20 to 100 in 0.22.
  "10 in version 0.20 to 100 in 0.22.", FutureWarning)
```

In [75]:
```
print(pd.DataFrame(rf_scores2))
```

```
     test     train
15  0.470  0.550000
17  0.480  0.532667
19  0.474  0.545333
21  0.492  0.534667
23  0.482  0.528667
```

Using the following values for parameters, I will run a final estimator and look at the results from a confusion matrix. 1) min_leaf_split = 13 1) min_leaf_samples = 21

In [77]:
```
from sklearn.ensemble import  RandomForestClassifier as rfc
#fit
rfc = rfc(min_samples_split = 13, min_samples_leaf = 21).fit(X_train_final, y_
train_final)
#Predict
Y_test_predict_rfc = rfc.predict(X_test_final)
```

```
C:\Users\Dustin\Anaconda3\lib\site-packages\sklearn\ensemble\forest.py:245: F
utureWarning: The default value of n_estimators will change from 10 in versio
n 0.20 to 100 in 0.22.
  "10 in version 0.20 to 100 in 0.22.", FutureWarning)
```

In [79]:
```
#Confusion Matrix - Standard Confusion, Percent Confusion
confusion2 = cm(Y_test_final, Y_test_predict_rfc)
fun = lambda x: x/sum(x)
cm_perc2 = np.apply_along_axis(fun, 1, confusion2)

#Get the global Accuracy
global_acc2 = sum(cm_perc2.diagonal())/4

print('{}\n\n{}\n\nGlobal Accuracy of {:.2f}'.format(confusion2, cm_perc2, glo
bal_acc2))
```

```
[[177  28 215  92]
 [ 51 397  12  47]
 [107   3 282  78]
 [125  80 198 108]]

[[0.34570312 0.0546875  0.41992188 0.1796875 ]
 [0.10059172 0.78303748 0.02366864 0.09270217]
 [0.22765957 0.00638298 0.6        0.16595745]
 [0.2446184  0.15655577 0.38747554 0.21135029]]

Global Accuracy of 0.49
```

Test Train Split