04_deception_MNB_hints

October 27, 2019

In [1]:
```
"""
"""
#IST 736 - Text Mining
#Homework Hints 4
#Multinomial Naive Bayes

#feature tables
#Counclusion
# %%
#################
## Reading in and vectorizing
## various formats for text data
##
## This example shows what to do with
## a very poorly formatted and dirty
## csv file.
##
## Here is the name of the original
## file: deception_data_converted_final.csv
########################################

## Textmining Naive Bayes Example
import pandas as pd
import numpy as np
import re
from nltk.corpus import stopwords
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn import preprocessing
from sklearn.model_selection import KFold
from sklearn.naive_bayes import MultinomialNB
from sklearn.metrics import confusion_matrix, accuracy_score, precision_recall_fscore_s
import matplotlib.pyplot as plt
import copy
from wordcloud import WordCloud
```

In [2]:
```
 # Here is some code to create a confusion matrix ...
#Thanks to https://scikit-learn.org/stable/auto_examples/model_selection/plot_confusio
```

```python
def plot_confusion_matrix(y_true, y_pred, classes,
                          normalize=False,
                          title=None,
                          cmap=plt.cm.Blues):
    """
    This function prints and plots the confusion matrix.
    Normalization can be applied by setting `normalize=True`.
    """
    if not title:
        if normalize:
            title = 'Normalized confusion matrix'
        else:
            title = 'Confusion matrix, without normalization'

    # Compute confusion matrix
    cm = confusion_matrix(y_true, y_pred)
    # Only use the labels that appear in the data
    #classes = classes[unique_labels(y_true, y_pred)]
    if normalize:
        cm = cm.astype('float') / cm.sum(axis=1)[:, np.newaxis]

    fig, ax = plt.subplots()
    im = ax.imshow(cm, interpolation='nearest', cmap=cmap)
    ax.figure.colorbar(im, ax=ax)
    # We want to show all ticks...
    ax.set(xticks=np.arange(cm.shape[1]),
           yticks=np.arange(cm.shape[0]),
           # ... and label them with the respective list entries
           xticklabels=classes, yticklabels=classes,
           title=title,
           ylabel='True label',
           xlabel='Predicted label')

    # Rotate the tick labels and set their alignment.
    plt.setp(ax.get_xticklabels(), rotation=45, ha="right",
             rotation_mode="anchor")

    # Loop over data dimensions and create text annotations.
    fmt = '.2f' if normalize else 'd'
    thresh = cm.max() / 2.
    for i in range(cm.shape[0]):
        for j in range(cm.shape[1]):
            ax.text(j, i, format(cm[i, j], fmt),
                    ha="center", va="center",
                    color="white" if cm[i, j] > thresh else "black")
    fig.tight_layout()
    return ax
# %%
```

```
In [3]: ##################################

        ## Step 1: Read in the file
        ## We cannot read it in as csv because it is a mess
        ## One option is to convert it to text.

        RawfileName="deception_data_converted_final.csv"
        FILE=open(RawfileName,"r")

        ## We are going to clean it and then write it back to csv!
        ## So, we need an empty csv file - let's make one....
        filename="CleanText.csv"
        NEWFILE=open(filename,"w")
        ## In the first row, create a column called Label and a column Text...
        ToWrite="Lie_Label,Senti_Label,Review\n"
        ## Write this to new empty cs v file
        NEWFILE.write(ToWrite)
        ## Close it up
        NEWFILE.close()

In [4]: ### Now, we have an empty csv file called CLeanText.csv
        ### Above we created the first row of column names: Label and Review
        ### Next, we will open this file for "a" or append - so we can
        ### add things to it from where we left off
        ### NOTE: If you open this file again with "w" it will write over
        ### whatever is in the file!  USE "a"....
        ### This line of code opens the file for append and creates
        ### a variable (NEWFILE) that we can use to access and control the
        ### file.
        NEWFILE=open(filename, "a")

        ### We also will build a CLEAN dataframe.
        ### So for now, we need a blank one...
        MyFinalDF=pd.DataFrame()

        ###############################
        ## IMPORTANT
        ##
        ## Below, we will create a lot of
        ## prints and outputs that we want to see
        ## Let's write them all to a file so
        ## we can see what our code is doing
        ###################################
        OutputFile="MyOutputFile.txt"
        ## There are many ways to do this...
        ## I prefer to open the file with "w" to
        ## create it. Then, close and reopen with "a" to
        ## write to it.
```

```
## You can also use with open, etc
OUTFILE=open(OutputFile,"w")
OUTFILE.close()
OUTFILE=open(OutputFile,"a") ### REMEMBER to close this below....
```

In [5]:
```
###
### Let's go through it one row at a time....

# %%
#Get the nltk.corpus stopwords ready
nltkstopwords = stopwords.words('english')
nltkstopwords = [w for w in nltkstopwords if not w == 't']
nltkstopwords = nltkstopwords + [w.replace("'", "") for w in nltkstopwords]

# %%
#Get the lengths while reading and tokenizing
MyLength = []
MyLengthOrig = []
for row in FILE:
    RawRow="\n\nThe row is: " + row +"\n"
    OUTFILE.write(RawRow) ## I am going to write this later again for comp
    row=row.lstrip()  ## strip all spaces from the left
    row=row.rstrip()  ## strip all spaces from the right
    row=row.strip()   ## strip all extra spaces in general
    row=row.replace(","," ")
    #print(row)
    ## Split up the row of text by space - TOKENIZE IT into a LIST
    Mylist=row.split(" ")
    #Use OrigList to get the original length
    OrigList = Mylist
    #print(Mylist)
    ## Now, we will clean this list (row)
    ## We will place the results (cleaned) into a new list
    ## Therefore, we need to build a new empty list...
    NewList=[]

    for word in Mylist:
        #print("The next word is: ", word)
        PlaceInOutputFile = "The next word BEFORE is: " +  word + "\n"
        OUTFILE.write(PlaceInOutputFile)
        word=word.lower()
        word=word.lstrip()
        #word=word.strip("\n")
        #word=word.strip("\\n")
        word=word.replace(","," ")
        #for good measure, take out stopwords before and after general tokenization
        if word in nltkstopwords:
            word = ''
```

```
    #Replace any string that occurs more than two times in a row with that string
    word = re.sub(r"(.)\1{2,}", "\\1", word, 1)
    word=word.replace(" ","")
    word=word.replace("_","")
    word=re.sub('\+', ' ',word)
    word=re.sub('.*\+\n', '',word)
    word=word.replace("\t","")
    word=word.replace("\r", "")
    word=word.replace(".","")
    word=word.replace("'", "")
    word=word.replace("(", "")
    word=word.replace(")", "")
    #word=word.replace("\'s","")
    word=word.lstrip()
    word=word.rstrip()
    word=word.strip()

    if word in nltkstopwords:
        word = ''

    #word.replace("\","")
    if word not in ["", "\\", '"', "'", "*", ":", ";"]:
        if len(word) >= 1:
            if not re.search(r'\d', word): ##remove digits
                NewList.append(word)
                PlaceInOutputFile = "The next word AFTER is: " +  word + "\n"
                OUTFILE.write(PlaceInOutputFile)

#print(NewList)

#print(NewList[-1])  ## what is this??? Its the last element
## What is the last element?? Its the label!

NewList = [w for w in NewList if w not in nltkstopwords]

   ## Labels for our data set <------------!!!!!!!!!!!!!!!!!!!!!!!
llabel=NewList[0]
if llabel == "f":
    llabel="truth"
elif llabel == "t":
    llabel="lie"
else:
    llabel="NEITHER f or t"

slabel=NewList[1]
if slabel == "n":
    slabel="neg"
```

```
elif slabel == "p":
    slabel="pos"
else:
    slabel="NEITHER n or p"
## -------------------------------------------------------------------
PlaceInOutputFile = "\nThe label is: " +  llabel + "  "+ slabel +"\n"
OUTFILE.write(PlaceInOutputFile)

NewList.pop(0) ## removes first item
NewList.pop(0) ## removes first item

Text=" ".join(NewList)

#PlaceInOutputFile = "\nThe text  is: " +  Text + "\n"
#OUTFILE.write(PlaceInOutputFile)
#print(Text)

#print("LABEL\n")
#print(label)

### More cleaning....
Text=Text.replace("\\n","")
Text=Text.strip("\\n")
Text=Text.replace("\\'","")
Text=Text.replace("\\","")
Text=Text.replace('"',"")
Text=Text.replace("'","")
Text=Text.replace("s'","")
Text=Text.lstrip()

#if len(Text) < 2:
 #   print("SMALL",Text)
#print(type(Text))
#print(Text)

LastStopWords=Text.split(" ")
LastStopWords = [w for w in LastStopWords if w not in nltkstopwords]
Text = " ".join(LastStopWords)

## Create the string you want to write to the NEWFILE...
OriginalRow="ORIGINAL" + RawRow
OUTFILE.write(OriginalRow)
ToWrite=llabel+","+slabel+","+Text+"\n"

if "NEITHER" not in ToWrite:
    #Let's get the lengths of each tokenized review while we're here.
    MyLength.append(len(NewList))
    #Same for the original. Subtract two for the two labels
```

```
                MyLengthOrig.append(len(OrigList) - 2)
                NEWFILE.write(ToWrite)
                OUTFILE.write(ToWrite)

        ## CLOSE files - always close files!
        FILE.close()
        NEWFILE.close()
        OUTFILE.close()
```

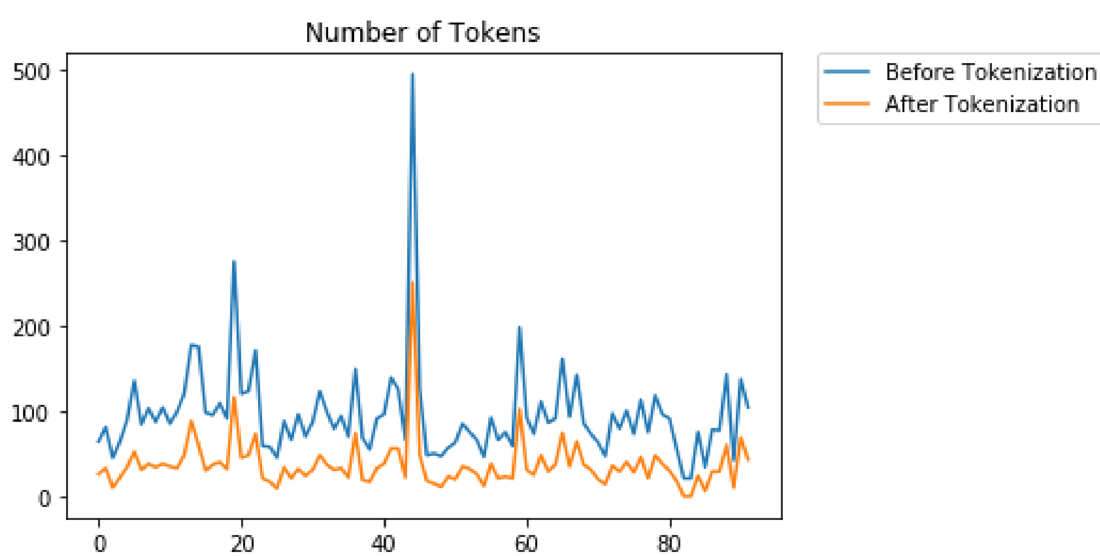
```
In [6]: ###########
        ## Read the new csv file you created into a DF or into CounterVectorizer
        #######
        ## recall that filename is CleanFile.csv - the file we just made
        ## Into DF
        MyTextDF=pd.read_csv(filename)
        ## remove any rows with NA
        MyTextDF = MyTextDF.dropna(how='any',axis=0)  ## axis 0 is rowwise
        print(MyTextDF.head())
        #print(MyTextDF["Label"])
        #print(MyTextDF.iloc[1,1])
```

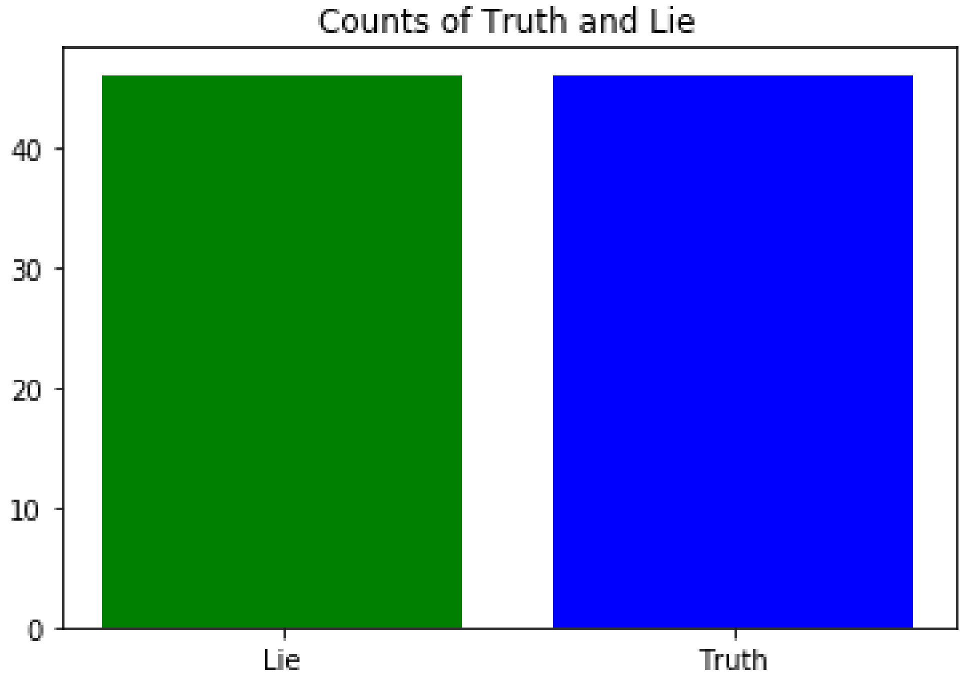
```
  Lie_Label Senti_Label                                             Review
0     truth         neg  mikes pizza high point ny service slow quality...
1     truth         neg  really like buffet restaurant marshall street ...
2     truth         neg  went shopping friend went dodo restaurant dinn...
3     truth         neg  olive oil garden disappointing expect good foo...
4     truth         neg  seven heaven restaurant never known superior s...
```

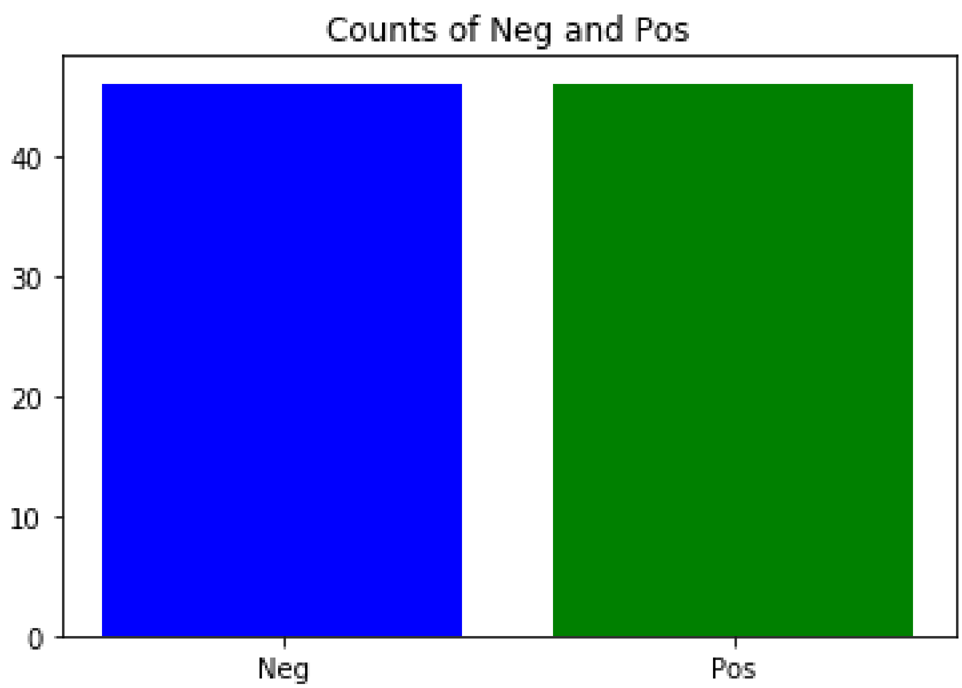
```
In [7]: ## KEEP THE LABELS!
        MyLieLabel = MyTextDF["Lie_Label"]
        MySentiLabel = MyTextDF["Senti_Label"]
        ## Remove the labels from the DF
        DF_noLabel= MyTextDF.drop(["Lie_Label"], axis=1)  #axis 1 is column
        DF_noLabel= DF_noLabel.drop(["Senti_Label"], axis=1)
        #print(DF_noLabel.head())
        ## Create a list where each element in the list is a row from
        ## the file/DF
        print(DF_noLabel.head())
        print("length: ", len(DF_noLabel))
```

```
                                              Review
0  mikes pizza high point ny service slow quality...
1  really like buffet restaurant marshall street ...
2  went shopping friend went dodo restaurant dinn...
3  olive oil garden disappointing expect good foo...
4  seven heaven restaurant never known superior s...
length:  92
```

In [8]:
```
# %% EDA

#Word count before and after.
dat = {'x': np.arange(len(MyLength)),
       'y1': MyLengthOrig,
       'y2': MyLength}

plt.plot(dat['x'], dat['y1'], label = 'Before Tokenization')
plt.plot(dat['x'], dat['y2'], label = 'After Tokenization')
plt.title('Number of Tokens')
plt.legend(bbox_to_anchor=(1.05, 1), loc=2, borderaxespad=0.)
plt.show()

#pos/neg & lie/truth ratios
dat = [len(MyTextDF[(MyTextDF["Lie_Label"] == "truth")]),
       len(MyTextDF[(MyTextDF["Lie_Label"] == "lie")]),
       len(MyTextDF[(MyTextDF["Senti_Label"] == "neg")]),
       len(MyTextDF[(MyTextDF["Senti_Label"] == "pos")])]

plt.bar(x = ['Truth', 'Lie'], height = dat[:1], color = ['b', 'g'])
plt.title('Counts of Truth and Lie')
plt.show()

plt.bar(x = ['Neg', 'Pos'], height = dat[2:], color = ['b', 'g'])
plt.title('Counts of Neg and Pos')
plt.show()

#Percent change in lengths
pctChange = [(MyLengthOrig[i] - MyLength[i])/MyLengthOrig[i] for i in range(len(MyLeng
print(np.mean(pctChange))
```




Counts of Truth and Lie
0
10
20
30
40
Lie
Truth


Counts of Neg and Pos
0
10
20
30
40
Neg
Pos

```
0.6314074158299527
```

```
In [9]: #Awesome. We have a clean dataset with all of the lower case, cleaned, tokenized revie
        #We also have the length of each tokenized review in MyLength
        print(DF_noLabel.head())
        print(MyLieLabel.head())
        print(MySentiLabel.head())
```

```
                                              Review
0  mikes pizza high point ny service slow quality...
1  really like buffet restaurant marshall street ...
2  went shopping friend went dodo restaurant dinn...
3  olive oil garden disappointing expect good foo...
4  seven heaven restaurant never known superior s...
0    truth
1    truth
2    truth
3    truth
4    truth
Name: Lie_Label, dtype: object
0    neg
1    neg
2    neg
3    neg
4    neg
Name: Senti_Label, dtype: object
```

```
In [10]: #We will now get the datasets we want to experiment with. (all will be vectorized and
         #Counts
         #Normalized Counts
         #tfidf
         #Standardized Counts
         #Standardized Normalized Counts
         #Standardized tfidf

         #Standard name for vectorizers, fits, and DFs are as follows:
         #Vectorizer: vect<Type>Orig
         #Fit: vect<Type>OrigFit
         #DF: vect<Type><Orig|Stand>DF
         #DF with label: vect<Type><Orig|Stand>DF<Lie|Senti>
         # %%
         #Define function to put labels back on. df defaults to DF_noLabel, lab as string
         def add_labels(df, lab = ''):
             tmp = copy.deepcopy(df)
             if lab == "Lie LABEL":
                 tmp[lab] = MyLieLabel
```

```
        elif lab == "Senti LABEL":
            tmp[lab] = MySentiLabel
        else:
            print('Use either Lie LABEL or Senti LABEL')
            return()
        return(tmp)

#Use: #new = add_labels(df = original, lab = 'Lie LABEL')

# %%

### BUILD the LIST that "content" in all vectorizers will expect

MyList=[]  #empty list
for i in range(0,len(DF_noLabel)):
    NextText=DF_noLabel.iloc[i,0]  ## what is this??
    ## PRINT TO FIND OUT!
    #print(MyTextDF.iloc[i,1])
    #print("Review #", i, "is: ", NextText, "\n\n")
    #print(type(NextText))
    ## This list is a collection of all the reviews. It will be HUGE
    MyList.append(NextText)

## see what this list looks like....
print(MyList[0:4])
```

```
['mikes pizza high point ny service slow quality low would think would know least make good pi
```

In [11]:

```
# %% COUNT VECTORIZER

vectCountOrig = CountVectorizer(input="content")

vectCountOrigFit = vectCountOrig.fit_transform(MyList)

MyColumnNames=vectCountOrig.get_feature_names()
#We can use MyColumnNames over and over again. As long as we keep using MyList
vectCountOrigDF=pd.DataFrame(vectCountOrigFit.toarray(), columns = MyColumnNames)
print(vectCountOrigDF.head(10))
vectCountOrigDFLie = add_labels(vectCountOrigDF, 'Lie LABEL')
vectCountOrigDFSenti = add_labels(vectCountOrigDF, 'Senti LABEL')
print(vectCountOrigDFLie.head(10))
print(vectCountOrigDFSenti.head(10))
```

```
   abc  abruptly  absolutely  acceptable  accord  acknowledge  across  actual  \
0    0         0           0           0       0            0       0       0
1    0         0           0           0       0            0       0       0
2    0         0           0           0       0            0       0       0
```

```
3    0         0           0           0       0            0       0       0
4    0         0           0           0       0            0       0       0
5    0         0           0           0       0            1       0       0
6    1         0           0           0       0            0       0       0
7    0         0           0           0       0            0       0       0
8    0         0           0           0       0            0       0       0
9    0         0           0           0       0            0       0       0

   actually  ad   ...    written  wrong  wrote  xyz  yeah  yelp  yesterday  \
0         0   0   ...          0      0      0    0     0     0          0
1         0   0   ...          0      0      0    0     0     0          0
2         0   0   ...          0      0      0    0     0     0          0
3         0   0   ...          0      0      0    0     0     0          0
4         0   0   ...          0      0      0    0     0     0          0
5         0   0   ...          0      0      0    1     0     1          0
6         0   0   ...          0      0      0    0     0     0          0
7         0   0   ...          0      1      0    0     0     0          0
8         0   0   ...          0      0      0    0     0     0          0
9         0   0   ...          0      0      0    0     1     0          1

   york  youll  yuenan
0     0      0       0
1     0      0       0
2     0      0       0
3     0      0       0
4     0      0       0
5     0      0       0
6     0      0       0
7     0      0       0
8     0      0       0
9     0      0       0

[10 rows x 1343 columns]
   abc  abruptly  absolutely  acceptable  accord  acknowledge  across  actual  \
0    0         0           0           0       0            0       0       0
1    0         0           0           0       0            0       0       0
2    0         0           0           0       0            0       0       0
3    0         0           0           0       0            0       0       0
4    0         0           0           0       0            0       0       0
5    0         0           0           0       0            1       0       0
6    1         0           0           0       0            0       0       0
7    0         0           0           0       0            0       0       0
8    0         0           0           0       0            0       0       0
9    0         0           0           0       0            0       0       0

   actually  ad    ...      wrong  wrote  xyz  yeah  yelp  yesterday  york  \
0         0   0    ...          0      0    0     0     0          0     0
1         0   0    ...          0      0    0     0     0          0     0
```

```
2          0   0    ...           0       0    0     0     0           0     0
3          0   0    ...           0       0    0     0     0           0     0
4          0   0    ...           0       0    0     0     0           0     0
5          0   0    ...           0       0    1     0     1           0     0
6          0   0    ...           0       0    0     0     0           0     0
7          0   0    ...           1       0    0     0     0           0     0
8          0   0    ...           0       0    0     0     0           0     0
9          0   0    ...           0       0    0     1     0           1     0

   youll  yuenan  Lie LABEL
0      0       0      truth
1      0       0      truth
2      0       0      truth
3      0       0      truth
4      0       0      truth
5      0       0      truth
6      0       0      truth
7      0       0      truth
8      0       0      truth
9      0       0      truth

[10 rows x 1344 columns]
   abc  abruptly  absolutely  acceptable  accord  acknowledge  across  actual  \
0    0         0           0           0       0            0       0       0
1    0         0           0           0       0            0       0       0
2    0         0           0           0       0            0       0       0
3    0         0           0           0       0            0       0       0
4    0         0           0           0       0            0       0       0
5    0         0           0           0       0            1       0       0
6    1         0           0           0       0            0       0       0
7    0         0           0           0       0            0       0       0
8    0         0           0           0       0            0       0       0
9    0         0           0           0       0            0       0       0

   actually  ad     ...       wrong  wrote  xyz  yeah  yelp  yesterday  york  \
0         0   0     ...           0      0    0     0     0          0     0
1         0   0     ...           0      0    0     0     0          0     0
2         0   0     ...           0      0    0     0     0          0     0
3         0   0     ...           0      0    0     0     0          0     0
4         0   0     ...           0      0    0     0     0          0     0
5         0   0     ...           0      0    1     0     1          0     0
6         0   0     ...           0      0    0     0     0          0     0
7         0   0     ...           1      0    0     0     0          0     0
8         0   0     ...           0      0    0     0     0          0     0
9         0   0     ...           0      0    0     1     0          1     0

   youll  yuenan  Senti LABEL
0      0       0          neg
```

```
1       0       0          neg
2       0       0          neg
3       0       0          neg
4       0       0          neg
5       0       0          neg
6       0       0          neg
7       0       0          neg
8       0       0          neg
9       0       0          neg

[10 rows x 1344 columns]
```

```
In [12]: #%% NORMALIZED COUNT VECTORIZER
         #We will normalize based on MyLength which is the number of tokens

         vectCountNormOrigDF = copy.deepcopy(vectCountOrigDF)
         vectCountNormOrigDF["_length"] = MyLength
         for col in MyColumnNames:
             vectCountNormOrigDF[col]= vectCountNormOrigDF[col] / vectCountNormOrigDF._length
         vectCountNormOrigDF = vectCountNormOrigDF.drop('_length', axis = 1)
         print(vectCountNormOrigDF.head(10))
         #Nice!
         vectCountNormOrigDFLie = add_labels(vectCountNormOrigDF, 'Lie LABEL')
         vectCountNormOrigDFSenti = add_labels(vectCountNormOrigDF, 'Senti LABEL')
```

```
       abc  abruptly  absolutely  acceptable  accord  acknowledge  across  \
0  0.00000       0.0         0.0         0.0     0.0     0.000000     0.0
1  0.00000       0.0         0.0         0.0     0.0     0.000000     0.0
2  0.00000       0.0         0.0         0.0     0.0     0.000000     0.0
3  0.00000       0.0         0.0         0.0     0.0     0.000000     0.0
4  0.00000       0.0         0.0         0.0     0.0     0.000000     0.0
5  0.00000       0.0         0.0         0.0     0.0     0.018868     0.0
6  0.03125       0.0         0.0         0.0     0.0     0.000000     0.0
7  0.00000       0.0         0.0         0.0     0.0     0.000000     0.0
8  0.00000       0.0         0.0         0.0     0.0     0.000000     0.0
9  0.00000       0.0         0.0         0.0     0.0     0.000000     0.0

   actual  actually   ad  ...  written     wrong  wrote       xyz  \
0     0.0       0.0  0.0  ...      0.0  0.000000    0.0  0.000000
1     0.0       0.0  0.0  ...      0.0  0.000000    0.0  0.000000
2     0.0       0.0  0.0  ...      0.0  0.000000    0.0  0.000000
3     0.0       0.0  0.0  ...      0.0  0.000000    0.0  0.000000
4     0.0       0.0  0.0  ...      0.0  0.000000    0.0  0.000000
5     0.0       0.0  0.0  ...      0.0  0.000000    0.0  0.018868
6     0.0       0.0  0.0  ...      0.0  0.000000    0.0  0.000000
7     0.0       0.0  0.0  ...      0.0  0.025641    0.0  0.000000
8     0.0       0.0  0.0  ...      0.0  0.000000    0.0  0.000000
```

```
9     0.0       0.0  0.0  ...        0.0  0.000000    0.0  0.000000

       yeah      yelp  yesterday  york  youll  yuenan
0  0.000000  0.000000   0.000000   0.0    0.0     0.0
1  0.000000  0.000000   0.000000   0.0    0.0     0.0
2  0.000000  0.000000   0.000000   0.0    0.0     0.0
3  0.000000  0.000000   0.000000   0.0    0.0     0.0
4  0.000000  0.000000   0.000000   0.0    0.0     0.0
5  0.000000  0.018868   0.000000   0.0    0.0     0.0
6  0.000000  0.000000   0.000000   0.0    0.0     0.0
7  0.000000  0.000000   0.000000   0.0    0.0     0.0
8  0.000000  0.000000   0.000000   0.0    0.0     0.0
9  0.025641  0.000000   0.025641   0.0    0.0     0.0

[10 rows x 1343 columns]
```

In [13]:
```python
# %% TFIDF VECTORIZER

# create the vectorizer
vectTFIDFOrig = TfidfVectorizer(input = 'content')
# tokenize and build vocab
vectTFIDFOrigFit = vectTFIDFOrig.fit_transform(MyList)
vectTFIDFOrigDF = pd.DataFrame(vectTFIDFOrigFit.toarray(), columns = MyColumnNames)
print(vectTFIDFOrigDF.head(10))
vectTFIDFOrigDFLie = add_labels(vectTFIDFOrigDF, 'Lie LABEL')
vectTFIDFOrigDFSenti = add_labels(vectTFIDFOrigDF, 'Senti LABEL')
```

```
        abc  abruptly  absolutely  acceptable  accord  acknowledge  across  \
0  0.000000       0.0         0.0         0.0     0.0     0.000000     0.0
1  0.000000       0.0         0.0         0.0     0.0     0.000000     0.0
2  0.000000       0.0         0.0         0.0     0.0     0.000000     0.0
3  0.000000       0.0         0.0         0.0     0.0     0.000000     0.0
4  0.000000       0.0         0.0         0.0     0.0     0.000000     0.0
5  0.000000       0.0         0.0         0.0     0.0     0.154473     0.0
6  0.240826       0.0         0.0         0.0     0.0     0.000000     0.0
7  0.000000       0.0         0.0         0.0     0.0     0.000000     0.0
8  0.000000       0.0         0.0         0.0     0.0     0.000000     0.0
9  0.000000       0.0         0.0         0.0     0.0     0.000000     0.0

   actual  actually   ad  ...    written     wrong  wrote      xyz      yeah  \
0     0.0       0.0  0.0  ...        0.0  0.000000    0.0  0.00000  0.000000
1     0.0       0.0  0.0  ...        0.0  0.000000    0.0  0.00000  0.000000
2     0.0       0.0  0.0  ...        0.0  0.000000    0.0  0.00000  0.000000
3     0.0       0.0  0.0  ...        0.0  0.000000    0.0  0.00000  0.000000
4     0.0       0.0  0.0  ...        0.0  0.000000    0.0  0.00000  0.000000
5     0.0       0.0  0.0  ...        0.0  0.000000    0.0  0.14153  0.000000
6     0.0       0.0  0.0  ...        0.0  0.000000    0.0  0.00000  0.000000
```

```
7     0.0       0.0  0.0   ...        0.0  0.172278    0.0  0.00000  0.000000
8     0.0       0.0  0.0   ...        0.0  0.000000    0.0  0.00000  0.000000
9     0.0       0.0  0.0   ...        0.0  0.000000    0.0  0.00000  0.137639

      yelp  yesterday  york  youll  yuenan
0  0.00000   0.000000   0.0    0.0     0.0
1  0.00000   0.000000   0.0    0.0     0.0
2  0.00000   0.000000   0.0    0.0     0.0
3  0.00000   0.000000   0.0    0.0     0.0
4  0.00000   0.000000   0.0    0.0     0.0
5  0.14153   0.000000   0.0    0.0     0.0
6  0.00000   0.000000   0.0    0.0     0.0
7  0.00000   0.000000   0.0    0.0     0.0
8  0.00000   0.000000   0.0    0.0     0.0
9  0.00000   0.150225   0.0    0.0     0.0

[10 rows x 1343 columns]
```

```
In [14]: # %% STANDARDIZED COUNT VECTORIZER

         vectCountStandDF = copy.deepcopy(vectCountOrigDF)
         scaler = preprocessing.MinMaxScaler()
         vectCountStandDF = pd.DataFrame(scaler.fit_transform(vectCountStandDF), columns = MyC
         print(vectCountStandDF.head())
         vectCountStandDFLie = add_labels(vectCountStandDF, 'Lie LABEL')
         vectCountStandDFSenti = add_labels(vectCountStandDF, 'Senti LABEL')
```

```
   abc  abruptly  absolutely  acceptable  accord  acknowledge  across  actual  \
0  0.0       0.0         0.0         0.0     0.0          0.0     0.0     0.0
1  0.0       0.0         0.0         0.0     0.0          0.0     0.0     0.0
2  0.0       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     0.0
4  0.0       0.0         0.0         0.0     0.0          0.0     0.0     0.0

   actually   ad   ...    written  wrong  wrote  xyz  yeah  yelp  yesterday  \
0       0.0  0.0   ...        0.0    0.0    0.0  0.0   0.0   0.0        0.0
1       0.0  0.0   ...        0.0    0.0    0.0  0.0   0.0   0.0        0.0
2       0.0  0.0   ...        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   0.0        0.0
4       0.0  0.0   ...        0.0    0.0    0.0  0.0   0.0   0.0        0.0

   york  youll  yuenan
0   0.0    0.0     0.0
1   0.0    0.0     0.0
2   0.0    0.0     0.0
3   0.0    0.0     0.0
4   0.0    0.0     0.0
```

```
[5 rows x 1343 columns]
```

```
In [15]: # %% STANDARDIZED NORMALIZED COUNT VECTORIZER

         vectCountNormStandDF = copy.deepcopy(vectCountNormOrigDF)
         scaler = preprocessing.MinMaxScaler()
         vectCountNormStandDF = pd.DataFrame(scaler.fit_transform(vectCountNormStandDF), colum
         print(vectCountNormStandDF.head(10))
         vectCountNormStandDFLie = add_labels(vectCountNormStandDF, 'Lie LABEL')
         vectCountNormStandDFSenti = add_labels(vectCountNormStandDF, 'Senti LABEL')
```

```
   abc  abruptly  absolutely  acceptable  accord  acknowledge  across  actual  \
0  0.0       0.0         0.0         0.0     0.0          0.0     0.0     0.0
1  0.0       0.0         0.0         0.0     0.0          0.0     0.0     0.0
2  0.0       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     0.0
4  0.0       0.0         0.0         0.0     0.0          0.0     0.0     0.0
5  0.0       0.0         0.0         0.0     0.0          1.0     0.0     0.0
6  1.0       0.0         0.0         0.0     0.0          0.0     0.0     0.0
7  0.0       0.0         0.0         0.0     0.0          0.0     0.0     0.0
8  0.0       0.0         0.0         0.0     0.0          0.0     0.0     0.0
9  0.0       0.0         0.0         0.0     0.0          0.0     0.0     0.0

   actually   ad  ...   written     wrong  wrote       xyz      yeah  \
0       0.0  0.0  ...       0.0  0.000000    0.0  0.000000  0.000000
1       0.0  0.0  ...       0.0  0.000000    0.0  0.000000  0.000000
2       0.0  0.0  ...       0.0  0.000000    0.0  0.000000  0.000000
3       0.0  0.0  ...       0.0  0.000000    0.0  0.000000  0.000000
4       0.0  0.0  ...       0.0  0.000000    0.0  0.000000  0.000000
5       0.0  0.0  ...       0.0  0.000000    0.0  0.471698  0.000000
6       0.0  0.0  ...       0.0  0.000000    0.0  0.000000  0.000000
7       0.0  0.0  ...       0.0  0.273504    0.0  0.000000  0.000000
8       0.0  0.0  ...       0.0  0.000000    0.0  0.000000  0.000000
9       0.0  0.0  ...       0.0  0.000000    0.0  0.000000  0.948718

       yelp  yesterday  york  youll  yuenan
0  0.000000        0.0   0.0    0.0     0.0
1  0.000000        0.0   0.0    0.0     0.0
2  0.000000        0.0   0.0    0.0     0.0
3  0.000000        0.0   0.0    0.0     0.0
4  0.000000        0.0   0.0    0.0     0.0
5  0.433962        0.0   0.0    0.0     0.0
6  0.000000        0.0   0.0    0.0     0.0
7  0.000000        0.0   0.0    0.0     0.0
8  0.000000        0.0   0.0    0.0     0.0
9  0.000000        1.0   0.0    0.0     0.0
```

```
[10 rows x 1343 columns]
```

```
In [16]: # %% STANDARDIZED TFIDF VECTORIZER

         vectTFIDFStandDF = copy.deepcopy(vectTFIDFOrigDF)
         scaler = preprocessing.MinMaxScaler()
         vectTFIDFStandDF = pd.DataFrame(scaler.fit_transform(vectTFIDFStandDF), columns = MyCo
         print(vectTFIDFStandDF.head(10))
         vectTFIDFStandDFLie = add_labels(vectTFIDFStandDF, 'Lie LABEL')
         vectTFIDFStandDFSenti = add_labels(vectTFIDFStandDF, 'Senti LABEL')
```

```
   abc  abruptly  absolutely  acceptable  accord  acknowledge  across  actual  \
0  0.0       0.0         0.0         0.0     0.0          0.0     0.0     0.0
1  0.0       0.0         0.0         0.0     0.0          0.0     0.0     0.0
2  0.0       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     0.0
4  0.0       0.0         0.0         0.0     0.0          0.0     0.0     0.0
5  0.0       0.0         0.0         0.0     0.0          1.0     0.0     0.0
6  1.0       0.0         0.0         0.0     0.0          0.0     0.0     0.0
7  0.0       0.0         0.0         0.0     0.0          0.0     0.0     0.0
8  0.0       0.0         0.0         0.0     0.0          0.0     0.0     0.0
9  0.0       0.0         0.0         0.0     0.0          0.0     0.0     0.0

   actually   ad  ...   written     wrong  wrote       xyz      yeah  \
0       0.0  0.0  ...       0.0  0.000000    0.0  0.000000  0.000000
1       0.0  0.0  ...       0.0  0.000000    0.0  0.000000  0.000000
2       0.0  0.0  ...       0.0  0.000000    0.0  0.000000  0.000000
3       0.0  0.0  ...       0.0  0.000000    0.0  0.000000  0.000000
4       0.0  0.0  ...       0.0  0.000000    0.0  0.000000  0.000000
5       0.0  0.0  ...       0.0  0.000000    0.0  0.594324  0.000000
6       0.0  0.0  ...       0.0  0.000000    0.0  0.000000  0.000000
7       0.0  0.0  ...       0.0  0.325975    0.0  0.000000  0.000000
8       0.0  0.0  ...       0.0  0.000000    0.0  0.000000  0.000000
9       0.0  0.0  ...       0.0  0.000000    0.0  0.000000  0.795374

       yelp  yesterday  york  youll  yuenan
0  0.000000        0.0   0.0    0.0     0.0
1  0.000000        0.0   0.0    0.0     0.0
2  0.000000        0.0   0.0    0.0     0.0
3  0.000000        0.0   0.0    0.0     0.0
4  0.000000        0.0   0.0    0.0     0.0
5  0.579993        0.0   0.0    0.0     0.0
6  0.000000        0.0   0.0    0.0     0.0
7  0.000000        0.0   0.0    0.0     0.0
8  0.000000        0.0   0.0    0.0     0.0
9  0.000000        1.0   0.0    0.0     0.0
```

[10 rows x 1343 columns]

In [17]:
```
#Alright!!!
#We have 12 datasets (6 vectorize methods, 2 label sets):

list_of_DF_Lie = [vectCountOrigDFLie, vectCountNormOrigDFLie, vectTFIDFOrigDFLie, vec
list_of_DF_Lie_Names = ['vectCountOrigDFLie', 'vectCountNormOrigDFLie', 'vectTFIDFOri

list_of_DF_Senti = [vectCountOrigDFSenti, vectCountNormOrigDFSenti, vectTFIDFOrigDFSe
list_of_DF_Senti_Names = ['vectCountOrigDFSenti', 'vectCountNormOrigDFSenti', 'vectTF
```

In [18]:
```
#Separate the different possible labels indices for both labels because we have a sma
TruthLie_ind = MyTextDF[(MyTextDF["Lie_Label"] == "truth")].index
LieLie_ind = MyTextDF[(MyTextDF["Lie_Label"] == "lie")].index

trainIndex_Lie = []
testIndex_Lie = []

trainIndex_LieT = []
testIndex_LieT = []

trainIndex_LieL = []
testIndex_LieL = []

#Get 10 folds
kfLieT = KFold(n_splits = 10, shuffle = True)
kfLieT.get_n_splits(TruthLie_ind)

kfLieL = KFold(n_splits = 10, shuffle = True)
kfLieL.get_n_splits(LieLie_ind)

for train_index, test_index in kfLieT.split(TruthLie_ind):
    trainIndex_LieT.append(train_index)
    testIndex_LieT.append(test_index)

for train_index, test_index in kfLieL.split(LieLie_ind):
    trainIndex_LieL.append(train_index)
    testIndex_LieL.append(test_index)

for i in range(10):
    trainIndex_Lie.append(trainIndex_LieT[i] + trainIndex_LieL[i])
    testIndex_Lie.append(testIndex_LieT[i] + testIndex_LieL[i])

#Repeat above for senti
NegSent_ind = MyTextDF[(MyTextDF["Senti_Label"] == "neg")].index
PosSent_ind = MyTextDF[(MyTextDF["Senti_Label"] == "pos")].index
```

```
trainIndex_Senti = []
testIndex_Senti = []

trainIndex_SentiN = []
testIndex_SentiN = []

trainIndex_SentiP = []
testIndex_SentiP = []

#Get 10 folds
kfSentiN = KFold(n_splits = 10, shuffle = True)
kfSentiN.get_n_splits(NegSent_ind)

kfSentiP = KFold(n_splits = 10, shuffle = True)
kfSentiP.get_n_splits(PosSent_ind)

for train_index, test_index in kfSentiN.split(NegSent_ind):
    trainIndex_SentiN.append(train_index)
    testIndex_SentiN.append(test_index)

for train_index, test_index in kfSentiP.split(PosSent_ind):
    trainIndex_SentiP.append(train_index)
    testIndex_SentiP.append(test_index)

for i in range(10):
    trainIndex_Senti.append(trainIndex_SentiN[i] + trainIndex_SentiP[i])
    testIndex_Senti.append(testIndex_SentiN[i] + testIndex_SentiP[i])
```

In [19]:
```
#Let's model with MNB!
#We will iterate through all 10 folds for each dataset
#We have our cross validation index list, so we can iterate through our CV list for e

# %%

## Which Features will be most important????!?!

#Define a function to produce a dictionary of features sorted by importance
def feat_imp(train_df, model):
    featLogProb = []
    features = {}
    ind = 0
    for feats in train_df.columns:
        ## the following line takes the difference of the log prob of feature given m
        ## thus it measure the importance of the feature for classification.
        featLogProb.append(abs(model.feature_log_prob_[1,ind] - model.feature_log_prob
        features[(feats)] = featLogProb[ind]
        ind = ind + 1
```

```
    sortedKeys = sorted(features, key = features.get, reverse = True)[:19]
    sortedVals = sorted(features.values(), reverse = True)[:19]
    features2  = {}
    for ki in range(len(sortedKeys)):
        features2[sortedKeys[ki]] = sortedVals[ki]
    return(features2)
```

In [20]:
```
# %% MULTINOMIAL NAIVE BAYES LIE
#https://scikit-learn.org/stable/modules/generated/sklearn.naive_bayes.MultinomialNB.h

#For each metric, we have a dictionary where the keys are the experiment dataframes an
cm_Lie = {}
acc_Lie = {}
prfs_Lie = {}

#Also get the most important features for each run
features_Lie = {}

for loc in range(len(list_of_DF_Lie)):
    DF = list_of_DF_Lie[loc]
    name = list_of_DF_Lie_Names[loc]
    cm_Lie[name] = []
    acc_Lie[name] = []
    prfs_Lie[name] = []
    features_Lie[name] = []
    ind = 1
    for train_ind, test_ind in zip(trainIndex_Lie, testIndex_Lie):
        train = DF.iloc[train_ind, ]
        test = DF.iloc[test_ind, ]
        #Remove labels
        trainLabels = train["Lie LABEL"]
        testLabels = test["Lie LABEL"]
        train = train.drop(["Lie LABEL"], axis = 1)
        test = test.drop(["Lie LABEL"], axis = 1)
        #Create the modeler
        MyModelNB= MultinomialNB()
        MyModelNB.fit(train, trainLabels)
        Prediction = MyModelNB.predict(test)
        y_true = (testLabels).tolist()
        y_predict = (Prediction).tolist()
        labels =['lie', 'truth']
        cm = confusion_matrix(y_true, y_predict, labels)
        cm_Lie[name].append(cm)
        acc = accuracy_score(y_true, y_predict)
        acc_Lie[name].append(acc)
        prfs = precision_recall_fscore_support(y_true, y_predict, pos_label = 'lie', a
```

```
        prfs_Lie[name].append(prfs)
        features_Lie[name].append(feat_imp(train, MyModelNB))

        #Plot the confusion matrix
#        title = str('Confusion Matrix\n' + name + ' fold ' + str(ind))
#        cm_plot = plot_confusion_matrix(y_true = y_true, y_pred = y_predict, classes
#        outpath = str('output/Lie/confmat/' + name + '_fold_' + str(ind) + '.png')
#        plt.savefig(outpath, bbox_inches='tight')
#
#        plt.clf()
#
#        #Create a word cloud
#        wc = WordCloud().generate_from_frequencies(features_Lie[name][ind - 1])
#        plt.imshow(wc)
#        plt.xticks(ticks = None)
#        plt.yticks(ticks = None)
#        outpath = str('output/Lie/wordclouds/' + name + '_fold_' + str(ind) + '.png'
#        plt.savefig(outpath, bbox_inches='tight')

        ind += 1
```

```
C:\Users\jerem\Anaconda3\lib\site-packages\sklearn\metrics\classification.py:1135: UndefinedMe
  'precision', 'predicted', average, warn_for)
```

In [25]:
```
# %% MULTINOMIAL NAIVE BAYES SENTI
#https://scikit-learn.org/stable/modules/generated/sklearn.naive_bayes.MultinomialNB.

#For each metric, we have a dictionary where the keys are the experiment dataframes a
cm_Senti = {}
acc_Senti = {}
prfs_Senti = {}

#Also get the most important features for each run
features_Senti = {}

for loc in range(len(list_of_DF_Senti)):
    DF = list_of_DF_Senti[loc]
    name = list_of_DF_Senti_Names[loc]
    cm_Senti[name] = []
    acc_Senti[name] = []
    prfs_Senti[name] = []
    features_Senti[name] = []
    ind = 1
    for train_ind, test_ind in zip(trainIndex_Senti, testIndex_Senti):
        train = DF.iloc[train_ind, ]
        test = DF.iloc[test_ind, ]
        #Remove labels
```

```
            trainLabels = train["Senti LABEL"]
            testLabels = test["Senti LABEL"]
            train = train.drop(["Senti LABEL"], axis = 1)
            test = test.drop(["Senti LABEL"], axis = 1)
            #Create the modeler
            MyModelNB= MultinomialNB()
            MyModelNB.fit(train, trainLabels)
            Prediction = MyModelNB.predict(test)
            y_true = (testLabels).tolist()
            y_predict = (Prediction).tolist()
            labels =['neg', 'pos']
            cm = confusion_matrix(y_true, y_predict, labels)
            cm_Senti[name].append(cm)
            acc = accuracy_score(y_true, y_predict)
            acc_Senti[name].append(acc)
            prfs = precision_recall_fscore_support(y_true, y_predict, pos_label = 'pos', a
            prfs_Senti[name].append(prfs)
            features_Senti[name].append(feat_imp(train, MyModelNB))

            #Plot the confusion matrix
#            title = str('Confusion Matrix\n' + name + ' fold ' + str(ind))
#            cm_plot = plot_confusion_matrix(y_true = y_true, y_pred = y_predict, classes
#            outpath = str('output/Senti/confmat/' + name + '_fold_' + str(ind) + '.png')
#            plt.savefig(outpath, bbox_inches='tight')
#
#            plt.clf()
#
#            #Create a word cloud
#            wc = WordCloud().generate_from_frequencies(features_Senti[name][ind - 1])
#            plt.imshow(wc)
#            plt.xticks(ticks = None)
#            plt.yticks(ticks = None)
#            outpath = str('output/Senti/wordclouds/' + name + '_fold_' + str(ind) + '.pn
#            plt.savefig(outpath, bbox_inches='tight')

            ind += 1
```

```
C:\Users\jerem\Anaconda3\lib\site-packages\sklearn\metrics\classification.py:1135: UndefinedMe
  'precision', 'predicted', average, warn_for)
```

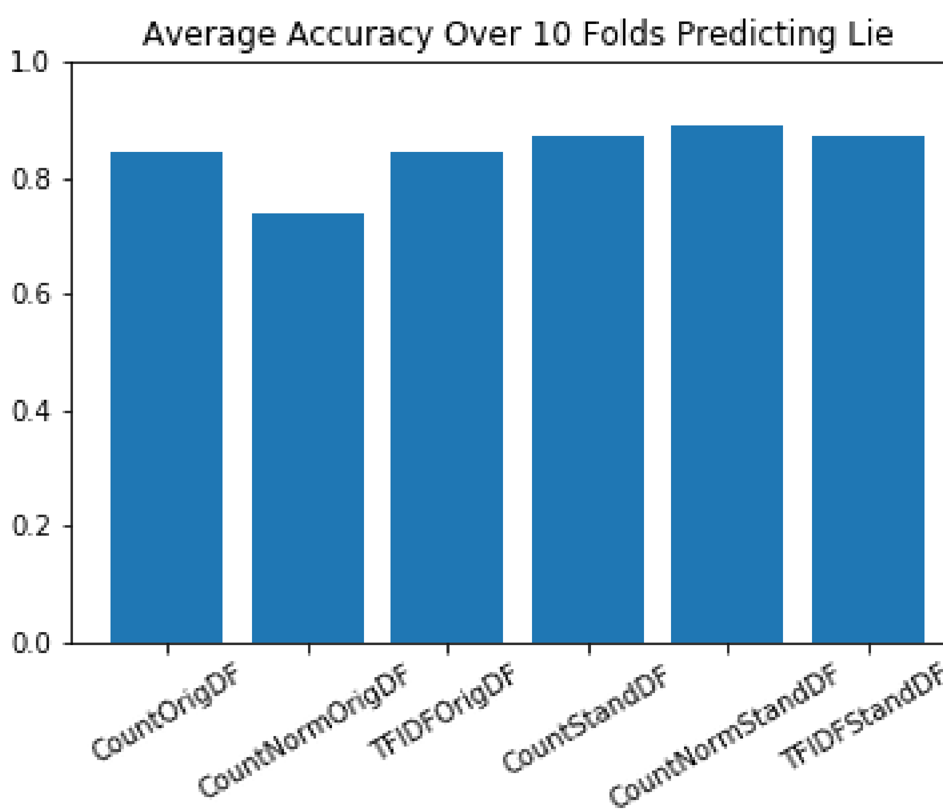
In [27]:
```
# Average across folds
#Compare model performance
# remember this list_of_DF_Lie_Names and list_of_DF_Senti_Names
model_summary_Lie = {}
model_summary_Senti = {}

acc_Dict_Lie = {}
```

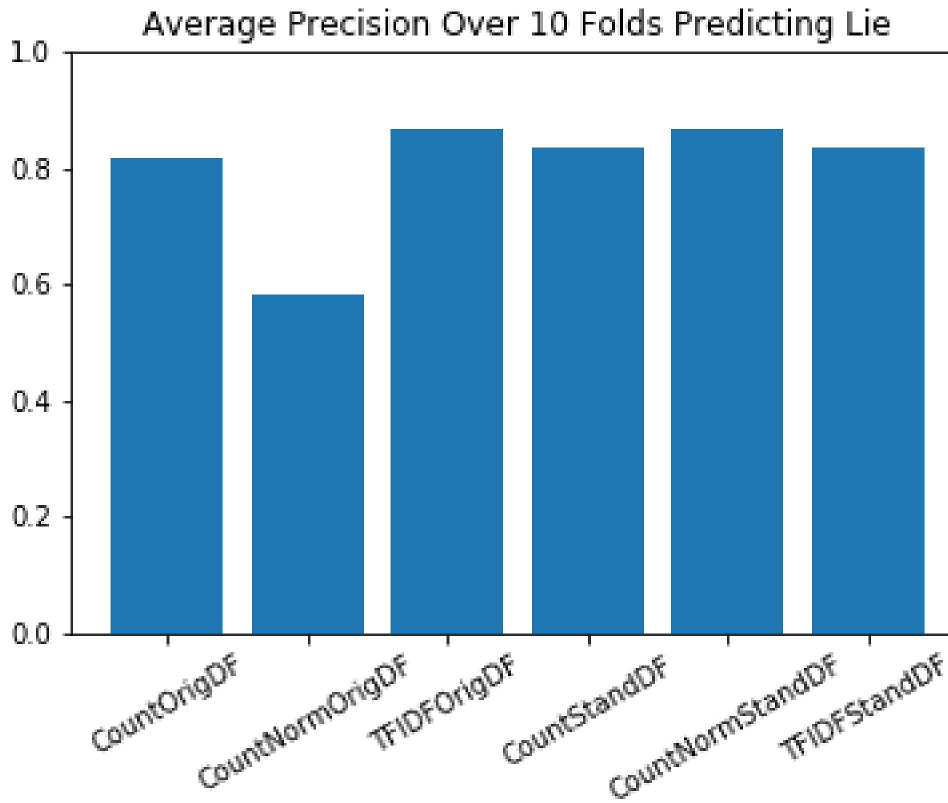
```python
prec_Dict_Lie = {}
rec_Dict_Lie = {}
F1_Dict_Lie = {}

acc_Dict_Senti = {}
prec_Dict_Senti = {}
rec_Dict_Senti = {}
F1_Dict_Senti = {}

for name in list_of_DF_Lie_Names:
    newname = name.replace('vect', '').replace('Lie', '')
    model_summary_Lie[newname] = {}
    #accuracy
    a_avg = 0
    for a in acc_Lie[name]:
        a_avg += a
    a_avg = a_avg / 10
    model_summary_Lie[newname]['acc'] = a_avg
    acc_Dict_Lie[newname] = a_avg

    #precision, recall, F1
    p_avg = 0
    r_avg = 0
    F1_avg = 0
    for m in prfs_Lie[name]:
        p_avg += m[0]
        r_avg += m[1]
        F1_avg += m[2]
    p_avg = p_avg / 10
    r_avg = r_avg / 10
    F1_avg = F1_avg / 10
    model_summary_Lie[newname]['prec'] = p_avg
    model_summary_Lie[newname]['rec'] = r_avg
    model_summary_Lie[newname]['F1'] = F1_avg
    prec_Dict_Lie[newname] = p_avg
    rec_Dict_Lie[newname] = r_avg
    F1_Dict_Lie[newname] = F1_avg

for name in list_of_DF_Senti_Names:
    newname = name.replace('vect', '').replace('Senti', '')
    model_summary_Senti[newname] = {}
    #accuracy
    a_avg = 0
    for a in acc_Senti[name]:
        a_avg += a
    a_avg = a_avg / 10
    model_summary_Senti[newname]['acc'] = a_avg
    acc_Dict_Senti[newname] = a_avg
```

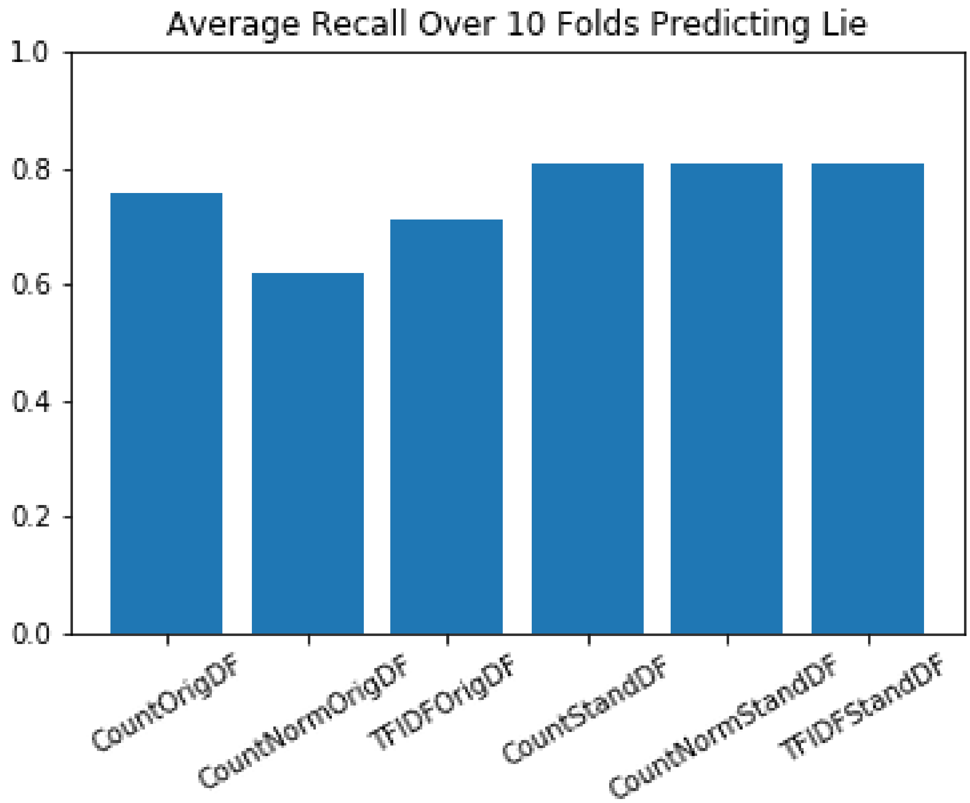
```
    #precision, recall, F1
    p_avg = 0
    r_avg = 0
    F1_avg = 0
    for m in prfs_Senti[name]:
        p_avg += m[0]
        r_avg += m[1]
        F1_avg += m[2]
    p_avg = p_avg / 10
    r_avg = r_avg / 10
    F1_avg = F1_avg / 10
    model_summary_Senti[newname]['prec'] = p_avg
    model_summary_Senti[newname]['rec'] = r_avg
    model_summary_Senti[newname]['F1'] = F1_avg
    prec_Dict_Senti[newname] = p_avg
    rec_Dict_Senti[newname] = r_avg
    F1_Dict_Senti[newname] = F1_avg
```

In [28]:
```
# %%LIE PLOTS

#Plot some of the metrics from above
#Accuracy
plt.bar(range(len(acc_Dict_Lie)), list(acc_Dict_Lie.values()), align='center')
plt.xticks(range(len(acc_Dict_Lie)), list(acc_Dict_Lie.keys()))
locs, labels = plt.xticks()
plt.setp(labels, rotation=30)
axes = plt.gca()
axes.set_ylim([0,1])
plt.title('Average Accuracy Over 10 Folds Predicting Lie')
plt.show()
plt.clf()


#Precision
plt.bar(range(len(prec_Dict_Lie)), list(prec_Dict_Lie.values()), align='center')
plt.xticks(range(len(prec_Dict_Lie)), list(prec_Dict_Lie.keys()))
locs, labels = plt.xticks()
plt.setp(labels, rotation=30)
axes = plt.gca()
axes.set_ylim([0,1])
plt.title('Average Precision Over 10 Folds Predicting Lie')
plt.show()
plt.clf()


#Recall
```

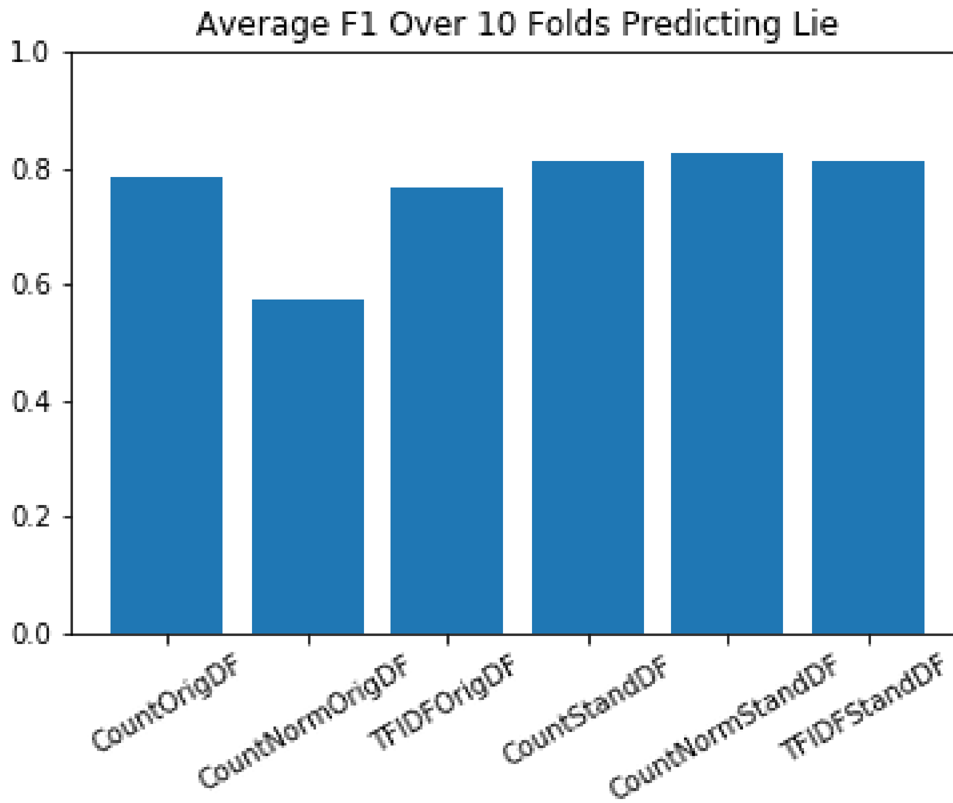
```python
plt.bar(range(len(rec_Dict_Lie)), list(rec_Dict_Lie.values()), align='center')
plt.xticks(range(len(rec_Dict_Lie)), list(rec_Dict_Lie.keys()))
locs, labels = plt.xticks()
plt.setp(labels, rotation=30)
axes = plt.gca()
axes.set_ylim([0,1])
plt.title('Average Recall Over 10 Folds Predicting Lie')
plt.show()
plt.clf()


#F1
plt.bar(range(len(F1_Dict_Lie)), list(F1_Dict_Lie.values()), align='center')
plt.xticks(range(len(F1_Dict_Lie)), list(F1_Dict_Lie.keys()))
locs, labels = plt.xticks()
plt.setp(labels, rotation=30)
axes = plt.gca()
axes.set_ylim([0,1])
plt.title('Average F1 Over 10 Folds Predicting Lie')
plt.show()
plt.clf()
```




Average Precision Over 10 Folds Predicting Lie
1.0
0.8
0.6
0.4
0.2
0.0
CountOrigDF
CountNormOrigDF
TFIDFOrigDF
CountStandDF
CountNormStandDF
TFIDFStandDF



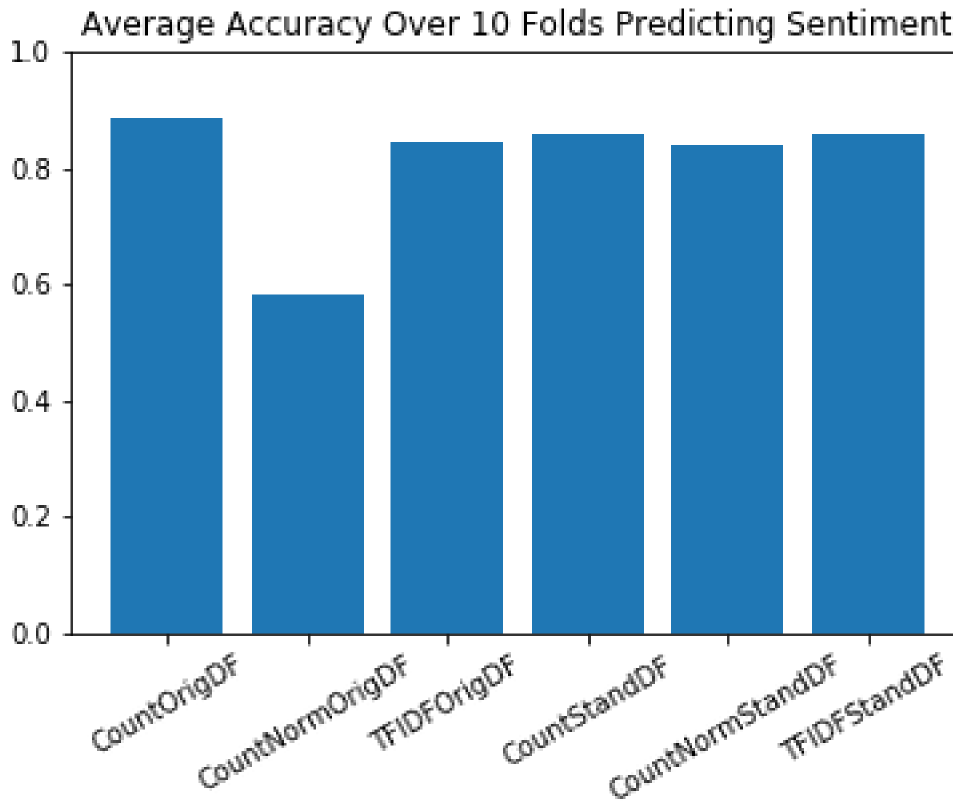
Average Recall Over 10 Folds Predicting Lie
1.0
0.8
0.6
0.4
0.2
0.0
CountOrigDF
CountNormOrigDF
TFIDFOrigDF
CountStandDF
CountNormStandDF
TFIDFStandDF



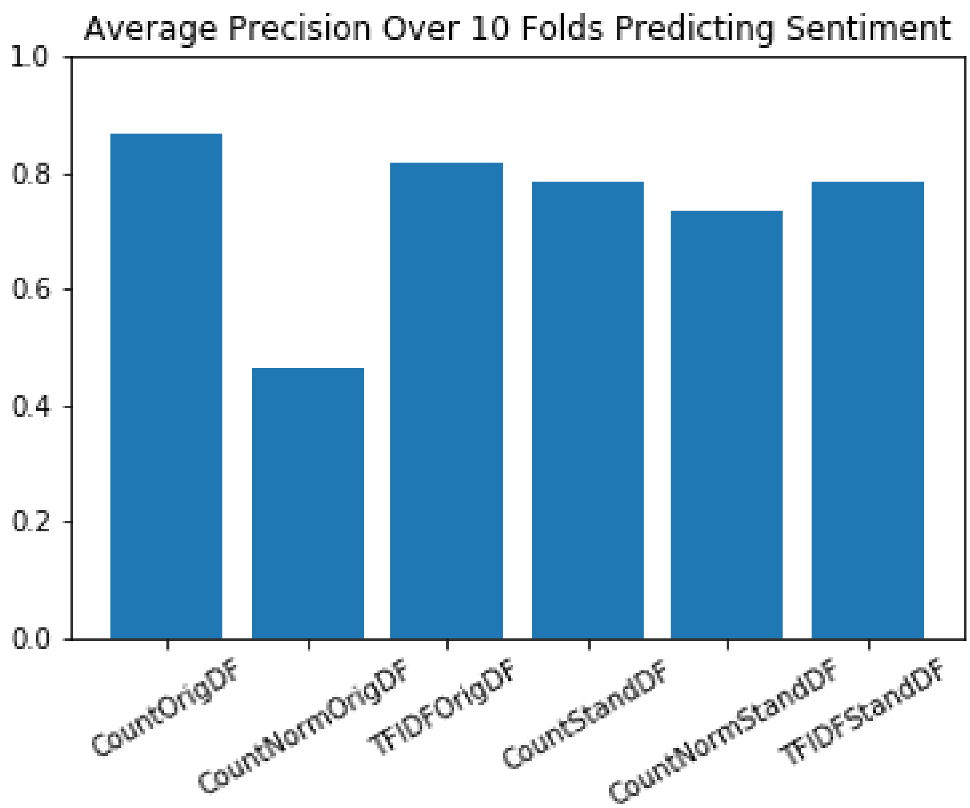

```
<matplotlib.figure.Figure at 0x1afca1d3c18>
```

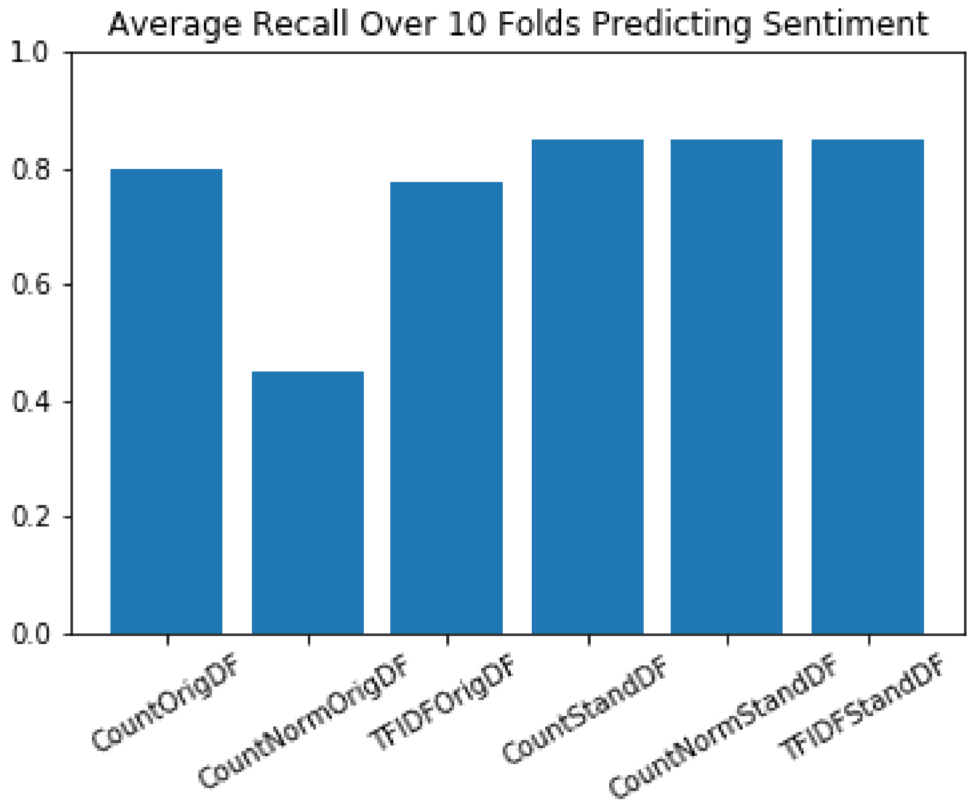
In [29]:
```
# %%SENTI PLOTS

#Plot some of the metrics from above
#Accuracy
plt.bar(range(len(acc_Dict_Senti)), list(acc_Dict_Senti.values()), align='center')
plt.xticks(range(len(acc_Dict_Senti)), list(acc_Dict_Senti.keys()))
locs, labels = plt.xticks()
plt.setp(labels, rotation=30)
axes = plt.gca()
axes.set_ylim([0,1])
plt.title('Average Accuracy Over 10 Folds Predicting Sentiment')
plt.show()
plt.clf()


#Precision
plt.bar(range(len(prec_Dict_Senti)), list(prec_Dict_Senti.values()), align='center')
plt.xticks(range(len(prec_Dict_Senti)), list(prec_Dict_Senti.keys()))
```

```
locs, labels = plt.xticks()
plt.setp(labels, rotation=30)
axes = plt.gca()
axes.set_ylim([0,1])
plt.title('Average Precision Over 10 Folds Predicting Sentiment')
plt.show()
plt.clf()


#Recall
plt.bar(range(len(rec_Dict_Senti)), list(rec_Dict_Senti.values()), align='center')
plt.xticks(range(len(rec_Dict_Senti)), list(rec_Dict_Senti.keys()))
locs, labels = plt.xticks()
plt.setp(labels, rotation=30)
axes = plt.gca()
axes.set_ylim([0,1])
plt.title('Average Recall Over 10 Folds Predicting Sentiment')
plt.show()
plt.clf()


#F1
plt.bar(range(len(F1_Dict_Senti)), list(F1_Dict_Senti.values()), align='center')
plt.xticks(range(len(F1_Dict_Senti)), list(F1_Dict_Senti.keys()))
locs, labels = plt.xticks()
plt.setp(labels, rotation=30)
axes = plt.gca()
axes.set_ylim([0,1])
plt.title('Average F1 Over 10 Folds Predicting Sentiment')
plt.show()
plt.clf()
```

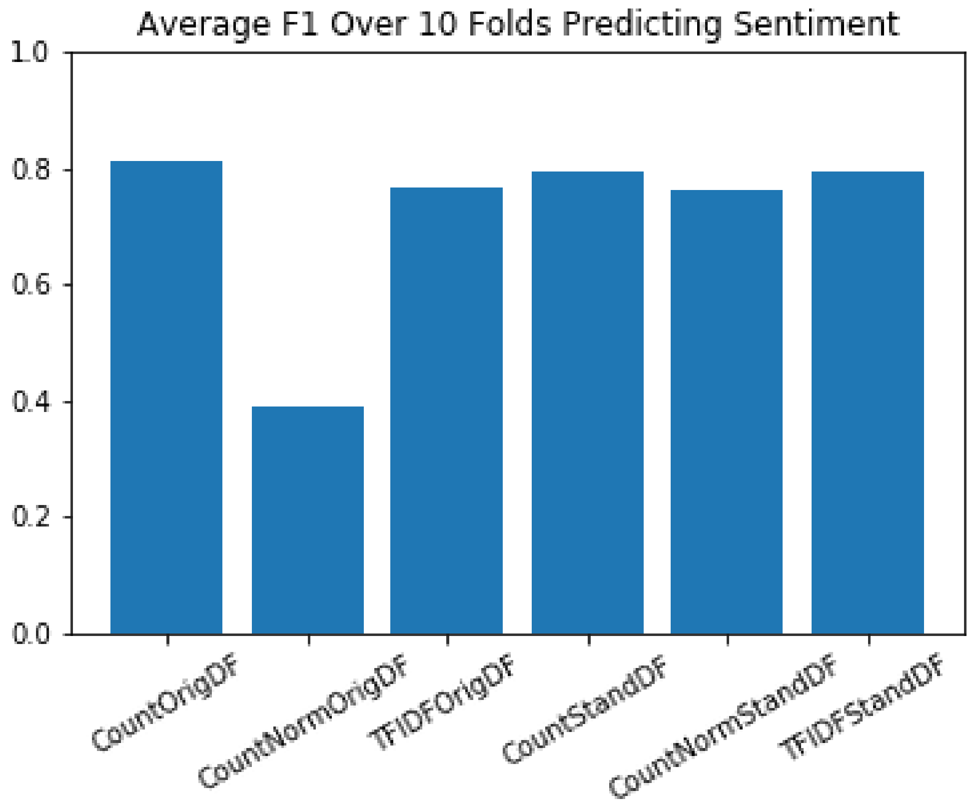


Average Accuracy Over 10 Folds Predicting Sentiment
1.0
0.8
0.6
0.4
0.2
0.0
CountOrigDF
CountNormOrigDF
TFIDFOrigDF
CountStandDF
CountNormStandDF
TFIDFStandDF


Average Precision Over 10 Folds Predicting Sentiment
1.0
0.8
0.6
0.4
0.2
0.0
CountOrigDF
CountNormOrigDF
TFIDFOrigDF
CountStandDF
CountNormStandDF
TFIDFStandDF



Average Recall Over 10 Folds Predicting Sentiment
1.0
0.8
0.6
0.4
0.2
0.0
CountOrigDF
CountNormOrigDF
TFIDFOrigDF
CountStandDF
CountNormStandDF
TFIDFStandDF




```
<matplotlib.figure.Figure at 0x1afc99c14e0>
```

```
In [35]: # %%
         features_dfs_Lie = {}
         features_dfs_Senti = {}
         #Get dataframes for each feature list

         for i in list_of_DF_Lie_Names:
             df = pd.DataFrame()
             dics = features_Lie[i]
             fold = 1
             for dic in dics:
                 col1 = 'fold_' + str(fold) + '_word'
                 col2 = 'fold_' + str(fold) + '_value'
                 sortedKeys = sorted(dic, key = dic.get, reverse = True)
                 sortedVals = sorted(dic.values(), reverse = True)
                 df[col1] = sortedKeys
                 df[col2] = sortedVals
                 fold += 1
```

```
    features_dfs_Lie[i] = df


for i in list_of_DF_Senti_Names:
    df = pd.DataFrame()
    dics = features_Senti[i]
    fold = 1
    for dic in dics:
        col1 = 'fold_' + str(fold) + '_word'
        col2 = 'fold_' + str(fold) + '_value'
        sortedKeys = sorted(dic, key = dic.get, reverse = True)
        sortedVals = sorted(dic.values(), reverse = True)
        df[col1] = sortedKeys
        df[col2] = sortedVals
        fold += 1

    features_dfs_Senti[i] = df
```