Jeffrey Levesque

jlevesqu@syr.edu | Naïve Bayes for Text Categorization

IST-736: HW3

PRofessor gates

**Introduction**

Main stream media is a traditional reference for news entities, as a whole capable of shaping and molding opinions of the general public. The dissemination of knowledge covers a wide variety of platforms, including television, online, as well as print. Vast capabilities of communication allow news agencies to maximize information delivery. Moreover, benefits potentially include a better-informed and engaged general public. However, the underlying facilities often relate to the ideals of a network, as well as general interest factions, and can be limited by social and technological constructs. MSNBC is principally known to favor a left-wing perspective[[1]](#footnote-1), while Fox News indulges more a right-wing viewpoint[[2]](#footnote-2). CNN attempts to report an unbiased perspective. However, in recent wave of political events, CNN has been perceived more leftist than prior years[[3]](#footnote-3).

People often gravitate to those with a similar belief, which may further polarize an already established belief structure[[4]](#footnote-4). Furthermore, in 2015 Microsoft released a study indicating that the average human attention span shortened from 12 to 8 just seconds (2010)[[5]](#footnote-5). A well-known idiom states “Road to hell is paved with good intentions”. In the advent of technology, individuals must carefully weigh information from all possible sources. Social interactions of the 21st century has allowed social media to be a focal point of social exchange theory[[6]](#footnote-6). Concepts of maximized rewards include receiving positive remarks from like-minded individuals, while minimized cost entail least effort of information retrieval.

**Analysis**

An analysis contrasted twitter accounts for two major news agencies. Sentiment for CNN, a recent left-wing reporting media was compared against Fox News, a well-known right-wing news group. A word cloud was generated for both, depicting word usages, which may characterize overall themes. Moreover, an overall word cloud was also created for completeness.

Next, sentiment analysis was conducted for both media outlets, as well as for both combined. Specifically, each tweet was measured for positive, negative, and neutral sentiment. Scores varied from 0 to 1 for each category. Since each tweet was collected with a timestamp, the sentiment scores were plotted as a time series plot.

Finally, an ensemble classifier was created, to predict whether a given tweet is from CNN or Fox News. The ensemble classifiers implemented the Naïve Bayes. The first utilized a TFIDF vectorizer on the unigram representation of corresponding tweets, while the second appended the part of s

peech (POS) tag after each associated word before the application of TFIDF.

**Data Preparation**

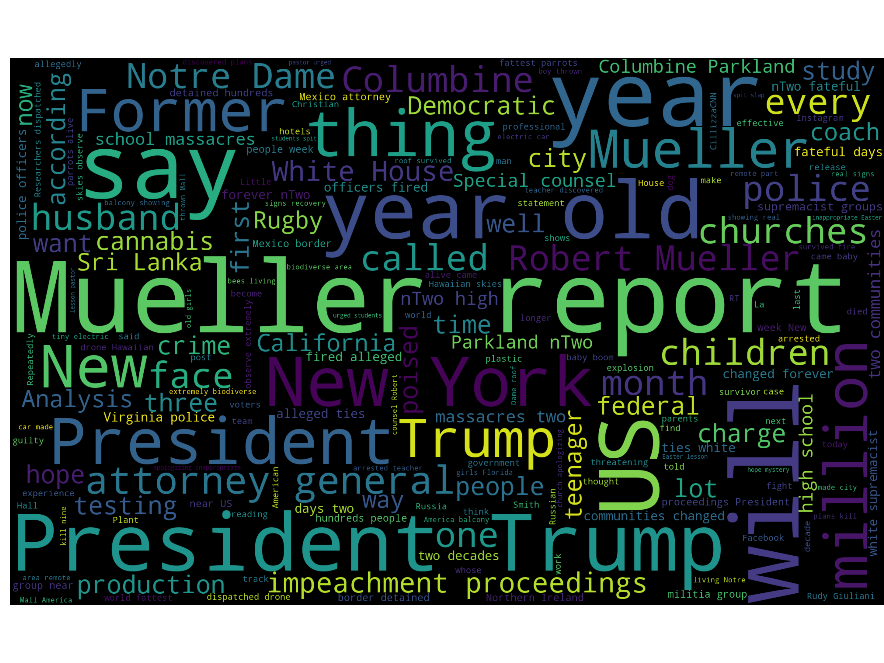
The Twitter API[[7]](#footnote-7) was implemented, requiring a Twitter developer[[8]](#footnote-8) account to be requested. Approval for an account was expedient, not longer than one day. Once granted, a template file was created (see Appendix F for associated code) containing the secret key and tokens provided via the Twitter developer account. A general class was created as a wrapper against the python Twython[[9]](#footnote-9) package. In this class, two main functionalities allow general querying through a set of parameters[[10]](#footnote-10) (see Appendix A for associated code), as well as querying for a defined twitter screen name timeline (see Appendix G for associated code).

More specifically, two screen names were queried for this study. The CNN and FoxNews user accounts were collected. Corresponding code was written such that results were in a dataframe format, then outputted to an associated csv file[[11]](#footnote-11). On future executions, if the corresponding csv file exists locally, then the twitter api did not scrape the locally existing data.

Furthermore, the parameters collected from the twitter accounts were screen\_name, created\_at, and text. Each account was collected with a count=200, which specifies the upper threshold number of tweets to return[[12]](#footnote-12). This was default parameter that could have been increased. Generally, the execution Twython package using this constraint was relatively fast, on order of seconds. Therefore, on a future study, it is appropriate to increase the count.

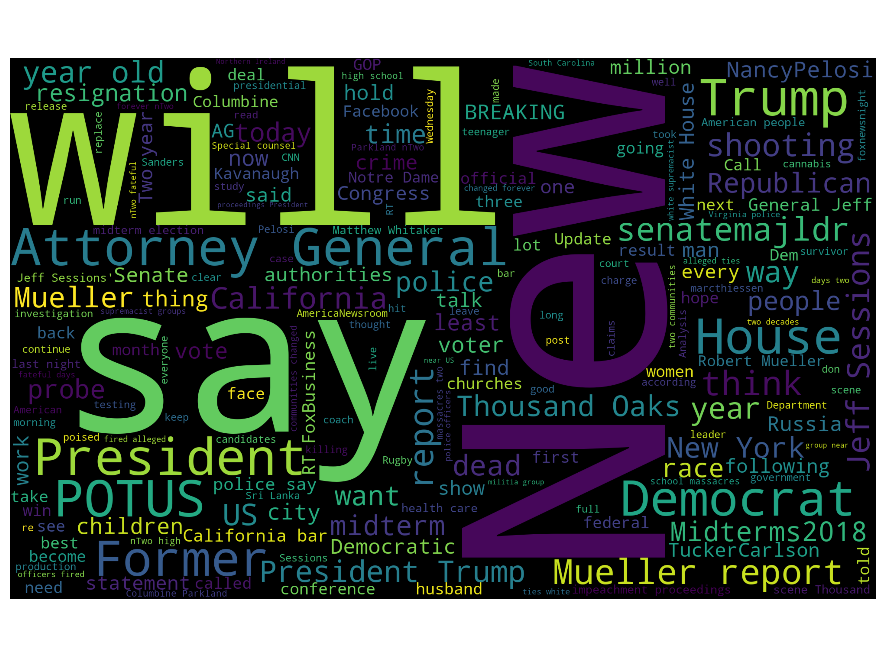
**Results**

A word cloud for the CNN media indicates a dispersion of words heavily related to the Mueller Report and President Trump.



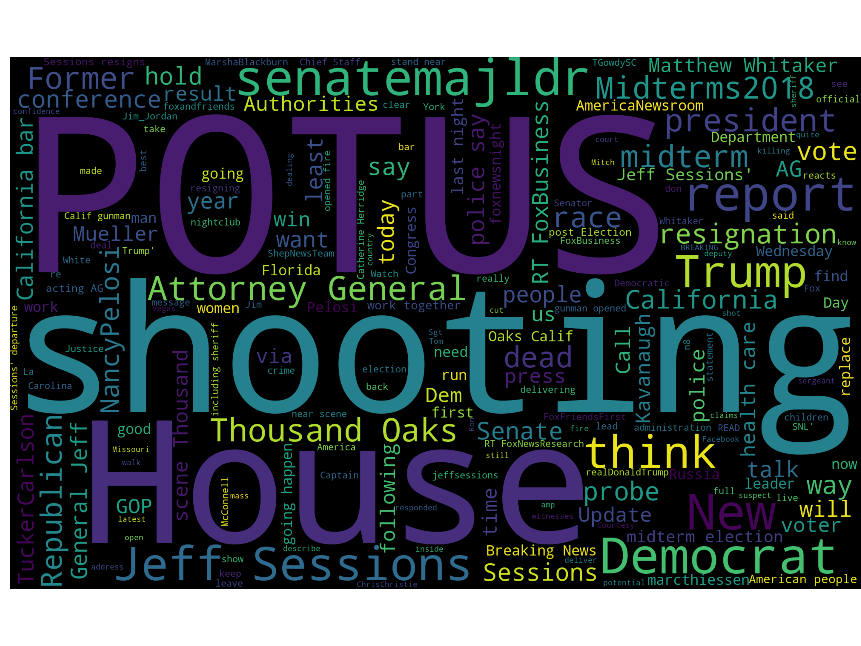
**Figure 1:** most frequent words for CNN twitter account (see Appendix E and Appendix G for associated code).

A word cloud for the Fox News media indicates large representation for single word verbs. However, political characterization still somewhat resembles that from CNN. Better choices of stop words may facilitate an improved visual corpus.



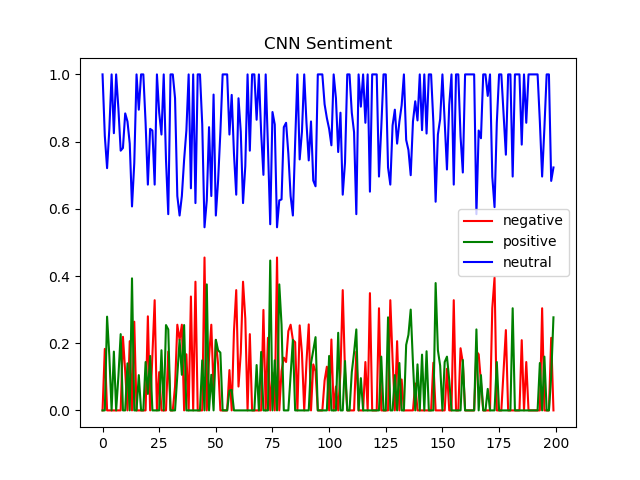
**Figure 2:** most frequent words for FoxNews twitter account (see Appendix E and Appendix G for associated code).

The combined corpus somewhat resembles the earlier word clouds. However, the largest frequent words were much smaller than the previous representations.



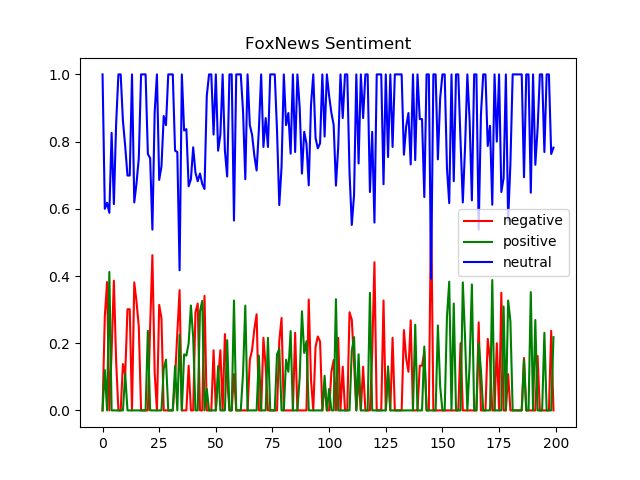
**Figure 3:** most frequent words used for combined CNN and FoxNews twitter accounts (see Appendix E and Appendix G for associated code).

Sentiment for CNN articles appear largely neutral dominant.



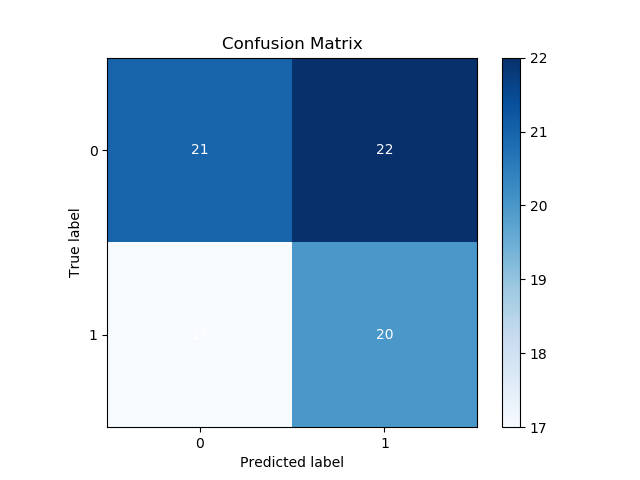
**Figure 4:** positive, negative, neutral sentiment time series characterized by CNN (see Appendix B and Appendix G for associated code).

Fox News depicts greater variability for neutral statements compared to CNN.



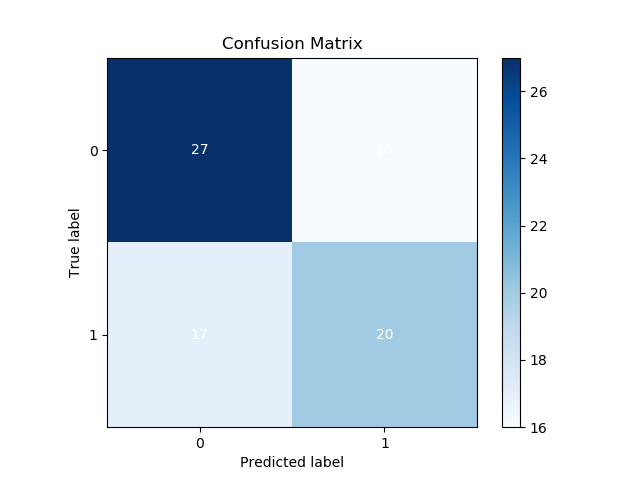
**Figure 5:** positive, negative, neutral sentiment time series characterized by FoxNews (see Appendix B and Appendix G for associated code).

Overall unigram confusion matrix indicates a slight better than chance predictive ability for identifying a tweet between CNN and FoxNews.



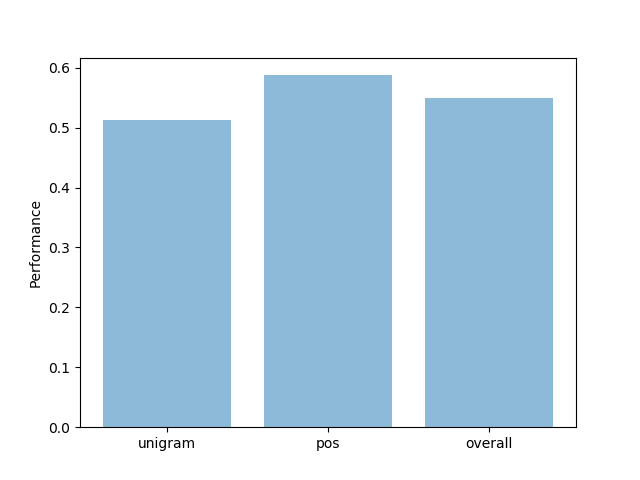
**Figure 6:** overall confusion matrix on the unigram representation indicating predictive ability between CNN and FoxNews using a Naïve Bayes model (see Appendix C and Appendix G for associated code).

Overall POS confusion matrix indicates a noticeable predictive ability to identify a tweet between CNN and FoxNews.



**Figure 7:** overall confusion matrix on the POS representation indicating predictive ability between CNN and FoxNews using a Naïve Bayes model (see Appendix C and Appendix G for associated code).

An ensembled score averages the unigram and POS Naïve Bayes models. Since the unigram model scored slightly better than average, the overall score is not significant in performance.



**Figure 8:** comparison between the unigram and POS Naïve Bayes model. An overall score was computed by averaging the two scores (see Appendix G for associated code).

**Conclusions**

In a digitized world, information travels through many different mediums. The same information may reach the end user from different sources, with a possibility of information loss, even distortion. The same concept borrowed from network packets[[13]](#footnote-13), fundamental to internet protocols, could be applicable to social theory. Specifically, multiple dimensions including social media, the internet that connects users to social media, attention span and interests of individuals and groups, as well as group theory vs individualism[[14]](#footnote-14) all play a similar pattern. Specifically, the potential bottlenecks in the social arena, should portray similar stochastic tendencies as those of network packets[[15]](#footnote-15).

Information theory contains a rich field of varying perspectives. Bottlenecks of information can throttle and impact the perspective of the mass. Short attention span may further limit an individual’s interest to acquire information from credible sources. These shortcomings may encourage users to rely on information loss, acquired from general conversations of varying social media. When enough individuals with a degree of these limitations share interest, information loss and distortion is more likely to occur.

Furthermore, media outlets like CNN and Fox News often provide differing viewpoints. Each tend to have a different fundamental ideology. To be cognizant of current events, it is important to know the limitations and degree of perspective from various sources. Having an open interpretation (at least initially), and not marrying one ideology is sometimes a good approach. An inquisitive stance especially to lower degree information is important. What exactly is a lower degree information? It is information that has traversed one or many nodes. In the concept of network packets, as information traverse’s routers, then across different networks, packets have a possibility of being dropped. Each router and network the information traverses, the greater likelihood of packet drops. Furthermore, the concept is similarly true with social sciences. Each time information traverses an individual, or group of individuals, the original message is subject to greater distortion. This traversal between individuals, or groups of individuals decreases the degree of the original information.

**Appendix A:** query twitter api using twython.

class TwitterQuery():

def \_\_init\_\_(self, key, secret):

self.conn = Twython(key, secret)

def get\_dict\_val(self, d, keys):

for k in keys:

d = d.get(k, None)

if d is None:

return(None)

return(d.encode('utf-8').decode('utf-8'))

def get\_dict\_path(self, d):

temp = []

result = []

for k,v in d.items():

if isinstance(v, dict):

temp.append(k)

self.get\_dict\_path(v)

else:

if isinstance(v, list):

temp.append(k)

[temp.append(x) for x in v]

result.append(temp)

temp = []

else:

temp.append(v)

result.append(temp)

temp = []

return(result)

def query(

self,

query,

params=[{'user': ['screen\_name']}, 'created\_at', 'text'],

sorted=None

):

keys = []

[keys.extend(self.get\_dict\_path(k)) if isinstance(k, dict) else keys.append([k]) for k in params]

results = {x[-1]: [] for x in keys}

for status in self.conn.search(\*\*query)['statuses']:

[results[k].append(self.get\_dict\_val(status, keys[i])) for i, (k,v) in enumerate(results.items())]

self.df\_query = pd.DataFrame(results)

return(self.df\_query)

def query\_user(

self,

screen\_name,

params=[{'user': ['screen\_name']}, 'created\_at', 'text'],

count=200

):

try:

timeline = self.conn.get\_user\_timeline(screen\_name=screen\_name, count=count)

except TwythonError as e:

print(e)

keys = []

[keys.extend(self.get\_dict\_path(k)) if isinstance(k, dict) else keys.append([k]) for k in params]

results = {x[-1]: [] for x in keys}

for tweet in timeline:

[results[k].append(self.get\_dict\_val(tweet, keys[i])) for i, (k,v) in enumerate(results.items())]

self.df\_timeline = pd.DataFrame(results)

return(self.df\_timeline)

**Appendix B:** use vader to perform sentiment analysis.

class Sentiment():

def \_\_init\_\_(self, data, column\_name):

# local variables

self.df = data

self.column\_name = column\_name

def vader\_analysis(self):

analyser = SentimentIntensityAnalyzer()

sid = SentimentIntensityAnalyzer()

result = {

'compound': [],

'negative': [],

'neutral': [],

'positive': []

}

# sentiment analysis

for i, s in enumerate(self.df[self.column\_name]):

ss = sid.polarity\_scores(s)

for k in sorted(ss):

if k == 'compound':

result['compound'].append(ss[k])

elif k == 'neg':

result['negative'].append(ss[k])

elif k == 'neu':

result['neutral'].append(ss[k])

elif k == 'pos':

result['positive'].append(ss[k])

# append results

self.df['compound'] = result['compound']

self.df['negative'] = result['negative']

self.df['neutral'] = result['neutral']

self.df['positive'] = result['positive']

# return scores

return(self.df)

def plot\_ts(self, title='Sentiment Analysis', filename='sentiment.png'):

# generate plot

plt.figure()

with pd.plotting.plot\_params.use('x\_compat', True):

self.df.negative.plot(color='r', legend=True)

self.df.positive.plot(color='g', legend=True)

self.df.neutral.plot(color='b', legend=True)

plt.title(title)

# save plot

plt.savefig(filename)

plt.show()

**Appendix C:** naïve bayes model, with utility methods for confusion matrix and overall accuracy.

class Model():

def \_\_init\_\_(

self,

df=None,

vectorize=True,

key\_text='SentimentText',

key\_class='Sentiment',

fp='{}/data/sample-sentiment.csv'.format(

Path(\_\_file\_\_).resolve().parents[1]

)

):

# class variables

if df is not None:

self.df = df

else:

self.df = pd.read\_csv(fp)

self.key\_text = key\_text

self.key\_class = key\_class

self.actual = None

self.predicted = None

# vectorize data

if vectorize:

self.split()

self.vectorize()

def split(self, test\_size=0.20, stem=True, pos\_split=False):

# clean

self.df[self.key\_text] = [re.sub('[#@]', '', x) for x in self.df[self.key\_text]]

# stem

if stem:

porter = PorterStemmer()

self.df[self.key\_text] = [porter.stem(word) for word in self.df[self.key\_text]]

# split

if pos\_split:

for i, row in self.df.iterrows():

# max length

if isinstance(self.df[self.key\_text].iloc[i], str):

max\_length = len(self.df[self.key\_text].iloc[i].split())

else:

max\_length = len(self.df[self.key\_text].iloc[i].str.split())

pos = self.df[['pos']].iloc[i]

# rebuild 'key-text' with pos suffix

combined = ''

for j in range(max\_length):

if isinstance(self.df[self.key\_text][i], str):

word = self.df[self.key\_text][i].split()[j]

else:

word = self.df[self.key\_text].iloc[i].split()[j]

combined = '{combined} {word}-{pos}'.format(

combined=combined,

word=word,

pos=pos[0][j]

)

self.df[self.key\_text].iloc[[i]] = combined

self.X\_train, self.X\_test, self.y\_train, self.y\_test = train\_test\_split(

self.df[self.key\_text],

self.df[self.key\_class],

test\_size=test\_size

)

def get\_split(self):

return({

'X\_train': self.X\_train,

'X\_test': self.X\_test,

'y\_train': self.y\_train,

'y\_test': self.y\_test,

})

def get\_pos(self, l, pos\_length=280):

result\_word = []

result\_pos = []

pos = [pos\_tag(x) for x in l]

for y in pos:

result\_word.append([x[0] for x in y if x[0] not in stop\_words])

result\_pos.append(

[penn\_scale[x[1]] if x[1] in penn\_scale and x[0] not in stop\_words else 1 for x in y]

)

# consistent length

for i,x in enumerate(result\_pos):

if len(x) < pos\_length:

difference = pos\_length - len(x)

result\_pos[i].extend([1] \* difference)

else:

difference = len(x) - pos\_length

result\_pos[i] = result\_pos[i][:len(result\_pos[i]) - difference]

return(result\_word, result\_pos)

def vectorize(self, stop\_words='english'):

# bag of words: with 'english' stopwords

self.count\_vect = CountVectorizer(stop\_words=stop\_words)

bow = self.count\_vect.fit\_transform(self.X\_train)

# tfidf weighting

self.tfidf\_transformer = TfidfTransformer()

self.X\_train\_tfidf = self.tfidf\_transformer.fit\_transform(bow)

def get\_tfidf(self):

return(self.X\_train\_tfidf)

def get\_df(self):

return(self.df)

def model(self, X, y, validate=False):

# fit model

self.clf = MultinomialNB().fit(X, y)

# validate

if validate and len(validate) == 2:

predictions = []

for item in list(validate[0]):

prediction = self.count\_vect.transform([item])

predictions.append(

self.clf.predict(self.tfidf\_transformer.fit\_transform(prediction))

)

self.actual = validate[1]

self.predicted = predictions

return({

'model': self.clf,

'actual': validate[1],

'predicted': predictions

})

return({

'model': self.clf,

'actual': None,

'predicted': None

})

def plot\_cm(

self,

actual=None,

predicted=None,

filename='confusion\_matrix.png'

):

if not actual:

actual = self.actual

if not predicted:

predicted = self.predicted

# generate plot

plt.figure()

skplt.metrics.plot\_confusion\_matrix(actual, predicted)

# save plot

plt.savefig(filename)

plt.show()

def get\_accuracy(self, actual=None, predicted=None):

if not actual:

actual = self.actual

if not predicted:

predicted = self.predicted

return(accuracy\_score(actual, predicted))

**Appendix D:** penn tree lookup hashtable.

penn\_scale = {

'CC': 2,

'CD': 3,

'DT': 4,

'EX': 5,

'FW': 6,

'IN': 7,

'JJ': 8,

'JJR': 9,

'JJS': 10,

'LS': 11,

'MD': 12,

'NN': 13,

'NNS': 14,

'NNP': 15,

'NNPS': 16,

'PDT': 17,

'POS': 18,

'PRP': 19,

'PRP$': 20,

'RB': 21,

'RBR': 22,

'RBS': 23,

'RP': 24,

'SYM': 25,

'TO': 26,

'UH': 27,

'VB': 28,

'VBD': 29,

'VBG': 30,

'VBN': 31,

'VBP': 32,

'VBZ': 33,

'WDT': 34,

'WP': 35,

'WP$': 36,

'WRB': 37

}

**Appendix E:** generate word cloud using provided dataframe.

def word\_cloud(

df,

filename='plot.png',

width=2000,

height=1250,

background\_color='black',

stopwords=[]

):

# extend stopwords

stopwords = stopwords.extend(STOPWORDS)

# generate wordcloud

text = df.values

wordcloud = WordCloud(

width = width,

height = height,

background\_color = background\_color,

stopwords = stopwords

).generate(str(text))

fig = plt.figure(

figsize = (40, 30),

facecolor = 'k',

edgecolor = 'k'

)

# generate plot

plt.imshow(wordcloud, interpolation = 'bilinear')

plt.axis('off')

plt.tight\_layout(pad=0)

# save plot

plt.savefig(filename)

plt.show()

**Appendix F:** twitter configuration template file.

twitter\_api = {

'CONSUMER\_KEY': 'REPLACE-ME',

'CONSUMER\_SECRET': 'REPLACE-ME',

'ACCESS\_TOKEN': 'REPLACE-ME',

'ACCESS\_SECRET': 'REPLACE-ME'

}

**Appendix G:** implementation of above functions and class methods.

# local variables

csv\_cnn = '../data/tweets\_cnn.csv'

csv\_foxnews = '../data/tweets\_foxnews.csv'

q = TwitterQuery(creds['CONSUMER\_KEY'], creds['CONSUMER\_SECRET'])

# tweets: cnn

if Path(csv\_cnn).is\_file():

df\_cnn = pd.read\_csv(csv\_cnn)

else:

df\_cnn = q.query\_user('cnn')

df\_cnn['text'] = df\_cnn['text'].str.replace(

'http\S+|www.\S+',

'',

case=False

)

df\_cnn.to\_csv(csv\_cnn)

# tweets: foxnews

if Path(csv\_foxnews).is\_file():

df\_foxnews = pd.read\_csv(csv\_foxnews)

else:

df\_foxnews = q.query\_user('FoxNews')

df\_foxnews['text'] = df\_foxnews['text'].str.replace(

'http\S+|www.\S+',

'',

case=False

)

df\_foxnews.to\_csv(csv\_foxnews)

# combine dataframes

df = df\_cnn.append(df\_foxnews)

# generate wordcloud

if not os.path.exists('viz'):

os.makedirs('viz')

word\_cloud(df\_cnn['text'], filename='viz/wc\_cnn.png')

word\_cloud(df\_foxnews['text'], filename='viz/wc\_foxnews.png')

word\_cloud(df['text'], filename='viz/wc\_cnn\_foxnews.png')

# sentiment analysis

sent\_cnn = Sentiment(df\_cnn, 'text')

sent\_foxnews = Sentiment(df\_foxnews, 'text')

sent\_overall = Sentiment(df, 'text')

df\_cnn = sent\_cnn.vader\_analysis()

df\_foxnews = sent\_foxnews.vader\_analysis()

df\_overall = sent\_overall.vader\_analysis()

# vectorize 'screen\_name'

df\_overall = df\_overall.replace({'screen\_name': {'CNN': 0, 'FoxNews': 1}})

sent\_cnn.plot\_ts(title='CNN Sentiment', filename='viz/sentiment\_cnn.png')

sent\_foxnews.plot\_ts(title='FoxNews Sentiment', filename='viz/sentiment\_foxnews.png')

sent\_overall.plot\_ts(title='Overall Sentiment', filename='viz/sentiment\_overall.png')

# unigram: perform unigram analysis.

unigram = nb(df=df\_overall, key\_text='text', key\_class='screen\_name')

# unigram vectorize

unigram.split()

unigram\_params = unigram.get\_split()

unigram\_vectorized = unigram.get\_tfidf()

# unigram classifier

model\_unigram = unigram.model(

unigram\_vectorized,

unigram\_params['y\_train'],

validate=(unigram\_params['X\_test'], unigram\_params['y\_test'])

)

# plot unigram

unigram.plot\_cm(filename='viz/cm\_unigram.png')

# perform pos analysis

df\_pos = unigram.get\_df()

# define pos

df\_pos['pos'] = unigram.get\_pos(

df\_pos['text'].apply(lambda x: x.split())

)[1]

# new dataframe

pos = nb(df\_pos, key\_text='text', key\_class='screen\_name')

pos.split(pos\_split=True)

# pos vectorize

pos\_params = pos.get\_split()

pos\_vectorized = pos.get\_tfidf()

# pos classifier

model\_pos = pos.model(

pos\_vectorized,

pos\_params['y\_train'],

validate=(unigram\_params['X\_test'], unigram\_params['y\_test'])

)

# plot pos

pos.plot\_cm(filename='viz/cm\_pos.png')

# ensembled scored

score\_unigram = unigram.get\_accuracy()

score\_pos = pos.get\_accuracy()

score\_good = (score\_unigram + score\_pos) / 2

score\_bad = 1 - score\_good

objects = ('unigram', 'pos', 'overall')

y\_pos = np.arange(len(objects))

performance = [score\_unigram, score\_pos, score\_good]

plt.bar(y\_pos, performance, align='center', alpha=0.5)

plt.xticks(y\_pos, objects)

plt.ylabel('Performance')

plt.savefig('viz/accuracy\_overall.png')

plt.show()

1. <https://en.wikipedia.org/wiki/MSNBC_controversies> [↑](#footnote-ref-1)
2. <https://en.wikipedia.org/wiki/Fox_News_controversies> [↑](#footnote-ref-2)
3. <https://www.allsides.com/blog/yes-cnns-media-bias-has-shifted-left> [↑](#footnote-ref-3)
4. <https://github.com/jeff1evesque/ist-736-hw/blob/master/hw3/resources/the-law-of-group-polarization.pdf> [↑](#footnote-ref-4)
5. <https://github.com/jeff1evesque/ist-736-hw/blobl/master/hw3/resources/microsoft-attention-spans-research-report.pdf> [↑](#footnote-ref-5)
6. <https://github.com/jeff1evesque/ist-736-hw/blob/master/hw3/resources/E758.pdf> [↑](#footnote-ref-6)
7. <https://developer.twitter.com/en/docs/tweets/search/api-reference/get-search-tweets> [↑](#footnote-ref-7)
8. <https://developer.twitter.com/en/apps> [↑](#footnote-ref-8)
9. <https://twython.readthedocs.io/en/latest/> [↑](#footnote-ref-9)
10. <https://developer.twitter.com/en/docs/tweets/search/api-reference/get-search-tweets.html> [↑](#footnote-ref-10)
11. <https://github.com/jeff1evesque/ist-736-hw/tree/master/data> [↑](#footnote-ref-11)
12. <https://developer.twitter.com/en/docs/tweets/timelines/api-reference/get-statuses-user_timeline.html> [↑](#footnote-ref-12)
13. <https://www.youtube.com/watch?v=7rLROSYcQU8> [↑](#footnote-ref-13)
14. <https://www.sciencedirect.com/topics/psychology/social-identity-theory> [↑](#footnote-ref-14)
15. <https://github.com/jeff1evesque/ist-736-hw/blob/master/hw3/resources/stochastic-networks.pdf> [↑](#footnote-ref-15)