Twitter Hate Speech Detection

The goal of this project is to classify tweets as normal or hate speech (defined vaguely as being either racist or sexist). The training set is already labeled (method of training set labeling is unknown and may be biased). Classification models will aim to label the test set based on the training set labels.

In [615]:
```
import pandas as pd
import ftfy
import nltk
import string
import pattern
import sklearn
import sklearn.ensemble
import scipy
from pattern.en import suggest, sentiment
from nltk.corpus import words, wordnet
```

Data Pre-Processing

Load data sets.

In [2]:
```
traindf = pd.read_csv('C:\Datasets\TrainTweets.csv',encoding = 'utf-8')
testdf = pd.read_csv('C:\Datasets\TestTweets.csv',encoding = 'utf-8')
```

First take a look at the training data frame.

In [4]:
```
pd.set_option('max_colwidth',-1)
```

In [5]:
```
traindf
```

Out[5]:

	id	label	tweet
0	1	0	@user when a father is dysfunctional and is so selfish he drags his kids into his dysfunction. #run
1	2	0	@user @user thanks for #lyft credit i can't use cause they don't offer wheelchair vans in pdx. #disapointed #getthanked
2	3	0	bihday your majesty
3	4	0	#model i love u take with u all the time in urð□□±!!! ð□□□ð□□□ð□□□ð□□□ ð□□¦ð□□¦ð□□¦
4	5	0	factsguide: society now #motivation
5	6	0	[2/2] huge fan fare and big talking before they leave. chaos and pay disputes when they get there. #allshowandnogo
6	7	0	@user camping tomorrow @user @user @user @user @user @user @user dannyâ□¦
7	8	0	the next school year is the year for exams.ð□□¯ can't think about that ð□□ #school #exams #hate #imagine #actorslife #revolutionschool #girl
8	9	0	we won!!! love the land!!! #allin #cavs #champions #cleveland #clevelandcavaliers â□¦

There are clearly some encoding errors. These errors can be fixed using ftfy.

In [6]:
```
traindf.tweet = traindf.tweet.apply(ftfy.fix_encoding)
traindf
```

Out[6]:

	id	label	tweet
0	1	0	@user when a father is dysfunctional and is so selfish he drags his kids into his dysfunction. #run
1	2	0	@user @user thanks for #lyft credit i can't use cause they don't offer wheelchair vans in pdx. #disapointed #getthanked
2	3	0	bihday your majesty
3	4	0	#model i love u take with u all the time in ur📱!!! 😙😎👄👅💦💦💦
4	5	0	factsguide: society now #motivation
5	6	0	[2/2] huge fan fare and big talking before they leave. chaos and pay disputes when they get there. #allshowandnogo
6	7	0	@user camping tomorrow @user @user @user @user @user @user @user danny…
7	8	0	the next school year is the year for exams.😯 can't think about that 😭 #school #exams #hate #imagine #actorslife #revolutionschool #girl
8	9	0	we won!!! love the land!!! #allin #cavs #champions #cleveland #clevelandcavaliers …

The tweets are tokenized below. Hashtags are treated separately from other words and emojis since they are concatenated words, and since they have extra emphasis. Note that in the data set everything is already lower case, which will make processing easier.

In [309]:
```
traindf['tokens'] = traindf.tweet.apply(nltk.tokenize.TweetTokenizer().tokenize)
traindf['hashtags'] = traindf.tokens.apply(lambda x: [a.replace('#','') for a in x i
```

In [310]:
```
traindf
```

Out[310]:

	id	label	tweet	tokens	hashtags
0	1	0	@user when a father is dysfunctional and is so selfish he drags his kids into his dysfunction. #run	[@user, when, a, father, is, dysfunctional, and, is, so, selfish, he, drags, his, kids, into, his, dysfunction, ., #run]	[run]
1	2	0	@user @user thanks for #lyft credit i can't use cause they don't offer wheelchair vans in pdx. #disapointed #getthanked	[@user, @user, thanks, for, #lyft, credit, i, can't, use, cause, they, don't, offer, wheelchair, vans, in, pdx, ., #disapointed, #getthanked]	[lyft, disapointed, getthanked]
2	3	0	bihday your majesty	[bihday, your, majesty]	[]

To help in pre-processing of tokens, a list of all words is created, along with a data frame of common words with their frequencies in the corpus. Most single letters will be removed from the list and contractions with the apostrophe omitted will be added.

In [345]:
```
wordlist = list(set(nltk.corpus.words.words()+[a for a in nltk.corpus.wordnet.all_le
# List of all letters excluding a, b (i.e. be), i, o, u (i.e. you)
nonwordletters = ['c','d','e','f','g','h','j','k','l','m','n','p','q','r','s','t','v
for letter in nonwordletters:
    wordlist.remove(letter)
# Add contractions
contractions = ['isnt','arent','wasnt','werent','havent','hasnt','hadnt','wont','wou
               'cant','couldnt','shouldnt','mightnt','mustnt','wouldve','shouldve','
               'im','youre','hes','shes','theyre','thats','whos','whats','whatre','w
               'youll','itll','theyll','thatll','wholl','whatll','wherell','whenll',
               'id','youd','hed','itd','theyd','thatd','whod','whatd','whered','when
               'ive','youve','weve','theyve']
wordlist = wordlist + contractions
wordlist.sort(key=str.lower)
commonwords = pd.read_excel('C:\Datasets\CommonWords.xlsx')
commonwords.iloc[:,1] = commonwords.iloc[:,1].str.replace('\xa0','')
```

For faster searching, a dictionary will be used to help search alphabetically.

In [362]:
```
pairlist = []
for letter in string.ascii_lowercase:
    for nextletter in string.ascii_lowercase:
        pairlist.append(''.join([letter,nextletter]))
for letter in string.ascii_uppercase:
        pairlist.append(letter)

truncatedwordlist = wordlist[:]
for i in range(len(truncatedwordlist)):
    truncatedwordlist[i]=truncatedwordlist[i][:2]

positionlist = []
for x in pairlist:
    if x in truncatedwordlist:
        positionlist.append(truncatedwordlist.index(x))
    else:
        positionlist.append(None)
for i in range(len(positionlist)):
    if positionlist[i]==None:
        positionlist[i]=positionlist[i-1]

pairdict = dict(zip(pairlist, positionlist))
```

This function determines whether a word is in the wordlist.

In [396]:
```
def inwordlist(word):
    if word == '':
        shortenedlist = wordlist[:334]
    else:
        if word[0] not in string.ascii_lowercase:
            shortenedlist = wordlist[:334]
        else:
            if len(word) ==1:
                shortenedlist = wordlist[pairdict[word[0].upper()]:pairdict[''.join(
            else:
                if word[1] not in string.ascii_lowercase:
                    shortenedlist = wordlist[pairdict[word[0].upper()]:pairdict[''.j
                else:
                    shortenedlist = wordlist[pairdict[word[0:2]]:(pairdict[word[0:2]
    if word in shortenedlist:
        return True
    else:
        return False
```

The function below is designed to split a hashtag into a list of words. All resulting words in the list must be actual words (lemmatized versions of the words must be in the word lists defined above). The function is designed to favor more common words as well as smaller numbers of words.

In [391]:
```
def splithashtag(hashtag):
    finallist = []
    templist = []

    # Returns word at the beginning of a string, the remainder of the string, and wh
    def firstwords(string):
        realwords = []
        n = len(string)
        for i in range(n):
            word = string[:(i+1)]
            lemmatizer = nltk.stem.WordNetLemmatizer()
            lemmatizedword = lemmatizer.lemmatize(word, pos = 'v')
            lemmatizedword2 = lemmatizer.lemmatize(word, pos = 'n')
            if (inwordlist(word)) or (inwordlist(lemmatizedword)) or (inwordlist(lem
                remainder = string[(i+1):]
                lemmatizedremainder = lemmatizer.lemmatize(remainder, pos = 'v')
                lemmatizedremainder2 = lemmatizer.lemmatize(remainder, pos = 'n')
                if (inwordlist(remainder)) or (inwordlist(lemmatizedremainder)) or\
                (inwordlist(lemmatizedremainder2)) or (remainder == ''):
                    complete = True
                else:
                    complete = False
                realwords.append([word,remainder,complete])
        return realwords

    # Recursive search through all word combinations, similar to depth first search
    def combosearch(string):
        nonlocal finallist
        nonlocal templist
        initiallist = firstwords(string)
        if initiallist != []:
            for i in range(len(initiallist)):
                templist.append(initiallist[i][0])
                if initiallist[i][2] == False:
                    combosearch(initiallist[i][1])
                else:
                    templist.append(initiallist[i][1])
                    finallist.append(templist[:])
                    templist.pop()
                templist.pop()

    combosearch(hashtag)
    # Selects most likely combination based on mean word frequency in corpus (single
    score = 0
    parse = []
    for x in finallist:
        tempscore = 0
        if (len(x) == 2) and (x[1]==''):
            tempscore = 25000000
        else:
            for a in x:
                if a in list(commonwords.iloc[:,1]):
                    tempscore += commonwords.iloc[list(commonwords.iloc[:,1]).index(
            tempscore = tempscore/len(x)**10
        if tempscore >= score:
            score = tempscore
```

```
            parse = x
    if (parse == []) or ((len(parse) == 2) and (parse[1]=='')):
        return [hashtag]
    else:
        return parse
```

These functions are used to process tokens and hashtags. Processing is only performed if the token or hashtag is not already a word. Tokens are processed using a spelling correction function from the pattern.en library. A threshold probability of 0.8 is required for the spelling correction to be accepted. Hashtags are processed with the same function first before trying to split the hashtag into multiple words.

In [405]:
```
def tokenprocess(token):
    lemmatizer = nltk.stem.WordNetLemmatizer()
    lemmatizedword = lemmatizer.lemmatize(token, pos = 'v')
    lemmatizedword2 = lemmatizer.lemmatize(token, pos = 'n')
    if (inwordlist(token)) or (inwordlist(lemmatizedword)) or (inwordlist(lemmatized
        return token
    elif suggest(token)[0][1] >= 0.8:
        return suggest(token)[0][0]
    else:
        return token

def hashtagprocess(hashtag):
    lemmatizer = nltk.stem.WordNetLemmatizer()
    lemmatizedword = lemmatizer.lemmatize(hashtag, pos = 'v')
    lemmatizedword2 = lemmatizer.lemmatize(hashtag, pos = 'n')
    if (inwordlist(hashtag)) or (inwordlist(lemmatizedword)) or (inwordlist(lemmatiz
        return [hashtag]
    elif suggest(hashtag)[0][1] >= 0.8:
        return [suggest(hashtag)[0][0]]
    else:
        return splithashtag(hashtag)
```

With the functions defined, pre-processing can continue. First process the hashtags.

In [408]:
```
traindf.hashtags = traindf.hashtags.apply(lambda tags: [hashtagprocess(tag) for tag
```

Since the hashtag processing took about 3+ hours, the result will be exported to a .csv file as a backup.

In [418]:
```
traindf.hashtags.apply(lambda tags:' #'.join([' '.join(tag) for tag in tags])).to_cs
```

Process the non-hashtag tokens.

In [427]:
```
traindf.tokens = traindf.tokens.apply(lambda tokens: [tokenprocess(token) if token[0
```

In [429]:
```
traindf.tokens.apply(lambda tags:' '.join(tags)).to_csv('C:\Datasets\ProcessedTokens
```

Create a new column for corrected tweet and tokens (for sentiment analysis). Merge hashtag lists and create a text from the list for hashtag sentiment analysis.

In [447]:
```
def subhashtag(words, hashtags):
    hashpositions = []
    newwords = words[:]
    for i in range(len(words)):
        if words[i][0] == '#':
            hashpositions.append(i)
    if hashpositions != []:
        for i in range(len(hashpositions)):
            newwords = newwords[:hashpositions[i]]+hashtags[i]+newwords[hashposition
            for j in range(len(hashpositions)):
                hashpositions[j] += len(hashtags[i])-1
    return newwords

def mergetags(hashtags):
    merged = []
    for x in hashtags:
        for a in x:
            merged.append(a)
    return merged


traindf['correcttokens'] = traindf.apply(lambda row: subhashtag(row.tokens, row.hash
traindf['corrected']= traindf.correcttokens.apply(lambda tokens: ' '.join(tokens))
traindf['hashtokens']= traindf.hashtags.apply(mergetags)
traindf['hashtext']= traindf.hashtokens.apply(lambda tokens: ' '.join(tokens))
```

In [449]:
```
traindf.head(30)
```

Out[449]:

	id	label	tweet	tokens	hashtags	correcttokens	corrected	hash
0	1	0	@user when a father is dysfunctional and is so selfish he drags his kids into his dysfunction. #run	[user, when, a, father, is, dysfunctional, and, is, so, selfish, he, drags, his, kids, into, his, dysfunction, ., #run]	[[run]]	[user, when, a, father, is, dysfunctional, and, is, so, selfish, he, drags, his, kids, into, his, dysfunction, ., run]	user when a father is dysfunctional and is so selfish he drags his kids into his dysfunction . run	
1	2	0	@user @user thanks for #lyft credit i can't use cause they don't offer wheelchair vans in pdx. #disapointed #getthanked	[user, user, thanks, for, #lyft, credit, i, canst, use, cause, they, dont, offer, wheelchair, vans, in, pox, ., #disapointed, #getthanked]	[[left], [disappointed], [get, thanked]]	[user, user, thanks, for, left, credit, i, canst, use, cause, they, dont, offer, wheelchair, vans, in, pox, .,	user user thanks for left credit i canst use cause they dont offer wheelchair vans in pox . disappointed	disap get, t

Now we are ready to generate features.

Feature Extraction

First, evaluate sentiments for tweets and hashtags.

In [452]:
```
traindf['sentiment']= traindf.corrected.apply(lambda text: sentiment(text)[0])
traindf['subjectivity']= traindf.corrected.apply(lambda text: sentiment(text)[1])
traindf['hashsentiment']=traindf.hashtext.apply(lambda text: sentiment(text)[0])
traindf['hashsubjectivity']= traindf.hashtext.apply(lambda text: sentiment(text)[1])
```

In [453]:
```
traindf.head(30)
```

Out[453]:

	id	label	tweet	tokens	hashtags	correcttokens	corrected	hash
0	1	0	@user when a father is dysfunctional and is so selfish he drags his kids into his dysfunction. #run	[user, when, a, father, is, dysfunctional, and, is, so, selfish, he, drags, his, kids, into, his, dysfunction, ., #run]	[[run]]	[user, when, a, father, is, dysfunctional, and, is, so, selfish, he, drags, his, kids, into, his, dysfunction, ., run]	user when a father is dysfunctional and is so selfish he drags his kids into his dysfunction . run	
1	2	0	@user @user thanks for #lyft credit i can't use cause they don't offer wheelchair vans in pdx. #disapointed #getthanked	[user, user, thanks, for, #lyft, credit, i, canst, use, cause, they, dont, offer, wheelchair, vans, in, pox, ., #disapointed, #getthanked]	[[left], [disappointed], [get, thanked]]	[user, user, thanks, for, left, credit, i, canst, use, cause, they, dont, offer, wheelchair, vans, in, pox,	user user thanks for left credit i canst use cause they dont offer wheelchair vans in pox . disappointed	disap get, t

Create features using tf-idf.

In [474]:
```
vectorizer = sklearn.feature_extraction.text.TfidfVectorizer(stop_words='english', n
X = vectorizer.fit_transform(traindf['corrected'])
```

Evaluate which terms appear in tweets labeled as hate speech.

In [478]:
```
Xhate = vectorizer.transform(traindf.loc[traindf.label==1,'corrected'])
```

In [536]:
```
hatefeatures = pd.DataFrame(Xhate.sum(axis=0),index=['total'],columns=vectorizer.get
hatefeatures = hatefeatures.sort_values('total',axis=1,ascending=False).T
hatefeatures.loc[hatefeatures.total >= 12]
```

Out[536]:

	total
user	206.564950
user user	87.448148
trump	64.233825
white	54.490779
black	45.001670
library	43.663376
racist	35.560455
oil	31.439107
allahs	28.876879
allahs oil	28.876879
racism	28.784168

The word "library" appears in the list a few times. It is likely this is a spelling correction for a word that is not in the word list. All the hate speech tweets containing "library" are shown below.

In [544]:
```
traindf.loc[traindf.correcttokens.apply(lambda tokens: True if 'library' in tokens e
```

Out[544]:

	tweet	tokens
192	you might be a libtard if... #libtard #sjw #liberal #politics	[you, might, be, a, library, if, ..., #libtard, #sjw, #liberal, #politics]
354	you might be a libtard if... #libtard #sjw #liberal #politics	[you, might, be, a, library, if, ..., #libtard, #sjw, #liberal, #politics]
621	you might be a libtard if... #libtard #sjw #liberal #politics	[you, might, be, a, library, if, ..., #libtard, #sjw, #liberal, #politics]
1717	@user you might be a libtard if... #libtard #sjw #liberal #politics	[user, you, might, be, a, library, if, ..., #libtard, #sjw, #liberal, #politics]
1860	you might be a libtard if... #libtard #sjw #liberal #politics	[you, might, be, a, library, if, ..., #libtard, #sjw, #liberal, #politics]
2155	you might be a libtard if... #libtard #sjw #liberal #politics	[you, might, be, a, library, if, ..., #libtard, #sjw, #liberal, #politics]
2839	you might be a libtard if... #libtard #sjw #liberal #politics	[you, might, be, a, library, if, ..., #libtard, #sjw, #liberal, #politics]

It is clear that in most cases the word "libtard" (not a real word) was spell corrected to "library." This must be corrected for. First corrections are made to spelling corrected tokens.

In [547]:

```
traindf['originaltokens'] = traindf.tweet.apply(nltk.tokenize.TweetTokenizer().token
def restorelibtard(originaltokens, tokens):
    for i in range(len(tokens)):
        if originaltokens[i]=='libtard':
            tokens[i]='libtard'

traindf.apply(lambda row: restorelibtard(row.originaltokens, row.tokens), axis=1)
traindf.loc[traindf.correcttokens.apply(lambda tokens: True if 'library' in tokens e
```

Out[547]:

	tweet	tokens
192	you might be a libtard if... #libtard #sjw #liberal #politics	[you, might, be, a, libtard, if, ..., #libtard, #sjw, #liberal, #politics]
354	you might be a libtard if... #libtard #sjw #liberal #politics	[you, might, be, a, libtard, if, ..., #libtard, #sjw, #liberal, #politics]
621	you might be a libtard if... #libtard #sjw #liberal #politics	[you, might, be, a, libtard, if, ..., #libtard, #sjw, #liberal, #politics]
1717	@user you might be a libtard if... #libtard #sjw #liberal #politics	[user, you, might, be, a, libtard, if, ..., #libtard, #sjw, #liberal, #politics]
1860	you might be a libtard if... #libtard #sjw #liberal #politics	[you, might, be, a, libtard, if, ..., #libtard, #sjw, #liberal, #politics]
2155	you might be a libtard if... #libtard #sjw #liberal #politics	[you, might, be, a, libtard, if, ..., #libtard, #sjw, #liberal, #politics]
2839	you might be a libtard if... #libtard #sjw #liberal #politics	[you, might, be, a, libtard, if, ..., #libtard, #sjw, #liberal, #politics]

Then corrections are made to hashtags.

In [555]:
```
traindf['originalhashtags'] = traindf.tokens.apply(lambda x: [a.replace('#','') for
def restorelibtard2(originalhashtags, hashtags):
    for i in range(len(hashtags)):
        if originalhashtags[i]=='libtard':
            hashtags[i]=['libtard']
traindf.apply(lambda row: restorelibtard2(row.originalhashtags, row.hashtags), axis=
traindf.loc[traindf.correcttokens.apply(lambda tokens: True if 'library' in tokens e
```

Out[555]:

	tweet	hashtags
192	you might be a libtard if... #libtard #sjw #liberal #politics	[[libtard], [saw], [liberal], [politics]]
354	you might be a libtard if... #libtard #sjw #liberal #politics	[[libtard], [saw], [liberal], [politics]]
621	you might be a libtard if... #libtard #sjw #liberal #politics	[[libtard], [saw], [liberal], [politics]]
1717	@user you might be a libtard if... #libtard #sjw #liberal #politics	[[libtard], [saw], [liberal], [politics]]
1860	you might be a libtard if... #libtard #sjw #liberal #politics	[[libtard], [saw], [liberal], [politics]]
2155	you might be a libtard if... #libtard #sjw #liberal #politics	[[libtard], [saw], [liberal], [politics]]
2839	you might be a libtard if... #libtard #sjw #liberal #politics	[[libtard], [saw], [liberal], [politics]]

All other columns are recalculated based on corrected tokens and hashtags.

In [556]:

```
traindf['correcttokens'] = traindf.apply(lambda row: subhashtag(row.tokens, row.hash
traindf['corrected']= traindf.correcttokens.apply(lambda tokens: ' '.join(tokens))
traindf['hashtokens']= traindf.hashtags.apply(mergetags)
traindf['hashtext']= traindf.hashtokens.apply(lambda tokens: ' '.join(tokens))
traindf['sentiment']= traindf.corrected.apply(lambda text: sentiment(text)[0])
traindf['subjectivity']= traindf.corrected.apply(lambda text: sentiment(text)[1])
traindf['hashsentiment']=traindf.hashtext.apply(lambda text: sentiment(text)[0])
traindf['hashsubjectivity']= traindf.hashtext.apply(lambda text: sentiment(text)[1])
traindf.head(30)
```

Out[556]:

	id	label	tweet	tokens	hashtags	correcttokens	corrected	hash
0	1	0	@user when a father is dysfunctional and is so selfish he drags his kids into his dysfunction. #run	[user, when, a, father, is, dysfunctional, and, is, so, selfish, he, drags, his, kids, into, his, dysfunction, ., #run]	[[run]]	[user, when, a, father, is, dysfunctional, and, is, so, selfish, he, drags, his, kids, into, his, dysfunction, ., run]	user when a father is dysfunctional and is so selfish he drags his kids into his dysfunction . run	
1	2	0	@user @user thanks for #lyft credit i can't use cause they don't offer wheelchair vans in pdx. #disapointed #getthanked	[user, user, thanks, for, #lyft, credit, i, canst, use, cause, they, dont, offer, wheelchair, vans, in, pox, ., #disapointed, #getthanked]	[[left], [disappointed], [get, thanked]]	[user, user, thanks, for, left, credit, i, canst, use, cause, they, dont, offer, wheelchair, vans, in, pox, [illegible]	user user thanks for left credit i canst use cause they dont offer wheelchair vans in pox . disappointed	disap get, t

Now the the more accurate list of terms appearing in hate tweets is generated.

In [557]:
```
X = vectorizer.fit_transform(traindf['corrected'])
Xhate = vectorizer.transform(traindf.loc[traindf.label==1,'corrected'])
hatefeatures = pd.DataFrame(Xhate.sum(axis=0),index=['total'],columns=vectorizer.get
hatefeatures = hatefeatures.sort_values('total',axis=1,ascending=False).T
hatefeatures.loc[hatefeatures.total >= 12]
```

Out[557]:

	total
user	206.532078
user user	87.448148
trump	64.233141
white	54.490779
black	45.000963
libtard	44.465259
racist	35.560455
oil	31.439107
allahs	28.876879
allahs oil	28.876879
racism	28.784168

Next, the td-idf must be calculated for the hashtags.

In [558]:
```
vectorizerhash = sklearn.feature_extraction.text.TfidfVectorizer(stop_words='english
Xhash = vectorizerhash.fit_transform(traindf['hashtext'])
```

The matrices for tweet, hashtag, sentiment, and subjectivity must be combined into one matrix.

In [575]:
```
Xcombined = scipy.sparse.hstack([X, Xhash, traindf.iloc[:,9:13]])
y = traindf.label
```

In [606]:
```
featurenames = vectorizer.get_feature_names()+vectorizerhash.get_feature_names()+['s
```

Modeling

Now that we have our final feature matrix, we can begin modeling. SciKit Learn supports SVM, kNN, random forest, and gradient boosted tree models for sparse matrices.

F1 score will be used for evaluation since the data set is so imbalanced.

Support Vector Machine

In [648]:
```
SVM = sklearn.svm.LinearSVC(class_weight='balanced',max_iter=10000)
Grid = {'C':[0.0001,0.001,0.01,0.1,0.2,0.3,0.4,0.5,0.6,0.7,0.8,0.9,1],'loss':['hinge
SVMmodel = sklearn.model_selection.GridSearchCV(SVM,param_grid=Grid,scoring='f1',cv=
SVMmodel.fit(Xcombined,y)
for x in list(zip(SVMmodel.cv_results_['mean_test_score'], SVMmodel.cv_results_['par
    print(x)
print(SVMmodel.best_params_)
```

```
(0.30676220211892535, {'C': 0.0001, 'loss': 'hinge'})
(0.20197303235473343, {'C': 0.0001, 'loss': 'squared_hinge'})
(0.18874035784058887, {'C': 0.001, 'loss': 'hinge'})
(0.2565683881944514, {'C': 0.001, 'loss': 'squared_hinge'})
(0.3205897300957599, {'C': 0.01, 'loss': 'hinge'})
(0.5318128269395964, {'C': 0.01, 'loss': 'squared_hinge'})
(0.5652336979302235, {'C': 0.1, 'loss': 'hinge'})
(0.604125236816496, {'C': 0.1, 'loss': 'squared_hinge'})
(0.5734663439879185, {'C': 0.2, 'loss': 'hinge'})
(0.6105860302936489, {'C': 0.2, 'loss': 'squared_hinge'})
(0.5778316539268468, {'C': 0.3, 'loss': 'hinge'})
(0.616933056594532, {'C': 0.3, 'loss': 'squared_hinge'})
(0.5819162225568001, {'C': 0.4, 'loss': 'hinge'})
(0.6171363466868479, {'C': 0.4, 'loss': 'squared_hinge'})
(0.5850501072973379, {'C': 0.5, 'loss': 'hinge'})
(0.6182600193000377, {'C': 0.5, 'loss': 'squared_hinge'})
(0.5907890504111737, {'C': 0.6, 'loss': 'hinge'})
(0.6167304672653708, {'C': 0.6, 'loss': 'squared_hinge'})
(0.5968724351187653, {'C': 0.7, 'loss': 'hinge'})
(0.6141404307691753, {'C': 0.7, 'loss': 'squared_hinge'})
(0.6021226164408784, {'C': 0.8, 'loss': 'hinge'})
(0.6129796339103365, {'C': 0.8, 'loss': 'squared_hinge'})
(0.6046220915356476, {'C': 0.9, 'loss': 'hinge'})
(0.6137548030140273, {'C': 0.9, 'loss': 'squared_hinge'})
(0.6042873511008013, {'C': 1, 'loss': 'hinge'})
(0.6137526998377282, {'C': 1, 'loss': 'squared_hinge'})
{'C': 0.5, 'loss': 'squared_hinge'}
```

In [652]:
```
SVM = sklearn.svm.LinearSVC(class_weight='balanced',max_iter=10000, C=0.5)
SVM.fit(Xcombined, y)
```

...

Below is the list of terms in order of decreasing coefficients.

In [653]:
```
pd.DataFrame(list(zip(SVM.coef_.tolist()[0],featurenames))).sort_values(0, ascending
```

Out[653]:

	0	1
3437	3.135486	racism
4839	2.831291	white
2271	2.434019	jews
3438	2.385982	racist
3784	2.385048	shitty
2697	2.305447	marijuana
4844	2.284568	whites
1354	2.283586	equality
1166	2.259757	discrimination
2393	2.207019	latest
97	2.191069	abuse

k Nearest Neighbors

In [654]:
```
kNN = sklearn.neighbors.KNeighborsClassifier()
Grid = {'n_neighbors':[5, 10, 50, 100, 150]}
kNNmodel = sklearn.model_selection.GridSearchCV(kNN,param_grid=Grid,scoring='f1',cv=
kNNmodel.fit(Xcombined,y)
for x in list(zip(kNNmodel.cv_results_['mean_test_score'], kNNmodel.cv_results_['par
    print(x)
print(kNNmodel.best_params_)
```

```
C:\Users\Nolan\Anaconda3\lib\site-packages\sklearn\metrics\classification.py:114
3: UndefinedMetricWarning: F-score is ill-defined and being set to 0.0 due to no
predicted samples.
  'precision', 'predicted', average, warn_for)
C:\Users\Nolan\Anaconda3\lib\site-packages\sklearn\metrics\classification.py:114
3: UndefinedMetricWarning: F-score is ill-defined and being set to 0.0 due to no
predicted samples.
  'precision', 'predicted', average, warn_for)
C:\Users\Nolan\Anaconda3\lib\site-packages\sklearn\metrics\classification.py:114
3: UndefinedMetricWarning: F-score is ill-defined and being set to 0.0 due to no
predicted samples.
  'precision', 'predicted', average, warn_for)
C:\Users\Nolan\Anaconda3\lib\site-packages\sklearn\metrics\classification.py:114
3: UndefinedMetricWarning: F-score is ill-defined and being set to 0.0 due to no
predicted samples.
  'precision', 'predicted', average, warn_for)
C:\Users\Nolan\Anaconda3\lib\site-packages\sklearn\metrics\classification.py:114
3: UndefinedMetricWarning: F-score is ill-defined and being set to 0.0 due to no
predicted samples.
```

Overall, nearest neighbors performs poorly in terms of F1 score. Lower k leads to higher F1 score, but such small k values could lead to overfitting.

Random Forest

In [655]:
```
RF = sklearn.ensemble.RandomForestClassifier()
Grid = {'n_estimators':[10, 50, 100, 200, 300],'min_samples_split':[2, 5, 10, 20]}
RFmodel = sklearn.model_selection.GridSearchCV(RF,param_grid=Grid,scoring='f1',cv=3)
RFmodel.fit(Xcombined,y)
for x in list(zip(RFmodel.cv_results_['mean_test_score'], RFmodel.cv_results_['param
    print(x)
print(RFmodel.best_params_)
```

```
(0.6097183427395928, {'min_samples_split': 2, 'n_estimators': 10})
(0.6267688904821788, {'min_samples_split': 2, 'n_estimators': 50})
(0.636488853264342, {'min_samples_split': 2, 'n_estimators': 100})
(0.63294662745829, {'min_samples_split': 2, 'n_estimators': 200})
(0.6338501099881841, {'min_samples_split': 2, 'n_estimators': 300})
(0.6159185473198057, {'min_samples_split': 5, 'n_estimators': 10})
(0.6299304857042703, {'min_samples_split': 5, 'n_estimators': 50})
(0.6308463055875501, {'min_samples_split': 5, 'n_estimators': 100})
(0.6283244255805156, {'min_samples_split': 5, 'n_estimators': 200})
(0.6290517406491568, {'min_samples_split': 5, 'n_estimators': 300})
(0.614351604336635, {'min_samples_split': 10, 'n_estimators': 10})
(0.6269441193019829, {'min_samples_split': 10, 'n_estimators': 50})
(0.6236530911503566, {'min_samples_split': 10, 'n_estimators': 100})
(0.6256484831800544, {'min_samples_split': 10, 'n_estimators': 200})
(0.6258832780415612, {'min_samples_split': 10, 'n_estimators': 300})
(0.6083329819436214, {'min_samples_split': 20, 'n_estimators': 10})
(0.6188103379964345, {'min_samples_split': 20, 'n_estimators': 50})
(0.6252700748243349, {'min_samples_split': 20, 'n_estimators': 100})
(0.6191439331792497, {'min_samples_split': 20, 'n_estimators': 200})
(0.6220176044621004, {'min_samples_split': 20, 'n_estimators': 300})
{'min_samples_split': 2, 'n_estimators': 100}
```

The F1 value for the random forest model is optimized for fairly large forests of deep trees. The best F1 values are slightly better than the linear SVM model, suggesting the decision boundary is slightly nonlinear. This may be expected given that hate speech was defined as being sexist or racist, which could result in a bimodal distribution.

In [656]:
```
RF = sklearn.ensemble.RandomForestClassifier(n_estimators=100)
RF.fit(Xcombined, y)
```

...

Below is the list of tokens in order of decreasing importance.

In [657]:
```
pd.DataFrame(list(zip(RF.feature_importances_.tolist(),featurenames))).sort_values(0
```
Out[657]:

	0	1
4839	0.017237	white
6872	0.015947	trump
7044	0.015758	sentiment
3437	0.015229	racism
3438	0.014253	racist
4475	0.013432	user
4379	0.012055	trump
7045	0.011921	subjectivity
485	0.009176	black
4873	0.007607	women
194	0.006004	allahs oil

Gradient Boosted Tree

In [658]:
```
GB = sklearn.ensemble.GradientBoostingClassifier()
Grid = {'n_estimators':[10, 50, 100, 200, 300],'min_samples_split':[2, 5, 10, 20]}
GBmodel = sklearn.model_selection.GridSearchCV(GB,param_grid=Grid,scoring='f1',cv=3)
GBmodel.fit(Xcombined,y)
for x in list(zip(GBmodel.cv_results_['mean_test_score'], GBmodel.cv_results_['param
    print(x)
print(GBmodel.best_params_)
```

```
(0.1261345084158636, {'min_samples_split': 2, 'n_estimators': 10})
(0.38445519731203703, {'min_samples_split': 2, 'n_estimators': 50})
(0.4635884745486337, {'min_samples_split': 2, 'n_estimators': 100})
(0.5155631232937532, {'min_samples_split': 2, 'n_estimators': 200})
(0.5367833300920853, {'min_samples_split': 2, 'n_estimators': 300})
(0.12610461803132303, {'min_samples_split': 5, 'n_estimators': 10})
(0.38777550235604763, {'min_samples_split': 5, 'n_estimators': 50})
(0.457962710256974, {'min_samples_split': 5, 'n_estimators': 100})
(0.5107311901352537, {'min_samples_split': 5, 'n_estimators': 200})
(0.5378043752042742, {'min_samples_split': 5, 'n_estimators': 300})
(0.1261345084158636, {'min_samples_split': 10, 'n_estimators': 10})
(0.3909332953922089, {'min_samples_split': 10, 'n_estimators': 50})
(0.46064941627292294, {'min_samples_split': 10, 'n_estimators': 100})
(0.5124585839678684, {'min_samples_split': 10, 'n_estimators': 200})
(0.5363092060805205, {'min_samples_split': 10, 'n_estimators': 300})
(0.12610461803132303, {'min_samples_split': 20, 'n_estimators': 10})
(0.38969792253541685, {'min_samples_split': 20, 'n_estimators': 50})
(0.4519501314716337, {'min_samples_split': 20, 'n_estimators': 100})
(0.5085676888285633, {'min_samples_split': 20, 'n_estimators': 200})
(0.5364580259377819, {'min_samples_split': 20, 'n_estimators': 300})
{'min_samples_split': 5, 'n_estimators': 300}
```

In [659]:

```
Grid = {'n_estimators':[300],'min_samples_split':[5],'learning_rate':[0.1, 0.2, 0.3,
GBmodel = sklearn.model_selection.GridSearchCV(GB,param_grid=Grid,scoring='f1',cv=3)
GBmodel.fit(Xcombined,y)
for x in list(zip(GBmodel.cv_results_['mean_test_score'], GBmodel.cv_results_['param
    print(x)
print(GBmodel.best_params_)
```

```
(0.5383138976750622, {'learning_rate': 0.1, 'min_samples_split': 5, 'n_estimators':
300})
(0.5736574484663055, {'learning_rate': 0.2, 'min_samples_split': 5, 'n_estimators':
300})
(0.5756903400548373, {'learning_rate': 0.3, 'min_samples_split': 5, 'n_estimators':
300})
(0.570228224191105, {'learning_rate': 0.5, 'min_samples_split': 5, 'n_estimators':
300})
{'learning_rate': 0.3, 'min_samples_split': 5, 'n_estimators': 300}
```

In [660]:

```
Grid = {'n_estimators':[300],'min_samples_split':[5],'learning_rate':[0.3],'max_dept
GBmodel = sklearn.model_selection.GridSearchCV(GB,param_grid=Grid,scoring='f1',cv=3)
GBmodel.fit(Xcombined,y)
for x in list(zip(GBmodel.cv_results_['mean_test_score'], GBmodel.cv_results_['param
    print(x)
print(GBmodel.best_params_)
```

```
(0.5695605275726003, {'learning_rate': 0.3, 'max_depth': 2, 'min_samples_split': 5,
'n_estimators': 300})
(0.5682547502753909, {'learning_rate': 0.3, 'max_depth': 3, 'min_samples_split': 5,
'n_estimators': 300})
(0.5731522071594352, {'learning_rate': 0.3, 'max_depth': 4, 'min_samples_split': 5,
'n_estimators': 300})
(0.5860009995722448, {'learning_rate': 0.3, 'max_depth': 5, 'min_samples_split': 5,
'n_estimators': 300})
(0.5821966309772971, {'learning_rate': 0.3, 'max_depth': 6, 'min_samples_split': 5,
'n_estimators': 300})
{'learning_rate': 0.3, 'max_depth': 5, 'min_samples_split': 5, 'n_estimators': 300}
```

The gradient boosted tree model does not perform as well as the random forest model or the support vector machine model. The small fraction of tweets that are labeled as hate speech may indicate very small cluster(s) in feature space that are associated with hate speech, which may require deeper trees (involving more features) to model. This may explain why the random forest performs better than gradient boosted trees.

In [661]:

```
GB = sklearn.ensemble.GradientBoostingClassifier(n_estimators=300, learning_rate = 0
GB.fit(Xcombined, y)
```

...

The features in order of decreasing importance are listed below.

In [662]:
```
pd.DataFrame(list(zip(GB.feature_importances_.tolist(),featurenames))).sort_values(0
```

Out[662]:

	0	1
6872	0.041407	trump
4839	0.034758	white
4909	0.034215	wow
5030	0.033538	allahs oil
3875	0.033044	smh
3438	0.027937	racist
2464	0.025991	libtard
7044	0.022907	sentiment
3437	0.020372	racism
4475	0.018178	user
485	0.017904	black

Overall, the random forest model performs the best in cross-validation, so it will be used to predict the test set classification.

The F1 score of 0.64 is mediocre. Improvements may be likely possible with semantic analysis (rather than using only the bag of words model). However, as such analysis is quite involved, it will be left for another time.

Processing and Prediction for the Test Dataset

First, based on the findings during feature extraction, the word "libtard" should be included in the wordlist before processing. And then the dictionary of positions in the word list must be re-created.

In [634]:
```
wordlist = wordlist + ['libtard']
wordlist.sort(key=str.lower)

truncatedwordlist = wordlist[:]
for i in range(len(truncatedwordlist)):
    truncatedwordlist[i]=truncatedwordlist[i][:2]

positionlist = []
for x in pairlist:
    if x in truncatedwordlist:
        positionlist.append(truncatedwordlist.index(x))
    else:
        positionlist.append(None)
for i in range(len(positionlist)):
    if positionlist[i]==None:
        positionlist[i]=positionlist[i-1]

pairdict = dict(zip(pairlist, positionlist))
```

Next, the test data frame will be pre-processed.

In [636]:
```
testdf.tweet = testdf.tweet.apply(ftfy.fix_encoding)
testdf['tokens'] = testdf.tweet.apply(nltk.tokenize.TweetTokenizer().tokenize)
testdf['hashtags'] = testdf.tokens.apply(lambda x: [a.replace('#','') for a in x if
```

In [638]:
```
testdf.hashtags = testdf.hashtags.apply(lambda tags: [hashtagprocess(tag) for tag in
```

In [639]:
```
testdf.hashtags.apply(lambda tags:' #'.join([' '.join(tag) for tag in tags])).to_csv
```

In [640]:
```
testdf.tokens = testdf.tokens.apply(lambda tokens: [tokenprocess(token) if token[0]!
```

In [641]:
```
testdf.tokens.apply(lambda tags:' '.join(tags)).to_csv('C:\Datasets\ProcessedTestTok
```

In [642]:
```
testdf['correcttokens'] = testdf.apply(lambda row: subhashtag(row.tokens, row.hashta
testdf['corrected']= testdf.correcttokens.apply(lambda tokens: ' '.join(tokens))
testdf['hashtokens']= testdf.hashtags.apply(mergetags)
testdf['hashtext']= testdf.hashtokens.apply(lambda tokens: ' '.join(tokens))
```

Then features are extracted for the test data set.

In [644]:
```
testdf['sentiment']= testdf.corrected.apply(lambda text: sentiment(text)[0])
testdf['subjectivity']= testdf.corrected.apply(lambda text: sentiment(text)[1])
testdf['hashsentiment']=testdf.hashtext.apply(lambda text: sentiment(text)[0])
testdf['hashsubjectivity']= testdf.hashtext.apply(lambda text: sentiment(text)[1])
```

In [645]:
```
testdf.head(30)
```

Out[645]:

	id	tweet	tokens	hashtags	co
0	31963	#studiolife #aislife #requires #passion #dedication #willpower to find #newmaterials…	[#studiolife, #aislife, #requires, #passion, #dedication, #willpower, to, find, #newmaterials, …]	[[studio, life], [ais, life], [requires], [passion], [dedication], [willpower], [new, materials]]	[studio, requir dedication, v find, new, n
1	31964	@user #white #supremacists want everyone to see the new ‘ #birds’ #movie — and here’s why	[user, #white, #supremacists, want, everyone, to, see, the, new, ‘, #birds, ’, #movie, —, and, here, ’, s, why]	[[white], [supremacists], [birds], [movie]]	suprema everyone, new, ‘, bir —, and, he
2	31965	safe ways to heal your #acne!! #altwaystoheal #healthy #healing!!	[safe, ways, to, heal, your, #acne, !, !, #altwaystoheal, #healthy, #healing, !, !]	[[acne], [alt, ways, to, heal], [healthy], [healing]]	[safe, wa your, a ways, to, h

In [646]:
```
Xtest = vectorizer.transform(testdf['corrected'])
Xtesthash = vectorizerhash.transform(testdf['hashtext'])
Xtestcombined = scipy.sparse.hstack([Xtest, Xtesthash, testdf.iloc[:,8:]])
```

Finally the predictions can be made.

In [663]:
```
ytest = RF.predict(Xtestcombined)
```

In [666]:
```
submissiondf = pd.DataFrame(testdf.id,columns=['id'])
submissiondf['label'] = ytest
submissiondf.to_csv('C:\Datasets\TwitterSubmission.csv')
```