Feature Selection Using TFIDF and NGrams

In [1]:
```
import pandas as pd
import numpy as np
from matplotlib import pyplot as plt
import sqlite3
%matplotlib inline
```

In [2]:
```
#import df
con = sqlite3.connect('twitter_hate.db')
with sqlite3.connect('twitter_hate.db') as con:
    df = pd.read_sql_query("SELECT * FROM tweets_nlp", con)
```

In [3]:
```
df
```

Out[3]:

	index	count	hate_speech	offensive_language	neither	class	tweet	tweet_clean	tweet_lemma	tweet_nouns	tweet_s
0	17	3	1	2	0	1	" bitch who do you love "	bitch love	bitch love	bitch love	
1	23	3	0	3	0	1	" fuck no that bitch dont even suck dick " &#1...	fuck bitch dont even suck dick ker...	fuck bitch do not even suck dick ...	bitch dick kermit videos bout fuck	
2	38	3	0	2	1	1	" lames crying over hoes thats tears of a clown "	lames crying hoes thats tears clown	lame cry hoe that s tear clown	lame hoe s tear clown	
3	59	3	0	3	0	1	"..All I wanna do is get money and fuck model ...	all i wanna get money fuck model bitches r...	all i wanna get money fuck model bitch ru...	wanna money fuck model bitch russell simmons	
							"@ARIZZLEINDACUT:	mentionhere females	mentionhere female think	mentionhere	

In [4]:
```
df.iloc[2827]['tweet_lemma']
```

Out[4]:
```
'tears     mentionhere   hashtaghere rt mentionhere mentionhere call sweetie fucking retard'
```

In [5]:

```python
#remove mentions, urls, hashtags, ;&, and 'rt' and other punctuation. keep a count of mentions, urls, ha
tweets = df['tweet_lemma']

mentions = []
urls = []
hashtags = []
i = 0
for tweet in tweets:
    tweet = tweet.split()
    mentions.append(tweet.count('mentionhere')+tweet.count('mentionhere:')+tweet.count('"mentionhere:')+
    urls.append(tweet.count('urlhere'))
    hashtags.append(tweet.count('hashtaghere'))
    tweet = [token for token in tweet if token not in [';&','']]
    tweet = [token for token in tweet if token not in ['&#;mentionhere:','mentionhere:','"mentionhere:',
    tweet = " ".join(tweet)
    tweets[i] = tweet
    i += 1

df['tweet_no_others'] = tweets
df['mention_count'] = mentions
df['url_count'] = urls
df['hashtag_count'] = hashtags
```

```
/Users/Colin/opt/anaconda3/envs/machinelearning/lib/python3.7/site-packages/ipykernel_launcher.py:16:
SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexin
g.html#returning-a-view-versus-a-copy (https://pandas.pydata.org/pandas-docs/stable/user_guide/indexin
g.html#returning-a-view-versus-a-copy)
  app.launch_new_instance()
```

In [6]:

```python
df.iloc[2827]['tweet_no_others']
```

Out[6]:

```
'tears call sweetie fucking retard'
```

In [7]:

```python
sum(count > 0 for count in mentions)
```

Out[7]:

```
1706
```

In [8]:

```
sum(count > 0 for count in urls)
```

Out[8]:

```
268
```

In [9]:

```
sum(count > 0 for count in hashtags)
```

Out[9]:

```
209
```

In [10]:

```
#just to check, find tweets with at least one of each count
mention_bool = df['mention_count'] > 0
url_bool = df['url_count'] > 0
hashtag_bool = df['hashtag_count'] > 0

df[mention_bool & url_bool | mention_bool & hashtag_bool | url_bool & hashtag_bool]
```

Out[10]:

	index	count	hate_speech	offensive_language	neither	class	tweet	tweet_clean	tweet_lemma	tweet_nouns	tı
4	62	3	0	3	0	1	"@ARIZZLEINDACUT: Females think dating a pussy...	mentionhere females think dating pussy cute n...	female think date pussy cute now stuff make pussy	mentionhere female cute stuff	
9	92	3	1	2	0	1	"@CaelanG15: "@22EdHam: @CaelanG15 that nigga ...	mentionhere mentionhere mentionhere nigga e...	nigga eat hoe lol hell yea lol john paul nigga...	mentionhere mentionhere mentionhere nigga hoe ...	
10	96	3	0	3	0	1	"@CauseWereGuys: On my way to fuck yo bitch ht...	mentionhere on way fuck yo bitch urlhere ye...	on way fuck yo bitch year old	mentionhere way fuck yo bitch year	
12	110	3	3	0	0	0	"@DevilGrimz: @VigxRArts you're fucking gay, b...	mentionhere mentionhere fucking gay blacklis...	fuck gay blacklist hoe hold anyway	mentionhere mentionhere gay hoe	
17	184	3	3	0	0	0	"@MarkRoundtreeJr: LMFAOOOO I HATE BLACK PEOPL...	mentionhere lmfaoooo i hate black people urlh...	lmfaoooo i hate black people this there s blac...	mentionhere lmfaoooo people people nigger	
...	...	...	...	...	...	...	...	...	...	...	
2769	23897	3	2	1	0	0	harm this pussy instead RT @ABC7: missing 26-y...	harm pussy instead rt mentionhere missing yr...	harm pussy instead miss yr old usc medical stu...	harm pussy mentionhere yr student tuesday	

	index	count	hate_speech	offensive_language	neither	class	tweet	tweet_clean	tweet_lemma	tweet_nouns	t
2787	24179	3	0	3	0	1	lol RT @_mykall: when you @ bae game &amp; an ...	lol rt mentionhere bae game amp unknown h...	lol bae game unknown hoe scream name loud asfck	lol rt mentionhere bae game amp hoe name loud ...	
2804	24314	3	2	1	0	0	omg RT @SaddyBey: Fat bitch. What's her @? htt...	omg rt mentionhere fat bitch what s urlhere	omg fat bitch what s	rt mentionhere fat bitch s	
2814	24410	3	2	1	0	0	she pooted &#8220;@Not1FuckisGiven: Either You...	pooted mentionhere either young thug gay ...	poote either young thug gay bitch poote	mentionhere thug gay bitch	
2827	24485	3	1	2	0	1	tears &#8220;@TheDouch3: #RelationshipGoals RT...	tears mentionhere hashtaghere rt mentionhe...	tears call sweetie fucking retard	tears mentionhere rt mentionhere mentionhere c...	

266 rows × 18 columns

```
In [11]: corpus = df['tweet_no_others']
```

In [12]:
```
from sklearn.feature_extraction.text import CountVectorizer

ngram = CountVectorizer(ngram_range=(2,2))
ngram_matrix = ngram.fit_transform(corpus)

ngram_matrix = ngram_matrix.toarray()
vocab = ngram.get_feature_names()
ngrams_df = pd.DataFrame(ngram_matrix, columns=vocab)
ngrams_df
```

Out[12]:

	aa	aaaaaaaaand	aap	aaron	aaronmacgruder	ability	abortion	about	abraham	absolute	...	zimmerman	zimmerman	zimme
	lol	begin	maoist	weak	stuff	block	get	act	lincoln	pussy		arrest	comin	cr
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	
2	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	
4	0	0	0	0	0	0	0	0	0	0	...	0	0	
...	...	...	...	...	...	...	...	...	...	...	...	...	...	
2855	0	0	0	0	0	0	0	0	0	0	...	0	0	
2856	0	0	0	0	0	0	0	0	0	0	...	0	0	
2857	0	0	0	0	0	0	0	0	0	0	...	0	0	
2858	0	0	0	0	0	0	0	0	0	0	...	0	0	
2859	0	0	0	0	0	0	0	0	0	0	...	0	0	

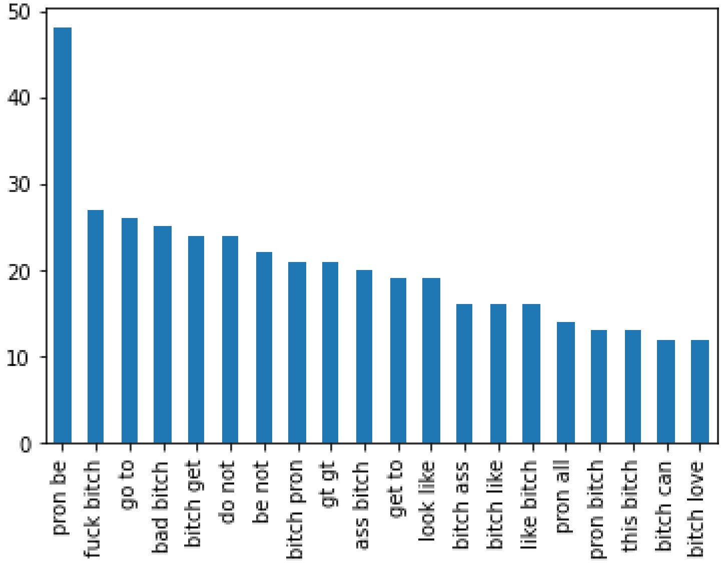
2860 rows × 16247 columns

In [13]:
```
ngrams_df['class'] = df['class']
```

In [14]:
```
ngrams_offensive = ngrams_df[ngrams_df['class'] == 1]
ngrams_hate = ngrams_df[ngrams_df['class'] == 0]
ngrams_offensive = ngrams_offensive.drop('class', axis='columns')
ngrams_hate = ngrams_hate.drop('class', axis='columns')
```

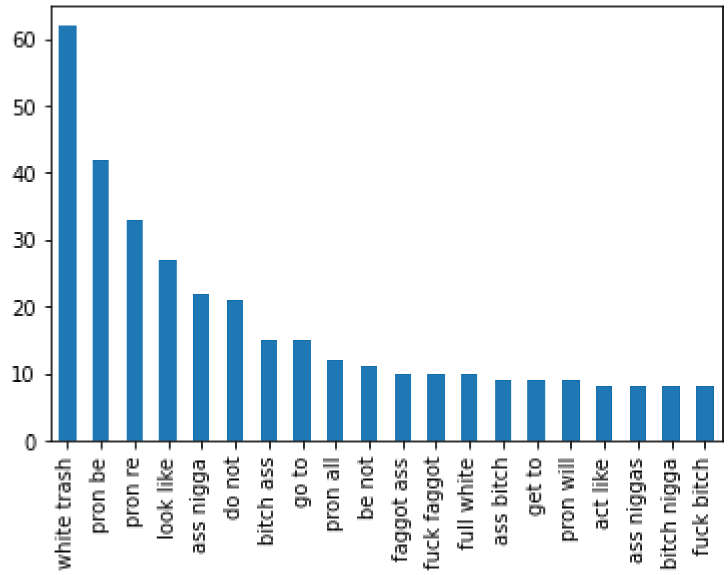
In [15]:
```
ng_count_off=ngrams_offensive.sum()
ng_off_largest = ng_count_off.nlargest(20)
ng_off_largest.plot(kind='bar')
```

Out[15]:
```
<AxesSubplot:>
```



```
In [16]: ng_count_hate=ngrams_hate.sum()
         ng_hate_largest = ng_count_hate.nlargest(20)
         ng_hate_largest.plot(kind='bar')
```

```
Out[16]: <AxesSubplot:>
```



In [17]:

```
ngram_3 = CountVectorizer(ngram_range=(3,3))
ngram_3_matrix = ngram_3.fit_transform(corpus)

ngram_3_matrix = ngram_3_matrix.toarray()
vocab = ngram_3.get_feature_names()
ngrams_3_df = pd.DataFrame(ngram_3_matrix, columns=vocab)
ngrams_3_df
```

Out[17]:

	aaaaaaaaand begin fuck	aap maoist terrorist	aaron weak last	aaronmacgruder stuff blow	abortion get cemetery	about act color	abraham lincoln quote	absolute pussy scared	absolve end today	abt bitch face	...	zimmerman arrest do	zimmerman comin yo
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
2	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
4	0	0	0	0	0	0	0	0	0	0	...	0	0
...	...	...	...	...	...	...	...	...	...	...	...	...	...
2855	0	0	0	0	0	0	0	0	0	0	...	0	0
2856	0	0	0	0	0	0	0	0	0	0	...	0	0
2857	0	0	0	0	0	0	0	0	0	0	...	0	0
2858	0	0	0	0	0	0	0	0	0	0	...	0	0
2859	0	0	0	0	0	0	0	0	0	0	...	0	0

2860 rows × 16725 columns

In [18]:

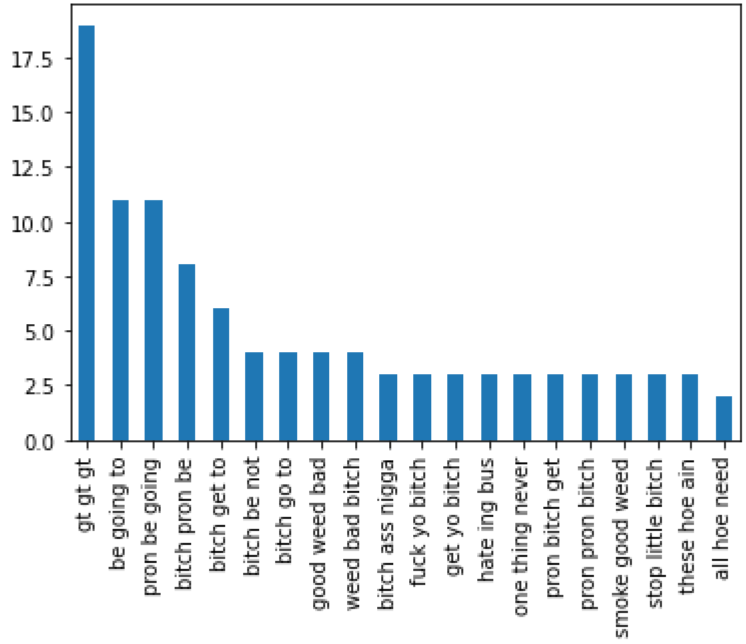
```
ngrams_3_df['class'] = df['class']
```

In [19]:

```
ngrams_3_offensive = ngrams_3_df[ngrams_df['class'] == 1]
ngrams_3_hate = ngrams_3_df[ngrams_df['class'] == 0]
ngrams_3_offensive = ngrams_3_offensive.drop('class', axis='columns')
ngrams_3_hate = ngrams_3_hate.drop('class', axis='columns')
```

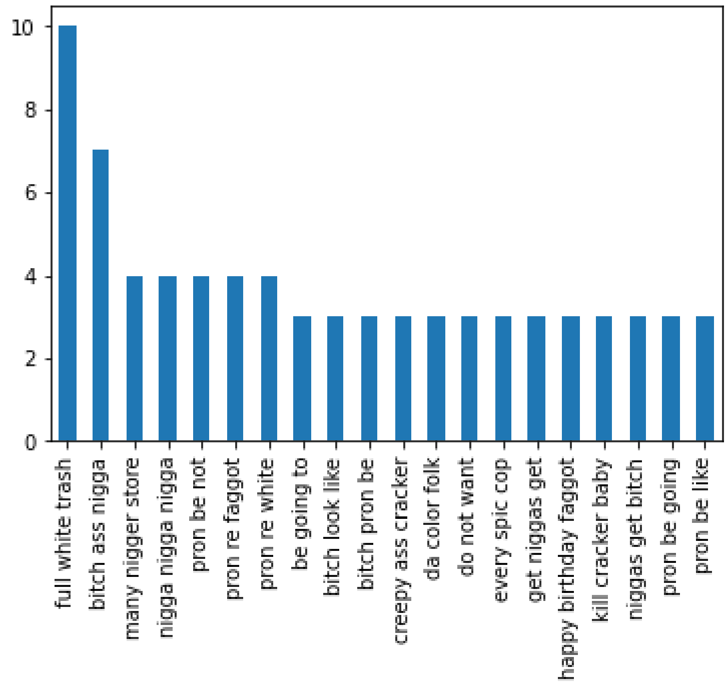
In [20]:
```
ng3_count_off=ngrams_3_offensive.sum()
ng3_off_largest = ng3_count_off.nlargest(20)
ng3_off_largest.plot(kind='bar')
```

Out[20]:
```
<AxesSubplot:>
```



```
In [21]: ng3_count_hate=ngrams_3_hate.sum()
         ng3_hate_largest = ng3_count_hate.nlargest(20)
         ng3_hate_largest.plot(kind='bar')
```

```
Out[21]: <AxesSubplot:>
```



In [22]:
```
from sklearn.feature_extraction.text import TfidfVectorizer

tfidfv = TfidfVectorizer(min_df=0., max_df=1., use_idf=True)
tfidfv_matrix = tfidfv.fit_transform(corpus)
tfidfv_matrix = tfidfv_matrix.toarray()

vocab = tfidfv.get_feature_names()
tfidf_df = pd.DataFrame(np.round(tfidfv_matrix, 2), columns=vocab)
tfidf_df
```

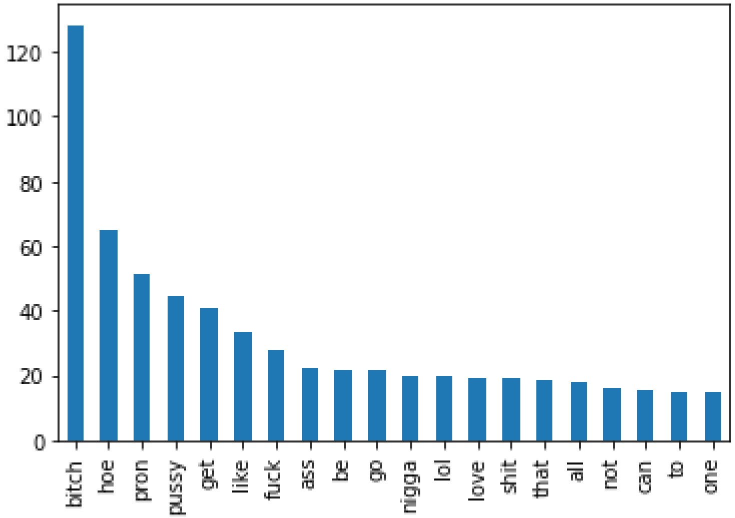
Out[22]:
```
       aa  aaaaaaaaand  aap  aaron  aaronmacgruder   ab  ability  abortion  about  abraham  ...  zimmerman  zimmy  zion  zionist  zippe
   0  0.0          0.0  0.0    0.0             0.0  0.0      0.0       0.0    0.0      0.0  ...        0.0    0.0   0.0      0.0
   1  0.0          0.0  0.0    0.0             0.0  0.0      0.0       0.0    0.0      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      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      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      0.0  ...        0.0    0.0   0.0      0.0
 ...  ...          ...  ...    ...             ...  ...      ...       ...    ...      ...  ...        ...    ...   ...      ...
2855  0.0          0.0  0.0    0.0             0.0  0.0      0.0       0.0    0.0      0.0  ...        0.0    0.0   0.0      0.0
2856  0.0          0.0  0.0    0.0             0.0  0.0      0.0       0.0    0.0      0.0  ...        0.0    0.0   0.0      0.0
2857  0.0          0.0  0.0    0.0             0.0  0.0      0.0       0.0    0.0      0.0  ...        0.0    0.0   0.0      0.0
2858  0.0          0.0  0.0    0.0             0.0  0.0      0.0       0.0    0.0      0.0  ...        0.0    0.0   0.0      0.0
2859  0.0          0.0  0.0    0.0             0.0  0.0      0.0       0.0    0.0      0.0  ...        0.0    0.0   0.0      0.0

2860 rows × 4451 columns
```

In [23]:
```
tfidf_df['class'] = df['class']
```

In [24]:
```
tfidf_offensive = tfidf_df[tfidf_df['class'] == 1]
tfidf_hate = tfidf_df[tfidf_df['class'] == 0]
tfidf_offensive = tfidf_offensive.drop('class', axis='columns')
tfidf_hate = tfidf_hate.drop('class', axis='columns')
```

In [25]:
```
tf_count_off=tfidf_offensive.sum()
tf_off_largest = tf_count_off.nlargest(20)
tf_off_largest.plot(kind='bar')
```

Out[25]:
```
<AxesSubplot:>
```

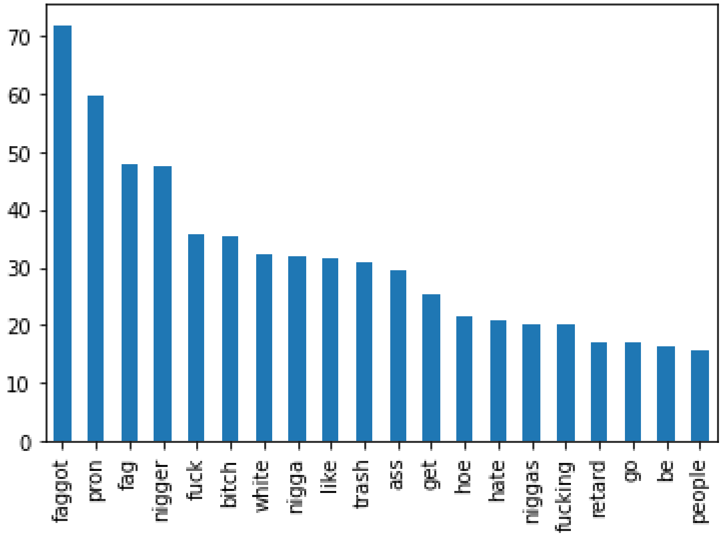


In [26]:

```
tf_count_hate=tfidf_hate.sum()
tf_hate_largest = tf_count_hate.nlargest(20)
tf_hate_largest.plot(kind='bar')
```

Out[26]:

```
<AxesSubplot:>
```



Aggregate the 3 different dataframes and try out some modeling

In [27]:

```
#add num_tokens, mention_count, url_count, hashtag_count

new_columns = ['num_tokens', 'mention_count', 'url_count', 'hashtag_count']

for col in new_columns:
    ngrams_df[col] = df[col]
    ngrams_3_df[col] = df[col]
    tfidf_df[col] = df[col]
```

Start with ngram, n=2

In [28]:

```
X = ngrams_df.drop('class', axis='columns')
y = ngrams_df['class'].astype(int)
```

In [29]:

```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, random_state=42)
```

In [30]:

```
from sklearn.pipeline import Pipeline
from sklearn.model_selection import KFold, GridSearchCV
from sklearn.naive_bayes import GaussianNB
from sklearn.feature_selection import SelectFromModel
from sklearn.linear_model import LogisticRegression
from sklearn import tree
```

Logistic Regression - ngrams (n=2)

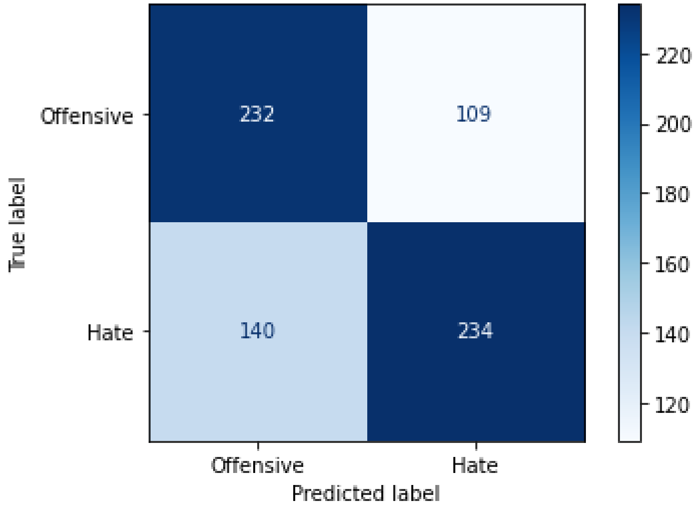
In [31]:

```
lgr = LogisticRegression(max_iter=1000)
param_grid = [{}]
lgr_n_2 = GridSearchCV(lgr,
                          param_grid,
                          cv=KFold(n_splits=5).split(X_train, y_train),
                          verbose=2)
y_preds_lgr_n_2 = lgr_n_2.fit(X_train, y_train).predict(X_test)
```

```
Fitting 5 folds for each of 1 candidates, totalling 5 fits
[CV]  ................................................................

[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.

[CV] ................................................. , total=   9.3s
[CV]  ................................................................

[Parallel(n_jobs=1)]: Done   1 out of   1 | elapsed:    9.3s remaining:    0.0s

[CV] ................................................. , total=   4.5s
[CV]  ................................................................
[CV] ................................................. , total=   7.8s
[CV]  ................................................................
[CV] ................................................. , total=   4.6s
[CV]  ................................................................
[CV] ................................................. , total=   6.6s

[Parallel(n_jobs=1)]: Done   5 out of   5 | elapsed:   32.9s finished
```

```
In [32]: from sklearn.metrics import plot_confusion_matrix
         class_names = ['Offensive', 'Hate']
         plot_confusion_matrix(lgr_n_2, X_test, y_test, cmap=plt.cm.Blues, display_labels = class_names, values_f
```

```
Out[32]: <sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay at 0x1a7fa48c10>
```



In [33]:

```
from sklearn.metrics import classification_report
report_lgr_n_2 = classification_report(y_test, y_preds_lgr_n_2)
print(report_lgr_n_2)
```

```
              precision    recall  f1-score   support

           0       0.62      0.68      0.65       341
           1       0.68      0.63      0.65       374

    accuracy                           0.65       715
   macro avg       0.65      0.65      0.65       715
weighted avg       0.65      0.65      0.65       715
```

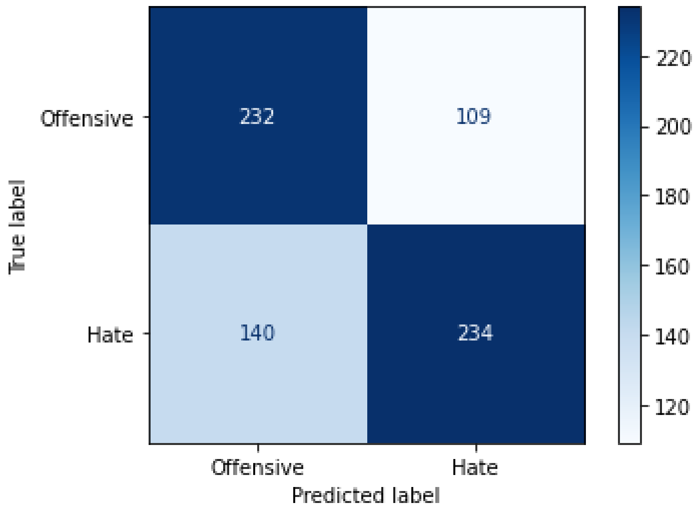
Decision Tree - ngram (n=2)

In [34]:
```
dec_tree = tree.DecisionTreeClassifier()
dec_tree_n_2 = GridSearchCV(lgr,
                           param_grid,
                           cv=KFold(n_splits=5).split(X_train, y_train),
                           verbose=2)
y_preds_dec_tree_n_2 = dec_tree_n_2.fit(X_train, y_train).predict(X_test)
```

```
Fitting 5 folds for each of 1 candidates, totalling 5 fits
[CV]  ................................................................

[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.

[CV] ................................................. , total=   8.7s
[CV]  ................................................................

[Parallel(n_jobs=1)]: Done   1 out of   1 | elapsed:    8.7s remaining:    0.0s

[CV] ................................................. , total=   4.8s
[CV]  ................................................................
[CV] ................................................. , total=   8.2s
[CV]  ................................................................
[CV] ................................................. , total=   4.7s
[CV]  ................................................................
[CV] ................................................. , total=   6.7s

[Parallel(n_jobs=1)]: Done   5 out of   5 | elapsed:   33.1s finished
```

```
In [35]: plot_confusion_matrix(dec_tree_n_2, X_test, y_test, cmap=plt.cm.Blues, display_labels = class_names, val
```

```
Out[35]: <sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay at 0x1a5cd6e690>
```



In [36]:

```
report_dec_tree_n_2 = classification_report(y_test, y_preds_dec_tree_n_2)
print(report_dec_tree_n_2)
```

```
              precision    recall  f1-score   support

           0       0.62      0.68      0.65       341
           1       0.68      0.63      0.65       374

    accuracy                           0.65       715
   macro avg       0.65      0.65      0.65       715
weighted avg       0.65      0.65      0.65       715
```

Naive Bayes - ngram (n=2)

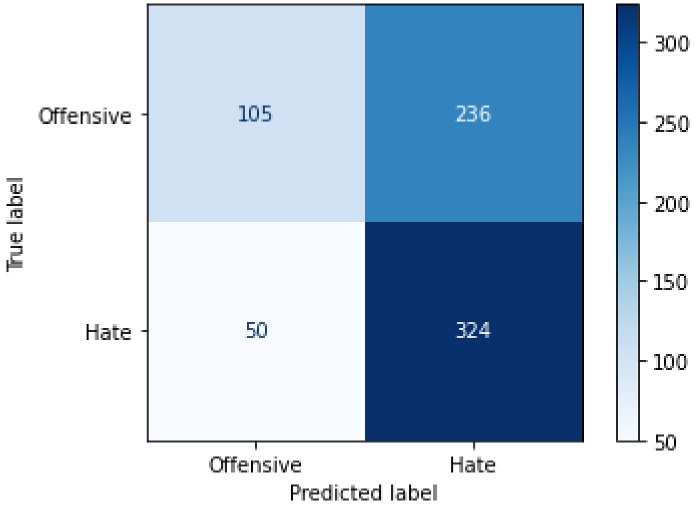
In [37]:

```
gnb = GaussianNB()
param_grid = [{}]
gnb_n_2 = GridSearchCV(gnb,
                           param_grid,
                           cv=KFold(n_splits=5).split(X_train, y_train),
                           verbose=2)
y_preds_gnb_n_2 = gnb_n_2.fit(X_train, y_train).predict(X_test)
```

```
Fitting 5 folds for each of 1 candidates, totalling 5 fits
[CV]  ................................................................

[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.

[CV] ................................................. , total=   1.9s
[CV]  ................................................................

[Parallel(n_jobs=1)]: Done   1 out of   1 | elapsed:    2.0s remaining:    0.0s

[CV] ................................................. , total=   2.0s
[CV]  ................................................................
[CV] ................................................. , total=   1.9s
[CV]  ................................................................
[CV] ................................................. , total=   1.8s
[CV]  ................................................................
[CV] ................................................. , total=   1.8s

[Parallel(n_jobs=1)]: Done   5 out of   5 | elapsed:    9.6s finished
```

```
In [38]: plot_confusion_matrix(gnb_n_2, X_test, y_test, cmap=plt.cm.Blues, display_labels = class_names, values_f
```

```
Out[38]: <sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay at 0x1a5ce43210>
```



In [39]:
```
report_gnb_n_2 = classification_report( y_test, y_preds_gnb_n_2)
print(report_gnb_n_2)
```

```
              precision    recall  f1-score   support

           0       0.68      0.31      0.42       341
           1       0.58      0.87      0.69       374

    accuracy                           0.60       715
   macro avg       0.63      0.59      0.56       715
weighted avg       0.63      0.60      0.56       715
```

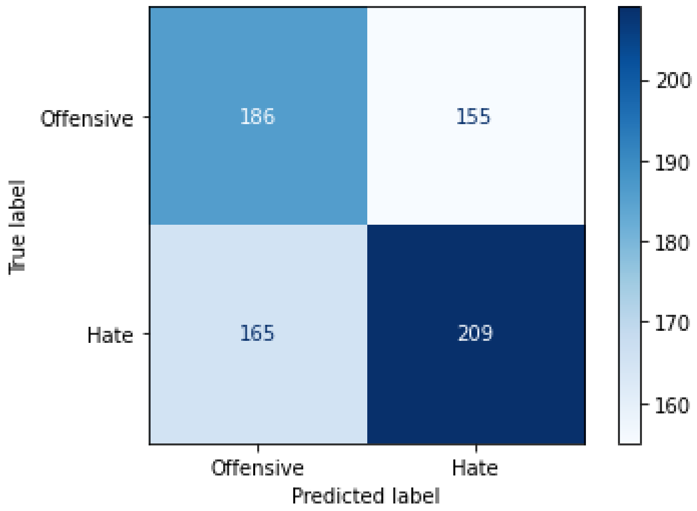
SVM - ngram (n=2)

In [40]:
```
from sklearn.svm import SVC
svc = SVC()
param_grid = [{}]
svm_n_2 = GridSearchCV(svc,
                           param_grid,
                           cv=KFold(n_splits=5).split(X_train, y_train),
                           verbose=2)
y_preds_svm_n_2 = svm_n_2.fit(X_train, y_train).predict(X_test)
```

```
Fitting 5 folds for each of 1 candidates, totalling 5 fits
[CV]  ................................................................

[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.

[CV] ................................................. , total= 1.1min
[CV]  ................................................................

[Parallel(n_jobs=1)]: Done   1 out of   1 | elapsed:  1.1min remaining:    0.0s

[CV] ................................................. , total= 1.1min
[CV]  ................................................................
[CV] ................................................. , total= 1.1min
[CV]  ................................................................
[CV] ................................................. , total= 1.1min
[CV]  ................................................................
[CV] ................................................. , total= 1.1min

[Parallel(n_jobs=1)]: Done   5 out of   5 | elapsed:  5.3min finished
```

```
In [41]: plot_confusion_matrix(svm_n_2, X_test, y_test, cmap=plt.cm.Blues, display_labels = class_names, values_f
```

```
Out[41]: <sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay at 0x1a5cc4fd50>
```



In [42]:

```
report_svm_n_2 = classification_report( y_test, y_preds_svm_n_2)
print(report_svm_n_2)
```

```
              precision    recall  f1-score   support

           0       0.53      0.55      0.54       341
           1       0.57      0.56      0.57       374

    accuracy                           0.55       715
   macro avg       0.55      0.55      0.55       715
weighted avg       0.55      0.55      0.55       715
```

ngram with n=3

In [43]:

```
X = ngrams_3_df.drop('class', axis='columns')
y = ngrams_3_df['class'].astype(int)
```

In [44]:

```
X_train, X_test, y_train, y_test = train_test_split(X, y, random_state=42)
```

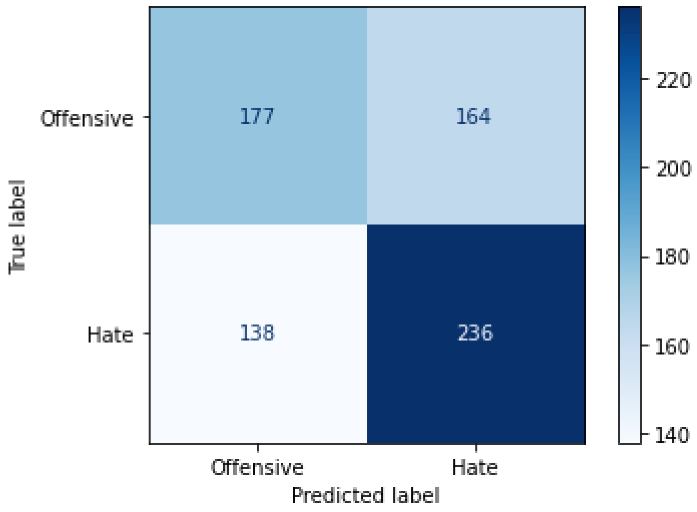
Logistic Regression - ngram (n=3)

```
In [45]: lgr_n_3 = GridSearchCV(lgr,
                                param_grid,
                                cv=KFold(n_splits=5).split(X_train, y_train),
                                verbose=2)
         y_preds_lgr_n_3 = lgr_n_3.fit(X_train, y_train).predict(X_test)
```

```
Fitting 5 folds for each of 1 candidates, totalling 5 fits
[CV]  ................................................................

[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.

[CV] ................................................. , total=   7.1s
[CV]  ................................................................

[Parallel(n_jobs=1)]: Done   1 out of   1 | elapsed:    7.2s remaining:    0.0s

[CV] ................................................. , total=   5.4s
[CV]  ................................................................
[CV] ................................................. , total=   5.9s
[CV]  ................................................................
[CV] ................................................. , total=   5.9s
[CV]  ................................................................
[CV] ................................................. , total=   4.6s

[Parallel(n_jobs=1)]: Done   5 out of   5 | elapsed:   29.1s finished
```

```
In [46]: plot_confusion_matrix(lgr_n_3, X_test, y_test, cmap=plt.cm.Blues, display_labels = class_names, values_f
```

```
Out[46]: <sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay at 0x1a5ce6e910>
```



```
In [47]: report_lgr_n_3 = classification_report( y_test, y_preds_lgr_n_3)
         print(report_lgr_n_3)
```

```
              precision    recall  f1-score   support

           0       0.56      0.52      0.54       341
           1       0.59      0.63      0.61       374

    accuracy                           0.58       715
   macro avg       0.58      0.58      0.57       715
weighted avg       0.58      0.58      0.58       715
```

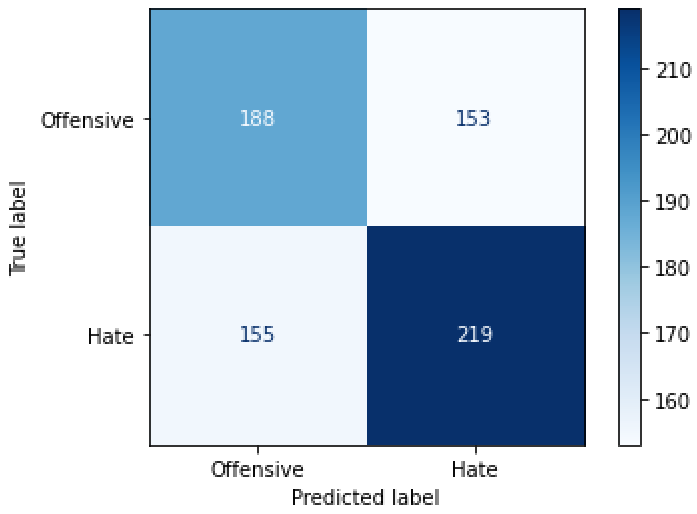
Decision Tree - ngram (n=3)

```
In [48]: dec_tree_n_3 = GridSearchCV(dec_tree,
                                     param_grid,
                                     cv=KFold(n_splits=5).split(X_train, y_train),
                                     verbose=2)
         y_preds_dec_tree_n_3 = dec_tree_n_3.fit(X_train, y_train).predict(X_test)
```

```
Fitting 5 folds for each of 1 candidates, totalling 5 fits
[CV]  ................................................................

[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.

[CV] ................................................. , total=   2.9s
[CV]  ................................................................

[Parallel(n_jobs=1)]: Done   1 out of   1 | elapsed:    2.9s remaining:    0.0s

[CV] ................................................. , total=   2.4s
[CV]  ................................................................
[CV] ................................................. , total=   2.6s
[CV]  ................................................................
[CV] ................................................. , total=   2.1s
[CV]  ................................................................
[CV] ................................................. , total=   2.5s


[Parallel(n_jobs=1)]: Done   5 out of   5 | elapsed:   12.7s finished
```

```
In [49]: plot_confusion_matrix(dec_tree_n_3, X_test, y_test, cmap=plt.cm.Blues, display_labels = class_names, val
```

```
Out[49]: <sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay at 0x1a562eba50>
```



```
In [50]: report_dec_tree_n_3 = classification_report( y_test, y_preds_dec_tree_n_3)
         print(report_dec_tree_n_3)
```

```
              precision    recall  f1-score   support

           0       0.55      0.55      0.55       341
           1       0.59      0.59      0.59       374

    accuracy                           0.57       715
   macro avg       0.57      0.57      0.57       715
weighted avg       0.57      0.57      0.57       715
```

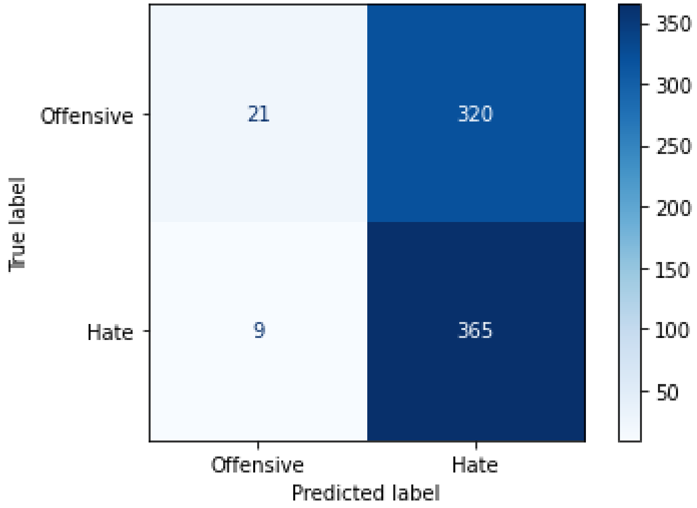
Naive Bayes - ngram (n=3)

In [51]:
```
gnb_n_3 = GridSearchCV(gnb,
                        param_grid,
                        cv=KFold(n_splits=5).split(X_train, y_train),
                        verbose=2)
y_preds_gnb_n_3 = gnb_n_3.fit(X_train, y_train).predict(X_test)
```

```
Fitting 5 folds for each of 1 candidates, totalling 5 fits
[CV]  ................................................................

[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.

[CV] ................................................. , total=   2.1s
[CV]  ................................................................

[Parallel(n_jobs=1)]: Done   1 out of   1 | elapsed:    2.1s remaining:    0.0s

[CV] ................................................. , total=   1.9s
[CV]  ................................................................
[CV] ................................................. , total=   1.8s
[CV]  ................................................................
[CV] ................................................. , total=   1.9s
[CV]  ................................................................
[CV] ................................................. , total=   1.9s


[Parallel(n_jobs=1)]: Done   5 out of   5 | elapsed:    9.7s finished
```

```
In [52]: plot_confusion_matrix(gnb_n_3, X_test, y_test, cmap=plt.cm.Blues, display_labels = class_names, values_f
```

```
Out[52]: <sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay at 0x1080869d0>
```



```
In [53]: report_gnb_n_3 = classification_report( y_test, y_preds_gnb_n_3)
         print(report_gnb_n_3)
```

```
              precision    recall  f1-score   support

           0       0.70      0.06      0.11       341
           1       0.53      0.98      0.69       374

    accuracy                           0.54       715
   macro avg       0.62      0.52      0.40       715
weighted avg       0.61      0.54      0.41       715
```

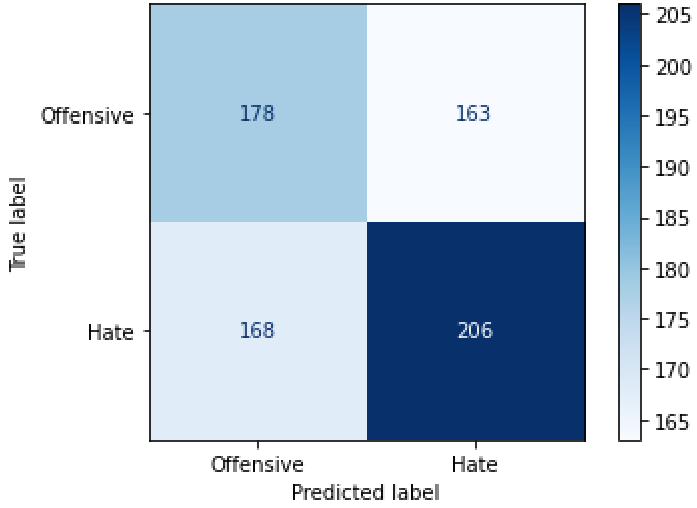
SVM - ngram (n=3)

In [54]:
```
svm_n_3 = GridSearchCV(svc,
                       param_grid,
                       cv=KFold(n_splits=5).split(X_train, y_train),
                       verbose=2)
y_preds_svm_n_3 = svm_n_3.fit(X_train, y_train).predict(X_test)
```

```
Fitting 5 folds for each of 1 candidates, totalling 5 fits
[CV]  ................................................................

[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.

[CV] ................................................. , total= 1.1min
[CV]  ................................................................

[Parallel(n_jobs=1)]: Done   1 out of   1 | elapsed:  1.1min remaining:    0.0s

[CV] ................................................. , total= 1.1min
[CV]  ................................................................
[CV] ................................................. , total= 1.1min
[CV]  ................................................................
[CV] ................................................. , total= 1.1min
[CV]  ................................................................
[CV] ................................................. , total= 1.2min


[Parallel(n_jobs=1)]: Done   5 out of   5 | elapsed:  5.6min finished
```

In [55]:
```
plot_confusion_matrix(svm_n_3, X_test, y_test, cmap=plt.cm.Blues, display_labels = class_names, values_f
```

Out[55]:
```
<sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay at 0x1a53fcda90>
```



In [56]:
```
report_svm_n_3 = classification_report( y_test, y_preds_svm_n_3)
print(report_svm_n_3)
```

```
              precision    recall  f1-score   support

           0       0.51      0.52      0.52       341
           1       0.56      0.55      0.55       374

    accuracy                           0.54       715
   macro avg       0.54      0.54      0.54       715
weighted avg       0.54      0.54      0.54       715
```

TFIDF

In [57]:
```
X = tfidf_df.drop('class', axis='columns')
y = tfidf_df['class'].astype(int)
```

In [58]:
```
X_train, X_test, y_train, y_test = train_test_split(X, y, random_state=42)
```

Logistic Regression - TFIDF

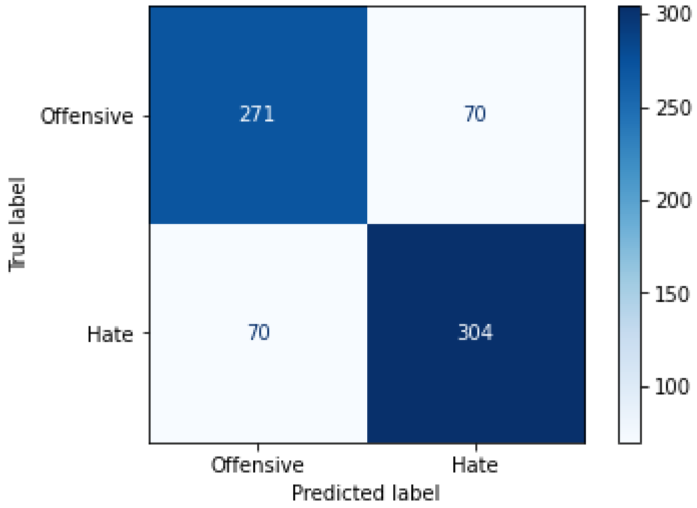
In [59]:

```
log_reg_tf = GridSearchCV(lgr,
                          param_grid,
                          cv=KFold(n_splits=5).split(X_train, y_train),
                          verbose=2)
y_preds_log_reg_tf = log_reg_tf.fit(X_train, y_train).predict(X_test)
```

```
Fitting 5 folds for each of 1 candidates, totalling 5 fits
[CV]  ................................................................

[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.

[CV] ................................................. , total=   2.8s
[CV]  ................................................................

[Parallel(n_jobs=1)]: Done   1 out of   1 | elapsed:    2.9s remaining:    0.0s

[CV] ................................................. , total=   1.8s
[CV]  ................................................................
[CV] ................................................. , total=   2.4s
[CV]  ................................................................
[CV] ................................................. , total=   1.4s
[CV]  ................................................................
[CV] ................................................. , total=   1.7s

[Parallel(n_jobs=1)]: Done   5 out of   5 | elapsed:   10.1s finished
```

In [60]:
```
plot_confusion_matrix(log_reg_tf, X_test, y_test, cmap=plt.cm.Blues, display_labels = class_names, value
```

Out[60]:
```
<sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay at 0x108064950>
```



In [61]:

```
report_log_reg_tf = classification_report( y_test, y_preds_log_reg_tf)
print(report_log_reg_tf)
```

```
              precision    recall  f1-score   support

           0       0.79      0.79      0.79       341
           1       0.81      0.81      0.81       374

    accuracy                           0.80       715
   macro avg       0.80      0.80      0.80       715
weighted avg       0.80      0.80      0.80       715
```

In [62]:

```
importance_logreg = log_reg_tf.best_estimator_.coef_.tolist()[0]

features = list(tfidf_df.drop('class', axis='columns').columns)
feature_importance_logreg = pd.DataFrame(list(zip(features,importance_logreg)), columns =['features','im
feature_importance_logreg = feature_importance_logreg.sort_values(by='importance')
feature_importance_logreg.head(20)
```

Out[62]:

	features	importance
1253	faggot	-4.594443
2639	nigger	-3.856314
2636	niggas	-2.768235
2633	nigga	-2.725411
4290	white	-2.599723
1251	fag	-2.595279
3097	queer	-1.813521
1493	gay	-1.792469
793	coon	-1.697835
2096	kill	-1.667324
2870	people	-1.607309
1679	hate	-1.598636
4014	trash	-1.503182
400	black	-1.423769
3213	retard	-1.385454
1430	fuck	-1.372237
317	beaner	-1.347287
4269	wetback	-1.224424
3539	smh	-1.216035
3110	racist	-1.166534

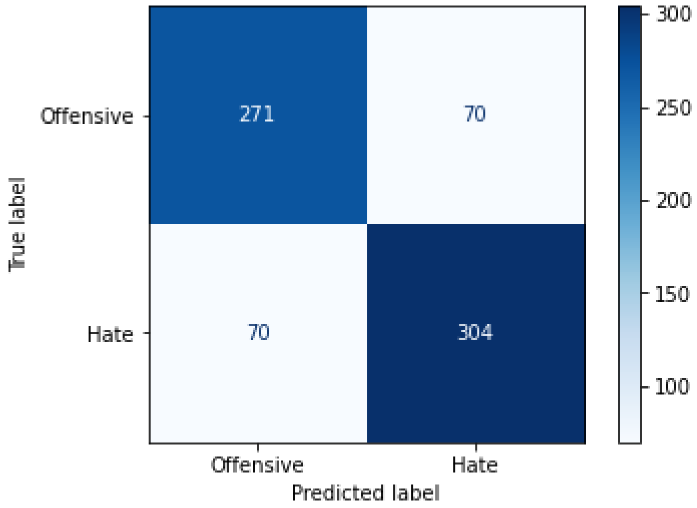
Decision Tree - TFIDF

In [63]:
```
dec_tree_tf = GridSearchCV(lgr,
                           param_grid,
                           cv=KFold(n_splits=5).split(X_train, y_train),
                           verbose=2)
y_preds_dec_tree_tf = dec_tree_tf.fit(X_train, y_train).predict(X_test)
```

```
Fitting 5 folds for each of 1 candidates, totalling 5 fits
[CV]  ................................................................

[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.

[CV] ................................................. , total=   2.8s
[CV]  ................................................................

[Parallel(n_jobs=1)]: Done   1 out of   1 | elapsed:    2.8s remaining:    0.0s

[CV] ................................................. , total=   1.7s
[CV]  ................................................................
[CV] ................................................. , total=   2.4s
[CV]  ................................................................
[CV] ................................................. , total=   1.4s
[CV]  ................................................................
[CV] ................................................. , total=   1.7s

[Parallel(n_jobs=1)]: Done   5 out of   5 | elapsed:   10.0s finished
```

```
In [64]: plot_confusion_matrix(dec_tree_tf, X_test, y_test, cmap=plt.cm.Blues, display_labels = class_names, valu
```

```
Out[64]: <sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay at 0x1a4dfa1790>
```



In [65]:
```
report_dec_tree_tf = classification_report( y_test, y_preds_dec_tree_tf)
print(report_dec_tree_tf)
```

```
              precision    recall  f1-score   support

           0       0.79      0.79      0.79       341
           1       0.81      0.81      0.81       374

    accuracy                           0.80       715
   macro avg       0.80      0.80      0.80       715
weighted avg       0.80      0.80      0.80       715
```

In [66]:

```
importance_dectree = dec_tree_tf.best_estimator_.coef_.tolist()[0]

features = list(tfidf_df.drop('class', axis='columns').columns)
feature_importance_dectree = pd.DataFrame(list(zip(features,importance_dectree)), columns =['features','
feature_importance_dectree = feature_importance_dectree.sort_values(by='importance')
feature_importance_dectree.head(20)
```

Out[66]:

	features	importance
1253	faggot	-4.594443
2639	nigger	-3.856314
2636	niggas	-2.768235
2633	nigga	-2.725411
4290	white	-2.599723
1251	fag	-2.595279
3097	queer	-1.813521
1493	gay	-1.792469
793	coon	-1.697835
2096	kill	-1.667324
2870	people	-1.607309
1679	hate	-1.598636
4014	trash	-1.503182
400	black	-1.423769
3213	retard	-1.385454
1430	fuck	-1.372237
317	beaner	-1.347287
4269	wetback	-1.224424
3539	smh	-1.216035
3110	racist	-1.166534

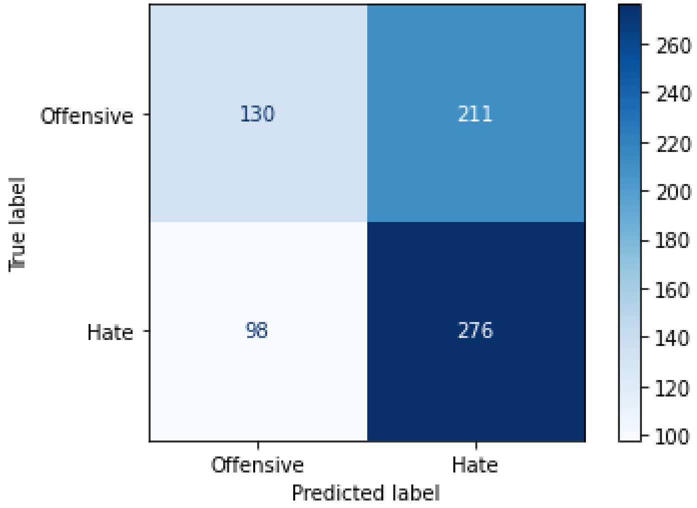
Naive Bayes - TFIDF

In [67]:
```
gnb_tf = GridSearchCV(gnb,
                           param_grid,
                           cv=KFold(n_splits=5).split(X_train, y_train),
                           verbose=2)
y_preds_gnb_tf = gnb_tf.fit(X_train, y_train).predict(X_test)
```

```
Fitting 5 folds for each of 1 candidates, totalling 5 fits
[CV]  ................................................................

[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.

[CV] ................................................. , total=   0.5s
[CV]  ................................................................

[Parallel(n_jobs=1)]: Done   1 out of   1 | elapsed:    0.5s remaining:    0.0s

[CV] ................................................. , total=   0.5s
[CV]  ................................................................
[CV] ................................................. , total=   0.4s
[CV]  ................................................................
[CV] ................................................. , total=   0.5s
[CV]  ................................................................
[CV] ................................................. , total=   0.4s

[Parallel(n_jobs=1)]: Done   5 out of   5 | elapsed:    2.5s finished
```

```
In [68]: plot_confusion_matrix(gnb_tf, X_test, y_test, cmap=plt.cm.Blues, display_labels = class_names, values_f
```

```
Out[68]: <sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay at 0x1a32387850>
```



In [69]:
```
report_gnb_tf = classification_report( y_test, y_preds_gnb_tf)
print(report_gnb_tf)
```

```
              precision    recall  f1-score   support

           0       0.57      0.38      0.46       341
           1       0.57      0.74      0.64       374

    accuracy                           0.57       715
   macro avg       0.57      0.56      0.55       715
weighted avg       0.57      0.57      0.55       715
```

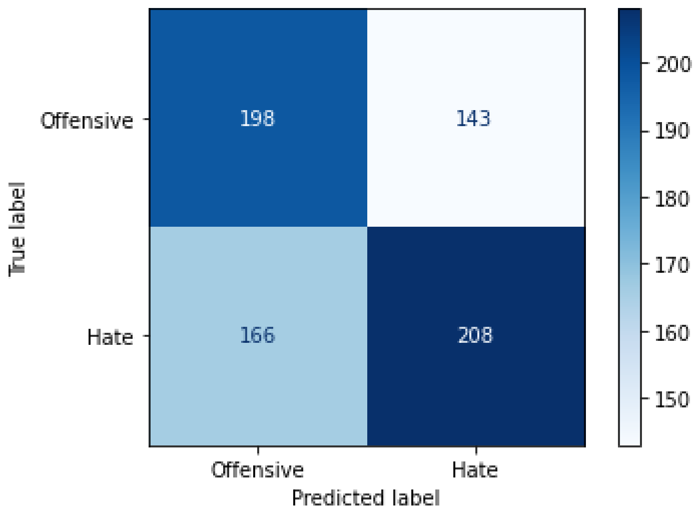
SVM - TFIDF

In [70]:
```
svm_tf = GridSearchCV(svc,
                           param_grid,
                           cv=KFold(n_splits=5).split(X_train, y_train),
                           verbose=2)
y_preds_svm_tf = svm_tf.fit(X_train, y_train).predict(X_test)
```

```
Fitting 5 folds for each of 1 candidates, totalling 5 fits
[CV]  ................................................................

[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.

[CV] ................................................. , total=  20.0s
[CV]  ................................................................

[Parallel(n_jobs=1)]: Done   1 out of   1 | elapsed:   20.0s remaining:    0.0s

[CV] ................................................. , total=  19.1s
[CV]  ................................................................
[CV] ................................................. , total=  17.7s
[CV]  ................................................................
[CV] ................................................. , total=  17.6s
[CV]  ................................................................
[CV] ................................................. , total=  17.5s

[Parallel(n_jobs=1)]: Done   5 out of   5 | elapsed:  1.5min finished
```

In [71]:
```
plot_confusion_matrix(svm_tf, X_test, y_test, cmap=plt.cm.Blues, display_labels = class_names, values_fo
```

Out[71]:
```
<sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay at 0x1a4e1ded90>
```



In [72]:
```
report_svm_tf = classification_report( y_test, y_preds_svm_tf)
print(report_svm_tf)
```

```
              precision    recall  f1-score   support

           0       0.54      0.58      0.56       341
           1       0.59      0.56      0.57       374

    accuracy                           0.57       715
   macro avg       0.57      0.57      0.57       715
weighted avg       0.57      0.57      0.57       715
```

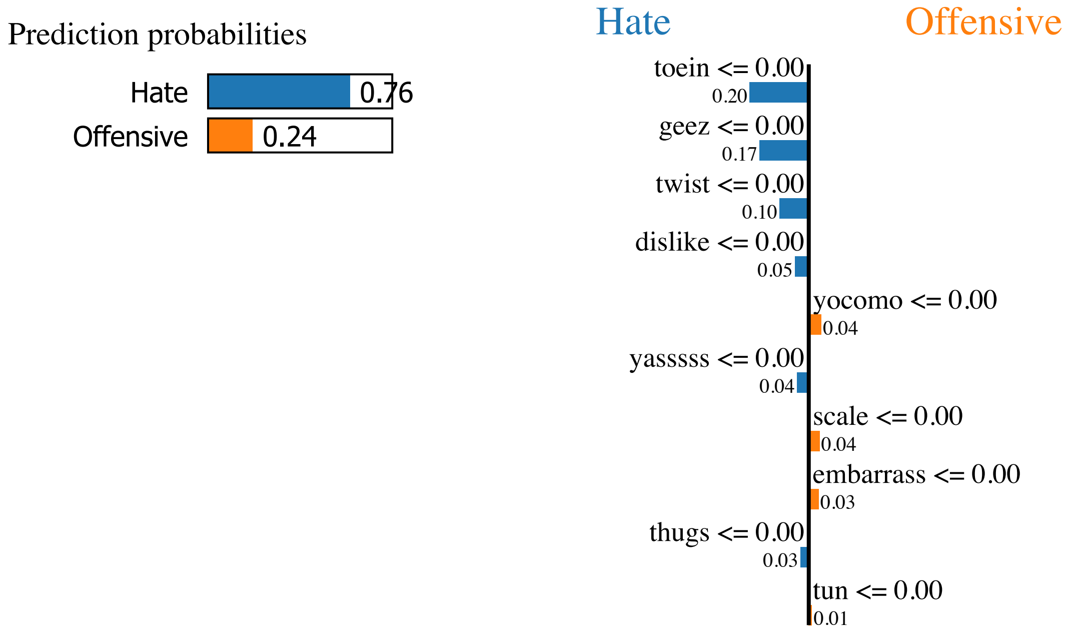
Feature importance using LIME

Going to start with Decision Tree using TF-IDF as it had good results and is interpretable

In [137]:

```
import lime
import lime.lime_tabular

i = np.random.randint(0, X_test.shape[0])

explainer = lime.lime_tabular.LimeTabularExplainer(training_data = X_train.to_numpy(),
                                                    mode = 'classification',
                                                    feature_names = features,
                                                   class_names = ['Hate', 'Offensive'])

exp = explainer.explain_instance(data_row = X_test.iloc[i].to_numpy(),
                                 predict_fn = dec_tree_tf.predict_proba)
actual = tfidf_df['class'][i]

if actual == 0:
    actual = 'Hate'
else:
    actual = 'Offensive'

print(f'Actual classification: {actual}')
exp.show_in_notebook()
```

```
Actual classification: Hate
```



Feature	Value
toein	0.00
geez	0.00
twist	0.00
dislike	0.00
yocomo	0.00
yasssss	0.00
scale	0.00
embarrass	0.00
thugs	0.00
tur	0.00

```
In [ ]:
```

In [1]:

```
import pandas as pd
import numpy as np
from matplotlib import pyplot as plt
import sqlite3
%matplotlib inline
```

In [2]:

```
#import previous df from sqlite
con = sqlite3.connect('twitter_hate.db')
sql = """
SELECT * FROM tweets_nlp
"""
with sqlite3.connect('twitter_hate.db') as con:
    df = pd.read_sql_query(sql, con)
```

In [3]:

```
tweets = df['tweet_clean']

mentions = []
urls = []
hashtags = []
i = 0
for tweet in tweets:
    tweet = tweet.split()
    mentions.append(tweet.count('mentionhere')+tweet.count('mentionhere:')+tw
    urls.append(tweet.count('urlhere'))
    hashtags.append(tweet.count('hashtaghere'))
    tweet = [token for token in tweet if token not in [';&','']]
    tweet = [token for token in tweet if token not in ['&#;mentionhere:','mer
    tweet = " ".join(tweet)
    tweets[i] = tweet
    i += 1

df['tweet_no_others'] = tweets
df['mention_count'] = mentions
df['url_count'] = urls
df['hashtag_count'] = hashtags
```

```
C:\Users\seanx\anaconda3\lib\site-packages\ipykernel_launcher.py:15: Settin
gWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-doc
s/stable/user_guide/indexing.html#returning-a-view-versus-a-copy (https://p
andas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-vi
ew-versus-a-copy)
  from ipykernel import kernelapp as app
```

In [4]:
```
df.head()
```

Out[4]:

	index	count	hate_speech	offensive_language	neither	class	tweet	tweet_
0	17	3	1	2	0	1	" bitch who do you love "	bitc
1	23	3	0	3	0	1	" fuck no that bitch dont even suck dick " &#1...	fuck don suc vide
2	38	3	0	2	1	1	" lames crying over hoes thats tears of a clown "	lames hoes tears
3	59	3	0	3	0	1	"..All I wanna do is get money and fuck model ...	all i get fuck b ru
4	62	3	0	3	0	1	"@ARIZZLEINDACUT: Females think dating a pussy...	fe think puss no m

In [5]:
```
corpus = df['tweet_no_others']
```

Bag of Words Features

In [6]:
```
from sklearn.feature_extraction.text import CountVectorizer

cv = CountVectorizer(min_df=0., max_df=1.)
cv_matrix = cv.fit_transform(corpus)
cv_matrix = cv_matrix.toarray()
cv_matrix
```

Out[6]:
```
array([[0, 0, 0, ..., 0, 0, 0],
       [0, 0, 0, ..., 0, 0, 0],
       [0, 0, 0, ..., 0, 0, 0],
       ...,
       [0, 0, 0, ..., 0, 0, 0],
       [0, 0, 0, ..., 0, 0, 0],
       [0, 0, 0, ..., 0, 0, 0]], dtype=int64)
```

In [7]:

```
# get all unique words in the corpus
vocab = cv.get_feature_names()
# show document feature vectors
df_BOW = pd.DataFrame(cv_matrix, columns=vocab)
df_BOW['class'] = df['class']
df_BOW
```

Out[7]:

```
      aa  aaaaaaaaand  aap  aaron  aaronmacgruder  ab  ability  abortion  about  abraham  ..
   0   0            0    0      0               0   0        0         0      0        0  ..
   1   0            0    0      0               0   0        0         0      0        0  ..
   2   0            0    0      0               0   0        0         0      0        0  ..
   3   0            0    0      0               0   0        0         0      0        0  ..
   4   0            0    0      0               0   0        0         0      0        0  ..
 ...  ...         ...  ...    ...             ... ...      ...       ...    ...      ...  ..
2855   0            0    0      0               0   0        0         0      0        0  ..
2856   0            0    0      0               0   0        0         0      0        0  ..
2857   0            0    0      0               0   0        0         0      0        0  ..
2858   0            0    0      0               0   0        0         0      0        0  ..
2859   0            0    0      0               0   0        0         0      0        0  ..

2860 rows × 5139 columns
```

In [8]:

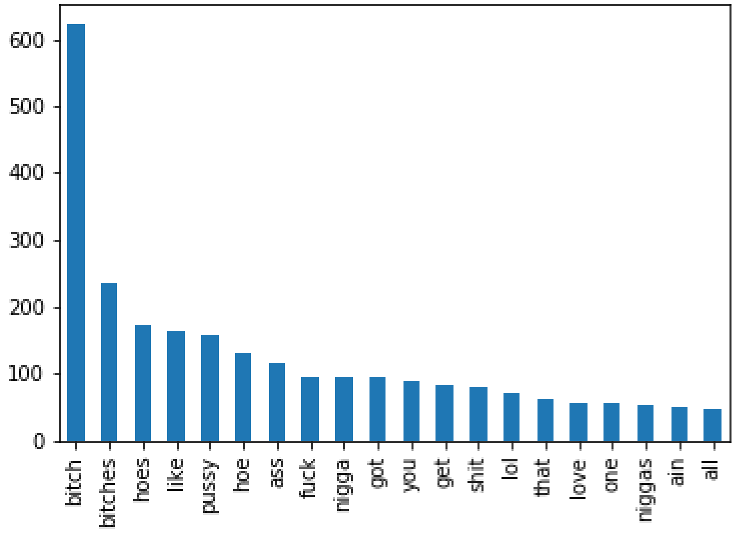
```
BOW_offensive = df_BOW[df_BOW['class'] == 1]
BOW_hate = df_BOW[df_BOW['class'] == 0]
BOW_offensive = BOW_offensive.drop(columns=['class'])
BOW_hate = BOW_hate.drop(columns=['class'])
```

In [9]:

```
BOW_count_off=BOW_offensive.sum()
BOW_off_largest = BOW_count_off.nlargest(20)
BOW_off_largest.plot(kind='bar')
```

Out[9]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x18401166448>
```

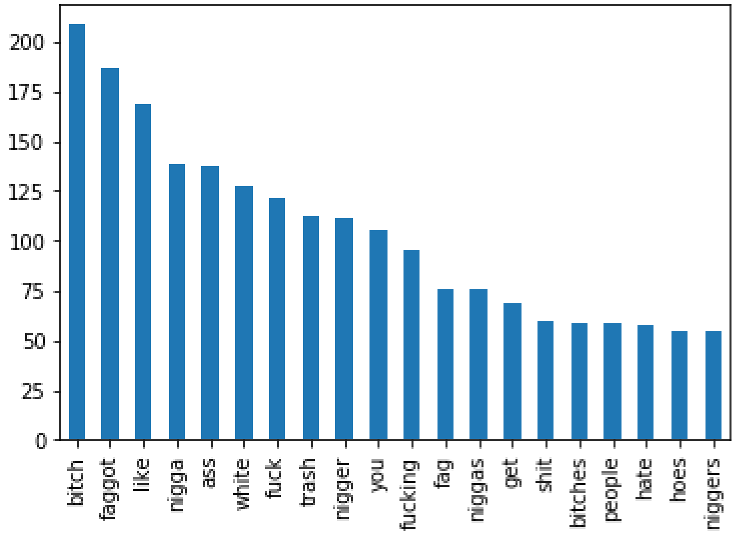


In [10]:

```
BOW_count_hate=BOW_hate.sum()
BOW_hate_largest = BOW_count_hate.nlargest(20)
BOW_hate_largest.plot(kind='bar')
```

Out[10]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x18403cd4388>
```



Training BOW with Logistic Regression and Decision Tree

In [11]:

```
X = pd.concat([df_BOW.drop(columns = ['class']), df[['mention_count', 'url_cc
y = df['class'].astype(int)
```

In [12]:

```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, random_state=42)
```

In [13]:

```
from sklearn.pipeline import Pipeline
from sklearn.model_selection import KFold, GridSearchCV
from sklearn.linear_model import LogisticRegression
from sklearn.tree import DecisionTreeClassifier
from sklearn.feature_selection import SelectFromModel
from sklearn.metrics import classification_report
```

In [14]:

```
param_grid = [{}]
lg = GridSearchCV(LogisticRegression(),
                           param_grid,
                           cv=KFold(n_splits=5,
                                   random_state=42).split(X_train,
                           verbose=2)
y_preds_lg = lg.fit(X_train, y_train).predict(X_test)
```

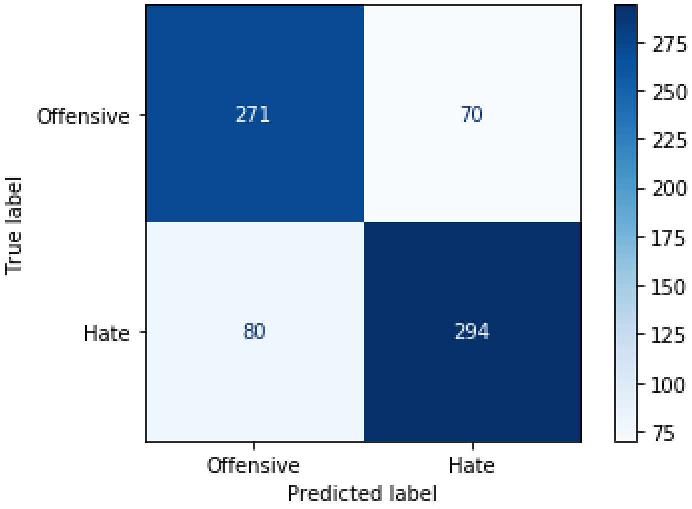
```
C:\Users\seanx\anaconda3\lib\site-packages\sklearn\model_selection\_split.p
y:296: FutureWarning: Setting a random_state has no effect since shuffle is
False. This will raise an error in 0.24. You should leave random_state to i
ts default (None), or set shuffle=True.
  FutureWarning
[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent wor
kers.

Fitting 5 folds for each of 1 candidates, totalling 5 fits
[CV]  ..............................................................
[CV] ................................................. , total=   0.7s
[CV]  ..............................................................

[Parallel(n_jobs=1)]: Done   1 out of   1 | elapsed:    0.7s remaining:
0.0s

[CV] ................................................. , total=   0.9s
[CV]  ..............................................................
[CV] ................................................. , total=   0.8s
[CV]  ..............................................................
[CV] ................................................. , total=   0.9s
[CV]  ..............................................................
[CV] ................................................. , total=   0.8s

[Parallel(n_jobs=1)]: Done   5 out of   5 | elapsed:    4.2s finished
```

In [15]:

```
from sklearn.metrics import plot_confusion_matrix
class_names = ['Offensive', 'Hate']
plot_confusion_matrix(lg, X_test, y_test, cmap=plt.cm.Blues, display_labels =
```

Out[15]:

```
<sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay at 0x1840ee1
4088>
```



In [43]:

```
from sklearn.metrics import classification_report
report_lg_BOW = classification_report( y_test, y_preds_lg)
print(report_lg_BOW)
```

```
              precision    recall  f1-score   support

           0       0.77      0.79      0.78       341
           1       0.81      0.79      0.80       374

    accuracy                           0.79       715
   macro avg       0.79      0.79      0.79       715
weighted avg       0.79      0.79      0.79       715
```

In [16]:

```
importance_logreg = lg.best_estimator_.coef_.tolist()[0]

features = list(df_BOW.columns)
feature_importance_logreg = pd.DataFrame(list(zip(features,importance_logreg)
feature_importance_logreg = feature_importance_logreg.sort_values(by='importa
feature_importance_logreg.head(20)
```

Out[16]:

	features	importance
1455	faggit	-2.108559
3030	niggaz	-2.079960
3027	niggahs	-1.998250
1456	faggot	-1.949334
3033	niggerous	-1.639718
1732	gave	-1.594995
3023	nigerian	-1.555284
2411	kike	-1.518631
4949	whistle	-1.385182
3546	queen	-1.284641
444	black	-1.274186
1305	dwn	-1.250660
1458	fagjo	-1.218483
1453	facts	-1.196915
911	cool	-1.137583
3283	pennsylvanians	-1.103863
4160	sperm	-1.076972
851	comfortable	-1.046262
348	beaner	-1.045217
4061	smfh	-1.037146

In [17]:

```
tree = GridSearchCV(DecisionTreeClassifier(),
                             param_grid,
                             cv=KFold(n_splits=5,
                                                   random_state=42).split(X_train,
                             verbose=2)

y_preds_tree = tree.fit(X_train, y_train).predict(X_test)
```

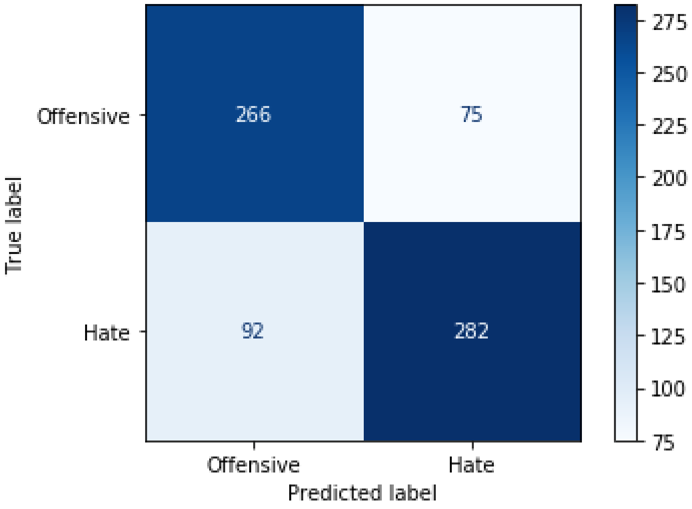
```
C:\Users\seanx\anaconda3\lib\site-packages\sklearn\model_selection\_split.p
y:296: FutureWarning: Setting a random_state has no effect since shuffle is
False. This will raise an error in 0.24. You should leave random_state to i
ts default (None), or set shuffle=True.
  FutureWarning
[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent wor
kers.

Fitting 5 folds for each of 1 candidates, totalling 5 fits
[CV]  ................................................................
[CV] ................................................. , total=   1.1s
[CV]  ................................................................

[Parallel(n_jobs=1)]: Done   1 out of   1 | elapsed:    1.0s remaining:
0.0s

[CV] ................................................. , total=   1.1s
[CV]  ................................................................
[CV] ................................................. , total=   1.2s
[CV]  ................................................................
[CV] ................................................. , total=   1.8s
[CV]  ................................................................
[CV] ................................................. , total=   1.9s

[Parallel(n_jobs=1)]: Done   5 out of   5 | elapsed:    7.2s finished
```

In [18]:

```
plot_confusion_matrix(tree, X_test, y_test, cmap=plt.cm.Blues, display_labels
```

Out[18]:

```
<sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay at 0x18403d0
a0c8>
```



In [19]:

```
report_tree = classification_report( y_test, y_preds_tree)
print(report_tree)
```

```
              precision    recall  f1-score   support

           0       0.74      0.78      0.76       341
           1       0.79      0.75      0.77       374

    accuracy                           0.77       715
   macro avg       0.77      0.77      0.77       715
weighted avg       0.77      0.77      0.77       715
```

In [20]:

```
importance_tree = tree.best_estimator_.feature_importances_.tolist()

features = list(df_BOW.columns)
feature_importance_logreg = pd.DataFrame(list(zip(features,importance_tree)),
feature_importance_logreg = feature_importance_logreg.sort_values(by='importa
feature_importance_logreg.head(20)
```

Out[20]:

	features	importance
432	bitch	0.090740
434	bitches	0.072084
3535	pussies	0.069759
2046	hoeing	0.064999
2044	hockey	0.054338
3027	niggahs	0.025053
3023	nigerian	0.023572
5138	zzzzzz	0.012307
3025	niggaa	0.008869
4949	whistle	0.007725
1305	dwn	0.007683
1667	fuccing	0.007497
224	ass	0.007476
1676	fuckin	0.007473
3937	shirts	0.006702
423	bird	0.006569
4683	tv	0.006502
4759	upset	0.006126
4716	ugliest	0.005897
1942	hat	0.005024

Word2vec embedding

In [21]:

```python
from gensim.models import word2vec
import nltk
```

In [22]:

```python
feature_size = 100    # Word vector dimensionality
window_context = 30          # Context window size
min_word_count = 1   # Minimum word count
sample = 1e-3   # Downsample setting for frequent words

wpt = nltk.WordPunctTokenizer()
tokenized_corpus = [wpt.tokenize(document) for document in corpus]

# Set values for various parameters
feature_size = 100    # Word vector dimensionality
window_context = 30          # Context window size
min_word_count = 1   # Minimum word count
sample = 1e-3   # Downsample setting for frequent words

w2v_model = word2vec.Word2Vec(tokenized_corpus, size=feature_size,
                          window=window_context, min_count=min_word_count,
                          sample=sample, iter=50)

# view similar words based on gensim's model
similar_words = {search_term: [item[0] for item in w2v_model.wv.most_similar(
                  for search_term in ['hate','love','nigger','faggot','bitch'
similar_words
```

Out[22]:

```
{'hate': ['goddamit', 'cripples', 'escape', 'dairy', 'ing'],
 'love': ['mitchell', 'bread', 'victoria', 'baltimore', 'emojis'],
 'nigger': ['hoodrats', 'ebloa', 'kidnapped', 'traditions', 'tyler'],
 'faggot': ['tear', 'little', 'bitching', 'ultimate', 'fag'],
 'bitch': ['next', 'knowin', 'tf', 'mobbin', 'meal'],
 'pussy': ['stank', 'swimm', 'strap', 'poo', 'another'],
 'cracker': ['hypocrisy', 'friday', 'fathom', 'blocked', 'statement'],
 'nigga': ['lame', 'scary', 'yah', 'dont', 'fuk'],
 'homo': ['bullet', 'snapchat', 'cortez', 'twisted', 'fosters'],
 'cunt': ['profile', 'managers', 'piss', 'dress', 'pm'],
 'fuck': ['fish', 'hmm', 'zima', 'kermit', 'whiney'],
 'trash': ['white',
  'westbrook',
  'trailer',
  'hashtagherehashtaghere',
  'doesnt'],
 'queer': ['yost', 'pathetic', 'traitor', 'project', 'lbum']}
```

In [23]:

```
def average_word_vectors(words, model, vocabulary, num_features):

    feature_vector = np.zeros((num_features,),dtype="float64")
    nwords = 0.

    for word in words:
        if word in vocabulary:
            nwords = nwords + 1.
            feature_vector = np.add(feature_vector, model[word])

    if nwords:
        feature_vector = np.divide(feature_vector, nwords)

    return feature_vector

def averaged_word_vectorizer(corpus, model, num_features):
    vocabulary = set(model.wv.index2word)
    features = [average_word_vectors(tokenized_sentence, model, vocabulary, r
                    for tokenized_sentence in corpus]
    return np.array(features)

w2v_feature_array = averaged_word_vectorizer(corpus=tokenized_corpus, model=w
                                             num_features=feature_size)
pd.DataFrame(w2v_feature_array)
```

```
C:\Users\seanx\anaconda3\lib\site-packages\ipykernel_launcher.py:9: Depreca
tionWarning: Call to deprecated `__getitem__` (Method will be removed in 4.
0.0, use self.wv.__getitem__() instead).
  if __name__ == '__main__':
```

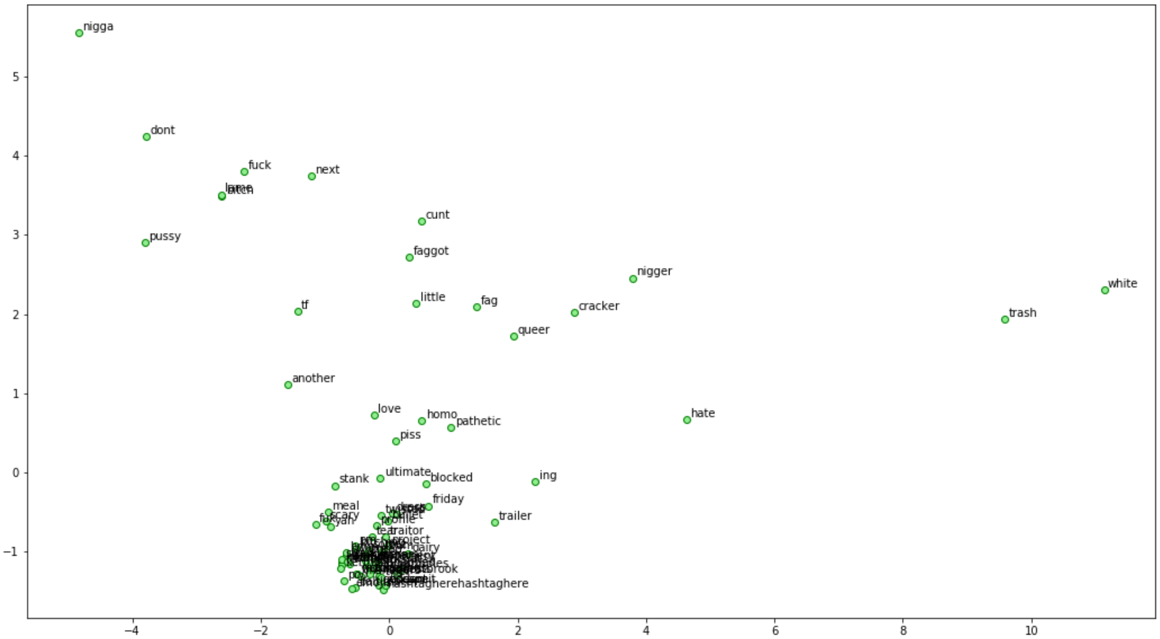
Out[23]:

	0	1	2	3	4	5	6	7	
0	0.727941	0.244606	0.683794	-0.061855	-0.406099	-0.468852	0.655809	-0.275200	-0.
1	0.842390	0.307193	0.882716	0.294135	-0.236299	-0.177043	0.579353	-0.165394	-0.
2	0.546560	0.224021	0.425261	-0.028157	-0.306209	-0.039596	0.662744	-0.180299	-0.
3	0.692076	0.149320	-0.006210	-0.070063	-0.443279	-0.198863	0.479766	-0.269004	-0.
4	0.518252	0.081868	0.732512	-0.030877	-0.265558	0.084669	0.658960	-0.165031	-0.
...	...	...	...	...	...	...	...	...	
2855	-0.051713	-0.128862	0.805924	-0.030887	-0.387116	-0.644496	0.266638	0.223754	-0.
2856	-1.109598	-0.135360	-0.197205	-0.557631	-0.728592	-0.173033	0.391669	0.502999	-1.
2857	0.140394	0.190423	0.698273	0.058907	-0.212432	-0.033794	0.315742	0.242145	-0.
2858	0.718705	0.362663	0.732503	-0.118662	-0.687060	0.056234	0.649663	-0.468604	-0.
2859	0.547129	0.186977	0.305199	-0.090508	-0.358534	-0.156237	0.567174	-0.303735	-0.

2860 rows × 100 columns

In [24]:

```
from sklearn.decomposition import PCA

words = sum([[k] + v for k, v in similar_words.items()], [])
wvs = w2v_model.wv[words]

pca = PCA(n_components=2)
np.set_printoptions(suppress=True)
P = pca.fit_transform(wvs)
labels = words

plt.figure(figsize=(18, 10))
plt.scatter(P[:, 0], P[:, 1], c='lightgreen', edgecolors='g')
for label, x, y in zip(labels, P[:, 0], P[:, 1]):
    plt.annotate(label, xy=(x+0.06, y+0.03), xytext=(0, 0), textcoords='offse
```



In [25]:

```
X_w2v = pd.DataFrame(w2v_feature_array)
y = df['class'].astype(int)
X_train_w2v, X_test_w2v, y_train_w2v, y_test_w2v = train_test_split(X_w2v, y,
```

In [26]:

```
lg_w2v = GridSearchCV(LogisticRegression(max_iter = 1000),
                           param_grid,
                           cv=KFold(n_splits=5,
                                              random_state=42).split(X_train_
                           verbose=2)
y_preds_w2v_lg = lg_w2v.fit(X_train_w2v, y_train_w2v).predict(X_test_w2v)
```

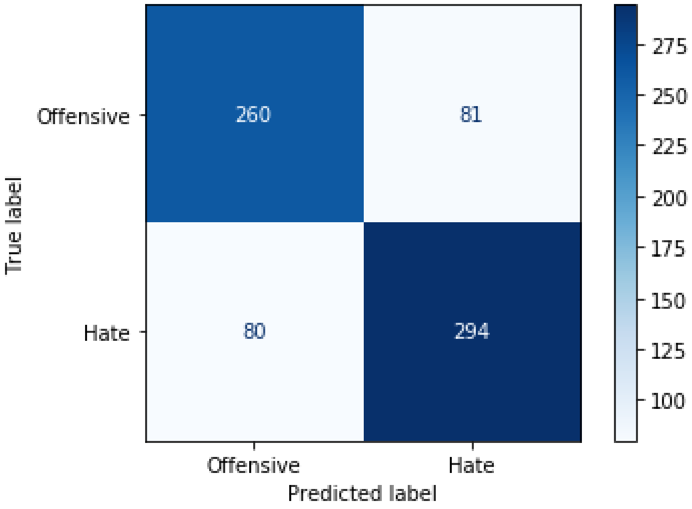
```
C:\Users\seanx\anaconda3\lib\site-packages\sklearn\model_selection\_split.p
y:296: FutureWarning: Setting a random_state has no effect since shuffle is
False. This will raise an error in 0.24. You should leave random_state to i
ts default (None), or set shuffle=True.
  FutureWarning
[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent wor
kers.
[Parallel(n_jobs=1)]: Done   1 out of   1 | elapsed:    0.0s remaining:
0.0s

Fitting 5 folds for each of 1 candidates, totalling 5 fits
[CV]  ................................................................
[CV] ................................................... , total=   0.1s
[CV]  ................................................................
[CV] ................................................... , total=   0.1s
[CV]  ................................................................
[CV] ................................................... , total=   0.1s
[CV]  ................................................................
[CV] ................................................... , total=   0.1s
[CV]  ................................................................
[CV] ................................................... , total=   0.1s

[Parallel(n_jobs=1)]: Done   5 out of   5 | elapsed:    0.3s finished
```

In [27]:

```
plot_confusion_matrix(lg_w2v, X_test_w2v, y_test_w2v, cmap=plt.cm.Blues, disp
```

Out[27]:

```
<sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay at 0x1840ede
0708>
```



In [28]:

```
report_w2v = classification_report( y_test_w2v, y_preds_w2v_lg)
print(report_w2v)
```

```
              precision    recall  f1-score   support

           0       0.76      0.76      0.76       341
           1       0.78      0.79      0.79       374

    accuracy                           0.77       715
   macro avg       0.77      0.77      0.77       715
weighted avg       0.77      0.77      0.77       715
```

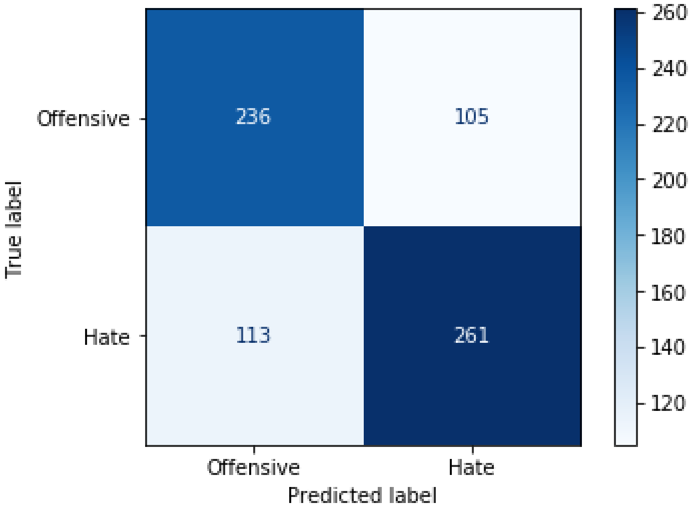
In [29]:

```
tree_w2v = GridSearchCV(DecisionTreeClassifier(),
                           param_grid,
                           cv=KFold(n_splits=5,
                                              random_state=42).split(X_train_
                           verbose=2)
y_preds_w2v_tree = tree_w2v.fit(X_train_w2v, y_train_w2v).predict(X_test_w2v)
```

```
Fitting 5 folds for each of 1 candidates, totalling 5 fits
[CV]  ...................................................................

C:\Users\seanx\anaconda3\lib\site-packages\sklearn\model_selection\_split.p
y:296: FutureWarning: Setting a random_state has no effect since shuffle is
False. This will raise an error in 0.24. You should leave random_state to i
ts default (None), or set shuffle=True.
  FutureWarning
[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent wor
kers.
[Parallel(n_jobs=1)]: Done   1 out of   1 | elapsed:    0.1s remaining:
0.0s

[CV] ................................................ , total=   0.2s
[CV]  ...................................................................
[CV] ................................................ , total=   0.2s
[CV]  ...................................................................
[CV] ................................................ , total=   0.2s
[CV]  ...................................................................
[CV] ................................................ , total=   0.2s
[CV]  ...................................................................
[CV] ................................................ , total=   0.2s

[Parallel(n_jobs=1)]: Done   5 out of   5 | elapsed:    0.8s finished
```

```
In [30]:  plot_confusion_matrix(tree_w2v, X_test_w2v, y_test_w2v, cmap=plt.cm.Blues, di
```

```
Out[30]: <sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay at 0x1841d37
         21c8>
```



```
In [31]:  report_w2v_tree = classification_report( y_test_w2v, y_preds_w2v_tree)
          print(report_w2v_tree)
```

```
              precision    recall  f1-score   support

           0       0.68      0.69      0.68       341
           1       0.71      0.70      0.71       374

    accuracy                           0.70       715
   macro avg       0.69      0.69      0.69       715
weighted avg       0.70      0.70      0.70       715
```

Combining BOW and W2V

```
In [32]:  X_mixed = pd.concat([pd.DataFrame(w2v_feature_array), df_BOW.drop(columns=['c
          y = df['class'].astype(int)
          X_train_mixed, X_test_mixed, y_train_mixed, y_test_mixed = train_test_split(X
```

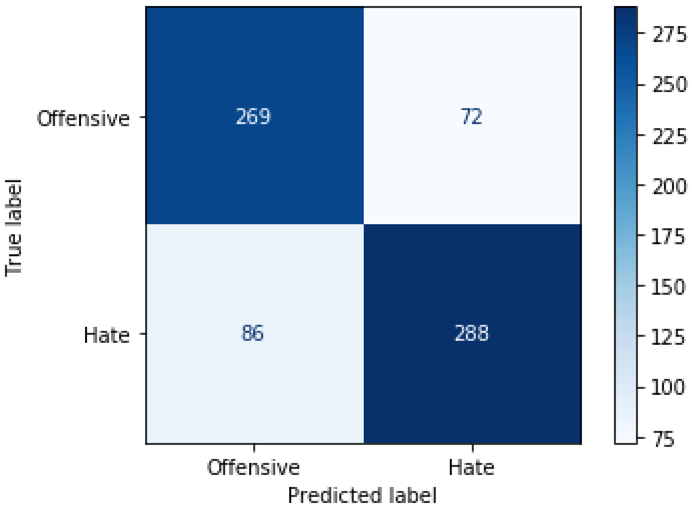
In [33]:

```
lg_mixed = GridSearchCV(LogisticRegression(max_iter = 1000),
                           param_grid,
                           cv=KFold(n_splits=5,
                                    random_state=42).split(X_train_
                           verbose=2)
y_preds_mixed = lg_mixed.fit(X_train_mixed, y_train_mixed).predict(X_test_mix
```

```
C:\Users\seanx\anaconda3\lib\site-packages\sklearn\model_selection\_split.p
y:296: FutureWarning: Setting a random_state has no effect since shuffle is
False. This will raise an error in 0.24. You should leave random_state to i
ts default (None), or set shuffle=True.
  FutureWarning
[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent wor
kers.


Fitting 5 folds for each of 1 candidates, totalling 5 fits
[CV]  ................................................................
[CV] ................................................. , total=   1.5s
[CV]  ................................................................


[Parallel(n_jobs=1)]: Done   1 out of   1 | elapsed:    1.4s remaining:
0.0s

[CV] ................................................. , total=   1.5s
[CV]  ................................................................
[CV] ................................................. , total=   1.3s
[CV]  ................................................................
[CV] ................................................. , total=   1.3s
[CV]  ................................................................
[CV] ................................................. , total=   1.4s


[Parallel(n_jobs=1)]: Done   5 out of   5 | elapsed:    6.9s finished
```

In [34]:
```
plot_confusion_matrix(lg_mixed, X_test_mixed, y_test_mixed, cmap=plt.cm.Blues
```

Out[34]:
```
<sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay at 0x1841d42
9808>
```



In [35]:
```
report_lg_mixed = classification_report( y_test_mixed, y_preds_mixed)
print(report_lg_mixed)
```

```
              precision    recall  f1-score   support

           0       0.76      0.79      0.77       341
           1       0.80      0.77      0.78       374

    accuracy                           0.78       715
   macro avg       0.78      0.78      0.78       715
weighted avg       0.78      0.78      0.78       715
```

In [36]:

```
features = list(X_mixed.columns)
feature_importance_logreg = pd.DataFrame(list(zip(features,importance_logreg)
feature_importance_logreg = feature_importance_logreg.sort_values(by='importa
feature_importance_logreg.head(20)
```

Out[36]:

	features	importance
1455	emoji	-2.108559
3030	muslims	-2.079960
3027	murdered	-1.998250
1456	emojis	-1.949334
3033	muthafucka	-1.639718
1732	found	-1.594995
3023	muhhfuckin	-1.555284
2411	jezzy	-1.518631
4949	wannabe	-1.385182
3546	pray	-1.284641
444	bc	-1.274186
1305	dm	-1.250660
1458	encrusted	-1.218483
1453	emm	-1.196915
911	closer	-1.137583
3283	otter	-1.103863
4160	smfh	-1.076972
851	chill	-1.046262
348	aunt	-1.045217
4061	shout	-1.037146

In [37]:

```
tree_mixed = GridSearchCV(DecisionTreeClassifier(),
                           param_grid,
                           cv=KFold(n_splits=5,
                                    random_state=42).split(X_train_
                           verbose=2)
y_preds_mixed_tree = tree_mixed.fit(X_train_mixed, y_train_mixed).predict(X_t
```

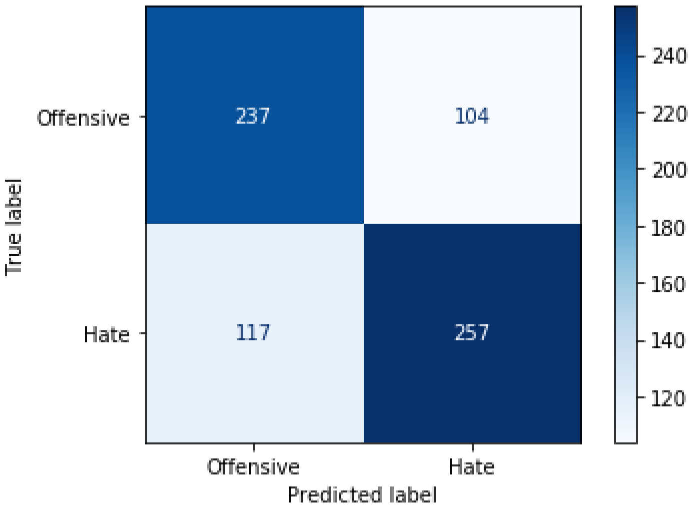
```
Fitting 5 folds for each of 1 candidates, totalling 5 fits
[CV]  ................................................................

C:\Users\seanx\anaconda3\lib\site-packages\sklearn\model_selection\_split.p
y:296: FutureWarning: Setting a random_state has no effect since shuffle is
False. This will raise an error in 0.24. You should leave random_state to i
ts default (None), or set shuffle=True.
  FutureWarning
[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent wor
kers.

[CV] ................................................. , total=   1.3s
[CV]  ................................................................

[Parallel(n_jobs=1)]: Done   1 out of   1 | elapsed:    1.2s remaining:
0.0s

[CV] ................................................. , total=   1.4s
[CV]  ................................................................
[CV] ................................................. , total=   1.5s
[CV]  ................................................................
[CV] ................................................. , total=   1.3s
[CV]  ................................................................
[CV] ................................................. , total=   1.3s

[Parallel(n_jobs=1)]: Done   5 out of   5 | elapsed:    6.7s finished
```

In [38]:

```
plot_confusion_matrix(tree_mixed, X_test_mixed, y_test_mixed, cmap=plt.cm.Blu
```

Out[38]:

```
<sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay at 0x18410ae
8648>
```



In [39]:

```
report_tree_mixed = classification_report( y_test_mixed, y_preds_mixed_tree)
print(report_tree_mixed)
```

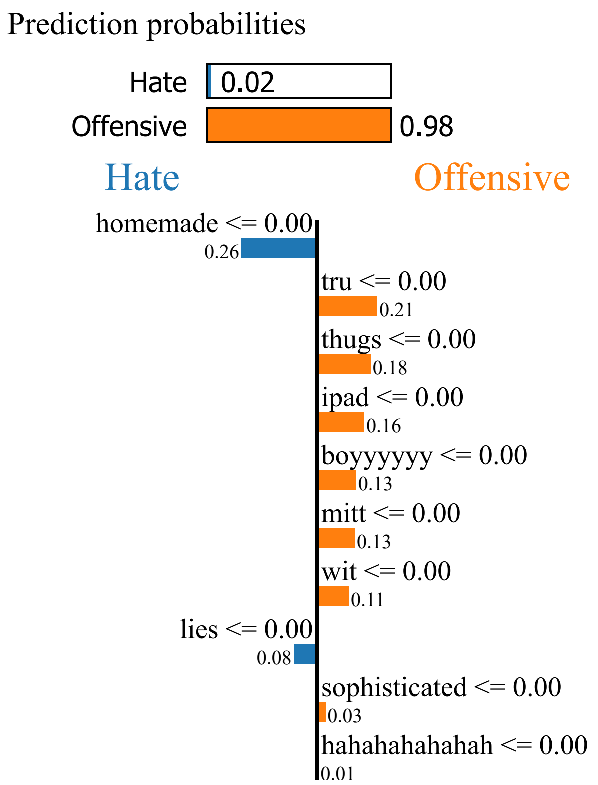
```
              precision    recall  f1-score   support

           0       0.67      0.70      0.68       341
           1       0.71      0.69      0.70       374

    accuracy                           0.69       715
   macro avg       0.69      0.69      0.69       715
weighted avg       0.69      0.69      0.69       715
```

Using LIME to interpret predictions

In [53]:

```
import lime
import lime.lime_tabular

i = np.random.randint(0, X_test.shape[0])

explainer = lime.lime_tabular.LimeTabularExplainer(training_data = X_train.tc
                                                    mode = 'classification',
                                                    feature_names = features,
                                                   class_names = ['Hate', 'Off

exp = explainer.explain_instance(data_row = X_test.iloc[i].to_numpy(),
                                 predict_fn = lg.predict_proba)
actual = df_BOW['class'][i]

if actual == 0:
    actual = 'Hate'
else:
    actual = 'Offensive'

print(f'Actual classification: {actual}')
exp.show_in_notebook()
```

```
Actual classification: Offensive
```



Feature	Value
homemade	0.00
tru	0.00
thugs	0.00
ipad	0.00
boyyyyyy	0.00

```
         mitt    0.00

          wit    0.00

         lies    0.00

sophisticated    0.00
```

In [ ]: