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	mentionhere	

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

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

In [5]:
```
#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]:
```
df.iloc[2827]['tweet_no_others']
```

Out[6]:
```
'tears call sweetie fucking retard'
```

In [7]:
```
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	t
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	tv
2787	24179	3	0	3	0	1	lol RT @_mykall: when you @ bae game & 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 “@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 “@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 lol	aaaaaaaaand begin	aap maoist	aaron weak	aaronmacgruder stuff	ability block	abortion get	about act	abraham lincoln	absolute pussy	...	zimmerman arrest	zimmerman comin	zimmer 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	

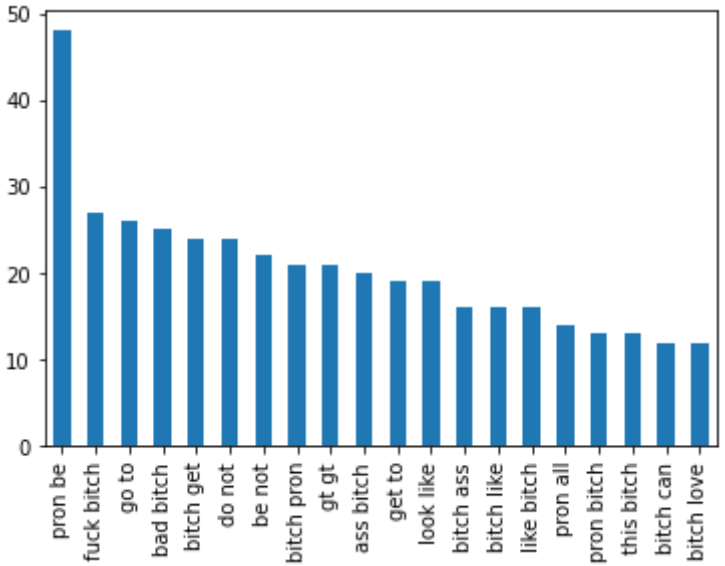
2860 rows × 16247 columns

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')
```

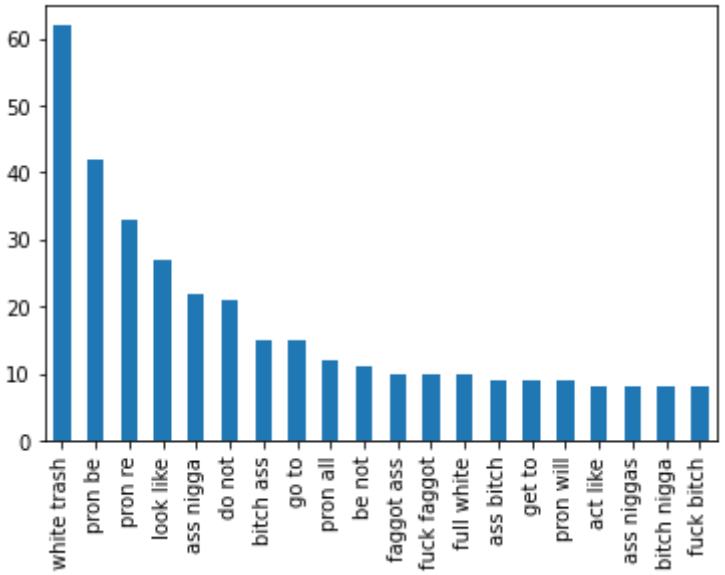
In [15]:
```
ng_count_off=ngrams_offensive.sum()
ng_off_largest = ng_count_off.nlargest(20)
ng_off_largest.plot(kind='bar')
```

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

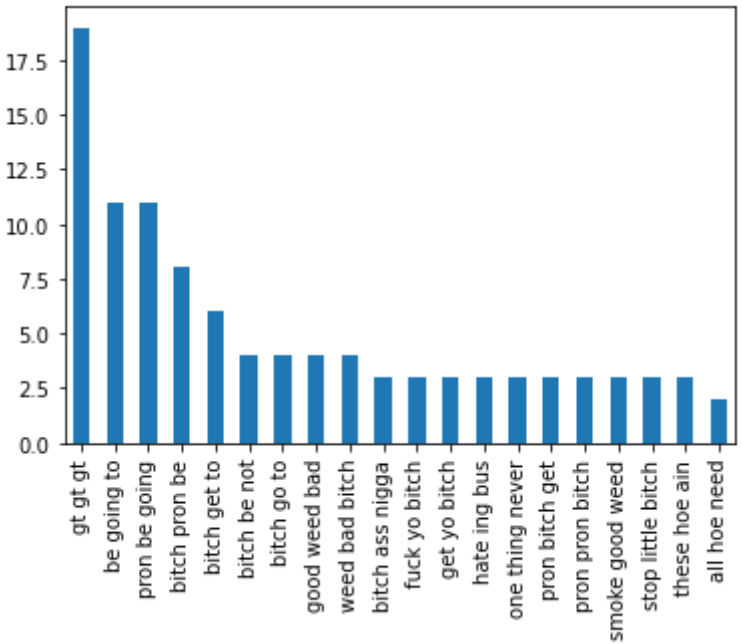
2860 rows × 16725 columns

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')
```

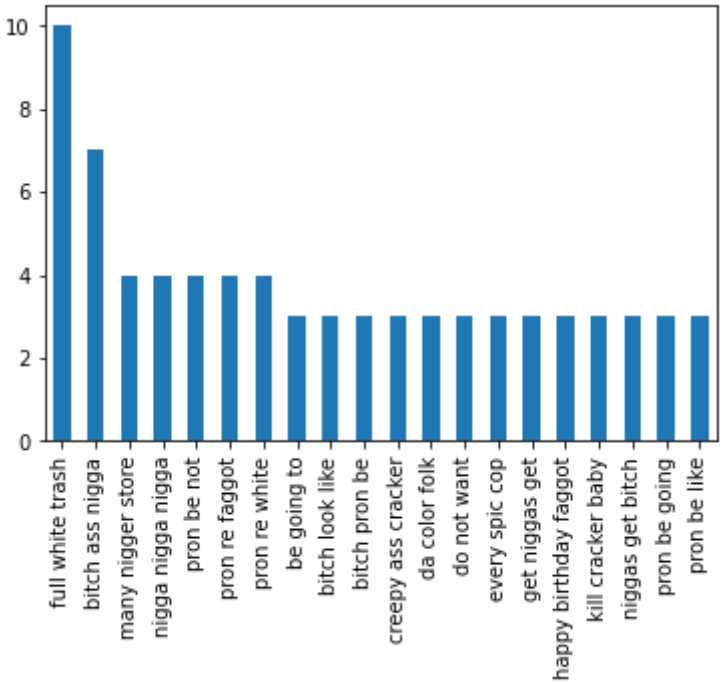
In [20]:
```
ng3_count_off=ngrams_3_offensive.sum()
ng3_off_largest = ng3_count_off.nlargest(20)
ng3_off_largest.plot(kind='bar')
```

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	

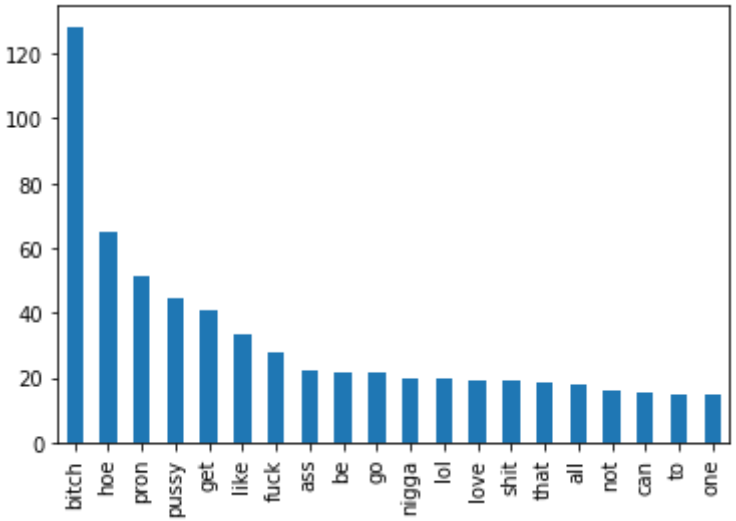
2860 rows × 4451 columns

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')
```

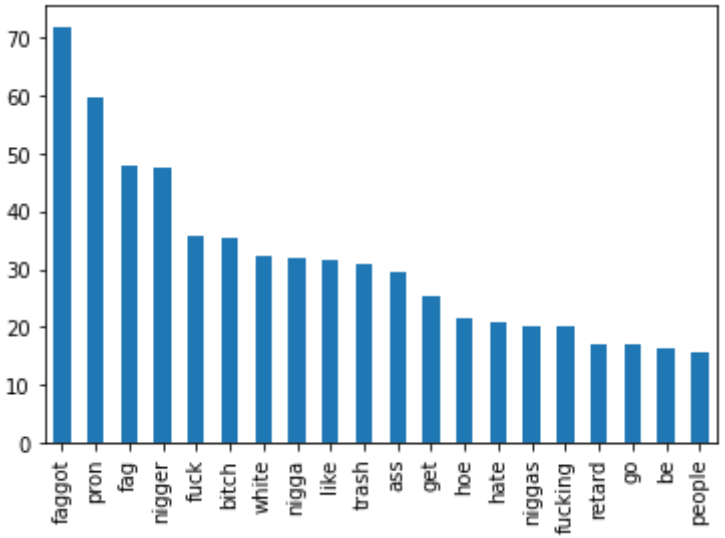
In [25]:
```
tf_count_off=tfidf_offensive.sum()
tf_off_largest = tf_count_off.nlargest(20)
tf_off_largest.plot(kind='bar')
```

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
```

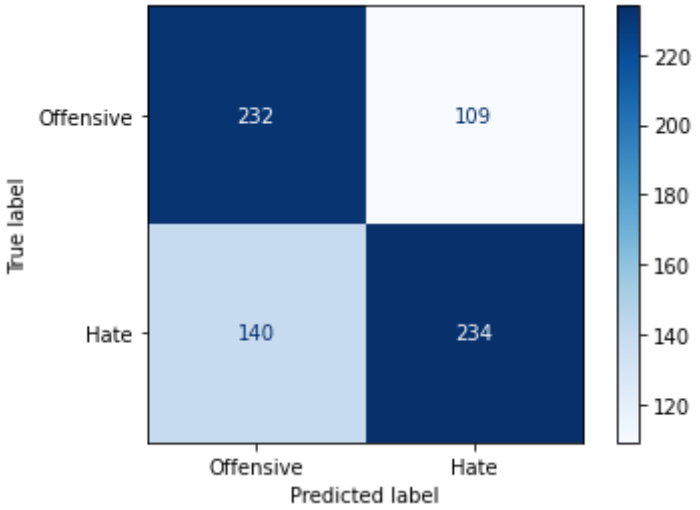
Logistic Regression - ngrams (n=2)

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
```

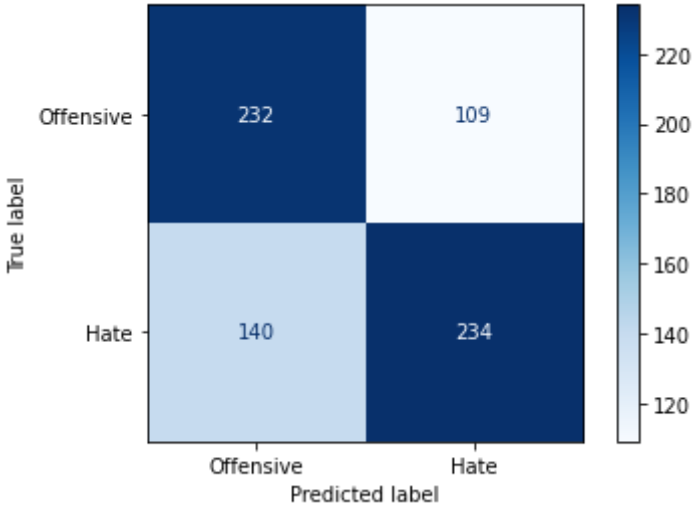
Decision Tree - ngram (n=2)

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
```

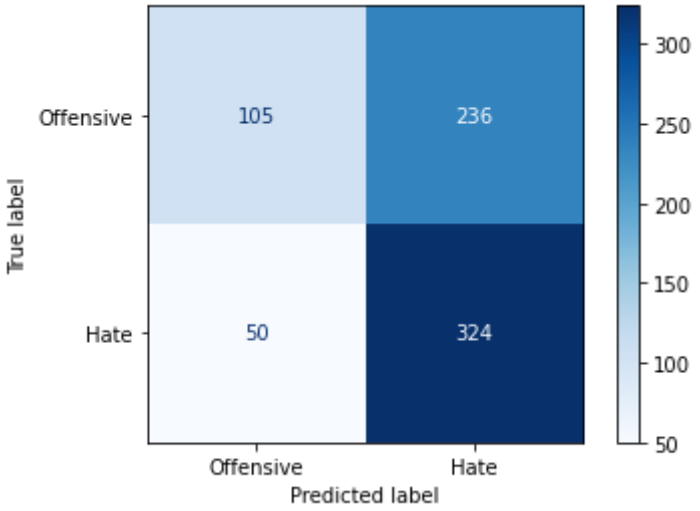
Naive Bayes - ngram (n=2)

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
```

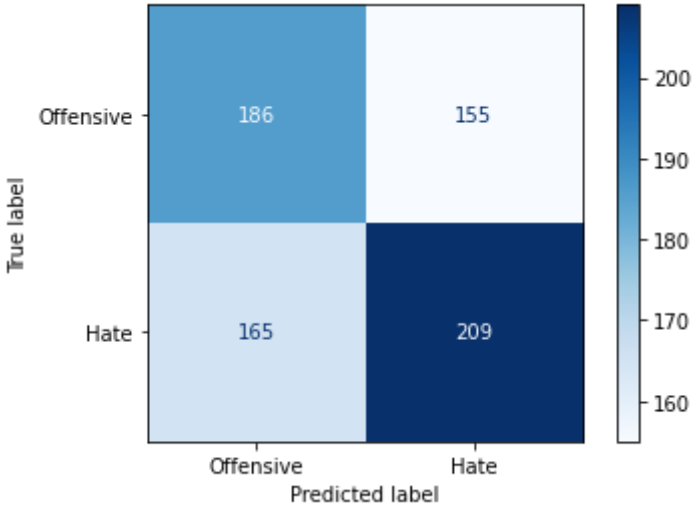
SVM - ngram (n=2)

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)
```

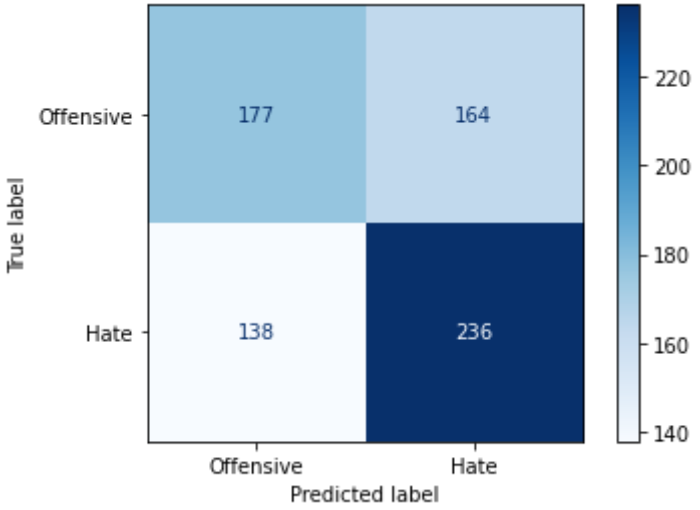
Logistic Regression - ngram (n=3)

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
```

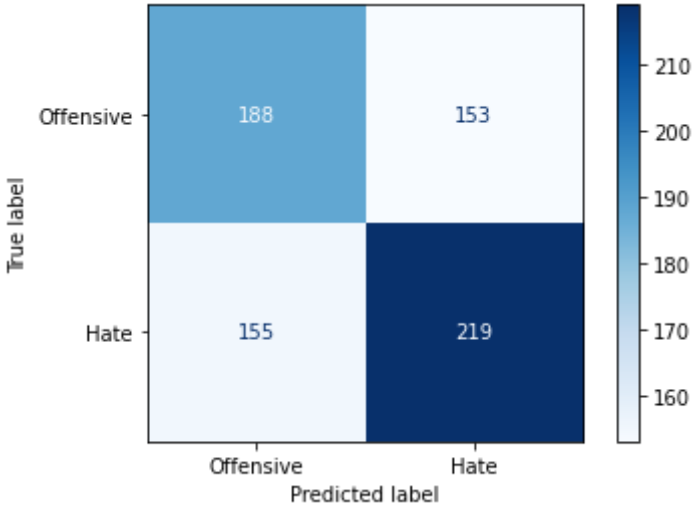
Decision Tree - ngram (n=3)

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
```

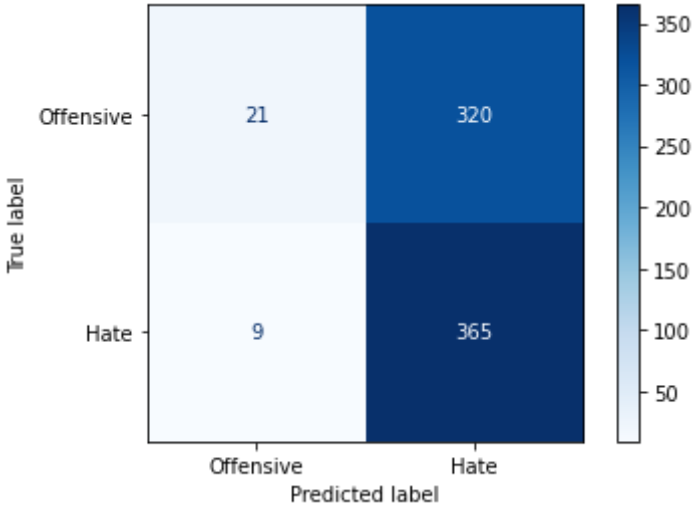
Naive Bayes - ngram (n=3)

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
```

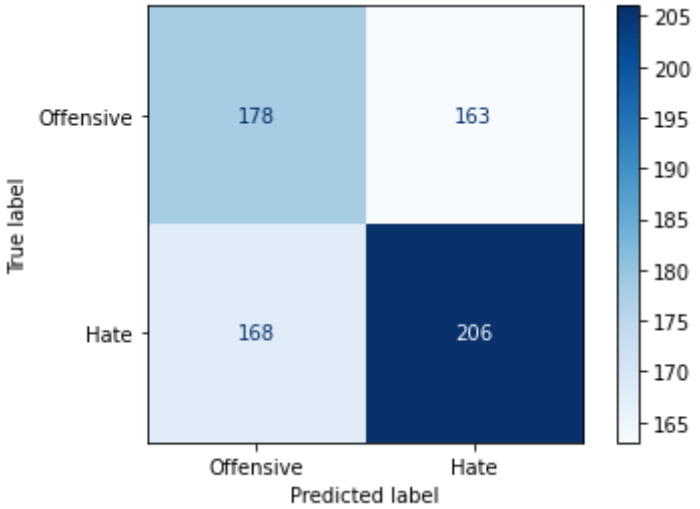
SVM - ngram (n=3)

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)
```

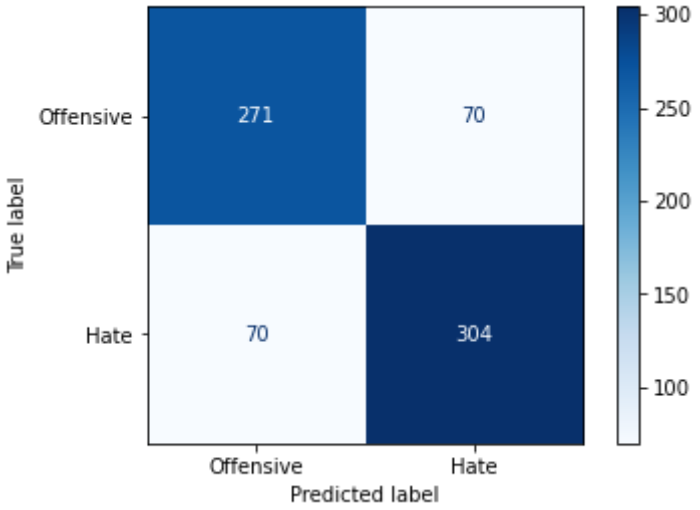
Logistic Regression - TFIDF

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

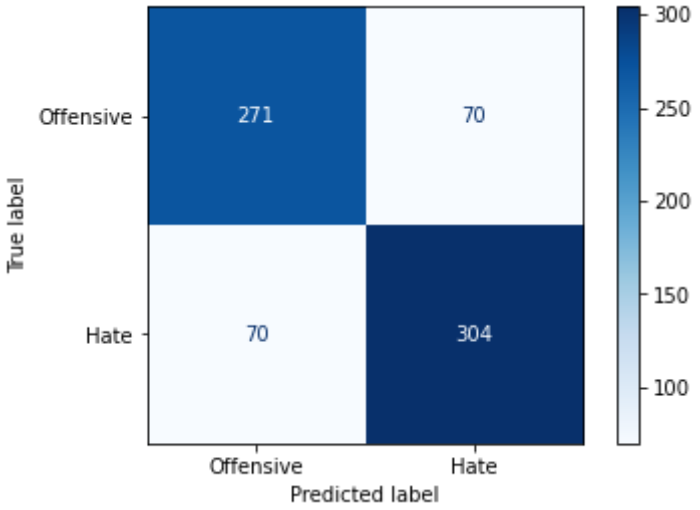
Decision Tree - TFIDF

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

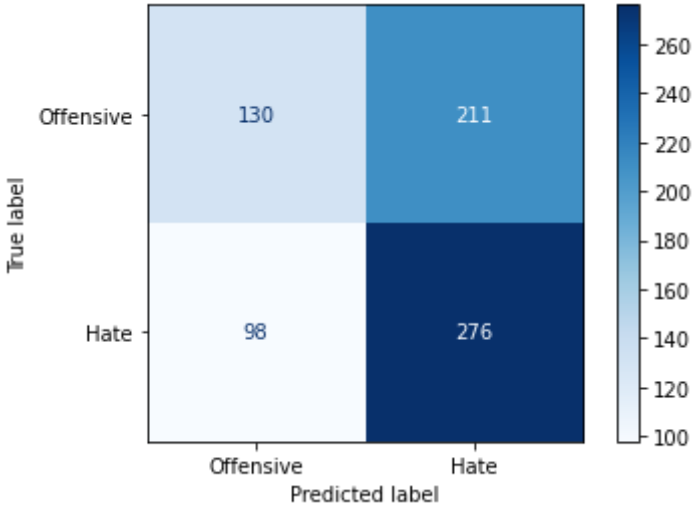
Naive Bayes - TFIDF

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_fo
```

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
```

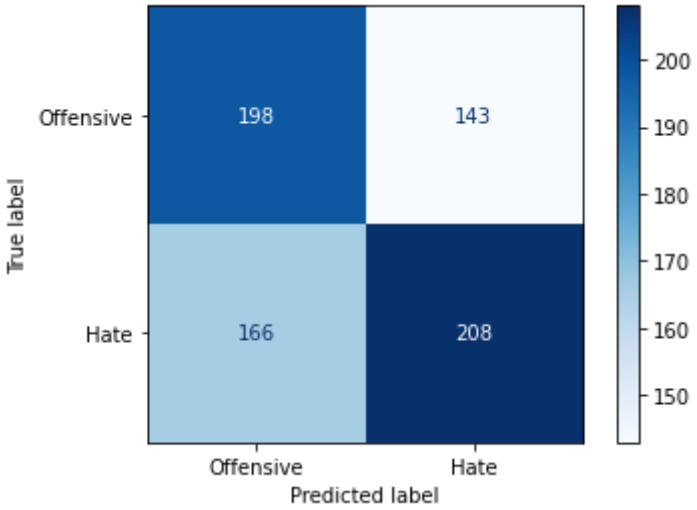
SVM - TFIDF

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)
```

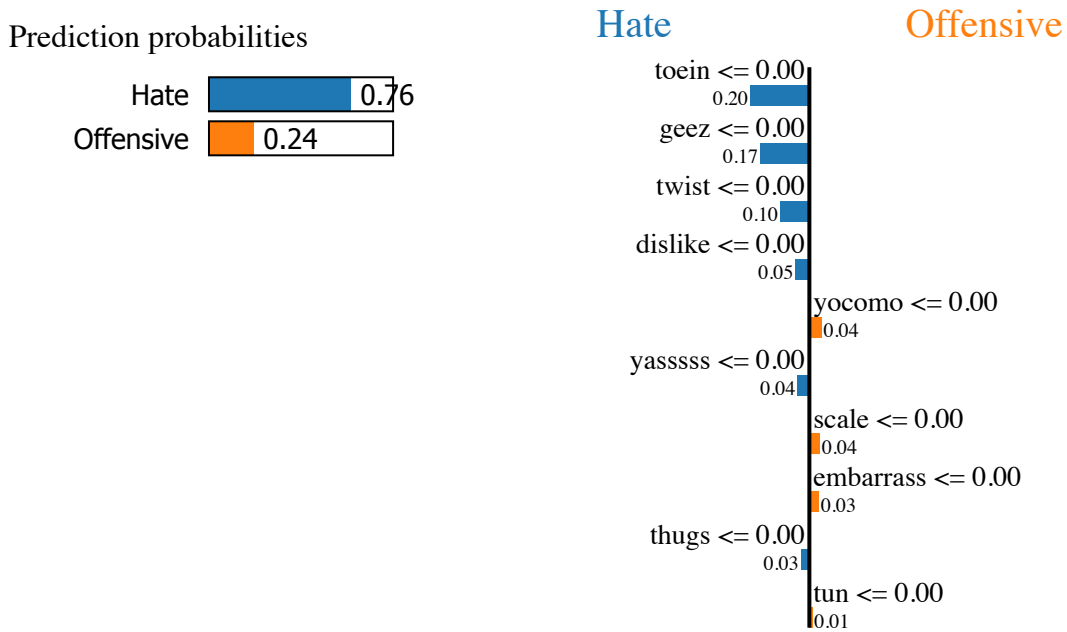
```
              precision    recall  f1-score   support

           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
tyn	0.00

In [ ]: