In [1]:
```
import jupyternotify
ip = get_ipython()
ip.register_magics(jupyternotify.JupyterNotifyMagics)
```

In [2]:
```
import pandas as pd
import numpy as np
from matplotlib import pyplot as plt
%matplotlib inline
```

In [3]:
```
df_full = pd.read_csv('full_tfidf_df.csv')
df_full.drop(columns=['Unnamed: 0'], inplace=True)
df_full[['num_tokens','mention_count','url_count','hashtag_count']] = df_full
df_full=df_full.astype('int')
```

In [4]:
```
df_full
```

Out[4]:

	aa	aaaaaaaaand	aaahhhhh	aahahah	aaliyah	aap	aaron	aaronmacgruder	aaryn	ab
0	0	0	0	0	0	0	0	0	0	C
1	0	0	0	0	0	0	0	0	0	C
2	0	0	0	0	0	0	0	0	0	C
3	0	0	0	0	0	0	0	0	0	C
4	0	0	0	0	0	0	0	0	0	C
...	...	...	...	...	...	...	...	...	...	..
20615	0	0	0	0	0	0	0	0	0	C
20616	0	0	0	0	0	0	0	0	0	C
20617	0	0	0	0	0	0	0	0	0	C
20618	0	0	0	0	0	0	0	0	0	C
20619	0	0	0	0	0	0	0	0	0	C

20620 rows × 11985 columns

In [86]:
```
features = df_full.drop(columns = 'class')
labels = df_full['class']
```

Learning Curve (for memory reasons, ran on another notebook)

from sklearn.model_selection import learning_curve from sklearn.linear_model import LogisticRegression

CV = 10 train_sizes = np.arange(500, 18500,500).tolist() train_sizes, train_scores, validation_scores = learning_curve(estimator = LogisticRegression(max_iter = 100000), X = features, y = labels, train_sizes = train_sizes, cv = CV, scoring = 'accuracy')

import matplotlib.pyplot as plt

train_scores_mean = train_scores.mean(axis = 1) validation_scores_mean = validation_scores.mean(axis = 1) train_error = 1- train_scores_mean validation_error = 1 - validation_scores_mean

plt.style.use('seaborn') plt.plot(train_sizes, train_error, label = 'Training error') plt.plot(train_sizes, validation_error, label = 'Validation error') plt.ylabel('Error', fontsize = 14) plt.xlabel('Training set size', fontsize = 14) plt.title('Learning curves for a logistic regression model', fontsize = 18, y = 1.03) plt.legend()

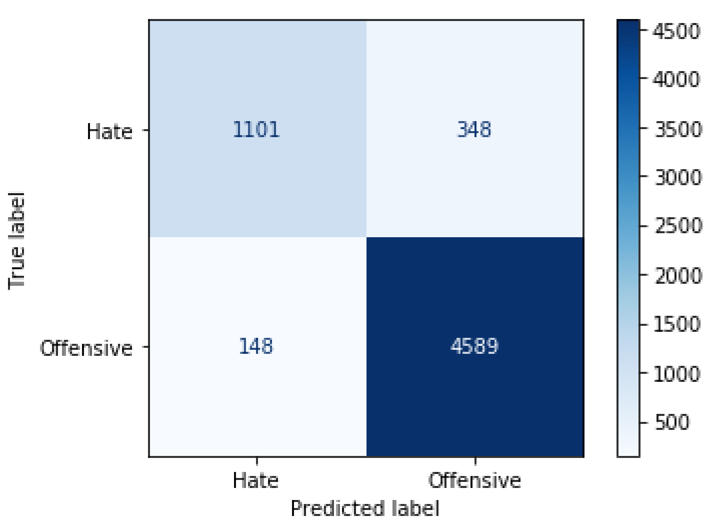
According to the learning curve, the validation error appears to increase after a training set of ~14500 (70%). Therefore, the optimal training size we'll use is 70%.

Base logistic regression

In [6]:
```
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import train_test_split
from sklearn.model_selection import cross_val_score, GridSearchCV, train_test
`
X_train, X_test, y_train, y_test = train_test_split(features, labels, test_si

clf = LogisticRegression(max_iter = 100000)
y_pred = clf.fit(X_train, y_train).predict(X_test)
```

In [7]:
```
from sklearn.metrics import plot_confusion_matrix
class_names = ['Hate', 'Offensive']
plot_confusion_matrix(clf, X_test, y_test, cmap=plt.cm.Blues, display_labels
```

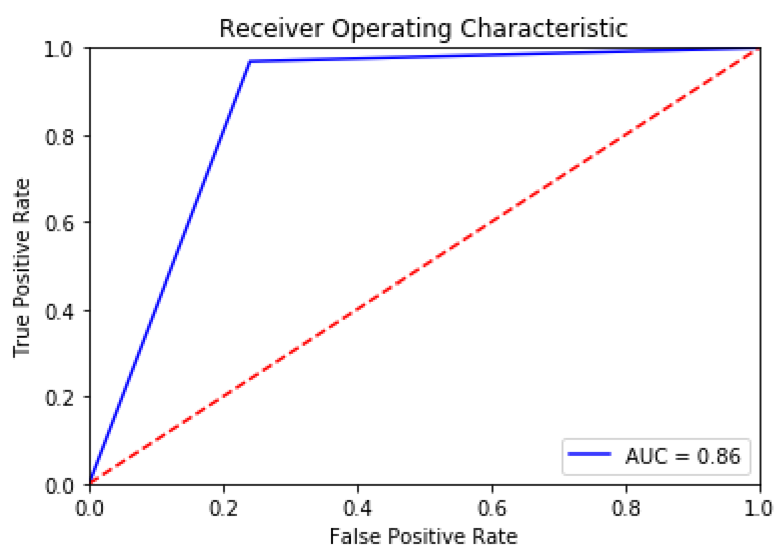
Out[7]:
```
<sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay at 0x1beceae
6048>
```



In [8]:
```
from sklearn.metrics import classification_report
report = classification_report(y_test, y_pred)
print(report)
```

```
              precision    recall  f1-score   support

           0       0.88      0.76      0.82      1449
           1       0.93      0.97      0.95      4737

    accuracy                           0.92      6186
   macro avg       0.91      0.86      0.88      6186
weighted avg       0.92      0.92      0.92      6186
```

In [9]:
```
import sklearn.metrics as metrics

fpr, tpr, threshold = metrics.roc_curve(y_test, y_pred)
roc_auc = metrics.auc(fpr, tpr)

plt.title('Receiver Operating Characteristic')
plt.plot(fpr, tpr, 'b', label = 'AUC = %0.2f' % roc_auc)
plt.legend(loc = 'lower right')
plt.plot([0, 1], [0, 1],'r--')
plt.xlim([0, 1])
plt.ylim([0, 1])
plt.ylabel('True Positive Rate')
plt.xlabel('False Positive Rate')
plt.show()
```



Using imblearn to balance samples

Random Undersampling

In [10]:
```
from imblearn.under_sampling import RandomUnderSampler
from imblearn.pipeline import Pipeline, make_pipeline

kf = KFold(n_splits = 10, random_state = 0)
param_grid = [{}]
clf_cv = GridSearchCV(LogisticRegression(max_iter = 100000), param_grid, cv=k


under_pipeline = make_pipeline(RandomUnderSampler(random_state=0), clf_cv)

under_pipeline
```

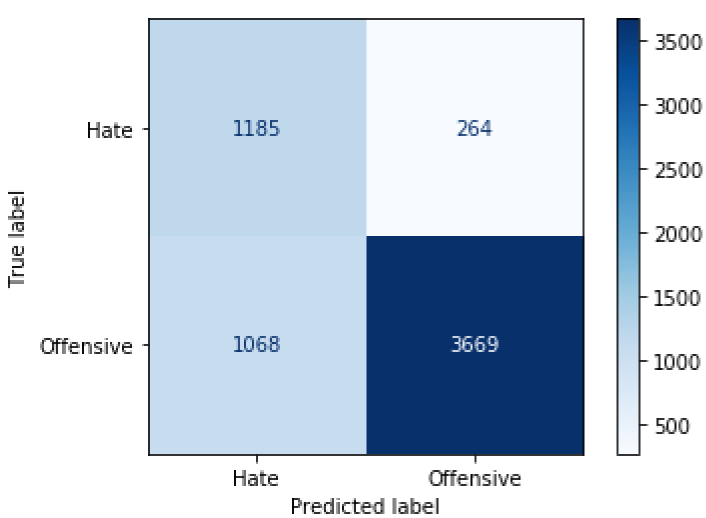
```
C:\Users\seanx\anaconda3\lib\site-packages\sklearn\model_selection\_split.p
y:297: FutureWarning: Setting a random_state has no effect since shuffle is
False. This will raise an error in 0.24. You should leave random_state to i
ts default (None), or set shuffle=True.
  FutureWarning
```

Out[10]:
```
Pipeline(steps=[('randomundersampler', RandomUnderSampler(random_state=0)),
                ('gridsearchcv',
                 GridSearchCV(cv=KFold(n_splits=10, random_state=0, shuffle
=False),
                              estimator=LogisticRegression(max_iter=10000
0),
                              param_grid=[{}]))])
```

In [11]:
```
y_preds_under = under_pipeline.fit(X_train, y_train).predict(X_test)
```

In [12]:
```
plot_confusion_matrix(under_pipeline, X_test, y_test, cmap=plt.cm.Blues, disp
```

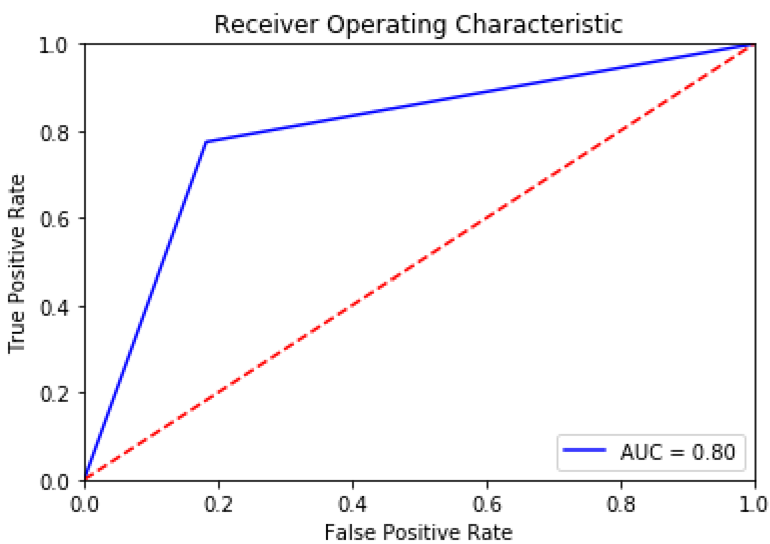
Out[12]:
```
<sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay at 0x1bed0da
e288>
```



In [13]:
```
report = classification_report(y_test, y_preds_under)
print(report)
```

```
              precision    recall  f1-score   support

           0       0.53      0.82      0.64      1449
           1       0.93      0.77      0.85      4737

    accuracy                           0.78      6186
   macro avg       0.73      0.80      0.74      6186
weighted avg       0.84      0.78      0.80      6186
```

In [14]:
```
fpr, tpr, threshold = metrics.roc_curve(y_test, y_preds_under)
roc_auc = metrics.auc(fpr, tpr)

plt.title('Receiver Operating Characteristic')
plt.plot(fpr, tpr, 'b', label = 'AUC = %0.2f' % roc_auc)
plt.legend(loc = 'lower right')
plt.plot([0, 1], [0, 1],'r--')
plt.xlim([0, 1])
plt.ylim([0, 1])
plt.ylabel('True Positive Rate')
plt.xlabel('False Positive Rate')
plt.show()
```



Random Oversampling

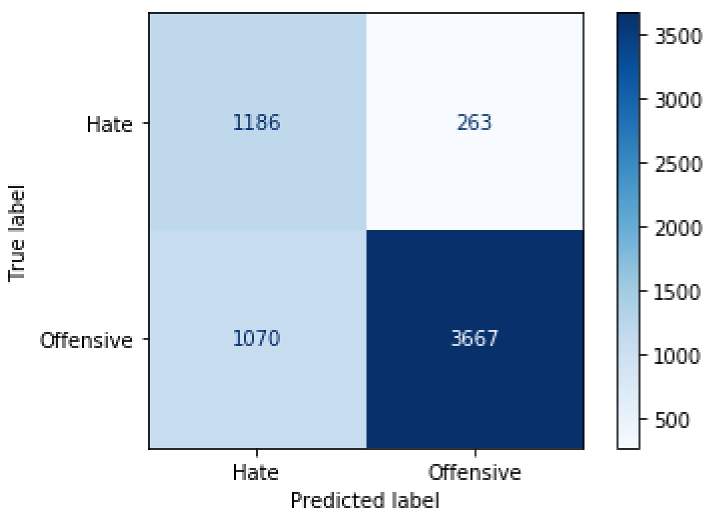
In [15]:
```
from imblearn.over_sampling import RandomOverSampler


param_grid = [{}]

over_pipeline = make_pipeline(RandomOverSampler(random_state=0), clf_cv)
over_pipeline
```

Out[15]:
```
Pipeline(steps=[('randomoversampler', RandomOverSampler(random_state=0)),
                ('gridsearchcv',
                 GridSearchCV(cv=KFold(n_splits=10, random_state=0, shuffle
=False),
                              estimator=LogisticRegression(max_iter=10000
0),
                              param_grid=[{}]))])
```

In [16]:
```
y_preds_over = over_pipeline.fit(X_train, y_train).predict(X_test)
```

In [17]:
```
plot_confusion_matrix(over_pipeline, X_test, y_test, cmap=plt.cm.Blues, displ
```

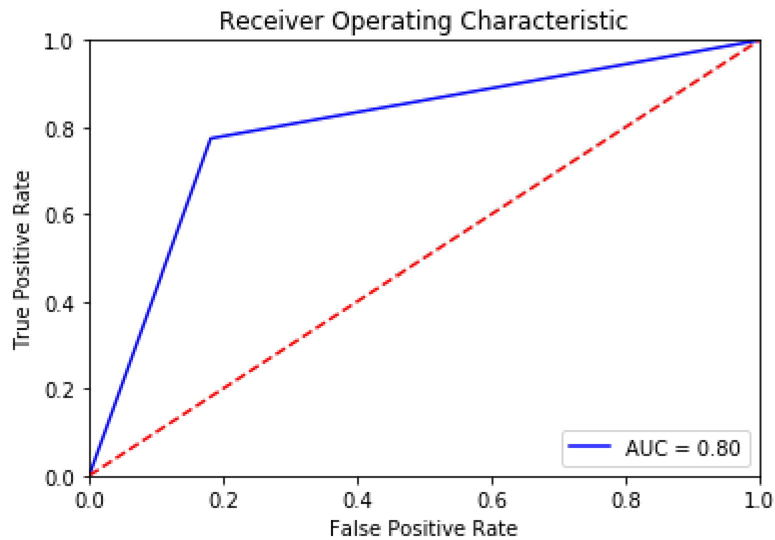
Out[17]:
```
<sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay at 0x1bed45f
bf08>
```



In [18]:
```
report = classification_report(y_test, y_preds_over)
print(report)
```

```
              precision    recall  f1-score   support

           0       0.53      0.82      0.64      1449
           1       0.93      0.77      0.85      4737

    accuracy                           0.78      6186
   macro avg       0.73      0.80      0.74      6186
weighted avg       0.84      0.78      0.80      6186
```

In [19]:
```
fpr, tpr, threshold = metrics.roc_curve(y_test, y_preds_over)
roc_auc = metrics.auc(fpr, tpr)

plt.title('Receiver Operating Characteristic')
plt.plot(fpr, tpr, 'b', label = 'AUC = %0.2f' % roc_auc)
plt.legend(loc = 'lower right')
plt.plot([0, 1], [0, 1],'r--')
plt.xlim([0, 1])
plt.ylim([0, 1])
plt.ylabel('True Positive Rate')
plt.xlabel('False Positive Rate')
plt.show()
```



Near Miss undersampling

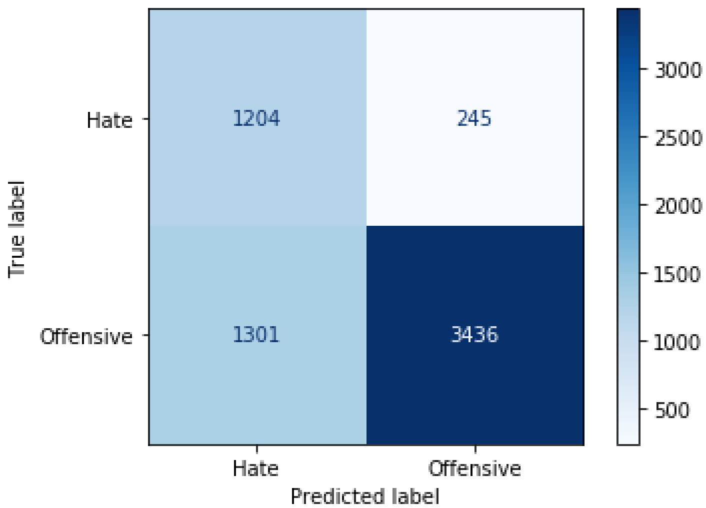
In [20]:
```
from imblearn.under_sampling import NearMiss

param_grid = [{}]
nm1 = NearMiss()
nearmiss1_pipeline = make_pipeline(nm1, clf_cv)

nearmiss1_pipeline
```

Out[20]:
```
Pipeline(steps=[('nearmiss', NearMiss()),
                ('gridsearchcv',
                 GridSearchCV(cv=KFold(n_splits=10, random_state=0, shuffle
=False),
                              estimator=LogisticRegression(max_iter=10000
0),
                              param_grid=[{}]))])
```

In [21]:
```
y_preds_nm1 = nearmiss1_pipeline.fit(X_train, y_train).predict(X_test)
```

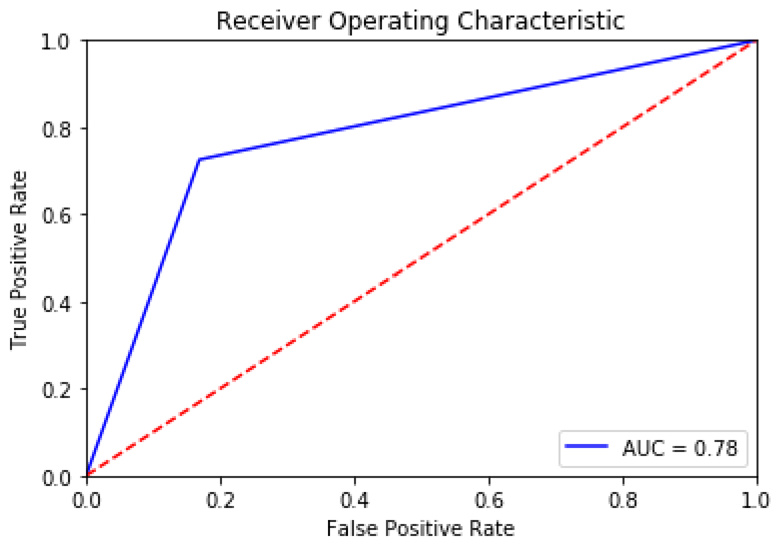
In [22]:
```
lot_confusion_matrix(nearmiss1_pipeline, X_test, y_test, cmap=plt.cm.Blues, d
eport = classification_report(y_test, y_preds_nm1)
rint(report)
```

```
              precision    recall  f1-score   support

           0       0.48      0.83      0.61      1449
           1       0.93      0.73      0.82      4737

    accuracy                           0.75      6186
   macro avg       0.71      0.78      0.71      6186
weighted avg       0.83      0.75      0.77      6186
```



In [23]:

```
fpr, tpr, threshold = metrics.roc_curve(y_test, y_preds_nm1)
roc_auc = metrics.auc(fpr, tpr)

plt.title('Receiver Operating Characteristic')
plt.plot(fpr, tpr, 'b', label = 'AUC = %0.2f' % roc_auc)
plt.legend(loc = 'lower right')
plt.plot([0, 1], [0, 1],'r--')
plt.xlim([0, 1])
plt.ylim([0, 1])
plt.ylabel('True Positive Rate')
plt.xlabel('False Positive Rate')
plt.show()
```



SMOTE oversampling

In [24]:

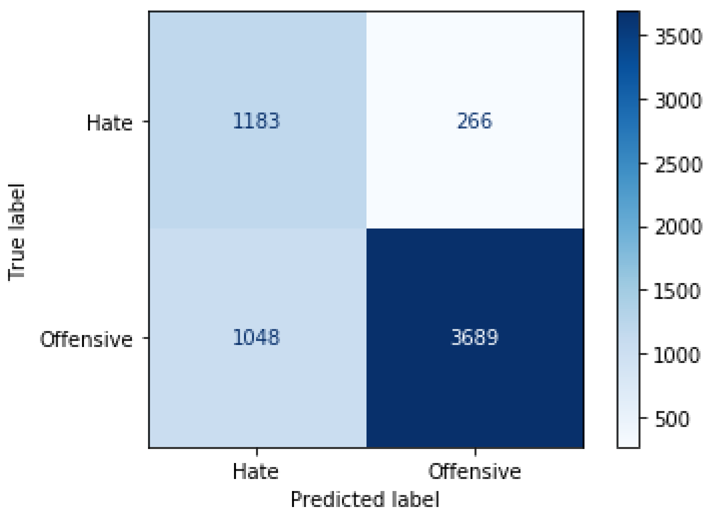
```
from imblearn.over_sampling import SMOTE

param_grid = [{}]
smote_pipeline = make_pipeline(SMOTE(), clf_cv)

smote_pipeline
```

Out[24]:

```
Pipeline(steps=[('smote', SMOTE()),
                ('gridsearchcv',
                 GridSearchCV(cv=KFold(n_splits=10, random_state=0, shuffle
=False),
                              estimator=LogisticRegression(max_iter=10000
0),
                              param_grid=[{}]))])
```

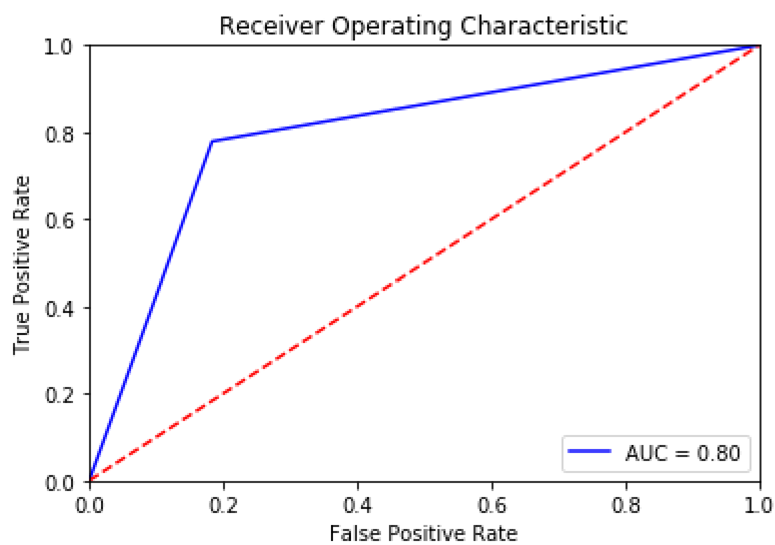
In [25]:
```
y_preds_smote = smote_pipeline.fit(X_train, y_train).predict(X_test)
plot_confusion_matrix(smote_pipeline, X_test, y_test, cmap=plt.cm.Blues, disp
report = classification_report(y_test, y_preds_smote)
print(report)
```

```
              precision    recall  f1-score   support

           0       0.53      0.82      0.64      1449
           1       0.93      0.78      0.85      4737

    accuracy                           0.79      6186
   macro avg       0.73      0.80      0.75      6186
weighted avg       0.84      0.79      0.80      6186
```



In [26]:

```
fpr, tpr, threshold = metrics.roc_curve(y_test, y_preds_smote)
roc_auc = metrics.auc(fpr, tpr)

plt.title('Receiver Operating Characteristic')
plt.plot(fpr, tpr, 'b', label = 'AUC = %0.2f' % roc_auc)
plt.legend(loc = 'lower right')
plt.plot([0, 1], [0, 1],'r--')
plt.xlim([0, 1])
plt.ylim([0, 1])
plt.ylabel('True Positive Rate')
plt.xlabel('False Positive Rate')
plt.show()
```



Doesn't seem like balancing the dataset actually increases model performance.

Grid Search - Hyperparameter Tuning for Logistic Regression

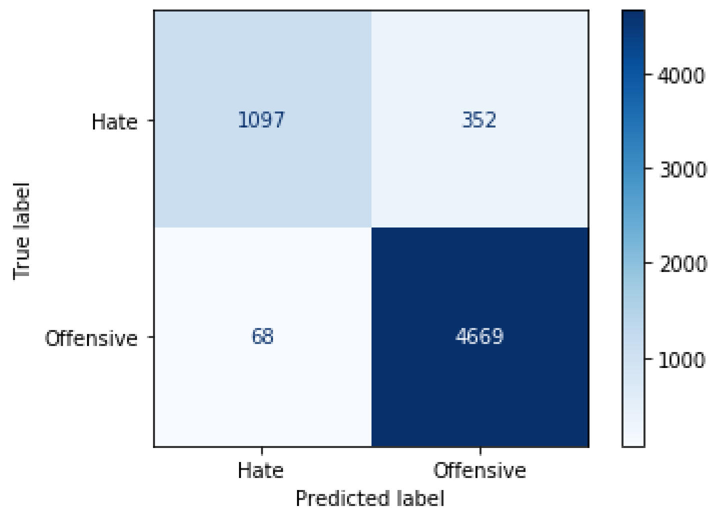
In [28]:

```
param_grid={'C': [0.001, 0.01, 0.1, 1, 10]}

grid_clf_acc = GridSearchCV(clf, param_grid = param_grid, cv=kf)
y_preds_grid = grid_clf_acc.fit(X_train, y_train).predict(X_test)
```

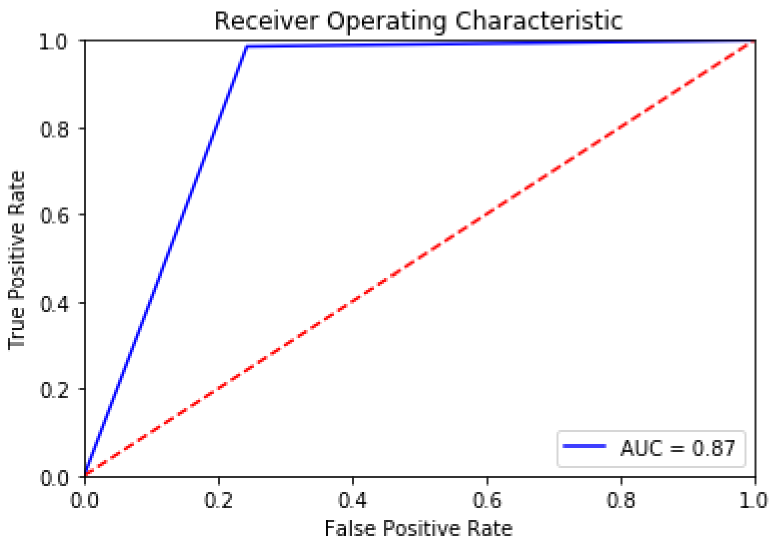
In [29]:

```
plot_confusion_matrix(grid_clf_acc, X_test, y_test, cmap=plt.cm.Blues, displa
report = classification_report(y_test, y_preds_grid)
print(report)
```

```
              precision    recall  f1-score   support

           0       0.94      0.76      0.84      1449
           1       0.93      0.99      0.96      4737

    accuracy                           0.93      6186
   macro avg       0.94      0.87      0.90      6186
weighted avg       0.93      0.93      0.93      6186
```



In [30]:
```
fpr, tpr, threshold = metrics.roc_curve(y_test, y_preds_grid)
roc_auc = metrics.auc(fpr, tpr)

plt.title('Receiver Operating Characteristic')
plt.plot(fpr, tpr, 'b', label = 'AUC = %0.2f' % roc_auc)
plt.legend(loc = 'lower right')
plt.plot([0, 1], [0, 1],'r--')
plt.xlim([0, 1])
plt.ylim([0, 1])
plt.ylabel('True Positive Rate')
plt.xlabel('False Positive Rate')
plt.show()
```



In [31]:
```
grid_clf_acc.best_estimator_
```

Out[31]:
```
LogisticRegression(C=0.001, max_iter=100000)
```

In [32]:
```
importance_logreg = grid_clf_acc.best_estimator_.coef_.tolist()[0]

feature_importance_logreg = pd.DataFrame(list(zip(features,importance_logreg)
feature_importance_logreg = feature_importance_logreg.sort_values(by='importa
feature_importance_logreg.head(10)
```

Out[32]:

	features	importance
11983	hashtag_count	-0.016116
8319	pussy	-0.002975
3077	dyke	-0.000998
7163	nicca	-0.000982
2485	darkie	-0.000970
7180	nig	-0.000965
987	bitch	-0.000712
7164	niccas	-0.000317
11826	yass	-0.000317
2257	cracker	-0.000317

Bagging, Boosting, and Stacking

In [33]:
```
best_clf = grid_clf_acc.best_estimator_
```

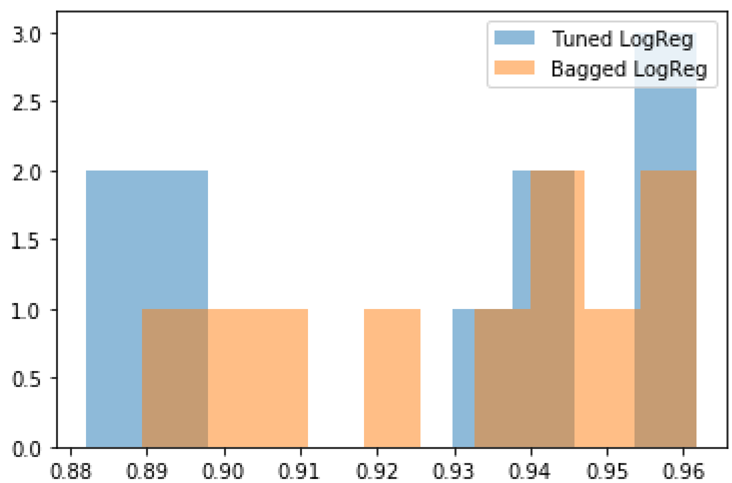
Bagging

In [34]:
```
from sklearn.ensemble import BaggingClassifier
from scipy import stats

vanilla_scores = cross_val_score(best_clf, features, labels, cv=10)
```

In [35]:
```
samples_list = [0.1, 0.2, 0.3, 0.4]
bagging_scores = []
clfs = []
for i in range(len(samples_list)):
    print(i)
    bagging_clf = BaggingClassifier(best_clf, max_samples=samples_list[i], ra
    clfs.append(bagging_clf)
    score = cross_val_score(bagging_clf, features, labels, cv=10)
    bagging_scores.append(score)
    print(score)
```

```
0
[0.94180407 0.88942774 0.8971872  0.90785645 0.92240543 0.9442289
 0.93743938 0.95392823 0.95538312 0.96168768]
1
[0.94180407 0.88942774 0.8971872  0.90785645 0.92240543 0.9442289
 0.93743938 0.95392823 0.95538312 0.96168768]
2
[0.94180407 0.88942774 0.8971872  0.90785645 0.92240543 0.9442289
 0.93743938 0.95392823 0.95538312 0.96168768]
3
[0.94180407 0.88942774 0.8971872  0.90785645 0.92240543 0.9442289
 0.93743938 0.95392823 0.95538312 0.96168768]
```

In [63]:
```
plt.hist(vanilla_scores,  alpha=0.5, label='Tuned LogReg')
plt.hist(bagging_scores[0],  alpha=0.5, label='Bagged LogReg')
plt.legend(loc='upper right')
plt.show()
```



In [62]:
```
print(stats.ttest_ind(bagging_scores[0], vanilla_scores))
```

```
Ttest_indResult(statistic=0.4737809401484407, pvalue=0.6413519547364437)
```

In [58]:
```
print(vanilla_scores.mean())
print(vanilla_scores.std())
```

```
0.9250242483026188
0.030157870447736736
```

In [61]:
```
print(bagging_scores[0].mean())
print(bagging_scores[0].std())
```

```
0.9311348205625606
0.02424054192006813
```

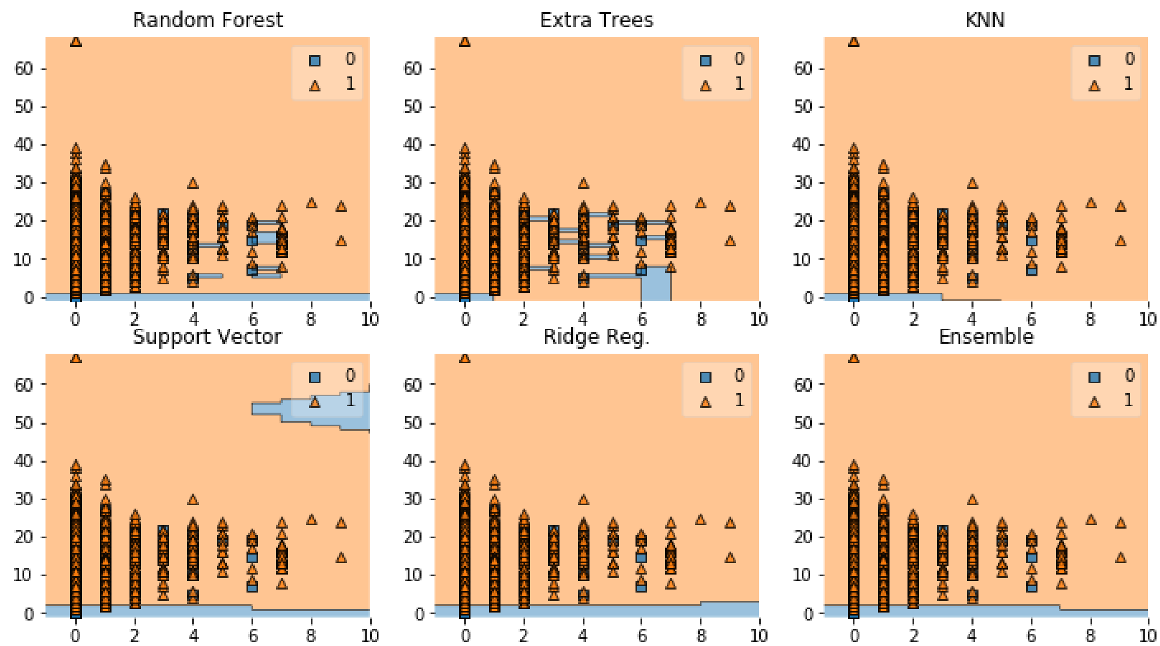
While bagging scores were slightly higher, there was no significant difference in accuracy between bagging and tuned logistic algorithm. Changing max_sample parameter had no effect.

Voting

In [77]:
```
from sklearn.ensemble import BaggingClassifier, ExtraTreesClassifier, RandomF
from sklearn.neighbors import KNeighborsClassifier
from sklearn.linear_model import RidgeClassifier
from sklearn.ensemble import VotingClassifier
from sklearn.svm import SVC

rf = RandomForestClassifier()
et = ExtraTreesClassifier()
knn = KNeighborsClassifier()
svc = SVC()
rg = RidgeClassifier()

clf_array = [rf, et, knn, svc, rg]

# Set up voting
eclf = VotingClassifier(estimators=[('Random Forests', rf), ('Extra Trees', e

for clf, label in zip([rf, et, knn, svc, rg,eclf], ['Random Forest', 'Extra T
    scores = cross_val_score(clf, features, labels, cv=10, scoring='accuracy'
    print("Mean: {0:.3f}, std: (+/-) {1:.3f} [{2}]".format(scores.mean(), scc
```

```
Mean: 0.939, std: (+/-) 0.018 [Random Forest]
Mean: 0.936, std: (+/-) 0.019 [Extra Trees]
Mean: 0.938, std: (+/-) 0.018 [KNeighbors]
Mean: 0.940, std: (+/-) 0.017 [SVC]
Mean: 0.935, std: (+/-) 0.019 [Ridge Classifier]
Mean: 0.940, std: (+/-) 0.017 [Ensemble]
```

In [97]:
```
from mlxtend.plotting import plot_decision_regions
import matplotlib.gridspec as gridspec
import itertools

gs = gridspec.GridSpec(3, 3)
fig = plt.figure(figsize=(12, 10))
label = ['Random Forest', 'Extra Trees', 'KNN', 'Support Vector','Ridge Reg.'

for clf, lab, grd in zip([rf, et, knn, svc, rg, eclf], label, itertools.produ
    clf.fit(features[['hashtag_count', 'num_tokens']], labels)
    ax = plt.subplot(gs[grd[0], grd[1]])
    fig = plot_decision_regions(X=np.array(features[['hashtag_count', 'num_to
                                y=np.array(labels), clf=clf)
    plt.title(lab)
```



Stacking

In [100]:

```
from itertools import combinations
from mlens.ensemble import SuperLearner
from sklearn.metrics import accuracy_score


names = ['Random Forest', 'Extra Trees', 'KNeighbors', 'SVC', 'Ridge Classifi
X_train, X_test, y_train, y_test = train_test_split(features, labels, test_si

def zip_stacked_classifiers(*args):
    to_zip = []
    for arg in args:
        combined_items = sum([list(map(list, combinations(arg, i))) for i in
        combined_items = filter(lambda x: len(x) > 0, combined_items)
        to_zip.append(combined_items)

    return zip(to_zip[0], to_zip[1])

stacked_clf_list = zip_stacked_classifiers(clf_array, names)
best_combination = [0.00, ""]
for clf in stacked_clf_list:

    ensemble = SuperLearner(scorer = accuracy_score,
                            random_state = 0,
                            folds = 10)
    ensemble.add(clf[0])
    ensemble.add_meta(best_clf)
    ensemble.fit(X_train, y_train)
    preds = ensemble.predict(X_test)
    accuracy = accuracy_score(preds, y_test)

    #if accuracy > best_combination[0]:
     #    best_combination[0] = accuracy
    #    best_combination[1] = clf[1]
   # print("Accuracy score: {0:.3f} {1}").format(accuracy, clf[1])

#print("\nBest stacking model is {} with accuracy of: {:.3f}").format(best_co
```

```
 itertools import combinations\nfrom mlens.en....format(best_combination
[1], best_combination[0])', 'silent': False, 'stop_on_error': True, 'store_
history': True, 'user_expressions': {}}, 'header': {'date': datetime.dateti
me(2020, 12, 20, 14, 46, 50, 571962, tzinfo=datetime.timezone.utc), 'msg_i
d': '0137129743d14d668a145b3d3b0afe82', 'msg_type': 'execute_request', 'ses
sion': '25cdb9d2e16b4b85861613ebb4cb8619', 'username': 'username', 'versio
n': '5.2'}, 'metadata': {}, 'msg_id': '0137129743d14d668a145b3d3b0afe82',
 'msg_type': 'execute_request', 'parent_header': {}})
    536             self._publish_execute_input(code, parent, self.executio
n_count)
    537
    538         reply_content = yield gen.maybe_future(
    539             self.do_execute(
    540                 code, silent, store_history,
--> 541                 user_expressions, allow_stdin,
        user_expressions = {}
        allow_stdin = True
    542             )
    543         )
```

In [ ]: