Random Forest and Logistic Regression with Boosting and Bagging

Random Forest using Grid Search for Optimization

In [4]:
```
import pandas as pd
import numpy as np
from matplotlib import pyplot as plt
%matplotlib inline
```

In [5]:
```
df = pd.read_csv('tfidf_df.csv')
df.drop(df.columns[0], axis=1,inplace=True)
#df['class'] = df['class'].astype('int')
df
```

Out[5]:

	aa	aaaaaaaaand	aap	aaron	aaronmacgruder	ab	ability	abortion	about	abraham	...	z
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	...	
2	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	...	
4	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	...	
2856	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	...	
2858	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	...	

2860 rows × 4455 columns

In [60]:
```
features = df.drop(columns = 'class')
labels = df['class']
```

In [7]:
```
from sklearn.model_selection import RandomizedSearchCV
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import train_test_split
from sklearn.model_selection import cross_val_score, train_test_split, KFold

X_train, X_test, y_train, y_test = train_test_split(features, labels, test_size =
rf = RandomForestClassifier(random_state=123)
rf.get_params()
```

Out[7]:
```
{'bootstrap': True,
 'ccp_alpha': 0.0,
 'class_weight': None,
 'criterion': 'gini',
 'max_depth': None,
 'max_features': 'auto',
 'max_leaf_nodes': None,
 'max_samples': None,
 'min_impurity_decrease': 0.0,
 'min_impurity_split': None,
 'min_samples_leaf': 1,
 'min_samples_split': 2,
 'min_weight_fraction_leaf': 0.0,
 'n_estimators': 100,
 'n_jobs': None,
 'oob_score': False,
 'random_state': 123,
 'verbose': 0,
 'warm_start': False}
```

Use random search to find the approximate range for the best model

In [8]:
```
# Number of trees in random forest
n_estimators = np.arange(100,2000,50).tolist()
# Number of features to consider at every split
max_features = ['auto', 'sqrt']
# Maximum number of levels in tree
max_depth = np.arange(2,40,1).tolist()
max_depth.append(None)
# Minimum number of samples required to split a node
min_samples_split = np.arange(2,20,2).tolist()
# Minimum number of samples required at each leaf node
min_samples_leaf = [1, 2, 3, 4]
# Method of selecting samples for training each tree
bootstrap = [True, False]

# Create the random grid
random_grid = {'n_estimators': n_estimators,
               'max_features': max_features,
               'max_depth': max_depth,
               'min_samples_split': min_samples_split,
               'min_samples_leaf': min_samples_leaf,
               'bootstrap': bootstrap}
```

In [13]:
```
# Use the random grid to search for best hyperparameters
rf_random = RandomizedSearchCV(estimator=rf, param_distributions=random_grid,
                              n_iter = 50, scoring='accuracy',
                              cv = 5, verbose=2, random_state=42,
                              return_train_score=True)

# Fit the random search model
rf_random.fit(X_train, y_train)
```

```
s=sqrt, max_depth=39, bootstrap=True, total= 11.7s
[CV] n_estimators=1600, min_samples_split=4, min_samples_leaf=4, max_features
=sqrt, max_depth=39, bootstrap=True
[CV]  n_estimators=1600, min_samples_split=4, min_samples_leaf=4, max_feature
s=sqrt, max_depth=39, bootstrap=True, total=  11.6s
[CV] n_estimators=1450, min_samples_split=10, min_samples_leaf=3, max_feature
s=sqrt, max_depth=6, bootstrap=False
[CV]  n_estimators=1450, min_samples_split=10, min_samples_leaf=3, max_featur
es=sqrt, max_depth=6, bootstrap=False, total=   5.3s
[CV] n_estimators=1450, min_samples_split=10, min_samples_leaf=3, max_feature
s=sqrt, max_depth=6, bootstrap=False
[CV]  n_estimators=1450, min_samples_split=10, min_samples_leaf=3, max_featur
es=sqrt, max_depth=6, bootstrap=False, total=   5.3s
[CV] n_estimators=1450, min_samples_split=10, min_samples_leaf=3, max_feature
s=sqrt, max_depth=6, bootstrap=False
[CV]  n_estimators=1450, min_samples_split=10, min_samples_leaf=3, max_featur
es=sqrt, max_depth=6, bootstrap=False, total=   5.3s
[CV] n_estimators=1450, min_samples_split=10, min_samples_leaf=3, max_feature
s=sqrt, max_depth=6, bootstrap=False
[CV]  n_estimators=1450, min_samples_split=10, min_samples_leaf=3, max_featur
```

In [14]:
```
rf_random.best_params_
```

Out[14]:
```
{'n_estimators': 1750,
 'min_samples_split': 8,
 'min_samples_leaf': 2,
 'max_features': 'auto',
 'max_depth': 33,
 'bootstrap': True}
```

Run random search again with more precise ranges

In [15]:
```
# Number of trees in random forest
n_estimators = np.arange(1550,1870,20).tolist()
# Number of features to consider at every split
max_features = ['auto']
# Maximum number of levels in tree
max_depth = np.arange(3,40,3).tolist()
max_depth.append(None)
# Minimum number of samples required to split a node
min_samples_split = np.arange(6,17,2).tolist()
# Minimum number of samples required at each leaf node
min_samples_leaf = [2, 3, 4, 5]
# Method of selecting samples for training each tree
bootstrap = [True]

# Create the random grid
random_grid = {'n_estimators': n_estimators,
               'max_features': max_features,
               'max_depth': max_depth,
               'min_samples_split': min_samples_split,
               'min_samples_leaf': min_samples_leaf,
               'bootstrap': bootstrap}
```

In [16]:
```
# Use the random grid to search for best hyperparameters
rf_random = RandomizedSearchCV(estimator=rf, param_distributions=random_grid,
                              n_iter = 50, scoring='accuracy',
                              cv = 4, verbose=2, random_state=42, n_jobs=1,
                              return_train_score=True)

# Fit the random search model
rf_random.fit(X_train, y_train)
```

```
                                        'max_depth': [3, 6, 9, 12, 15, 18, 2
1,
                                                      24, 27, 30, 33, 36, 39,
                                                      None],
                                        'max_features': ['auto'],
                                        'min_samples_leaf': [2, 3, 4, 5],
                                        'min_samples_split': [6, 8, 10, 12, 1
4,
                                                              16],
                                        'n_estimators': [1550, 1570, 1590, 16
10,
                                                         1630, 1650, 1670, 16
90,
                                                         1710, 1730, 1750, 17
70,
                                                         1790, 1810, 1830,
                                                         1850]},
                   random_state=42, return_train_score=True, scoring='accurac
y',
                   verbose=2)
```

In [17]:
```
rf_random.best_params_
```

Out[17]:
```
{'n_estimators': 1570,
 'min_samples_split': 12,
 'min_samples_leaf': 4,
 'max_features': 'auto',
 'max_depth': 3,
 'bootstrap': True}
```

Use grid search to find the best model

In [20]:
```python
from sklearn.model_selection import GridSearchCV

# Create the parameter grid based on the results of random search
param_grid = {
    'bootstrap': [True],
    'max_depth': [3,4,5],
    'max_features': ['auto', 'sqrt'],
    'min_samples_leaf': [3, 4],
    'min_samples_split': [8, 9, 10, 11, 12],
    'n_estimators': [1550,1570,1590,1610, 1630]
}

# Create a base model
rf = RandomForestClassifier(random_state = 42)

# Instantiate the grid search model
grid_search = GridSearchCV(estimator = rf, param_grid = param_grid,
                          cv = 4, verbose = 2, return_train_score=True)
```

In [21]:
```python
# Fit the grid search to the data
grid_search.fit(X_train, y_train)
```

```
_samples_split=9, n_estimators=1590, total=   4.2s
[CV] bootstrap=True, max_depth=5, max_features=sqrt, min_samples_leaf=4, min_
samples_split=9, n_estimators=1610
[CV]  bootstrap=True, max_depth=5, max_features=sqrt, min_samples_leaf=4, min
_samples_split=9, n_estimators=1610, total=   4.4s
[CV] bootstrap=True, max_depth=5, max_features=sqrt, min_samples_leaf=4, min_
samples_split=9, n_estimators=1610
[CV]  bootstrap=True, max_depth=5, max_features=sqrt, min_samples_leaf=4, min
_samples_split=9, n_estimators=1610, total=   4.3s
[CV] bootstrap=True, max_depth=5, max_features=sqrt, min_samples_leaf=4, min_
samples_split=9, n_estimators=1610
[CV]  bootstrap=True, max_depth=5, max_features=sqrt, min_samples_leaf=4, min
_samples_split=9, n_estimators=1610, total=   4.3s
[CV] bootstrap=True, max_depth=5, max_features=sqrt, min_samples_leaf=4, min_
samples_split=9, n_estimators=1610
[CV]  bootstrap=True, max_depth=5, max_features=sqrt, min_samples_leaf=4, min
_samples_split=9, n_estimators=1610, total=   4.5s
[CV] bootstrap=True, max_depth=5, max_features=sqrt, min_samples_leaf=4, min_
samples_split=9, n_estimators=1630
```

In [22]:
```python
grid_search.best_params_
```

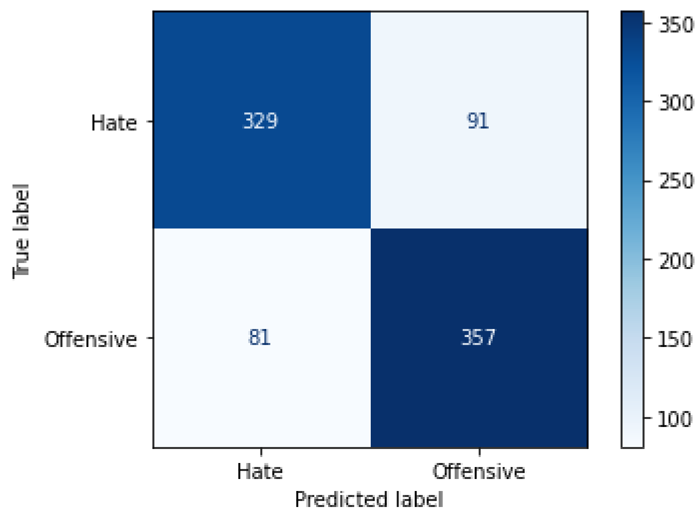
Out[22]:
```
{'bootstrap': True,
 'max_depth': 5,
 'max_features': 'auto',
 'min_samples_leaf': 3,
 'min_samples_split': 8,
 'n_estimators': 1550}
```

In [23]:
```python
best_rf = grid_search.best_estimator_
y_pred_best_rf = best_rf.fit(X_train, y_train).predict(X_test)
```

View the confusion matrix for the best model

In [24]:
```
from sklearn.metrics import plot_confusion_matrix
class_names = ['Hate', 'Offensive']
plot_confusion_matrix(best_rf, X_test, y_test, cmap=plt.cm.Blues, display_labels
```

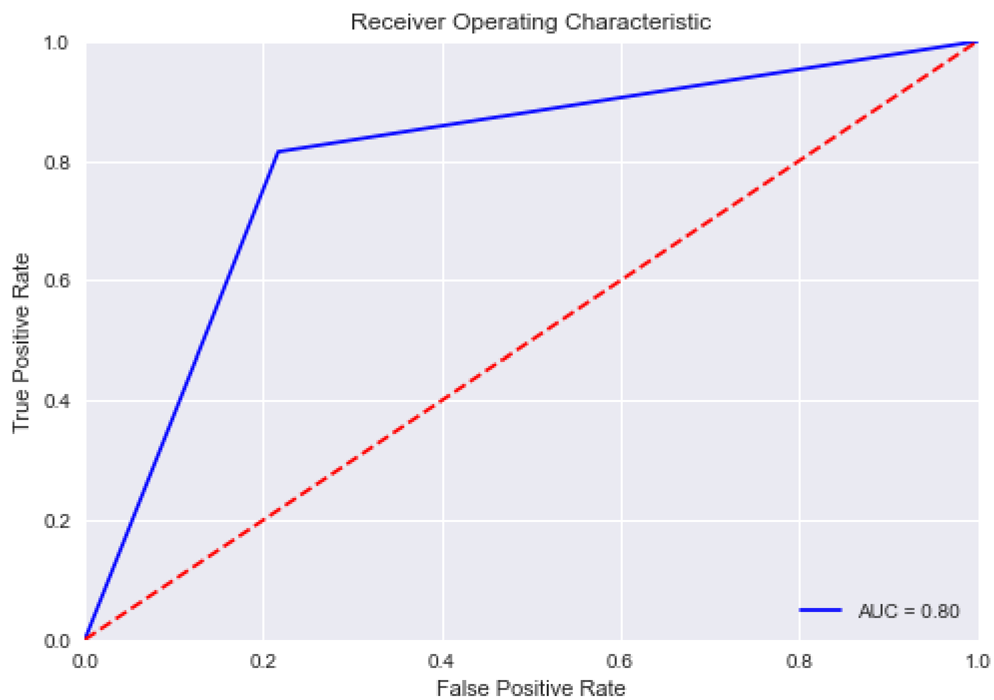
Out[24]:
```
<sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay at 0x19eb19c5f60
>
```



In [25]:
```
from sklearn.metrics import classification_report
report = classification_report(y_test, y_pred_best_rf)
print(report)
```

```
              precision    recall  f1-score   support

           0       0.80      0.78      0.79       420
           1       0.80      0.82      0.81       438

    accuracy                           0.80       858
   macro avg       0.80      0.80      0.80       858
weighted avg       0.80      0.80      0.80       858
```

In [77]:
```
import sklearn.metrics as metrics

fpr, tpr, threshold = metrics.roc_curve(y_test, y_pred_best_rf)
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()
```



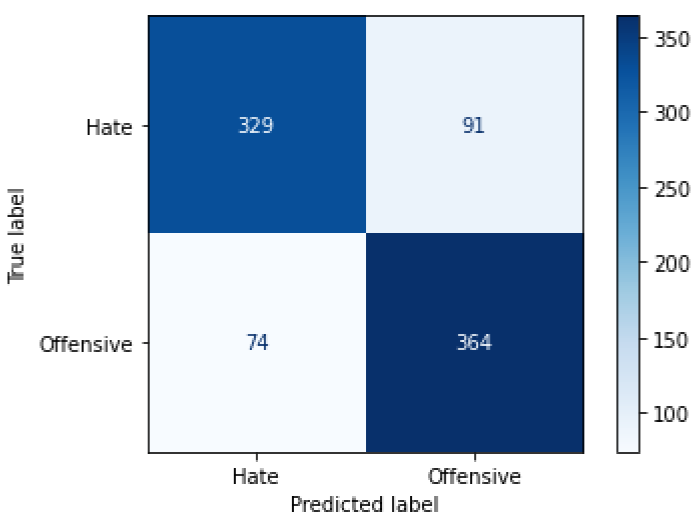
Compared to the base model:

In [26]:
```
rf_base = RandomForestClassifier()
rf_base.get_params()
```

Out[26]:
```
{'bootstrap': True,
 'ccp_alpha': 0.0,
 'class_weight': None,
 'criterion': 'gini',
 'max_depth': None,
 'max_features': 'auto',
 'max_leaf_nodes': None,
 'max_samples': None,
 'min_impurity_decrease': 0.0,
 'min_impurity_split': None,
 'min_samples_leaf': 1,
 'min_samples_split': 2,
 'min_weight_fraction_leaf': 0.0,
 'n_estimators': 100,
 'n_jobs': None,
 'oob_score': False,
 'random_state': None,
 'verbose': 0,
 'warm_start': False}
```

In [27]:
```
y_pred_base = rf_base.fit(X_train, y_train).predict(X_test)
plot_confusion_matrix(rf_base, X_test, y_test, cmap=plt.cm.Blues, display_labels
```

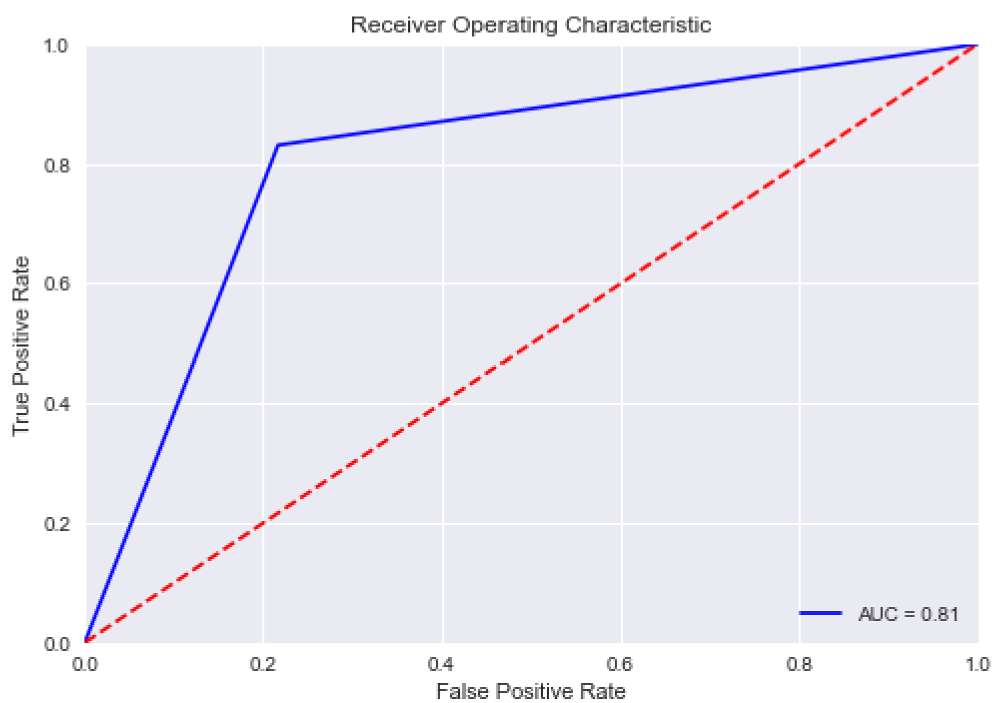
Out[27]:
```
<sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay at 0x19eb1a2f940
>
```



In [28]:
```
report = classification_report(y_test, y_pred_base)
print(report)
```

```
              precision    recall  f1-score   support

           0       0.82      0.78      0.80       420
           1       0.80      0.83      0.82       438

    accuracy                           0.81       858
   macro avg       0.81      0.81      0.81       858
weighted avg       0.81      0.81      0.81       858
```

In [78]:
```
fpr, tpr, threshold = metrics.roc_curve(y_test, y_pred_base)
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()
```



Grid Search to tune Logistic Regression Model

In [53]:
```
from sklearn.linear_model import LogisticRegression
lr = LogisticRegression()

# Create the parameter grid based on the results of random search
param_grid = {
    'penalty': ['l1', 'l2', 'elasticnet', 'none'],
    'solver': ['newton-cg', 'lbfgs', 'liblinear', 'sag', 'saga'],
    'max_iter': [500, 1000, 1500, 2000]
}

# Instantiate the grid search model
grid_search = GridSearchCV(estimator = lr, param_grid = param_grid,
                           cv = 4, verbose = 2, return_train_score=True)
```

In [54]:
```
# Fit the grid search to the data
grid_search.fit(X_train, y_train)
```

```
Fitting 4 folds for each of 80 candidates, totalling 320 fits
[CV] max_iter=500, penalty=l1, solver=newton-cg ......................
[CV] ....... max_iter=500, penalty=l1, solver=newton-cg, total=   0.1s
[CV] max_iter=500, penalty=l1, solver=newton-cg ......................

[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent worke
rs.
[Parallel(n_jobs=1)]: Done   1 out of   1 | elapsed:    0.1s remaining:    0.
0s
```

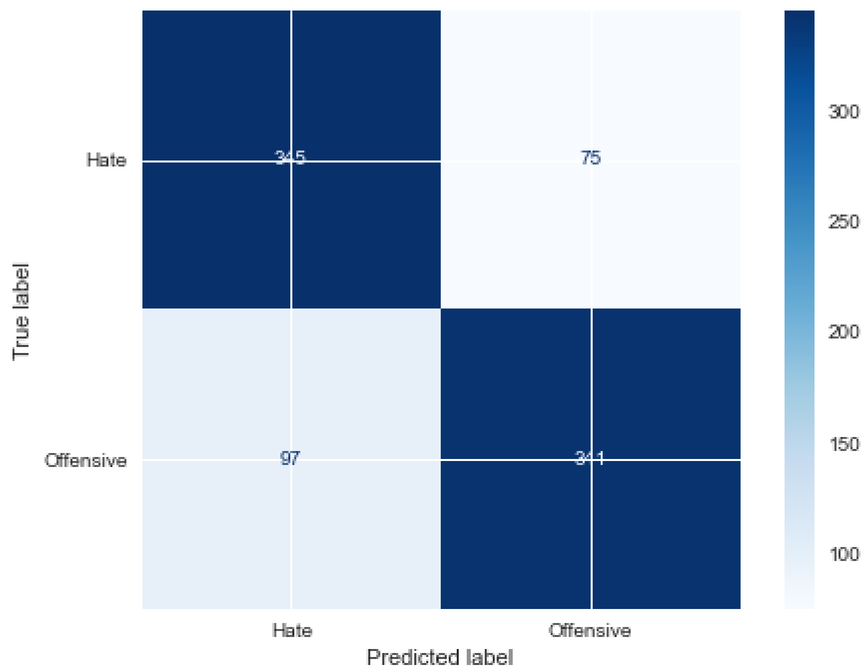
In [55]:
```
grid_search.best_params_
```

Out[55]:
```
{'max_iter': 500, 'penalty': 'l1', 'solver': 'liblinear'}
```

In [56]:
```
best_lr = grid_search.best_estimator_
y_pred_best_lr = best_lr.fit(X_train, y_train).predict(X_test)
```

In [57]:
```
plot_confusion_matrix(best_lr, X_test, y_test, cmap=plt.cm.Blues, display_labels
```

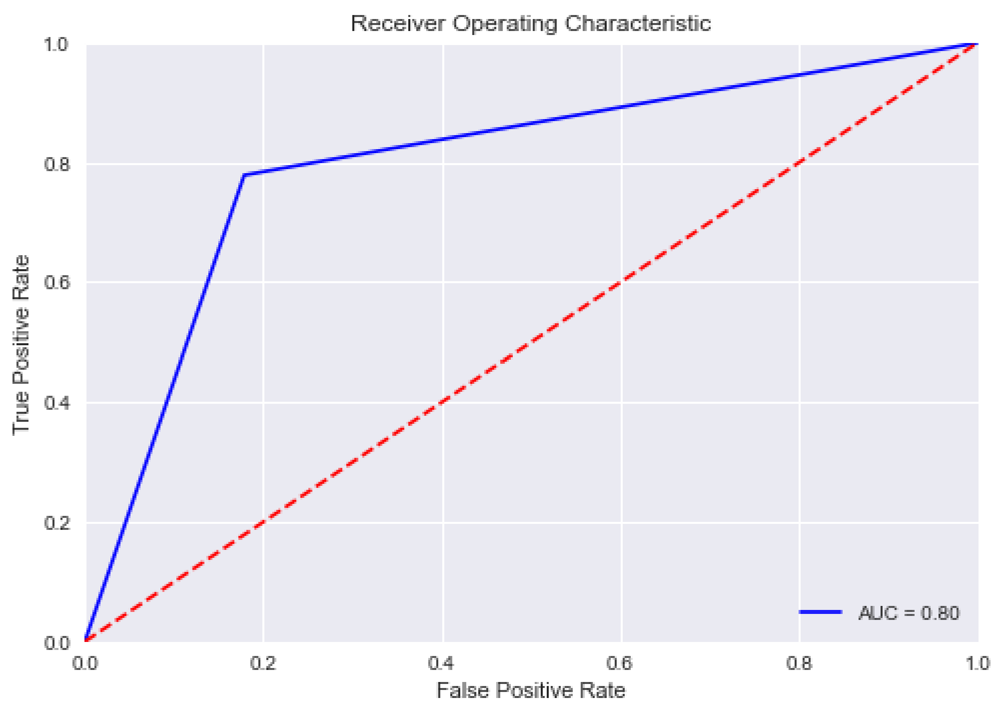
Out[57]:
```
<sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay at 0x19eb1f00908
>
```



In [58]:
```
report = classification_report(y_test, y_pred_best_lr)
print(report)
```

```
              precision    recall  f1-score   support

           0       0.78      0.82      0.80       420
           1       0.82      0.78      0.80       438

    accuracy                           0.80       858
   macro avg       0.80      0.80      0.80       858
weighted avg       0.80      0.80      0.80       858
```

In [79]:
```
import sklearn.metrics as metrics

fpr, tpr, threshold = metrics.roc_curve(y_test, y_pred_best_lr)
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()
```



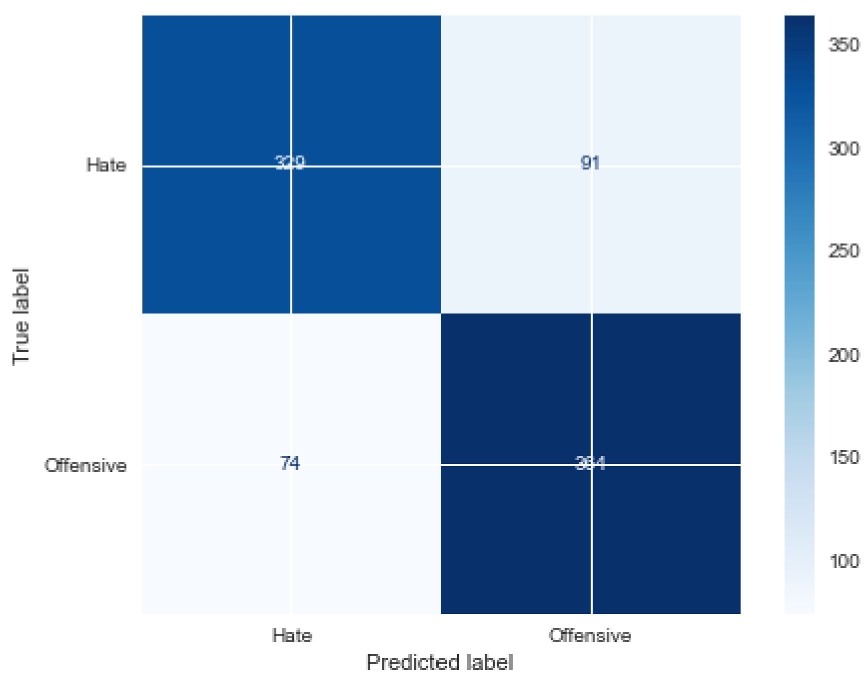
Base Logistic Regression Model

In [47]:
```
lr_base.get_params()
```

Out[47]:
```
{'C': 1.0,
 'class_weight': None,
 'dual': False,
 'fit_intercept': True,
 'intercept_scaling': 1,
 'l1_ratio': None,
 'max_iter': 1000,
 'multi_class': 'auto',
 'n_jobs': None,
 'penalty': 'l2',
 'random_state': None,
 'solver': 'lbfgs',
 'tol': 0.0001,
 'verbose': 0,
 'warm_start': False}
```

In [48]:
```
y_pred_base_lr = lr_base.fit(X_train, y_train).predict(X_test)
plot_confusion_matrix(rf_base, X_test, y_test, cmap=plt.cm.Blues, display_labels
```

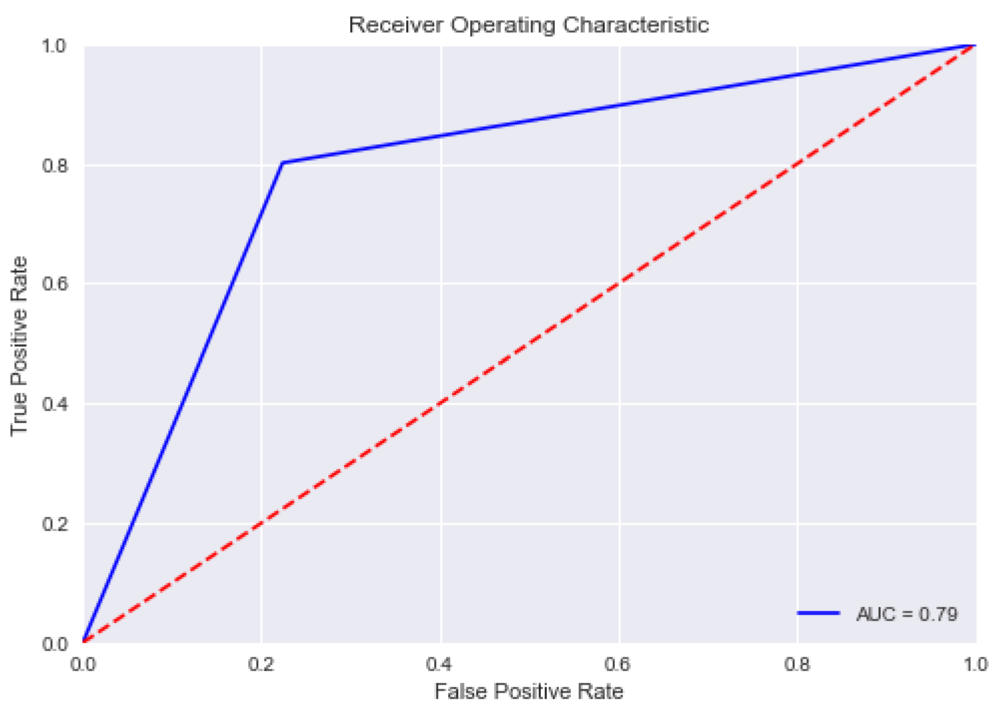
Out[48]:
```
<sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay at 0x19eb253d0f0
>
```



In [49]:
```
report = classification_report(y_test, y_pred_base_lr)
print(report)
```

```
              precision    recall  f1-score   support

           0       0.79      0.78      0.78       420
           1       0.79      0.80      0.80       438

    accuracy                           0.79       858
   macro avg       0.79      0.79      0.79       858
weighted avg       0.79      0.79      0.79       858
```

In [80]:
```
fpr, tpr, threshold = metrics.roc_curve(y_test, y_pred_base_lr)
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()
```



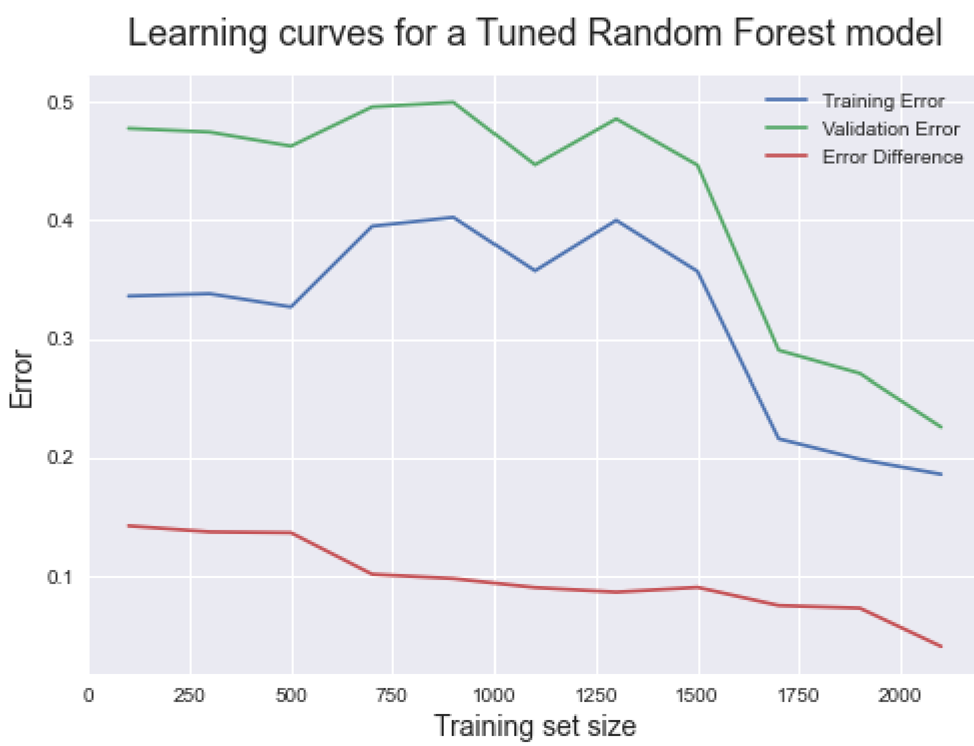
Learning Curves for all models

In [29]:
```
from sklearn.model_selection import learning_curve
train_sizes = np.arange(100, 2300, 200).tolist()

cv = 5
train_sizes, train_scores, validation_scores = learning_curve(
estimator = best_rf,
X = features,
y = labels, train_sizes = train_sizes, cv = cv,
scoring = 'accuracy')
```

In [30]:
```
train_scores_mean = train_scores.mean(axis = 1)
validation_scores_mean = validation_scores.mean(axis = 1)
```

In [31]:
```
plt.style.use('seaborn')
plt.plot(train_sizes, (1-train_scores_mean), label = 'Training Error')
plt.plot(train_sizes, (1-validation_scores_mean), label = 'Validation Error')
plt.plot(train_sizes, train_scores_mean - validation_scores_mean, label = 'Error
plt.ylabel('Error', fontsize = 14)
plt.xlabel('Training set size', fontsize = 14)
plt.title('Learning curves for a Tuned Random Forest model', fontsize = 18, y = 1
plt.legend()
```

Out[31]: `<matplotlib.legend.Legend at 0x19eb1e7c860>`

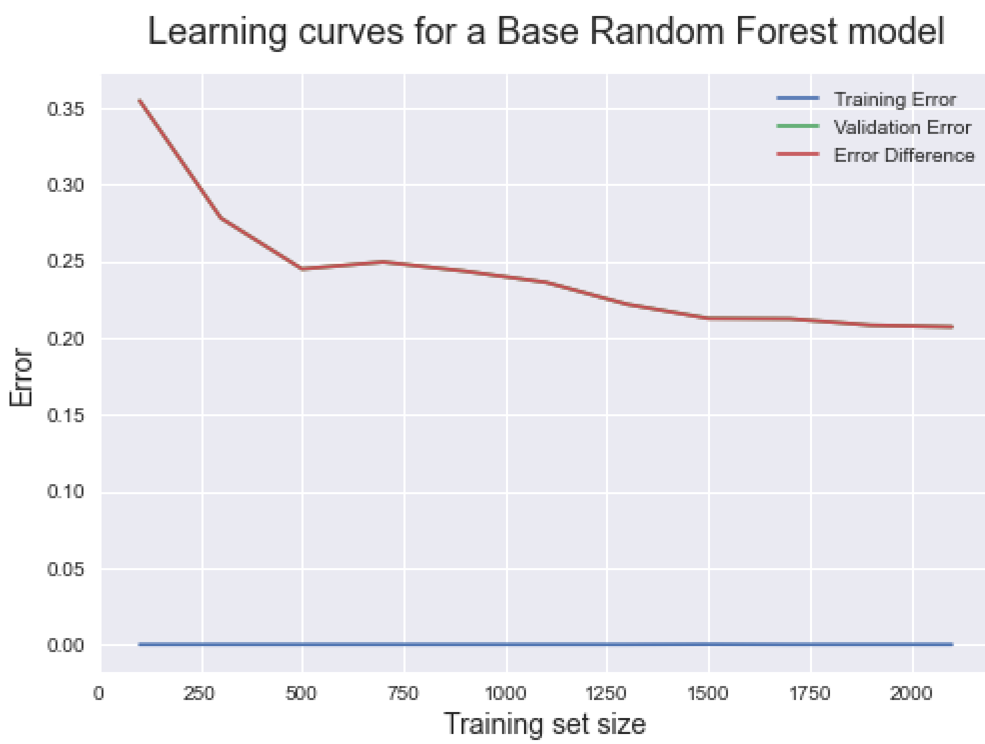


In [32]:
```
train_sizes, train_scores, validation_scores = learning_curve(
estimator = rf_base,
X = features,
y = labels, train_sizes = train_sizes, cv = cv,
scoring = 'accuracy')
```

In [33]:
```
train_scores_mean = train_scores.mean(axis = 1)
validation_scores_mean = validation_scores.mean(axis = 1)
```

In [34]:
```
plt.style.use('seaborn')
plt.plot(train_sizes, (1-train_scores_mean), label = 'Training Error')
plt.plot(train_sizes, (1-validation_scores_mean), label = 'Validation Error')
plt.plot(train_sizes, train_scores_mean - validation_scores_mean, label = 'Error
plt.ylabel('Error', fontsize = 14)
plt.xlabel('Training set size', fontsize = 14)
plt.title('Learning curves for a Base Random Forest model', fontsize = 18, y = 1.
plt.legend()
```

Out[34]: <matplotlib.legend.Legend at 0x19eb2353a20>



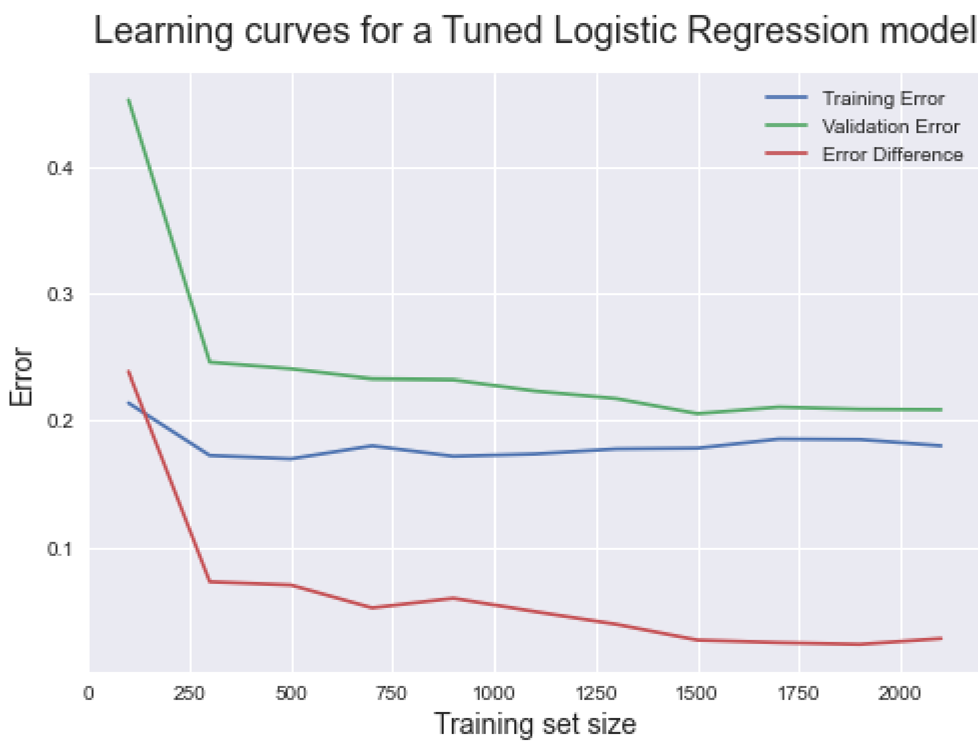
The base model seems to be overfitting the data

In [61]:
```
train_sizes, train_scores, validation_scores = learning_curve(
estimator = best_lr,
X = features,
y = labels, train_sizes = train_sizes, cv = cv,
scoring = 'accuracy')
```

In [62]:
```
train_scores_mean = train_scores.mean(axis = 1)
validation_scores_mean = validation_scores.mean(axis = 1)
```

In [63]:
```
plt.style.use('seaborn')
plt.plot(train_sizes, (1-train_scores_mean), label = 'Training Error')
plt.plot(train_sizes, (1-validation_scores_mean), label = 'Validation Error')
plt.plot(train_sizes, train_scores_mean - validation_scores_mean, label = 'Error
plt.ylabel('Error', fontsize = 14)
plt.xlabel('Training set size', fontsize = 14)
plt.title('Learning curves for a Tuned Logistic Regression model', fontsize = 18,
plt.legend()
```

Out[63]: <matplotlib.legend.Legend at 0x19ee5be4f98>

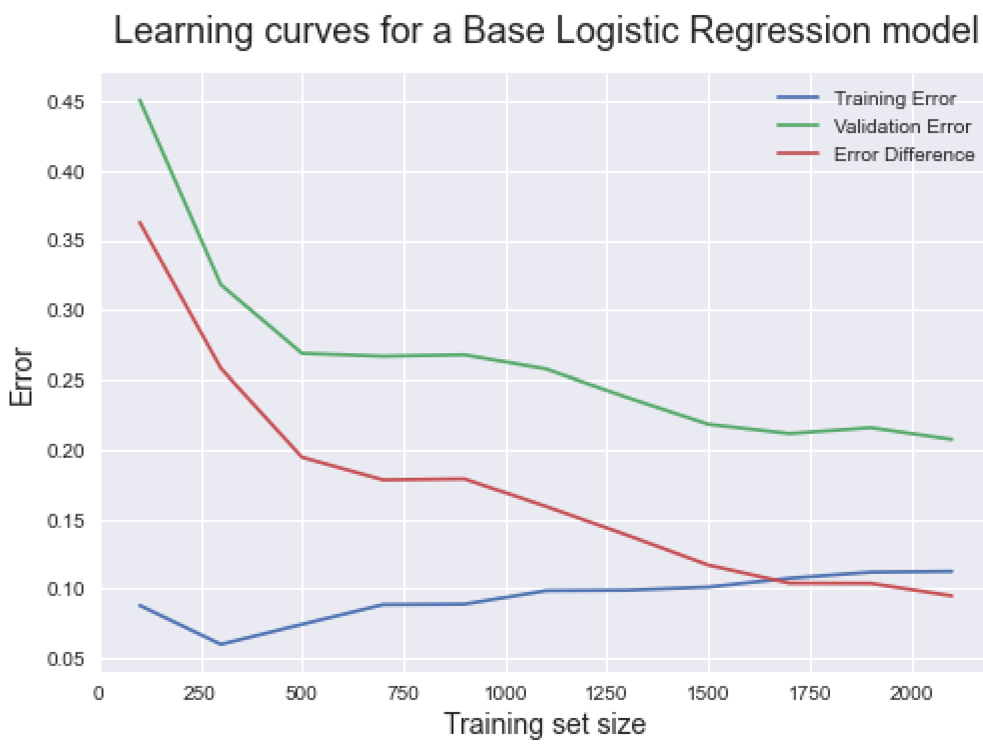


In [64]:
```
train_sizes, train_scores, validation_scores = learning_curve(
estimator = lr_base,
X = features,
y = labels, train_sizes = train_sizes, cv = cv,
scoring = 'accuracy')
```

In [65]:
```
train_scores_mean = train_scores.mean(axis = 1)
validation_scores_mean = validation_scores.mean(axis = 1)
```

In [66]:
```
plt.style.use('seaborn')
plt.plot(train_sizes, (1-train_scores_mean), label = 'Training Error')
plt.plot(train_sizes, (1-validation_scores_mean), label = 'Validation Error')
plt.plot(train_sizes, train_scores_mean - validation_scores_mean, label = 'Error
plt.ylabel('Error', fontsize = 14)
plt.xlabel('Training set size', fontsize = 14)
plt.title('Learning curves for a Base Logistic Regression model', fontsize = 18,
plt.legend()
```

Out[66]: <matplotlib.legend.Legend at 0x19eb25631d0>



Ensembles

Bagging

In [69]:
```
from sklearn.model_selection import cross_val_score
from sklearn.ensemble import BaggingClassifier

models = [rf_base, best_rf, lr_base, best_lr]
model_names = ['Base Random Forest', 'Tuned Random Forest', 'Base Logistic Regres
i = 0
for model in models:
    vanilla_scores = cross_val_score(model, features, labels, cv=5)
    bagging_clf = BaggingClassifier(model,
       max_samples=0.4, random_state=123)
    bagging_scores = cross_val_score(bagging_clf, features, labels, cv=5)
    print(model_names[i])
    i = i + 1
    print("Mean of: {1:.3f}, std: (+/-) {2:.3f} [{0}]".format(model.__class__.__n
                                                              vanilla_scores.mean
    print("Mean of: {1:.3f}, std: (+/-) {2:.3f} [Bagging {0}]\n".format(model.__c
                                                                        bagging_s
```

```
Base Random Forest
Mean of: 0.793, std: (+/-) 0.027 [RandomForestClassifier]
Mean of: 0.792, std: (+/-) 0.033 [Bagging RandomForestClassifier]

Tuned Random Forest
Mean of: 0.790, std: (+/-) 0.030 [RandomForestClassifier]
Mean of: 0.784, std: (+/-) 0.029 [Bagging RandomForestClassifier]

Base Logistic Regression
Mean of: 0.795, std: (+/-) 0.019 [LogisticRegression]
Mean of: 0.776, std: (+/-) 0.029 [Bagging LogisticRegression]

Tuned Logistic Regression
Mean of: 0.791, std: (+/-) 0.017 [LogisticRegression]
Mean of: 0.793, std: (+/-) 0.017 [Bagging LogisticRegression]
```

In [72]:
```
# Get some classifiers to evaluate
from sklearn.ensemble import ExtraTreesClassifier, VotingClassifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn.linear_model import RidgeClassifier
from sklearn.svm import SVC

seed = 123
np.random.seed(seed)

# Create classifiers
et = ExtraTreesClassifier()
knn = KNeighborsClassifier()
svc = SVC()
rg = RidgeClassifier()
# Set up voting
eclf = VotingClassifier(estimators=[('Random Forests', best_rf), ('Extra Trees',
                                    ('KNeighbors', knn), ('SVC', svc), ('Ridge Cl

for clf, label in zip([best_rf, et, knn, svc, rg, best_lr, eclf], ['Random Forest
                                                          'KNeighbors', 'SVC', 'Ridge
    scores = cross_val_score(clf, features, labels, cv=4, scoring='accuracy')
    print("Mean: {0:.3f}, std: (+/-) {1:.3f} [{2}]".format(scores.mean(), scores.
```

```
Mean: 0.786, std: (+/-) 0.031 [Random Forest]
Mean: 0.781, std: (+/-) 0.030 [Extra Trees]
Mean: 0.593, std: (+/-) 0.043 [KNeighbors]
Mean: 0.544, std: (+/-) 0.048 [SVC]
Mean: 0.764, std: (+/-) 0.019 [Ridge Classifier]
Mean: 0.785, std: (+/-) 0.023 [Logistic Regression]
Mean: 0.785, std: (+/-) 0.025 [Ensemble]
```

In [75]:
```
# Set up ensemble voting for bagging
ebclf_array = []
clf_array = [best_rf, et, knn, svc, rg, best_lr]

for clf in clf_array:
    ebclf_array.append(BaggingClassifier(clf, max_samples=0.5, random_state=seed)

v_eclf = VotingClassifier(estimators=zip(['Bagging Random Forest', 'Bagging Extra
                                          'Bagging SVC', 'Bagging Ridge Classifie
                                         ebclf_array),
                          voting='hard')

ebclf_array.append(v_eclf)

for clf, label in zip(ebclf_array, ['Bagging Random Forest', 'Bagging Extra Trees
                              'Bagging SVC', 'BaggingRidge Classifier', 'Bagging
    scores = cross_val_score(clf, features, labels, cv=5, scoring='accuracy')
    print("Mean: {0:.3f}, std: (+/-) {1:.3f} [{2}]".format(scores.mean(), scores.
```

```
Mean: 0.789, std: (+/-) 0.025 [Bagging Random Forest]
Mean: 0.785, std: (+/-) 0.028 [Bagging Extra Trees]
Mean: 0.613, std: (+/-) 0.035 [Bagging KNeighbors]
Mean: 0.545, std: (+/-) 0.044 [Bagging SVC]
Mean: 0.777, std: (+/-) 0.024 [BaggingRidge Classifier]
Mean: 0.789, std: (+/-) 0.017 [Bagging Logistic Regression]
Mean: nan, std: (+/-) nan [Bagging Ensemble]
```

Boosting

In [76]:
```
from mlxtend.classifier import EnsembleVoteClassifier
import warnings
from xgboost import XGBClassifier, plot_importance
from sklearn.ensemble import AdaBoostClassifier, GradientBoostingClassifier

warnings.filterwarnings('ignore')

# Create boosting classifiers
ada_boost = AdaBoostClassifier()
grad_boost = GradientBoostingClassifier()
xgb_boost = XGBClassifier()

boost_array = [ada_boost, grad_boost, xgb_boost]

eclf = EnsembleVoteClassifier(clfs=[ada_boost, grad_boost, xgb_boost], voting='ha

labels = ['Ada Boost', 'Grad Boost', 'XG Boost', 'Ensemble']

for clf, label in zip([ada_boost, grad_boost, xgb_boost, eclf], labels):
    scores = cross_val_score(clf, X_train, y_train, cv=5, scoring='accuracy')
    print("Mean: {0:.3f}, std: (+/-) {1:.3f} [{2}]".format(scores.mean(), scores.
```

```
Mean: 0.785, std: (+/-) 0.023 [Ada Boost]
Mean: 0.802, std: (+/-) 0.010 [Grad Boost]
Mean: 0.801, std: (+/-) 0.019 [XG Boost]
Mean: 0.803, std: (+/-) 0.019 [Ensemble]
```

In [ ]: