In [1]:
```
import matplotlib.pyplot as plt
import time
import numpy as np
import pandas as pd
import seaborn as sns
from numpy import ravel
from imblearn.over_sampling import SMOTE
from sklearn.model_selection import train_test_split
from sklearn.svm import SVR
from sklearn.decomposition import PCA
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import roc_auc_score
from sklearn.preprocessing import StandardScaler
from sklearn import datasets, model_selection, tree, preprocessing, metrics, l
inear_model
from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import RandomForestClassifier
from sklearn.manifold import TSNE
from sklearn.model_selection import RandomizedSearchCV
from sklearn.linear_model import SGDClassifier
from sklearn.naive_bayes import GaussianNB
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import GradientBoostingClassifier
from sklearn.svm import LinearSVC
from sklearn.ensemble import BaggingClassifier
from sklearn.ensemble import AdaBoostClassifier
from sklearn.model_selection import GridSearchCV
from sklearn import svm
from sklearn.ensemble import VotingClassifier
from scipy.stats import randint as sp_randint
import datetime
import xgboost as xgb
from xgboost import XGBClassifier
from sklearn.metrics import classification_report
from sklearn.metrics import confusion_matrix
import itertools


%matplotlib inline
```

In [2]:
```
def warn(*args, **kwargs):
    pass
import warnings
warnings.warn = warn
```

In [3]:
```
pd.set_option('display.max_columns', None)
pd.set_option('display.max_rows', None)
```

In [4]:
```
#read data
data = pd.read_csv('C:/application/interview_prep/bank-additional/bank-additio
nal/bank_full_processed.csv')
# visualize the data
data.head()
```

Out[4]:

	age	default	housing	loan	campaign	previous	poutcome	emp.var.rate	cons.price.idx
0	0.481481	1	1	1	0.0	0.0	0	0.9375	0.698753
1	0.493827	0	1	1	0.0	0.0	0	0.9375	0.698753
2	0.246914	1	-1	1	0.0	0.0	0	0.9375	0.698753
3	0.283951	1	1	1	0.0	0.0	0	0.9375	0.698753
4	0.481481	1	1	-1	0.0	0.0	0	0.9375	0.698753

Model Building and Evaluation

- **Train Test Split** : Divide the Data set into Train class and Test class for model building and Evaluation
- Used **Stratification split** since the data is imbalanced. A random split might probably have changed the target distribution

In [5]:
```
X = data.drop('y', axis=1)
y = data['y']
```

SMOTE for oversampling the dataset

- Since the dataset is highly imbalanced containing approximate 88% of NO values in term deposite , training our model on such dataset will create a bias for undersample class(12% of YES values)
- To deal with this situation using SMOTE (Synthetic Minority Oversampling TEchnique) is a good option
- SMOTE first selects a minority class instance a at random and finds its k nearest minority class neighbors. The synthetic instance is then created by choosing one of the k nearest neighbors b at random and connecting a and b to form a line segment in the feature space. The synthetic instances are generated as a convex combination of the two chosen instances a and b.
- The combination of SMOTE and under-sampling performs better than plain under-sampling

In [6]:
```
print("Dataset shape before SMOTE:-")
print("X:", X.shape)
print("y:",y.shape)
```

```
Dataset shape before SMOTE:-
X: (41188, 51)
y: (41188,)
```

In [7]:
```
sm = SMOTE(random_state=2)
X, y = sm.fit_resample(X, y.values.ravel())
```

In [8]:
```
print("Dataset shape after SMOTE:-")
print("X:", X.shape)
print("y:",y.shape)
```

```
Dataset shape after SMOTE:-
X: (73096, 51)
y: (73096,)
```

- Split performed without Stratification

In [9]:
```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, rando
m_state=1)
print('Original:', (data.y).mean(), 'Train:', (y_train).mean(), 'Test:', (y_te
st).mean())
```

```
Original: 0.11265417111780131 Train: 0.5010089609412408 Test: 0.4959644322845
4173
```

- Split performed with Stratification

In [10]:
```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, strat
ify=y, random_state=1)
print('Original:', (data.y).mean(), 'Train:', (y_train).mean(), 'Test:', (y_te
st).mean())
```

```
Original: 0.11265417111780131 Train: 0.5 Test: 0.5
```

Performing Different Algorithms to find the best fit for our dataset

- Will try following different algorithms to see which algorithm fits best on our dataset in terms of accuracy, ROC, AUC
- Performing Random Hyperparameter search and GridSearch for Hyperparameters, to select the best hyperparameters for a given algorithm

In [11]:
```
# Function that runs the requested algorithm and returns the accuracy metrics
def fit_ml_algo(algo, X_train, y_train, X_test, cv):
    # One Pass
    model = algo.fit(X_train, y_train)
    test_pred = model.predict(X_test)
    if (isinstance(algo, (LogisticRegression,
                          KNeighborsClassifier,
                          GaussianNB,
                          DecisionTreeClassifier,
                          RandomForestClassifier,
                          GradientBoostingClassifier,
                          BaggingClassifier,
                          AdaBoostClassifier,
                          XGBClassifier,


                          ))):
        probs = model.predict_proba(X_test)[:,1]
    else:
        probs = "Not Available"
    acc = round(model.score(X_test, y_test) * 100, 2)
    # CV
    train_pred = model_selection.cross_val_predict(algo,
                                                   X_train,
                                                   y_train,
                                                   cv=cv,
                                                   n_jobs = -1)
    acc_cv = round(metrics.accuracy_score(y_train, train_pred) * 100, 2)
    try:
        feature_importances = np.mean([tree.feature_importances_ for tree in a
lgo.estimators_], axis=0)
    except:
        feature_importances = 'none'
    return train_pred, test_pred, acc, acc_cv, probs, feature_importances
```

In [12]:
```
# calculate the fpr and tpr for all thresholds of the classification
def plot_roc_curve(y_test, preds):
    fpr, tpr, threshold = metrics.roc_curve(y_test, preds)
    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([-0.01, 1.01])
    plt.ylim([-0.01, 1.01])
    plt.ylabel('True Positive Rate')
    plt.xlabel('False Positive Rate')
    plt.show()
```

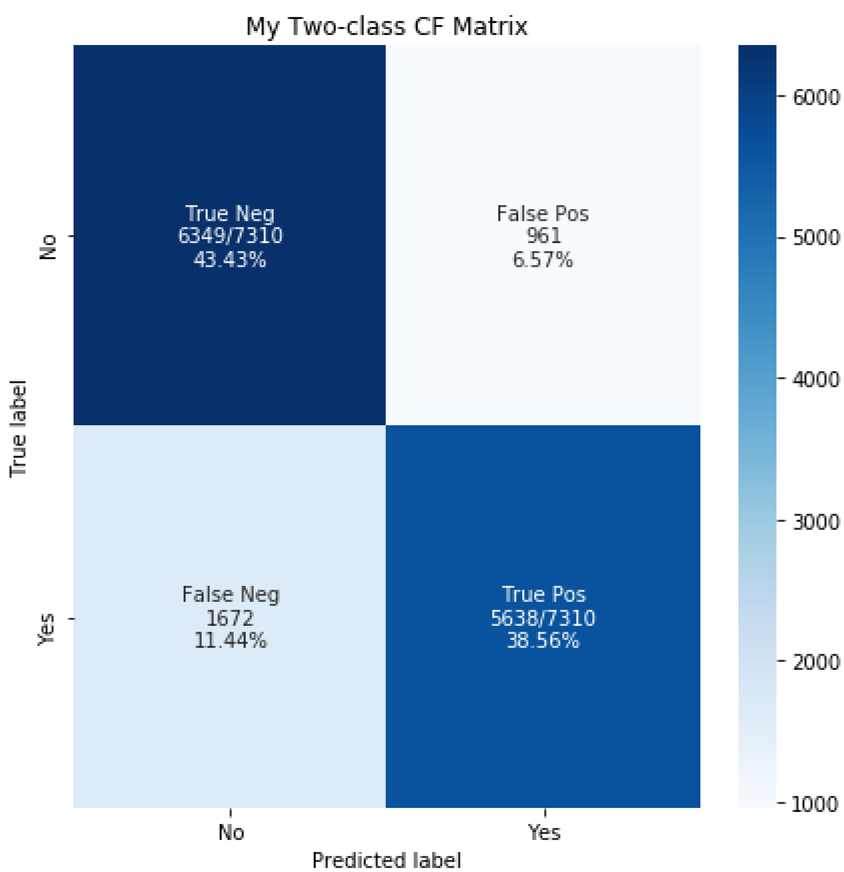
In [13]:
```
def make_confusion_matrix(y,y_pred,categories='auto',count=True,percent=True,c
bar=True,xyticks=True,
                          figsize=(7,7),cmap='Blues',title=None):
    cf = confusion_matrix(y, y_pred)
    blanks = ['' for i in range(cf.size)]
    group_names = ['True Neg','False Pos','False Neg','True Pos']
    categories = ['No', 'Yes']

    if group_names and len(group_names)==cf.size:
        group_labels = ["{}\n".format(value) for value in group_names]
    else:
        group_labels = blanks

    if count:
        cm_sum = np.sum(cf, axis=1, keepdims=True)
        cm_perc = cf / cm_sum.astype(float) * 100
        annot = np.empty_like(cf).astype(str)
        nrows, ncols = cf.shape
        for i in range(nrows):
            for j in range(ncols):
                c = cf[i, j]
            #p = cm_perc[i, j]
                if i == j:
                    s = cm_sum[i]
                    annot[i, j] = '%d/%d\n' % (c, s)
                elif c == 0:
                    annot[i, j] = ''
                else:
                    annot[i, j] = '%d\n' % (c)
        group_counts = list(annot.flat)

    else:
        group_counts = blanks
    if percent:
        group_percentages = ["{0:.2%}".format(value) for value in cf.flatten()
/np.sum(cf)]
    else:
        group_percentages = blanks

    box_labels = [f"{v1}{v2}{v3}".strip() for v1, v2, v3 in zip(group_labels,g
roup_counts,group_percentages)]
    box_labels = np.asarray(box_labels).reshape(cf.shape[0],cf.shape[1])
     # SET FIGURE PARAMETERS ACCORDING TO OTHER ARGUMENTS
    if figsize==None:
        #Get default figure size if not set
        figsize = plt.rcParams.get('figure.figsize')

    if xyticks==False:
        #Do not show categories if xyticks is False
        categories=False


    # MAKE THE HEATMAP VISUALIZATION
    plt.figure(figsize=figsize)
    sns.heatmap(cf,annot=box_labels,fmt="",cmap=cmap,cbar=cbar,xticklabels=cat
egories,yticklabels=categories)
```

```
    plt.ylabel('True label')
    plt.xlabel('Predicted label')
    if title:
        plt.title(title)
```

Logistic Regression

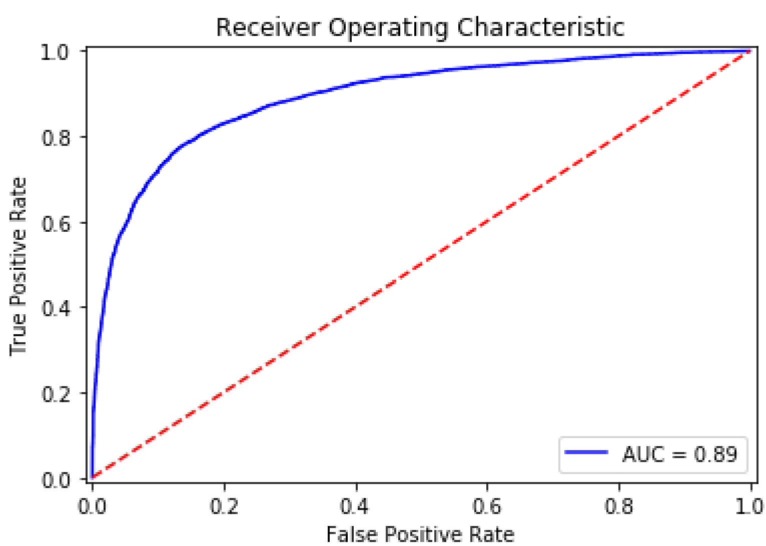
In [14]:
```
start_time = time.time()
train_pred_log, test_pred_log, acc_log, acc_cv_log, probs_log,feature_log = fi
t_ml_algo(LogisticRegression(n_jobs = -1),
                                                                 X_train,
                                                                 y_train,
                                                                 X_test,
                                                                 10)
log_time = (time.time() - start_time)
print('\033[1m' + "Accuracy before CV: %s" % acc_log)
print("Accuracy CV 10-Fold: %s" % acc_cv_log)
print("Running Time for the Algorithm to train and pred: %s" % datetime.timede
lta(seconds=log_time))
make_confusion_matrix(y_test,test_pred_log, title='My Two-class CF Matrix')
```

```
Accuracy before CV: 81.99
Accuracy CV 10-Fold: 81.87
Running Time for the Algorithm to train and pred: 0:00:25.611744
```



In [15]:
```
print("Classification Report on Training :-")
print (metrics.classification_report(y_train, train_pred_log))
print("Classification Report on Testing :-")
print (metrics.classification_report(y_test, test_pred_log))
print("ROC and AUC curve :-")
plot_roc_curve(y_test, probs_log)
```

```
Classification Report on Training :-
              precision    recall  f1-score   support

           0       0.79      0.86      0.83     29238
           1       0.85      0.77      0.81     29238

    accuracy                           0.82     58476
   macro avg       0.82      0.82      0.82     58476
weighted avg       0.82      0.82      0.82     58476

Classification Report on Testing :-
              precision    recall  f1-score   support

           0       0.79      0.87      0.83      7310
           1       0.85      0.77      0.81      7310

    accuracy                           0.82     14620
   macro avg       0.82      0.82      0.82     14620
weighted avg       0.82      0.82      0.82     14620

ROC and AUC curve :-
```



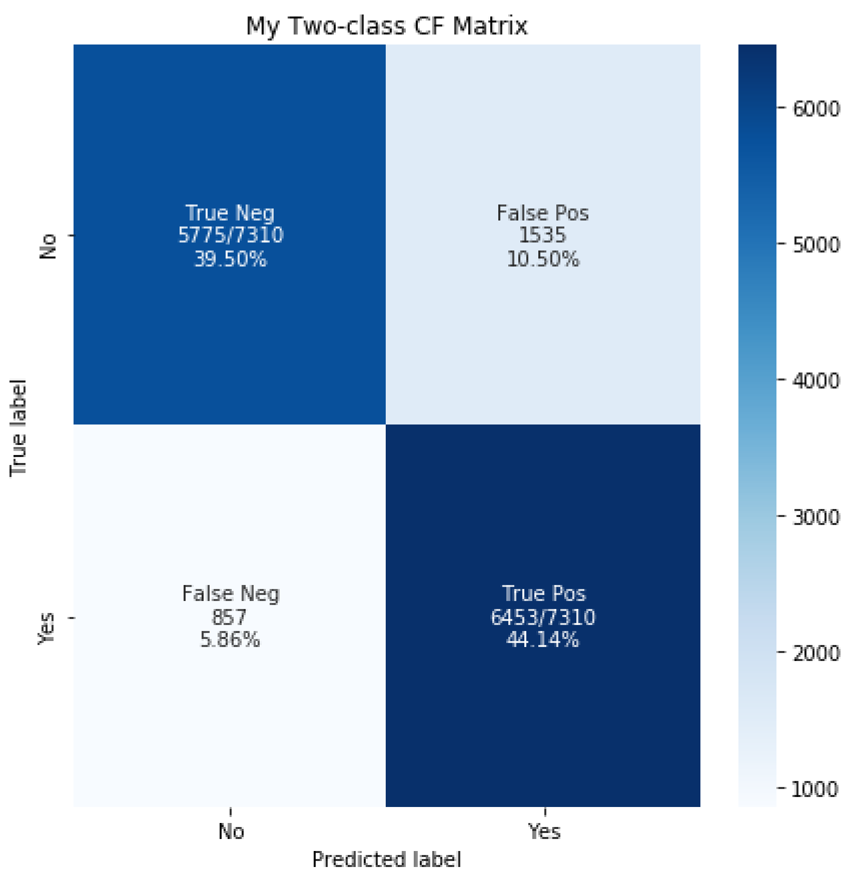
K-Nearest Neighbors

In [16]:
```
alpha = [x for x in range(1, 70, 7)]
cv_auc_array=[]
for i in alpha:
    k_cfl=KNeighborsClassifier(n_neighbors=i)
    k_cfl.fit(X_train,y_train)
    predict_y = k_cfl.predict_proba(X_test)
    cv_auc_array.append(roc_auc_score(y_test, predict_y[:,1]))
for i in range(len(cv_auc_array)):
    print ('AUC for k = ',alpha[i],'is',cv_auc_array[i])
best_alpha = np.argmax(cv_auc_array)
```

```
AUC for k =  1 is 0.8775649794801642
AUC for k =  8 is 0.9122914190968279
AUC for k =  15 is 0.8999171066002196
AUC for k =  22 is 0.8902493913290829
AUC for k =  29 is 0.8832063249376357
AUC for k =  36 is 0.8776175282253009
AUC for k =  43 is 0.8711960360131072
AUC for k =  50 is 0.8662328182633089
AUC for k =  57 is 0.8619666760860167
AUC for k =  64 is 0.8583707175486235
```

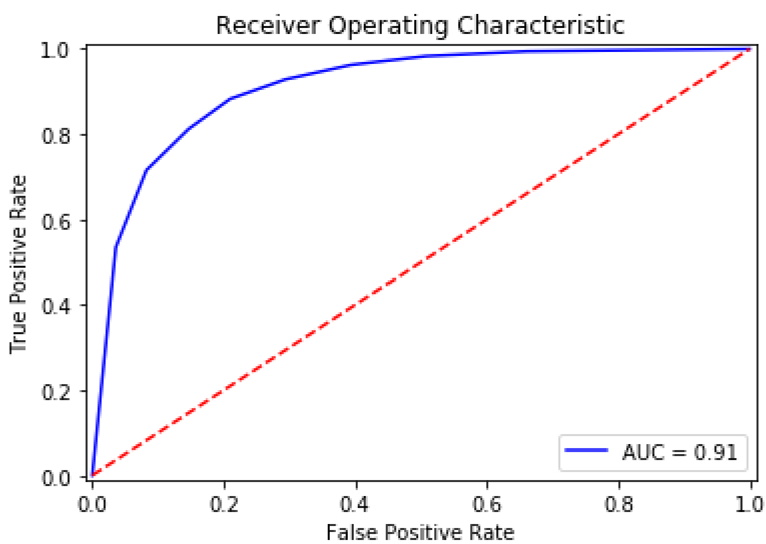
In [17]:
```
# k-Nearest Neighbors
start_time = time.time()
train_pred_knn, test_pred_knn, acc_knn, acc_cv_knn, probs_knn,feature_knn = fi
t_ml_algo(KNeighborsClassifier(n_neighbors=alpha[best_alpha],

n_jobs = -1),

X_train,

y_train,

X_test,

10)
knn_time = (time.time() - start_time)
print('\033[1m' + "Accuracy before CV: %s" % acc_knn)
print("Accuracy CV 10-Fold: %s" % acc_cv_knn)
print("Running Time for the Algorithm to train and pred: %s" % datetime.timede
lta(seconds=knn_time))
make_confusion_matrix(y_test,test_pred_knn, title='My Two-class CF Matrix')
```

Accuracy before CV: 83.64
Accuracy CV 10-Fold: 83.11
Running Time for the Algorithm to train and pred: 0:02:40.221750



In [18]:
```
print("Classification Report on Training :-")
print (metrics.classification_report(y_train, train_pred_knn))
print("Classification Report on Testing :-")
print (metrics.classification_report(y_test, test_pred_knn))
print("ROC and AUC curve :-")
plot_roc_curve(y_test, probs_knn)
```

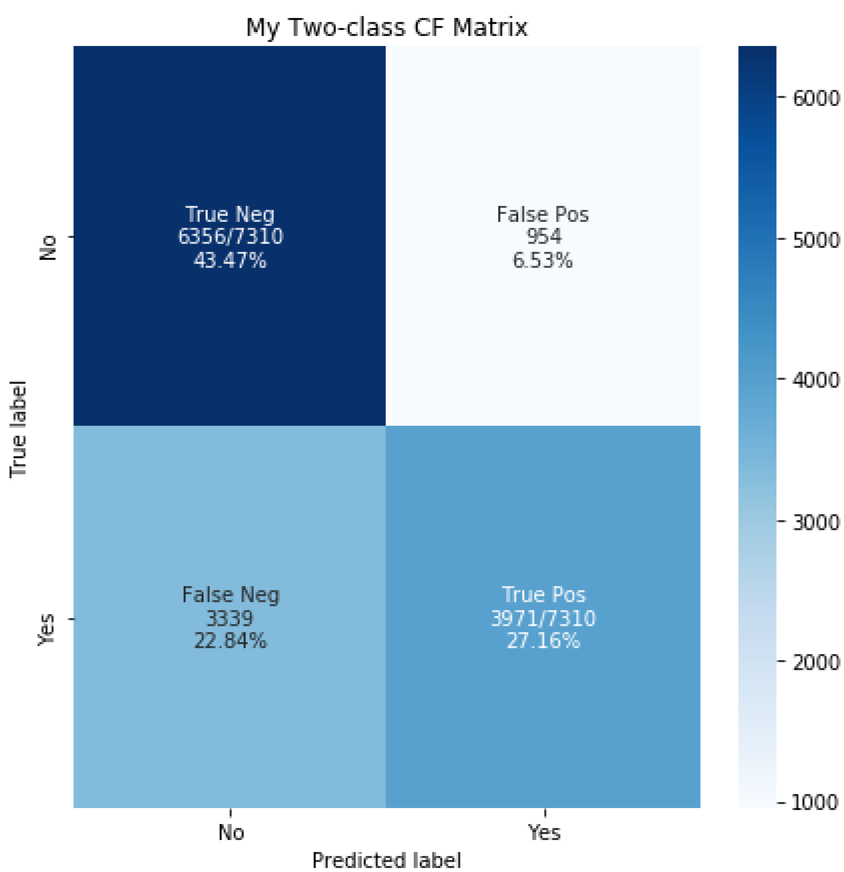
```
Classification Report on Training :-
              precision    recall  f1-score   support

           0       0.87      0.78      0.82     29238
           1       0.80      0.88      0.84     29238

    accuracy                           0.83     58476
   macro avg       0.83      0.83      0.83     58476
weighted avg       0.83      0.83      0.83     58476

Classification Report on Testing :-
              precision    recall  f1-score   support

           0       0.87      0.79      0.83      7310
           1       0.81      0.88      0.84      7310

    accuracy                           0.84     14620
   macro avg       0.84      0.84      0.84     14620
weighted avg       0.84      0.84      0.84     14620

ROC and AUC curve :-
```



Gaussian Naive Bayes

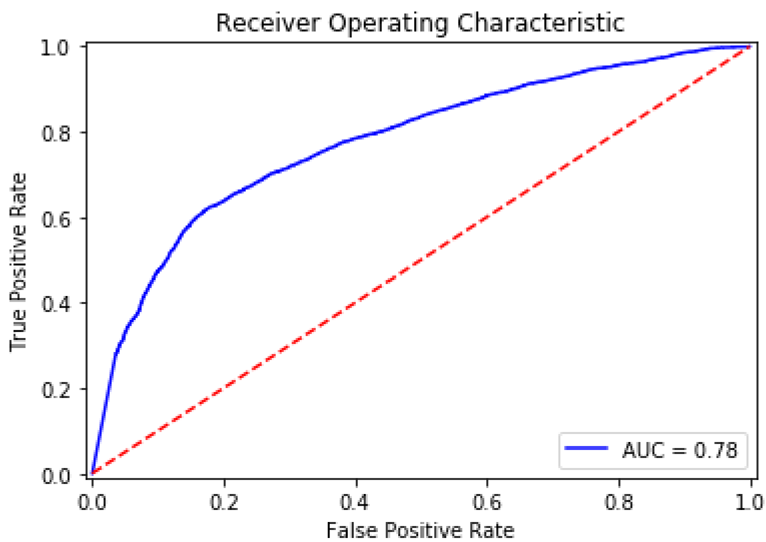
In [19]:
```
# Gaussian Naive Bayes
start_time = time.time()
train_pred_gaussian, test_pred_gaussian, acc_gaussian, acc_cv_gaussian, probs_
gau,feature_gau = fit_ml_algo(GaussianNB(),

X_train,

y_train,

X_test,

10)
gaussian_time = (time.time() - start_time)
print('\033[1m' +"Accuracy: %s" % acc_gaussian)
print("Accuracy CV 10-Fold: %s" % acc_cv_gaussian)
print("Running Time for the Algorithm to train and pred: %s" % datetime.timede
lta(seconds=gaussian_time))
make_confusion_matrix(y_test,test_pred_gaussian, title='My Two-class CF Matri
x')
```

```
Accuracy: 70.64
Accuracy CV 10-Fold: 71.31
Running Time for the Algorithm to train and pred: 0:00:01.335958
```



In [20]:
```
print("Classification Report on Training :-")
print (metrics.classification_report(y_train, train_pred_gaussian))
print("Classification Report on Testing :-")
print (metrics.classification_report(y_test, test_pred_gaussian))
print("ROC and AUC curve :-")
plot_roc_curve(y_test, probs_gau)
```

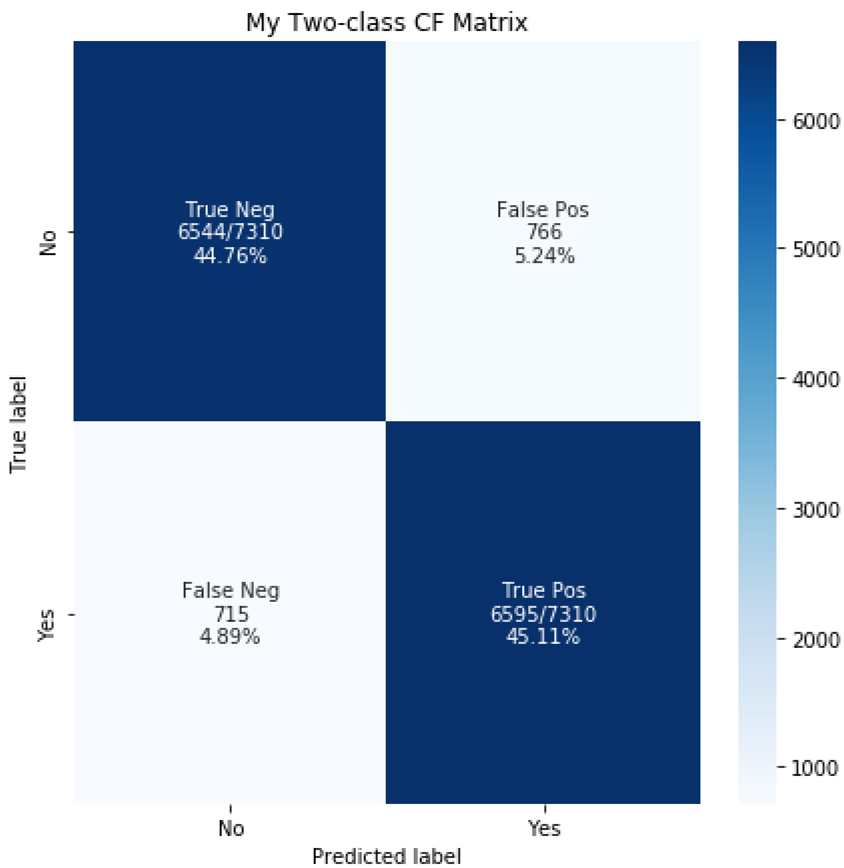
```
Classification Report on Training :-
              precision    recall  f1-score   support

           0       0.66      0.88      0.75     29238
           1       0.82      0.55      0.66     29238

    accuracy                           0.71     58476
   macro avg       0.74      0.71      0.71     58476
weighted avg       0.74      0.71      0.71     58476

Classification Report on Testing :-
              precision    recall  f1-score   support

           0       0.66      0.87      0.75      7310
           1       0.81      0.54      0.65      7310

    accuracy                           0.71     14620
   macro avg       0.73      0.71      0.70     14620
weighted avg       0.73      0.71      0.70     14620

ROC and AUC curve :-
```



Decision Tree Classifier

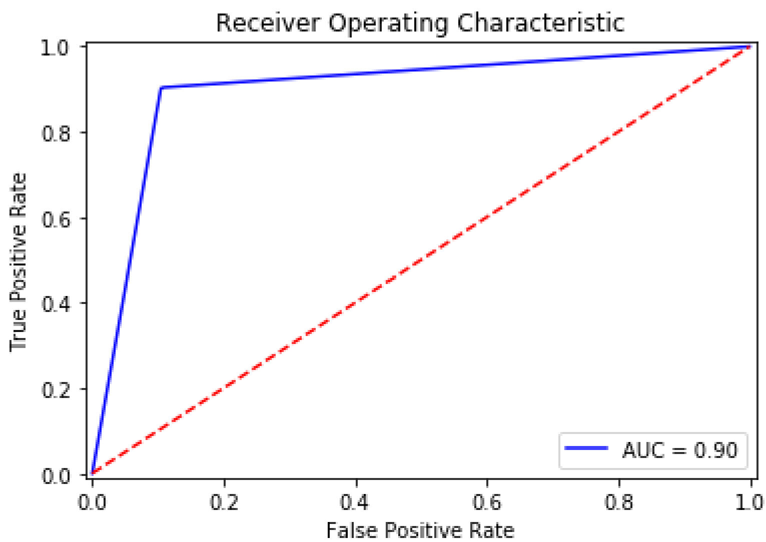
In [21]:
```
# Decision Tree Classifier
start_time = time.time()
train_pred_dt, test_pred_dt, acc_dt, acc_cv_dt, probs_dt,feature_dt = fit_ml_a
lgo(DecisionTreeClassifier(),
                                                              X_train,
                                                              y_train,
                                                              X_test,
                                                              10)
dt_time = (time.time() - start_time)
print('\033[1m' + "Accuracy: %s" % acc_dt)
print("Accuracy CV 10-Fold: %s" % acc_cv_dt)
print("Running Time for the Algorithm to train and pred: %s" % datetime.timede
lta(seconds=dt_time))
make_confusion_matrix(y_test,test_pred_dt, title='My Two-class CF Matrix')
```

```
Accuracy: 89.87
Accuracy CV 10-Fold: 89.59
Running Time for the Algorithm to train and pred: 0:00:04.503174
```



In [22]:
```python
print("Classification Report on Training :-")
print (metrics.classification_report(y_train, train_pred_dt))
print("Classification Report on Testing :-")
print (metrics.classification_report(y_test, test_pred_dt))
print("ROC and AUC curve :-")
plot_roc_curve(y_test, probs_dt)
```

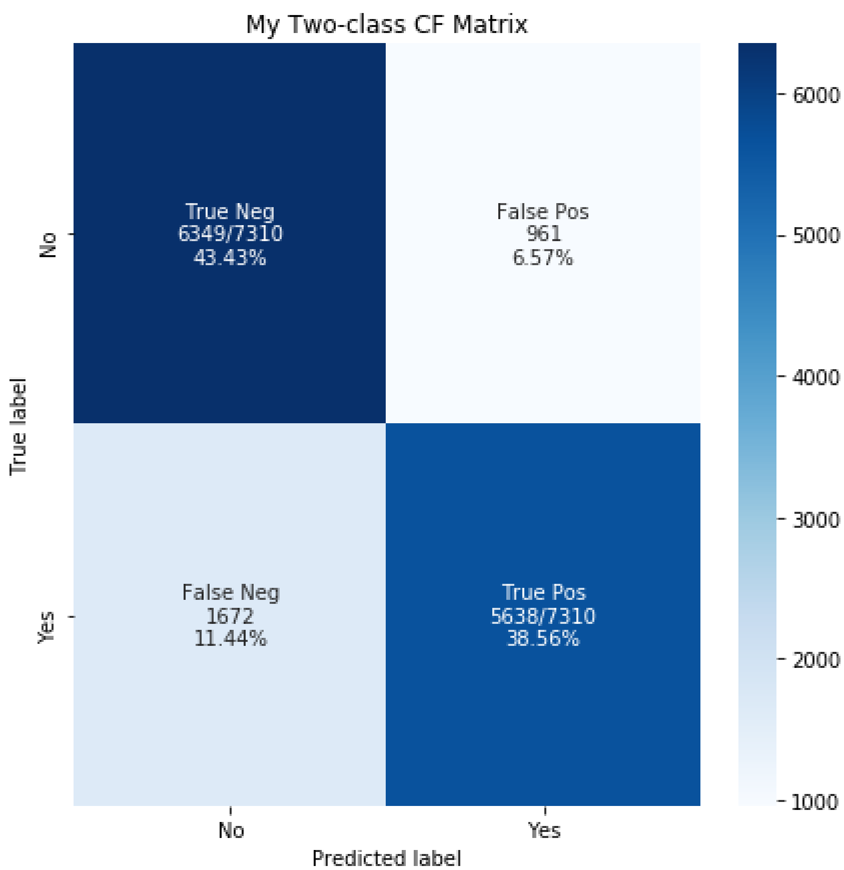
```
Classification Report on Training :-
              precision    recall  f1-score   support

           0       0.90      0.89      0.90     29238
           1       0.89      0.90      0.90     29238

    accuracy                           0.90     58476
   macro avg       0.90      0.90      0.90     58476
weighted avg       0.90      0.90      0.90     58476

Classification Report on Testing :-
              precision    recall  f1-score   support

           0       0.90      0.90      0.90      7310
           1       0.90      0.90      0.90      7310

    accuracy                           0.90     14620
   macro avg       0.90      0.90      0.90     14620
weighted avg       0.90      0.90      0.90     14620

ROC and AUC curve :-
```



Random Forest Classifier

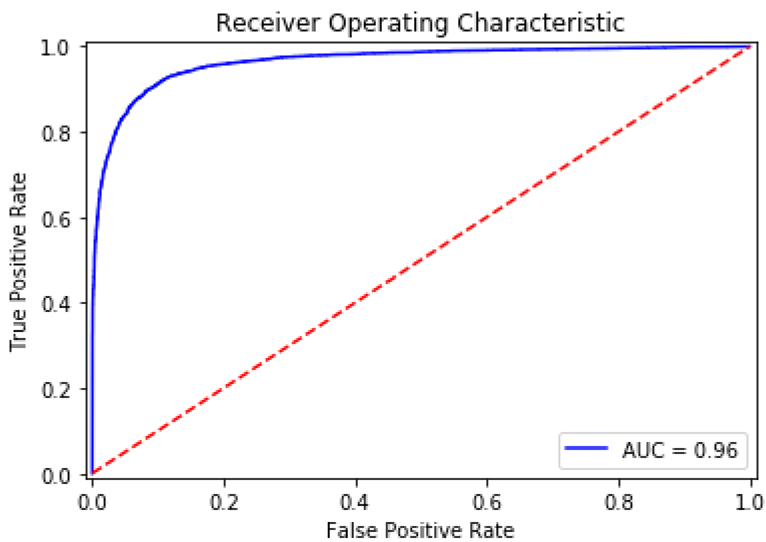
In [23]:
```
start_time = time.time()
rfc = RandomForestClassifier(n_estimators=10,
                             min_samples_leaf=2,
                             min_samples_split=17,
                             criterion='gini',
                             max_features=8)
train_pred_rf, test_pred_rf, acc_rf, acc_cv_rf, probs_rf,feature_rf = fit_ml_a
lgo(rfc,
                                                                X_train,
                                                                y_train,
                                                                X_test,
                                                                10)
rf_time = (time.time() - start_time)
print('\033[1m' + "Accuracy: %s" % acc_rf)
print("Accuracy CV 10-Fold: %s" % acc_cv_rf)
print("Running Time for the Algorithm to train and pred: %s" % datetime.timede
lta(seconds=rf_time))
make_confusion_matrix(y_test,test_pred_log, title='My Two-class CF Matrix')
```

Accuracy: 90.44
Accuracy CV 10-Fold: 90.34
Running Time for the Algorithm to train and pred: 0:00:06.729031



In [24]:
```
print("Classification Report on Training :-")
print (metrics.classification_report(y_train, train_pred_rf))
print("Classification Report on Testing :-")
print (metrics.classification_report(y_test, test_pred_rf))
print("ROC and AUC curve :-")
plot_roc_curve(y_test, probs_rf)
```

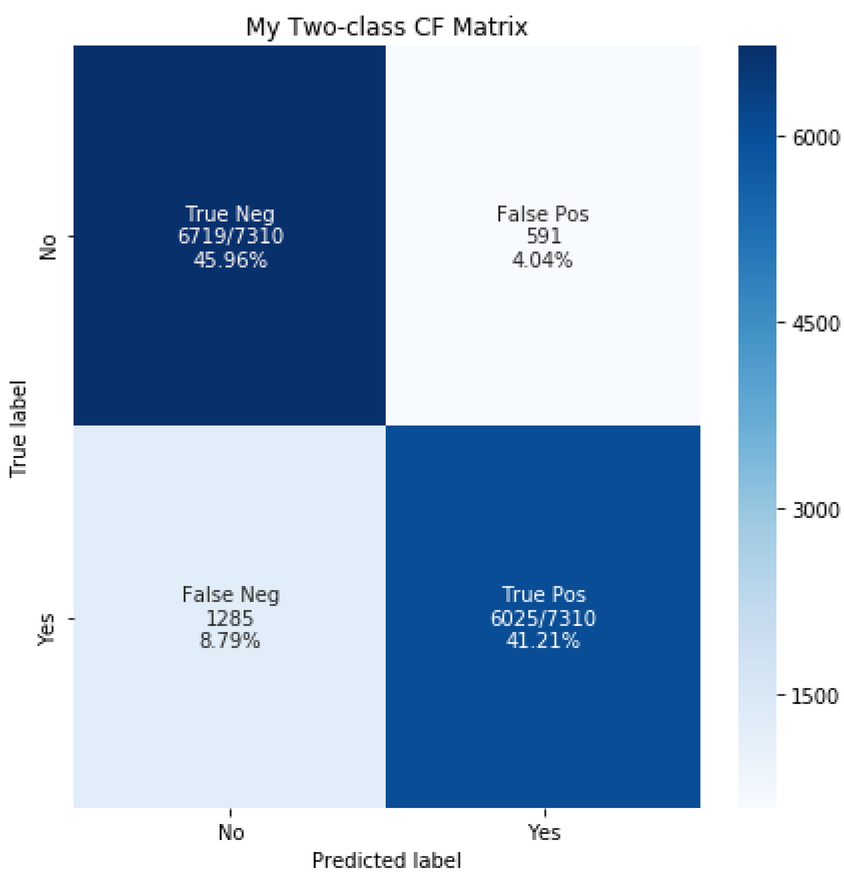
```
Classification Report on Training :-
              precision    recall  f1-score   support

           0       0.89      0.93      0.91     29238
           1       0.92      0.88      0.90     29238

    accuracy                           0.90     58476
   macro avg       0.90      0.90      0.90     58476
weighted avg       0.90      0.90      0.90     58476

Classification Report on Testing :-
              precision    recall  f1-score   support

           0       0.89      0.93      0.91      7310
           1       0.92      0.88      0.90      7310

    accuracy                           0.90     14620
   macro avg       0.91      0.90      0.90     14620
weighted avg       0.91      0.90      0.90     14620

ROC and AUC curve :-
```



Gradient Boosting Trees

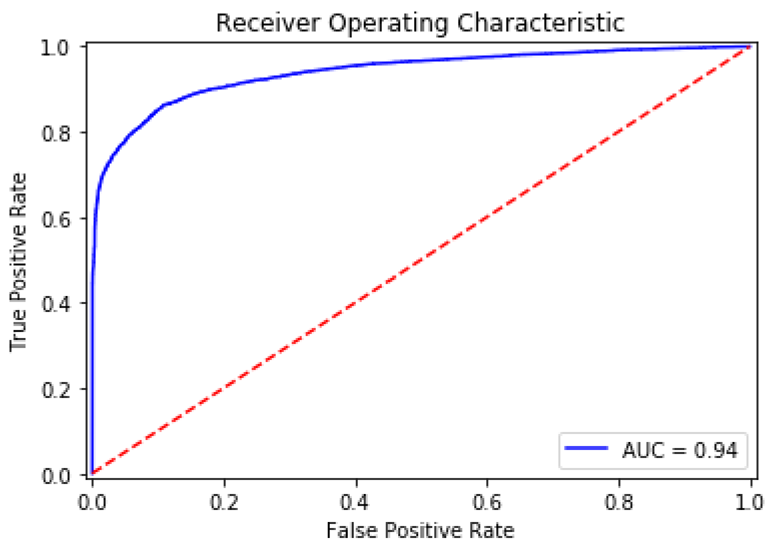
In [25]:
```
start_time = time.time()
train_pred_gbt, test_pred_gbt, acc_gbt, acc_cv_gbt, probs_gbt,feature_gbt = fi
t_ml_algo(GradientBoostingClassifier(),
                                                                 X_train,
                                                                 y_train,
                                                                 X_test,
                                                                 10)
gbt_time = (time.time() - start_time)
print('\033[1m' +"Accuracy: %s" % acc_gbt)
print("Accuracy CV 10-Fold: %s" % acc_cv_gbt)
print("Running Time for the Algorithm to train and pred: %s" % datetime.timede
lta(seconds=gbt_time))
make_confusion_matrix(y_test,test_pred_gbt, title='My Two-class CF Matrix')
```

Accuracy: 87.17
Accuracy CV 10-Fold: 87.28
Running Time for the Algorithm to train and pred: 0:01:27.451092



In [26]:
```
print("Classification Report on Training :-")
print (metrics.classification_report(y_train, train_pred_gbt))
print("Classification Report on Testing :-")
print (metrics.classification_report(y_test, test_pred_gbt))
print("ROC and AUC curve :-")
plot_roc_curve(y_test, probs_gbt)
```

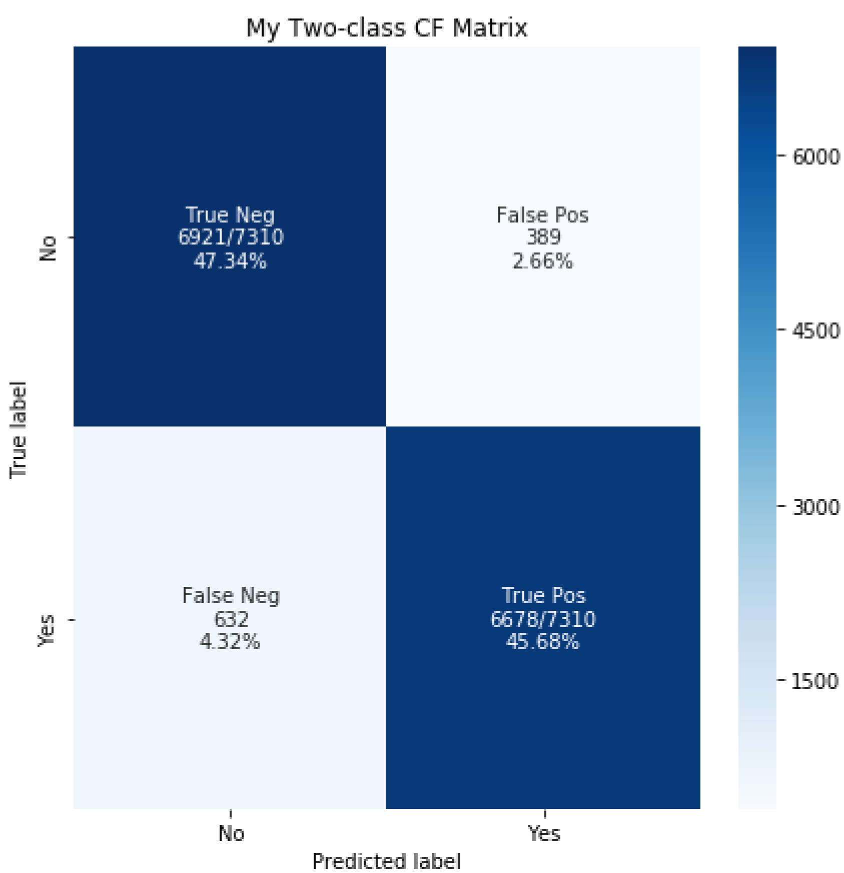
```
Classification Report on Training :-
              precision    recall  f1-score   support

           0       0.84      0.92      0.88     29238
           1       0.91      0.83      0.87     29238

    accuracy                           0.87     58476
   macro avg       0.88      0.87      0.87     58476
weighted avg       0.88      0.87      0.87     58476

Classification Report on Testing :-
              precision    recall  f1-score   support

           0       0.84      0.92      0.88      7310
           1       0.91      0.82      0.87      7310

    accuracy                           0.87     14620
   macro avg       0.88      0.87      0.87     14620
weighted avg       0.88      0.87      0.87     14620

ROC and AUC curve :-
```



BaggingClassifier

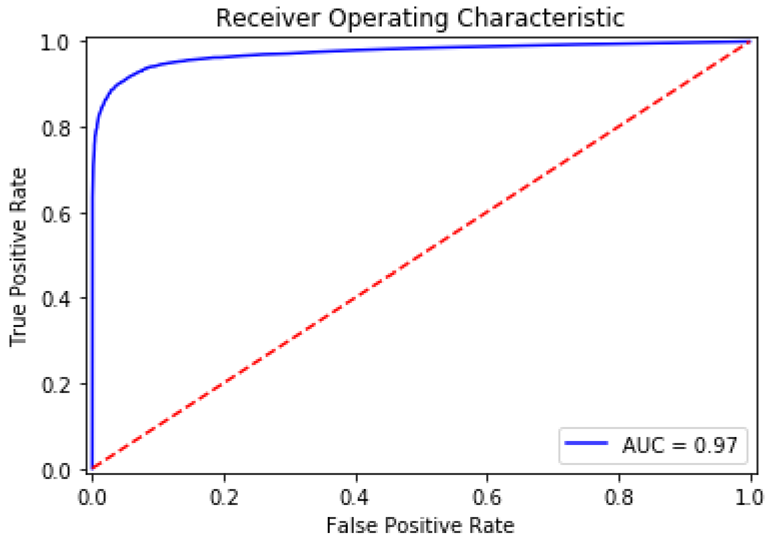
In [27]:
```
start_time = time.time()
dt_model = DecisionTreeClassifier(criterion = 'entropy',random_state=100)
train_pred_bc, test_pred_bc, acc_bc, acc_cv_bc, probs_bc,feature_bc = fit_ml_a
lgo(BaggingClassifier(base_estimator=dt_model, n_estimators=100,random_state=1
00),
                                                                 X_train,
                                                                 y_train,
                                                                 X_test,
                                                                 10)
bc_time = (time.time() - start_time)
print('\033[1m' + "Accuracy: %s" % acc_bc)
print("Accuracy CV 10-Fold: %s" % acc_cv_bc)
print("Running Time for the Algorithm to train and pred: %s" % datetime.timede
lta(seconds=bc_time))
make_confusion_matrix(y_test,test_pred_bc, title='My Two-class CF Matrix')
```

Accuracy: 93.02
Accuracy CV 10-Fold: 92.88
Running Time for the Algorithm to train and pred: 0:05:04.220410



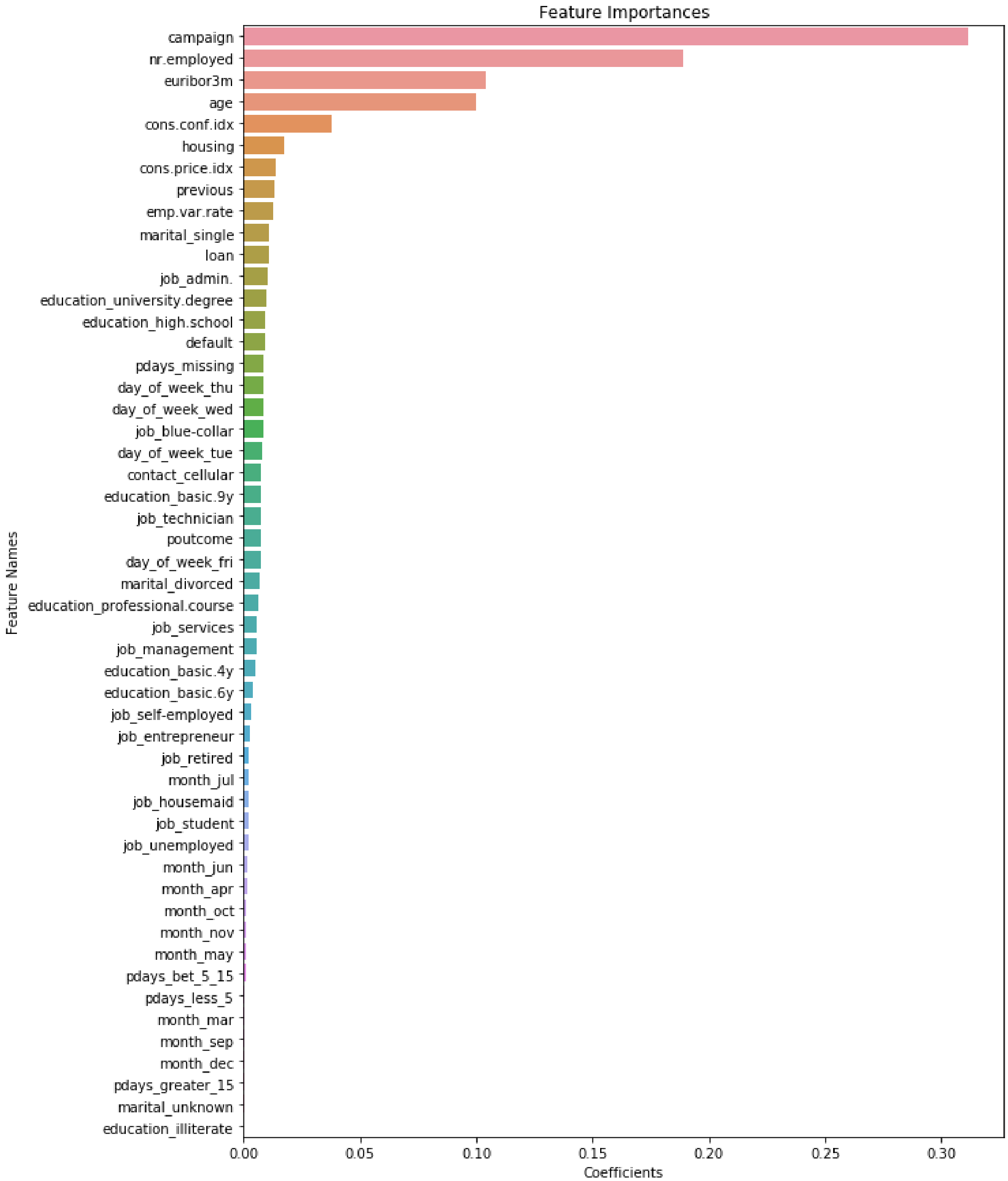
In [28]:
```
print("Classification Report on Training :-")
print (metrics.classification_report(y_train, train_pred_bc))
print("Classification Report on Testing :-")
print (metrics.classification_report(y_test, test_pred_bc))
print("ROC and AUC curve :-")
plot_roc_curve(y_test, probs_bc)
```

```
Classification Report on Training :-
              precision    recall  f1-score   support

           0       0.92      0.94      0.93     29238
           1       0.94      0.91      0.93     29238

    accuracy                           0.93     58476
   macro avg       0.93      0.93      0.93     58476
weighted avg       0.93      0.93      0.93     58476

Classification Report on Testing :-
              precision    recall  f1-score   support

           0       0.92      0.95      0.93      7310
           1       0.94      0.91      0.93      7310

    accuracy                           0.93     14620
   macro avg       0.93      0.93      0.93     14620
weighted avg       0.93      0.93      0.93     14620

ROC and AUC curve :-
```



In [29]:
```
'''Plots feature importance in a sorted order and shows the most significant v
ariables at the top'''
X1 = list(X_train.columns)
#X.remove('y_yes')
feature_importance_df = pd.DataFrame(data = feature_bc, index = X1, columns=[
'coefficient_values'])
feature_importance_df['sort'] = feature_importance_df.coefficient_values.abs()
sorted_feature_imp_df = feature_importance_df.sort_values(by='sort', ascending
=False).drop('sort', axis=1)
fig, ax = plt.subplots()
fig.set_size_inches(10, 15)
sns.barplot(np.array(sorted_feature_imp_df.coefficient_values), np.array(sorte
d_feature_imp_df.index.values))
plt.title('Feature Importances')
plt.xlabel('Coefficients')
plt.ylabel('Feature Names')
```

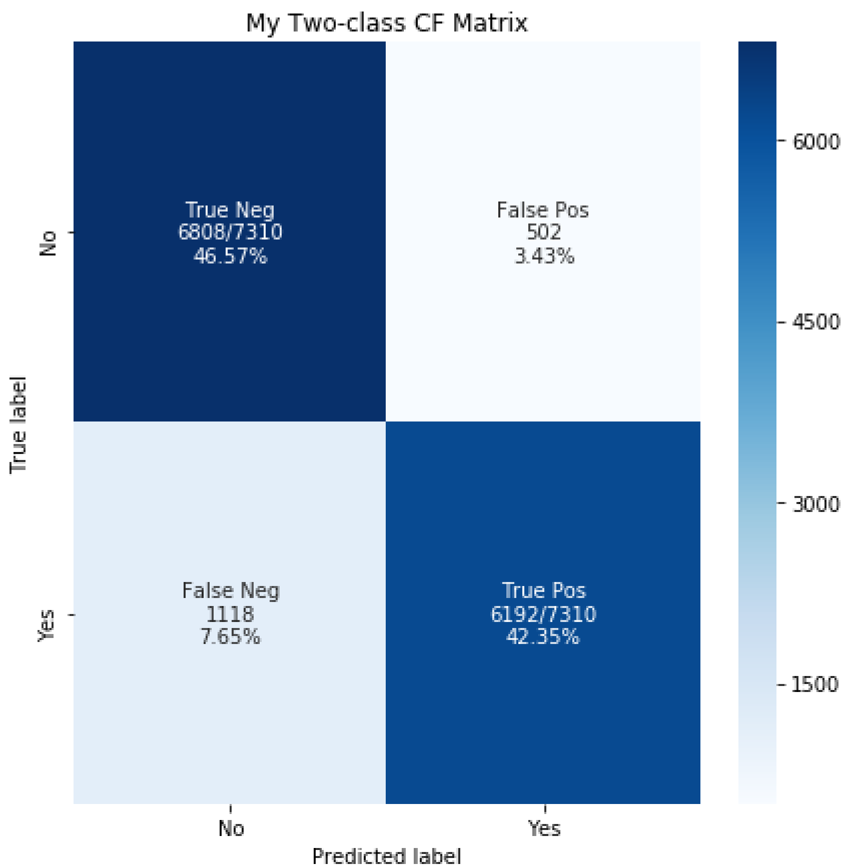
Out[29]: Text(0, 0.5, 'Feature Names')



XGBoost

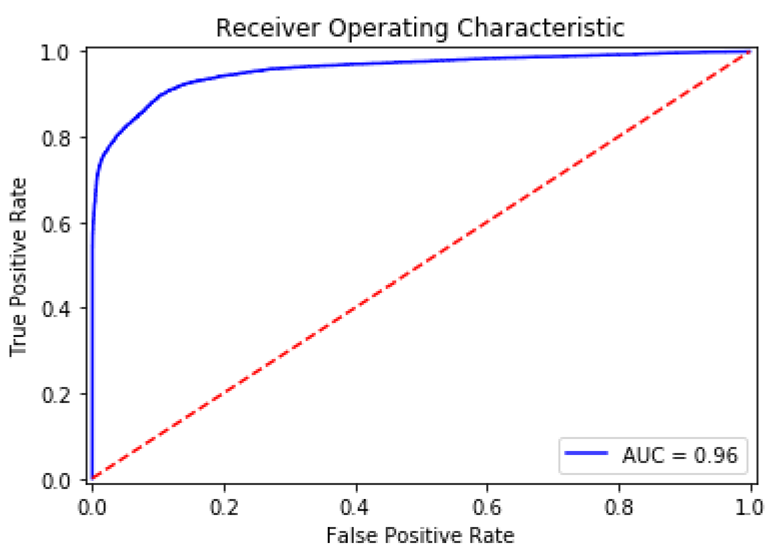
In [30]:
```
start_time = time.time()

train_pred_xgb1, test_pred_xgb1, acc_xgb1, acc_cv_xgb1, probs_xgb1, feature_xgb1 = fit_ml_algo(xgb.XGBClassifier(max_depth=5, n_estimators=100 ,class_weight='balanced', n_jobs=-1),
                                                                 X_train,
                                                                 y_train,
                                                                 X_test,
                                                                 10)
xgb_time1 = (time.time() - start_time)
print('\033[1m' + "Accuracy: %s" % acc_xgb1)
print("Accuracy CV 10-Fold: %s" % acc_cv_xgb1)
print("Running Time for the Algorithm to train and pred: %s" % datetime.timedelta(seconds=xgb_time1))
make_confusion_matrix(y_test,test_pred_xgb1, title='My Two-class CF Matrix')
```

Accuracy: 88.92
Accuracy CV 10-Fold: 89.13
Running Time for the Algorithm to train and pred: 0:01:58.062209

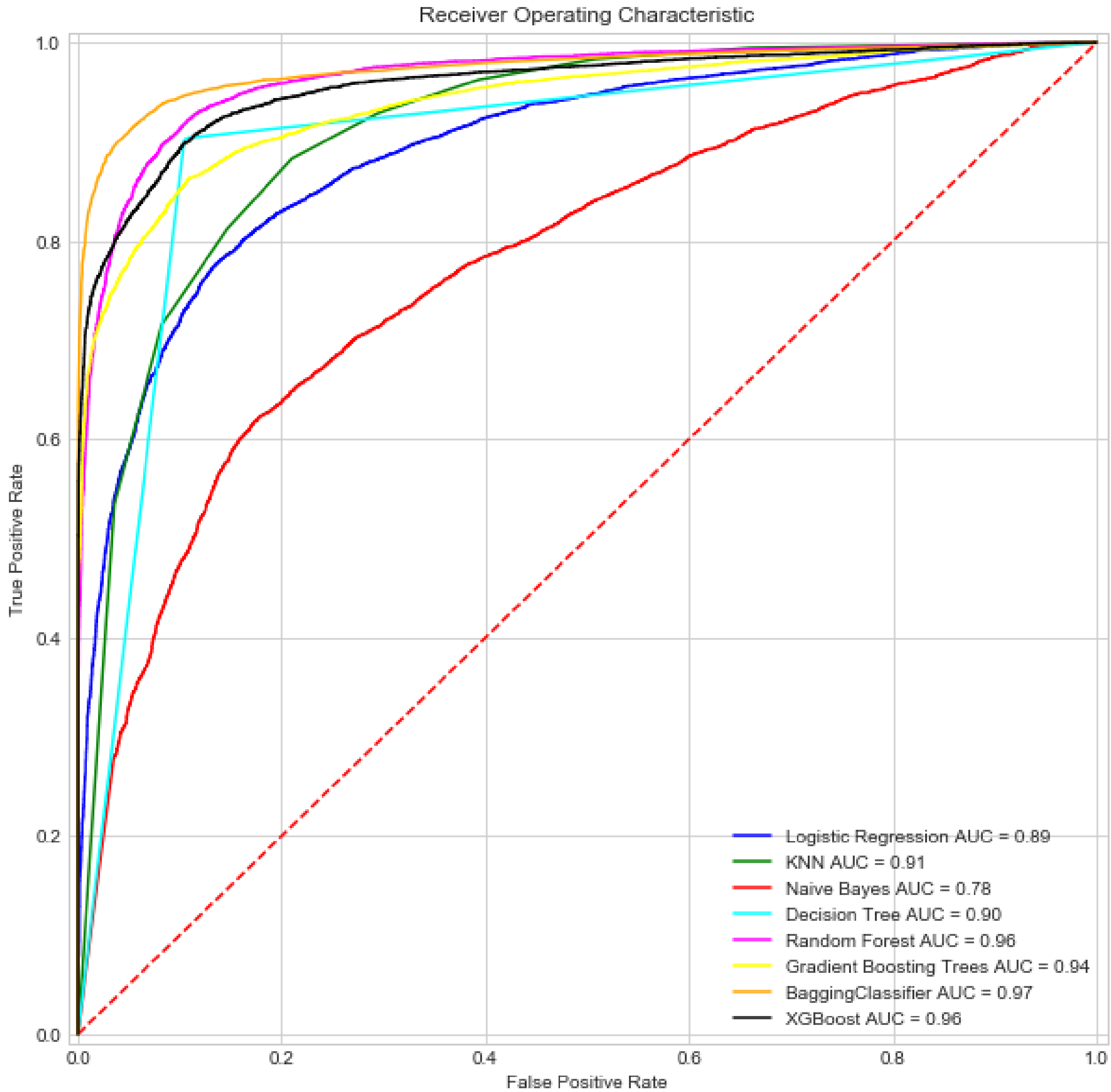


In [31]:
```
print("Classification Report on Training :-")
print (metrics.classification_report(y_train, train_pred_xgb1))
print("Classification Report on Testing :-")
print (metrics.classification_report(y_test, test_pred_xgb1))
print("ROC and AUC curve :-")
plot_roc_curve(y_test, probs_xgb1)
```

```
Classification Report on Training :-
              precision    recall  f1-score   support

           0       0.86      0.93      0.90     29238
           1       0.93      0.85      0.89     29238

    accuracy                           0.89     58476
   macro avg       0.89      0.89      0.89     58476
weighted avg       0.89      0.89      0.89     58476

Classification Report on Testing :-
              precision    recall  f1-score   support

           0       0.86      0.93      0.89      7310
           1       0.93      0.85      0.88      7310

    accuracy                           0.89     14620
   macro avg       0.89      0.89      0.89     14620
weighted avg       0.89      0.89      0.89     14620

ROC and AUC curve :-
```



In [32]:
```
plt.style.use('seaborn-whitegrid')
fig = plt.figure(figsize=(10,10))

models = [
    'Logistic Regression',
    'KNN',
    'Naive Bayes',
    'Decision Tree',
    'Random Forest',
    'Gradient Boosting Trees',
    'BaggingClassifier',
    'XGBoost'
]
probs = [
    probs_log,
    probs_knn,
    probs_gau,
    probs_dt,
    probs_rf,
    probs_gbt,
    probs_bc,
    probs_xgb1
]
colors = [
    'blue',
    'green',
    'red',
    'cyan',
    'magenta',
    'yellow',
    'orange',
    'black'
]

plt.title('Receiver Operating Characteristic')
plt.plot([0, 1], [0, 1],'r--')
plt.xlim([-0.01, 1.01])
plt.ylim([-0.01, 1.01])
plt.ylabel('True Positive Rate')
plt.xlabel('False Positive Rate')

def plot_roc_curves(y_test, prob, model):
    fpr, tpr, threshold = metrics.roc_curve(y_test, prob)
    roc_auc = metrics.auc(fpr, tpr)
    plt.plot(fpr, tpr, 'b', label = model + ' AUC = %0.2f' % roc_auc, color=co
lors[i])
    plt.legend(loc = 'lower right')

for i, model in list(enumerate(models)):
    plot_roc_curves(y_test, probs[i], models[i])

plt.show()
```


Receiver Operating Characteristic
True Positive Rate
False Positive Rate
0.0
0.2
0.4
0.6
0.8
1.0
Logistic Regression AUC = 0.89
KNN AUC = 0.91
Naive Bayes AUC = 0.78
Decision Tree AUC = 0.90
Random Forest AUC = 0.96
Gradient Boosting Trees AUC = 0.94
BaggingClassifier AUC = 0.97
XGBoost AUC = 0.96