In [158]:

```
import numpy as np
import pandas as pd
from sklearn.preprocessing import LabelEncoder
from sklearn.linear_model import ElasticNet
from sklearn.linear_model import RidgeCV
from sklearn.linear_model import LassoCV
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_squared_error
from sklearn.metrics import mean_absolute_error
from sklearn.metrics import r2_score
from sklearn.metrics import accuracy_score
from sklearn.linear_model import LogisticRegression
from sklearn.svm import SVC
from sklearn import tree
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import confusion_matrix
from sklearn.metrics import average_precision_score
from sklearn.metrics import f1_score
from sklearn.metrics import recall_score
from sklearn.metrics import classification_report
from sklearn import ensemble
from sklearn import preprocessing
from sklearn.naive_bayes import GaussianNB
```

Regression

In [131]:

```
train_data=pd.read_csv('train.csv')
test_data=pd.read_csv('test.csv')

train_data['MSSubClass']=train_data['MSSubClass'].apply(str)
train_data['OverallCond']=train_data['OverallCond'].astype(str)
train_data['YrSold']=train_data['YrSold'].astype(str)
train_data['MoSold']=train_data['MoSold'].astype(str)

test_data['MSSubClass']=test_data['MSSubClass'].apply(str)
test_data['OverallCond']=test_data['OverallCond'].astype(str)
test_data['YrSold']=test_data['YrSold'].astype(str)
test_data['MoSold']=test_data['MoSold'].astype(str)

columns=['MSZoning', 'FireplaceQu', 'BsmtQual', 'BsmtCond', 'GarageQual', 'GarageCo
for col in columns:
    label_enc=LabelEncoder()
    label_enc.fit(list(train_data[col].values))
    train_data[col]=label_enc.transform(list(train_data[col].values))
    label_enc.fit(list(test_data[col].values))
    test_data[col]=label_enc.transform(list(test_data[col].values))

train_data.drop("Id", axis=1, inplace=True)
test_data.drop("Id", axis=1, inplace=True)
train_data.drop("LandContour", axis=1, inplace=True)
test_data.drop("LandContour", axis=1, inplace=True)
train_data.drop("Utilities", axis=1, inplace=True)
test_data.drop("Utilities", axis=1, inplace=True)
train_data.drop("MiscFeature", axis=1, inplace=True)
test_data.drop("MiscFeature", axis=1, inplace=True)
train_data.drop("Neighborhood", axis=1, inplace=True)
test_data.drop("Neighborhood", axis=1, inplace=True)

for col in train_data.columns:
    if(col not in columns+['SalePrice']):
        try:
            if(col not in 'SalePrice'):
                train_data.drop(col, axis=1, inplace=True)
                test_data.drop(col, axis=1, inplace=True)
        except KeyError as e:
            continue

print(train_data.shape)
print(test_data.shape)

train_data.head()
test_data.head()

y_train=train_data.SalePrice.values
train_data.drop(['SalePrice'], axis=1, inplace=True)
print(train_data.shape)
print(y_train.shape)
```

```
(1460, 31)
(1459, 30)
(1460, 30)
(1460,)
```

In [39]:

```
train_data.head()
```

Out[39]:

	MSSubClass	MSZoning	Street	Alley	LotShape	LotConfig	LandSlope	OverallCond	Exter(
0	9	3	1	2	3	4	0	4	
1	4	3	1	2	3	2	0	7	
2	9	3	1	2	0	4	0	4	
3	10	3	1	2	0	0	0	4	
4	9	3	1	2	0	2	0	4	

5 rows × 30 columns

In [135]:

```
X_train, X_test, Y_train, Y_test=train_test_split(train_data, y_train, test_size=0.
normalizer=1e8

print('Ridge regression metrics:')
ridge_cv=RidgeCV(cv=5, alphas=[1e-3, 1e-2, 1e-1, 1]).fit(X_train, Y_train)
print('Training Regression score:', ridge_cv.score(X_train, Y_train))
print('Testing Regression score:', ridge_cv.score(X_test, Y_test))
print('Training MSE:', mean_squared_error(Y_train, ridge_cv.predict(X_train))/norma
print('Testing MSE:', mean_squared_error(Y_test, ridge_cv.predict(X_test))/normaliz
print('Training MAE:', mean_absolute_error(Y_train, ridge_cv.predict(X_train))*1e5/
print('Testing MAE', mean_absolute_error(Y_test, ridge_cv.predict(X_test))*1e5/norm
print('Training R2 score:', r2_score(Y_train, ridge_cv.predict(X_train)))
print('Testing R2 score:', r2_score(Y_test, ridge_cv.predict(X_test)))
print()

print('Lasso regression metrics:')
lasso_cv=LassoCV(cv=5, alphas=[1e-3, 1e-2, 1e-1, 1], random_state=0).fit(X_train, Y
print('Training Regression score:', lasso_cv.score(X_train, Y_train))
print('Testing Regression score:', lasso_cv.score(X_test, Y_test))
print('Training MSE:', mean_squared_error(Y_train, lasso_cv.predict(X_train))/norma
print('Testing MSE:', mean_squared_error(Y_test, lasso_cv.predict(X_test))/normaliz
print('Training MAE:', mean_absolute_error(Y_train, lasso_cv.predict(X_train))*1e5/
print('Testing MAE', mean_absolute_error(Y_test,lasso_cv.predict(X_test))*1e5/norma
print('Training R2 score:', r2_score(Y_train, lasso_cv.predict(X_train)))
print('Testing R2 score:', r2_score(Y_test, lasso_cv.predict(X_test)))
print()

print('Elastic Net regression metrics:')
elastic_net=ElasticNet(random_state=0).fit(X_train, Y_train)
print('Training Regression score:', elastic_net.score(X_train, Y_train))
print('Testing Regression score:', elastic_net.score(X_test, Y_test))
print('Training MSE:', mean_squared_error(Y_train, elastic_net.predict(X_train))/no
print('Testing MSE:', mean_squared_error(Y_test, elastic_net.predict(X_test))/norma
print('Training MAE:', mean_absolute_error(Y_train, elastic_net.predict(X_train))*1
print('Testing MAE', mean_absolute_error(Y_test, elastic_net.predict(X_test))*1e5/n
print('Training R2 score:', r2_score(Y_train, elastic_net.predict(X_train)))
print('Testing R2 score:', r2_score(Y_test, elastic_net.predict(X_test)))
print()

print('XGBoost regression metrics:')
xgb=ensemble.GradientBoostingRegressor().fit(X_train, Y_train)
print('Training Regression score:', xgb.score(X_train, Y_train))
print('Testing Regression score:', xgb.score(X_test, Y_test))
print('Training MSE:', mean_squared_error(Y_train, xgb.predict(X_train))/normalizer
print('Testing MSE:', mean_squared_error(Y_test, xgb.predict(X_test))/normalizer)
print('Training MAE:', mean_absolute_error(Y_train, xgb.predict(X_train))*1e5/norma
print('Testing MAE', mean_absolute_error(Y_test, xgb.predict(X_test))*1e5/normalize
print('Training R2 score:', r2_score(Y_train, xgb.predict(X_train)))
print('Testing R2 score:', r2_score(Y_test, xgb.predict(X_test)))
print()
```

```
Ridge regression metrics:
Training Regression score: 0.6623254810337607
Testing Regression score: 0.6726003825698206
Training MSE: 20.32319575989016
Testing MSE: 22.846252277234505
Training MAE: 30.704638945673267
Testing MAE 32.36687626519441
Training R2 score: 0.6623254810337607
```

```
Testing R2 score: 0.6726003825698206

Lasso regression metrics:
Training Regression score: 0.6623730341350089
Testing Regression score: 0.6726945147622702
Training MSE: 20.320333740604198
Testing MSE: 22.83968364458615
Training MAE: 30.713303846782047
Testing MAE 32.359331613657496
Training R2 score: 0.6623730341350089
Testing R2 score: 0.6726945147622702

Elastic Net regression metrics:
Training Regression score: 0.6189523323281737
Testing Regression score: 0.6384088257393613
Training MSE: 22.93364144754538
Testing MSE: 25.232171171188803
Training MAE: 32.177202274810206
Testing MAE 33.28663915304564
Training R2 score: 0.6189523323281737
Testing R2 score: 0.6384088257393613

XGBoost regression metrics:
Training Regression score: 0.8489002805042637
Testing Regression score: 0.7202370568077652
Training MSE: 9.094050649653434
Testing MSE: 19.52212048431751
Training MAE: 21.11383604509769
Testing MAE 27.756652115613733
Training R2 score: 0.8489002805042637
Testing R2 score: 0.7202370568077652
```

Classification

In [146]:

```python
data=pd.read_csv('bank-additional-full.csv', delimiter=';')

for col in data.columns:
    label_enc=LabelEncoder()
    label_enc.fit(list(data[col].values))
    data[col]=label_enc.transform(list(data[col].values))

data_x=data.iloc[:, :-1]
data_y=data.y

data_x=preprocessing.scale(data_x)
```

```
/home/vishaal/.local/lib/python3.5/site-packages/ipykernel_launcher.p
y:11: DataConversionWarning: Data with input dtype int64 were all conv
erted to float64 by the scale function.
  # This is added back by InteractiveShellApp.init_path()
```

In [151]:

```
X_train, X_test, Y_train, Y_test=train_test_split(data_x, data_y, test_size=0.3, ra

log_reg=LogisticRegression(random_state=0, solver='lbfgs', multi_class='multinomial
Y_pred_train=log_reg.predict(X_train)
print('Training accuracy:', accuracy_score(Y_train, Y_pred_train))
print('Training F1 score:', f1_score(Y_train, Y_pred_train, average='micro'))
print('Recall score:', recall_score(Y_train, Y_pred_train, average='micro'))
print('Classification report:', classification_report(Y_train, Y_pred_train))

Y_pred_test=log_reg.predict(X_test)
print('Test accuracy:', accuracy_score(Y_test, Y_pred_test))
print('Test F1 score:', f1_score(Y_test, Y_pred_test, average='micro'))
print('Recall score:', recall_score(Y_test, Y_pred_test, average='micro'))
print('Classification report:', classification_report(Y_test, Y_pred_test))

skplt.metrics.plot_roc_curve(y_test, y_proba)
plt.show()
```

```
Training accuracy: 0.90988866151018
Training F1 score: 0.90988866151018
Recall score: 0.90988866151018
Classification report:               precision    recall  f1-score   s
upport

           0       0.93      0.97      0.95     25580
           1       0.66      0.41      0.51      3251

   micro avg       0.91      0.91      0.91     28831
   macro avg       0.80      0.69      0.73     28831
weighted avg       0.90      0.91      0.90     28831

Test accuracy: 0.9126810714574735
Test F1 score: 0.9126810714574733
Recall score: 0.9126810714574735
Classification report:               precision    recall  f1-score   s
upport

           0       0.93      0.97      0.95     10968
           1       0.68      0.43      0.53      1389

   micro avg       0.91      0.91      0.91     12357
   macro avg       0.80      0.70      0.74     12357
weighted avg       0.90      0.91      0.90     12357
```

In [155]:

```
dt=tree.DecisionTreeClassifier().fit(X_train, Y_train)
Y_pred_train=dt.predict(X_train)
print('Training accuracy:', accuracy_score(Y_train, Y_pred_train))
print('Training F1 score:', f1_score(Y_train, Y_pred_train, average='micro'))
print('Recall score:', recall_score(Y_train, Y_pred_train, average='micro'))
print('Classification report:', classification_report(Y_train, Y_pred_train))

Y_pred_test=dt.predict(X_test)
print('Test accuracy:', accuracy_score(Y_test, Y_pred_test))
print('Test F1 score:', f1_score(Y_test, Y_pred_test, average='micro'))
print('Recall score:', recall_score(Y_test, Y_pred_test, average='micro'))
print('Classification report:', classification_report(Y_test, Y_pred_test))
```

```
Training accuracy: 1.0
Training F1 score: 1.0
Recall score: 1.0
Classification report:               precision    recall  f1-score   s
upport

           0       1.00      1.00      1.00     25580
           1       1.00      1.00      1.00      3251

   micro avg       1.00      1.00      1.00     28831
   macro avg       1.00      1.00      1.00     28831
weighted avg       1.00      1.00      1.00     28831

Test accuracy: 0.891964068948774
Test F1 score: 0.891964068948774
Recall score: 0.891964068948774
Classification report:               precision    recall  f1-score   s
upport

           0       0.94      0.94      0.94     10968
           1       0.52      0.53      0.52      1389

   micro avg       0.89      0.89      0.89     12357
   macro avg       0.73      0.73      0.73     12357
weighted avg       0.89      0.89      0.89     12357
```

In [156]:

```
rf=RandomForestClassifier(n_estimators=100, max_depth=2, random_state=0).fit(X_trai
Y_pred_train=rf.predict(X_train)
print('Training accuracy:', accuracy_score(Y_train, Y_pred_train))
print('Training F1 score:', f1_score(Y_train, Y_pred_train, average='micro'))
print('Recall score:', recall_score(Y_train, Y_pred_train, average='micro'))
print('Classification report:', classification_report(Y_train, Y_pred_train))

Y_pred_test=rf.predict(X_test)
print('Test accuracy:', accuracy_score(Y_test, Y_pred_test))
print('Test F1 score:', f1_score(Y_test, Y_pred_test, average='micro'))
print('Recall score:', recall_score(Y_test, Y_pred_test, average='micro'))
print('Classification report:', classification_report(Y_test, Y_pred_test))
```

```
Training accuracy: 0.9004543720301065
Training F1 score: 0.9004543720301065
Recall score: 0.9004543720301065
Classification report:               precision    recall  f1-score   s
upport

           0       0.90      1.00      0.95     25580
           1       0.83      0.15      0.25      3251

   micro avg       0.90      0.90      0.90     28831
   macro avg       0.86      0.57      0.60     28831
weighted avg       0.89      0.90      0.87     28831

Test accuracy: 0.9003803512179331
Test F1 score: 0.9003803512179331
Recall score: 0.9003803512179331
Classification report:               precision    recall  f1-score   s
upport

           0       0.90      1.00      0.95     10968
           1       0.85      0.14      0.24      1389

   micro avg       0.90      0.90      0.90     12357
   macro avg       0.87      0.57      0.59     12357
weighted avg       0.90      0.90      0.87     12357
```

In [159]:

```
gnb=GaussianNB().fit(X_train, Y_train)
Y_pred_train=gnb.predict(X_train)
print('Training accuracy:', accuracy_score(Y_train, Y_pred_train))
print('Training F1 score:', f1_score(Y_train, Y_pred_train, average='micro'))
print('Recall score:', recall_score(Y_train, Y_pred_train, average='micro'))
print('Classification report:', classification_report(Y_train, Y_pred_train))

Y_pred_test=gnb.predict(X_test)
print('Test accuracy:', accuracy_score(Y_test, Y_pred_test))
print('Test F1 score:', f1_score(Y_test, Y_pred_test, average='micro'))
print('Recall score:', recall_score(Y_test, Y_pred_test, average='micro'))
print('Classification report:', classification_report(Y_test, Y_pred_test))
```

```
Training accuracy: 0.847039644826749
Training F1 score: 0.847039644826749
Recall score: 0.847039644826749
Classification report:               precision    recall  f1-score   s
upport

           0       0.94      0.88      0.91     25580
           1       0.38      0.58      0.46      3251

   micro avg       0.85      0.85      0.85     28831
   macro avg       0.66      0.73      0.69     28831
weighted avg       0.88      0.85      0.86     28831

Test accuracy: 0.8507728413045238
Test F1 score: 0.8507728413045239
Recall score: 0.8507728413045238
Classification report:               precision    recall  f1-score   s
upport

           0       0.95      0.88      0.91     10968
           1       0.39      0.60      0.47      1389

   micro avg       0.85      0.85      0.85     12357
   macro avg       0.67      0.74      0.69     12357
weighted avg       0.88      0.85      0.86     12357
```

Regression Analysis:

The best model for the regression task was the XGBoost regressor. This is because XGBoost is an ensemble method and hence it generalizes better. Therefore, the test set regression performance is the best for XGBoost regressor. I have used the following metrics for regression:

- MSE
- MAE
- R2 score

For the dataset processing, I have dropped the columns which have the same values across all data samples since they do not add any importance to the regression task. Totally I have considered 30 features as the regression features to fit on the dataset.

Classification Analysis:

The best model for the classification task was the Logistic Regression model with no regularization. This is probably because of the bad fits of the other models. For the dataset, I have done no processing, I have taken all 20 features as is and scaled them by doing z-scoring.

The metrics used were:

- Accuracy
- Precision
- Recall
- F1 score