Imbalanced Datasets - Resolution Techniques

1. Load the Ames housing data set. Create a feature matrix using the following variables: Lot_Area, Year_Built, Gr_Liv_Area, Total_Bsmt_SF, and Full_Bath. Print the first few rows.

In [274]:
```
import numpy      as np
import pandas     as pd
import sklearn    as sk
import statistics as st
import pprint     as pp
import math       as mth
import seaborn    as sns
import matplotlib as mpl
import os         as os
import matplotlib.pyplot as plt
import imblearn

from sklearn.preprocessing import StandardScaler
from sklearn.preprocessing import PolynomialFeatures

from sklearn.linear_model  import LinearRegression
from sklearn.linear_model  import LogisticRegression
from sklearn.linear_model  import LogisticRegressionCV
from sklearn.linear_model  import Ridge
from sklearn.linear_model  import Lasso

from sklearn.model_selection import KFold
from sklearn.model_selection import GridSearchCV
from sklearn.model_selection import cross_validate
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 confusion_matrix

from sklearn.metrics         import accuracy_score
from sklearn.metrics         import recall_score
from sklearn.metrics         import precision_score
from sklearn.metrics         import f1_score
from sklearn.metrics         import roc_curve
from sklearn.metrics         import roc_auc_score

#from   tensorflow import keras
%matplotlib inline
```

In [275]:
```
from imblearn.over_sampling  import RandomOverSampler
from imblearn.under_sampling import RandomUnderSampler

df_model_results = pd.DataFrame(columns = ['model_name', 'accuracy_sco
re',
                                              'recall_score', 'precision_
score', 'f1_score'])
```

In [276]:
```
#Reading CSV Data File #1
data_folder = "C:/Users/HP/Google Drive/Education/Ashish/MCAS-UND/Data
sets/"
file_name = data_folder + "ames.csv"
ames_ds = pd.read_csv(file_name)
```

In [277]:
```
col_list = ['Lot_Area', 'Year_Built', 'Gr_Liv_Area', 'Total_Bsmt_SF',
'Full_Bath', 'Sale_Price']
ames_df = ames_ds[col_list].dropna()
ames_df.head()
```

Out[277]:

	Lot_Area	Year_Built	Gr_Liv_Area	Total_Bsmt_SF	Full_Bath	Sale_Price
0	31770	1960	1656	1080	1	215000
1	11622	1961	896	882	1	105000
2	14267	1958	1329	1329	1	172000
3	11160	1968	2110	2110	2	244000
4	13830	1997	1629	928	2	189900

2 Create a vector for the response. This will be 1 if the Sale_Price is greater than 300,000, and 0 otherwise. What is the proportion of homes that have a sale price greater than 300,000?

In [278]:
```
ames_df['Expensive'] = np.where(ames_df['Sale_Price'] > 300000, 1, 0)
```

In [279]:
```
prop = round(ames_df.Expensive.value_counts()[1] / ames_df.shape[0],2)
print ('Proportion of homes having the sales price > $300000 : ', pro
p)
```

```
Proportion of homes having the sales price > $300000 :  0.08
```

3. Split the data into training and testing sets using a 60/40 split, making sure to stratify based on the response variable. Print the dimensions of each set.

In [280]:
```
ames_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 2930 entries, 0 to 2929
Data columns (total 7 columns):
 #   Column         Non-Null Count  Dtype
---  ------         --------------  -----
 0   Lot_Area       2930 non-null   int64
 1   Year_Built     2930 non-null   int64
 2   Gr_Liv_Area    2930 non-null   int64
 3   Total_Bsmt_SF  2930 non-null   int64
 4   Full_Bath      2930 non-null   int64
 5   Sale_Price     2930 non-null   int64
 6   Expensive      2930 non-null   int32
dtypes: int32(1), int64(6)
memory usage: 171.7 KB
```

In [281]:
```
X = ames_df.drop(['Sale_Price','Expensive'], axis = 1).values
y = ames_df['Expensive'].values
X.shape, y.shape
```

Out[281]:
```
((2930, 5), (2930,))
```

In [282]:
```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size =
.4, stratify = y,
                                                    random_state
= 1)
X_train.shape, y_train.shape, X_test.shape, y_test.shape
```

Out[282]:
```
((1758, 5), (1758,), (1172, 5), (1172,))
```

4. Standardize the features from the training set and apply the transformation to the test set. Print the first few rows of the standardized features from the training set.

In [283]:
```
scaler = StandardScaler()
scaler.fit(X_train)
X_train = scaler.transform(X_train)
X_test  = scaler.transform(X_test)
```

In [284]:
```
X_train[0:9,]
```

Out[284]:
```
array([[-0.18393009, -0.14928268, -0.55295036,  0.43717178, -1.029404
59],
       [-0.35501203, -2.30202581, -1.59229585, -1.42233963, -1.029404
59],
       [ 0.31305404,  0.86185424,  1.50937297, -0.19220131,  0.785734
09],
       [-0.27793605, -0.14928268,  0.69712856, -0.68807102, -1.029404
59],
       [ 0.27919407, -0.47545589, -1.26085103, -2.4903667 , -1.029404
59],
       [-0.04626259, -0.67115981,  0.92832155, -0.16359345, -1.029404
59],
       [ 4.77956245,  0.69876764,  0.93036751,  1.2763359 ,  0.785734
09],
       [-0.32649838, -1.48659281,  0.10175545, -0.22080918,  0.785734
09],
       [ 0.77350504, -0.47545589, -0.55090441,  0.21307681, -1.029404
59]])
```

In [285]:
```
#
-----------------------------------------------------------------------
--
# Function for generating Decision Tree based Model
#
-----------------------------------------------------------------------
--
def f_build_classification_model(arg_model,
                                 arg_model_name,
                                 arg_X_train, arg_y_train,
                                 arg_X_test, arg_y_test):
    model = arg_model
    #
-----------------------------------------------------------------------
--
    # Fitting the Decision Tree Type Model
    #
-----------------------------------------------------------------------
--
    model.fit(arg_X_train, arg_y_train)

    #
-----------------------------------------------------------------------
--
    # Doing Predictions on Test dataset
    #
-----------------------------------------------------------------------
--
    y_test_pred = model.predict(arg_X_test)

    #
-----------------------------------------------------------------------
--
    # Creating a dictionary to store model metrics
    #
-----------------------------------------------------------------------
--
    model_metrics = {}

    #
-----------------------------------------------------------------------
--
    # Collecting metrics
    #
-----------------------------------------------------------------------
--

    # Calculating confusion Matrix for the LR Model
    #cm_list = confusion_matrix(y_test, y_test_pred)
    #[[tp, fp], [fn, tn]] = cm_list

    model_accuracy      = round(accuracy_score (arg_y_test, y_test_pre
d), 4)
    model_recall        = round(recall_score   (arg_y_test, y_test_pre
d), 4)
```

```
    model_precision     = round(precision_score(arg_y_test, y_test_pre
d), 4)
    fpr, tpr, thresholds = roc_curve (arg_y_test, y_test_pred)
    model_f1_score      = round(f1_score       (arg_y_test, y_test_pre
d), 4)

    #
----------------------------------------------------------------------
--
    # Storing metrics in the dictionary
    #
----------------------------------------------------------------------
--
    model_metrics['model_name'] = arg_model_name
    model_metrics['accuracy_score']  = model_accuracy
    model_metrics['recall_score']    = model_recall
    model_metrics['precision_score'] = model_precision
    model_metrics['f1_score']        = model_f1_score
    #model_metrics['true_positives'] = tp
    #model_metrics['false_positives'] = fp
    #model_metrics['true_negatives'] = tn
    #model_metrics['false_negatives'] = fn

    print('Logistic Model Precision Score : ', model_precision)
    print('Logistic Model Recall Score : ', model_recall)
    print('Logistic Model F1 Score : ', model_f1_score)

    plt.plot(fpr, tpr)
    plt.plot([0,1], [0,1])

    return model_metrics
# --------------------- END OF FUNCTION --------------------------
```

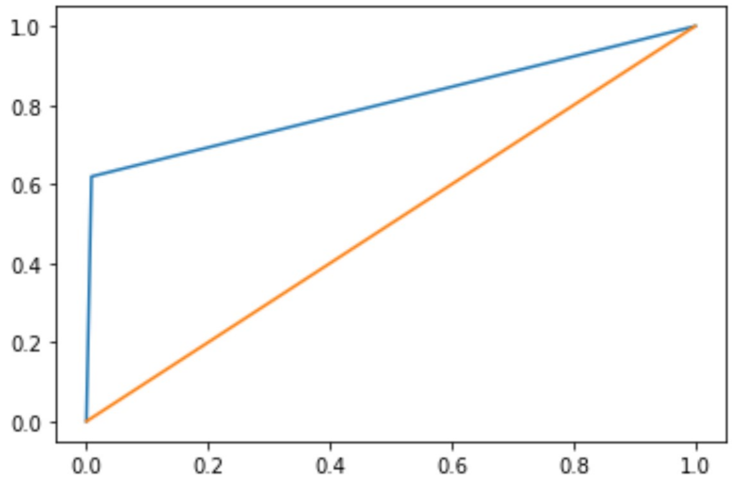
5. Model 1: Fit a model without any correction for the data being imbalanced.

a. Fit a Logistic Regression model to the training set (using regularization is optional).

b. Calculate and print the precision, recall, and F1 score for the test set.

In [286]:
```
lrm1 = LogisticRegression(solver = 'liblinear', random_state = 0)
model1_scores = f_build_classification_model(lrm1,
                                             'Model1 - Logistic - No O
ptimization',
                                             X_train,
                                             y_train,
                                             X_test,
                                             y_test)
df_model_results = df_model_results.append(model1_scores, ignore_index
= True)
```

```
Logistic Model Precision Score :  0.8636
Logistic Model Recall Score :  0.6196
Logistic Model F1 Score :  0.7215
```



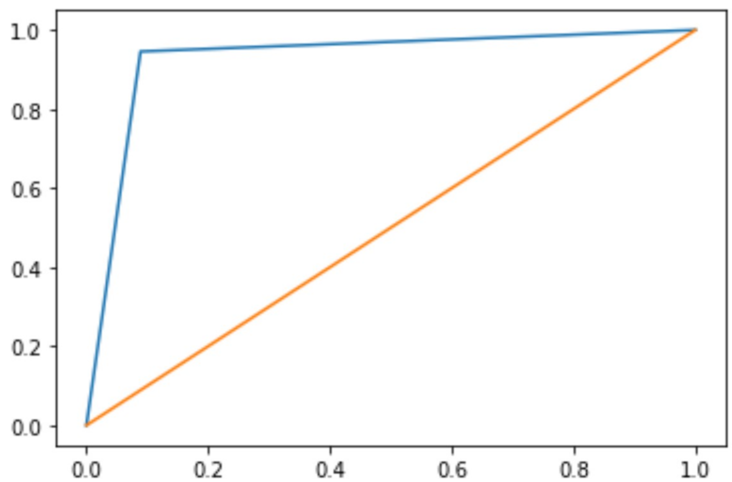
6. Model 2: Fit a model using weights to balance the classes.

Fit a Logistic Regression model to the training set again, but this time use different weights for the expensive and inexpensive houses. This is done by setting the class_weight parameter to 'balanced'.

Calculate and print the precision, recall, and F1 score for the test set.

In [287]:
```
lrm2 = LogisticRegression(class_weight = 'balanced')

model_scores = f_build_classification_model(lrm2,
                                            'Model2 - Logistic - Balan
ced Dataset',
                                            X_train,
                                            y_train,
                                            X_test,
                                            y_test)

df_model_results = df_model_results.append(model_scores, ignore_index
= True)
```

```
Logistic Model Precision Score :  0.4754
Logistic Model Recall Score :  0.9457
Logistic Model F1 Score :  0.6327
```



Preparing the data for undersampling and oversampling approaches for balancing Logistical Model Exercises

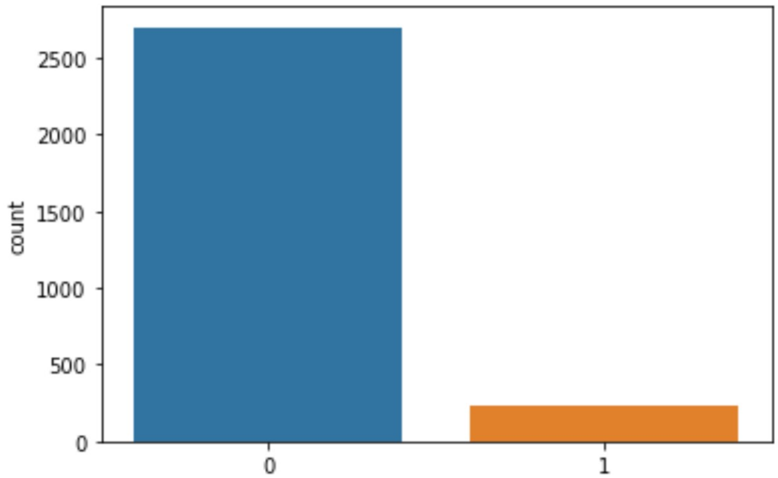
In [288]:
```
X = ames_df.drop(['Sale_Price','Expensive'], axis = 1)
y = ames_df['Expensive'].values
X.shape, y.shape
```

Out[288]:
```
((2930, 5), (2930,))
```

In [289]:
```
sns.countplot(y)
```

```
C:\Users\HP\.conda\envs\tensorflow\lib\site-packages\seaborn\_decorat
ors.py:43: FutureWarning: Pass the following variable as a keyword ar
g: x. From version 0.12, the only valid positional argument will be `
data`, and passing other arguments without an explicit keyword will r
esult in an error or misinterpretation.
  FutureWarning
```

Out[289]: <AxesSubplot:ylabel='count'>



7. Model 3: Fit a model after undersampling the majority class.

7.1 We first need to create a new training dataset after undersampling the majority class. Determine how many of the original observations you need to keep from the inexpensive homes to make the classes balanced.

7.2 Sample from the majority class without replacement so that the classes are balanced.

7.3 Create a new set of features and a new response vector. Confirm that the resampling worked by printing the proportion of expensive homes in the new dataset.

7.4 Print the dimensions of this new dataset.

7.5 Fit a Logistic Regression model to this new training data.

7.6 Calculate and print the precision, recall, and F1 score for the test set.

In [290]:
```
a1 = ames_df.Expensive.value_counts()[1]
print('Number of records needed to balance (Undersampling) the datase
t: ', a1)
```

```
Number of records needed to balance (Undersampling) the dataset:  230
```

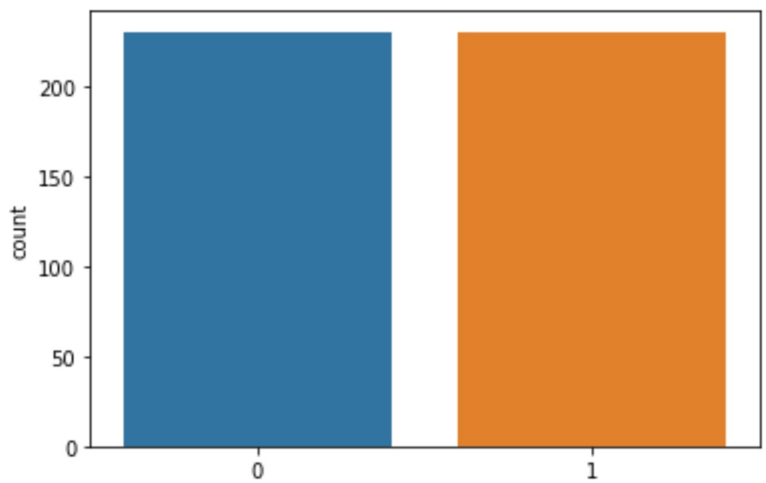
In [291]:
```
undersample = RandomUnderSampler(sampling_strategy='majority')

X_under, y_under = undersample.fit_resample(X, y)
X_under.shape, y_under.shape
```

Out[291]: ((460, 5), (460,))

In [292]:
```
sns.countplot(y_under)
```

```
C:\Users\HP\.conda\envs\tensorflow\lib\site-packages\seaborn\_decorat
ors.py:43: FutureWarning: Pass the following variable as a keyword ar
g: x. From version 0.12, the only valid positional argument will be `
data`, and passing other arguments without an explicit keyword will r
esult in an error or misinterpretation.
  FutureWarning
```

Out[292]: <AxesSubplot:ylabel='count'>

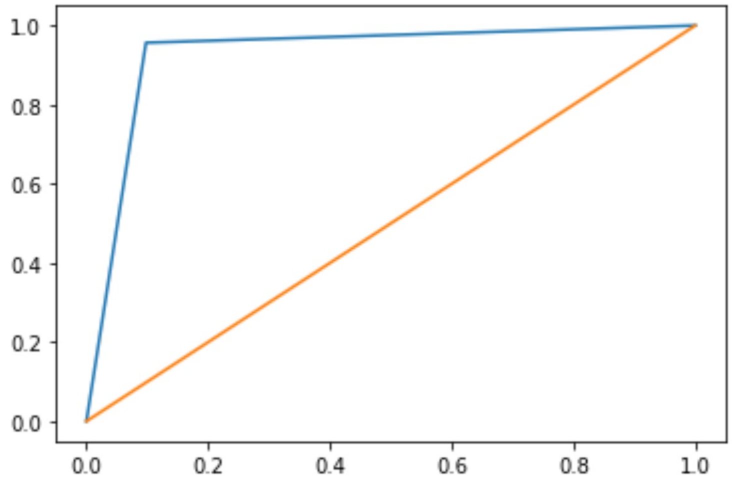


In [293]:
```
X_under_train, X_under_test, y_under_train, y_under_test = train_test_
split(X_under,

y_under,

test_size = .4,

stratify = y_under,

random_state = 1)

X_under_train.shape, y_under_train.shape, X_under_test.shape, y_under_
test.shape
```

Out[293]: ((276, 5), (276,), (184, 5), (184,))

In [294]:
```
lrm3 = LogisticRegression(solver = 'liblinear', random_state = 0)
model3_scores = f_build_classification_model(lrm3,
                                             'Model 3 - Logistic - Und
er Sampling',
                                             X_under_train,
                                             y_under_train,
                                             X_under_test,
                                             y_under_test)
```

```
Logistic Model Precision Score :  0.9072
Logistic Model Recall Score :  0.9565
Logistic Model F1 Score :  0.9312
```



In [295]:
```
df_model_results = df_model_results.append(model3_scores, ignore_index
= True)
```

8. Model 4: Fit a model after oversampling the minority class.

8.1 We first need to create a new training dataset with resampled data from the minority class. Determine how many additional observations you need from the expensive homes to make the classes balanced.

8.2 Sample from the minority class with replacement so that the classes are balanced. There are many ways to do this using functions from numpy or pandas, but you may find the np.random.choice function useful to sample the indices of the expensive homes.

8.3 Create a new set of features and a new response vector. Confirm that the resampling worked by printing the proportion of expensive homes in the new dataset.

8.4 Print the dimensions of this new dataset.

8.5 Fit a Logistic Regression model to this new training data.

8.6 Calculate and print the precision, recall, and F1 score for the test set.

In [296]:
```
a1 = ames_df.Expensive.value_counts()[0] - ames_df.Expensive.value_cou
nts()[1]
print('Number of records needed to balance (Oversampling) the dataset:
', a1)
```

```
Number of records needed to balance (Oversampling) the dataset:  2470
```

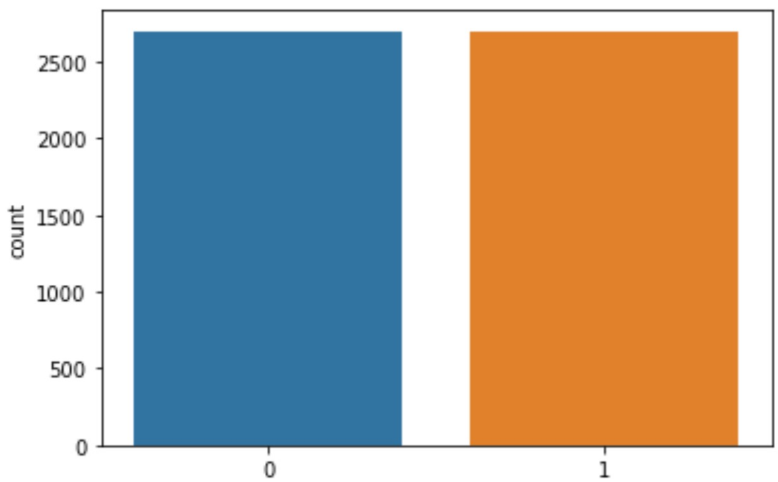
In [297]:
```
oversample = RandomOverSampler(sampling_strategy = 'minority')
X_over, y_over = oversample.fit_resample(X, y)
X_over.shape, y_over.shape
```

Out[297]:
```
((5400, 5), (5400,))
```

In [298]:
```
sns.countplot(y_over)
```

```
C:\Users\HP\.conda\envs\tensorflow\lib\site-packages\seaborn\_decorat
ors.py:43: FutureWarning: Pass the following variable as a keyword ar
g: x. From version 0.12, the only valid positional argument will be `
data`, and passing other arguments without an explicit keyword will r
esult in an error or misinterpretation.
  FutureWarning
```

Out[298]:
```
<AxesSubplot:ylabel='count'>
```

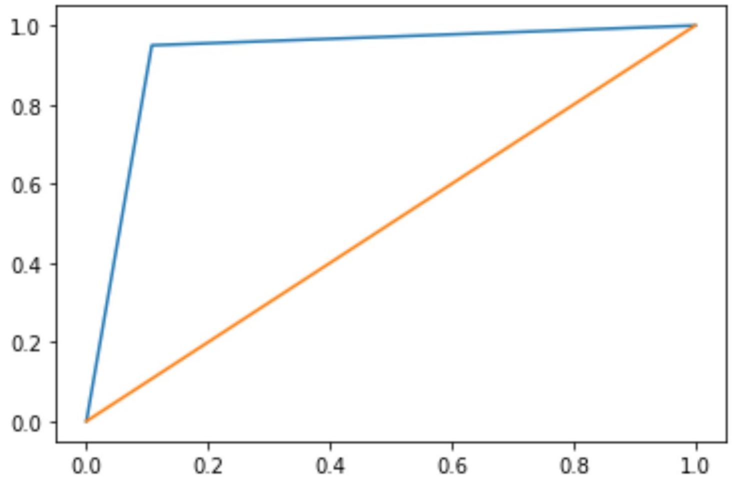


In [299]:
```
X_over_train, X_over_test, y_over_train, y_over_test = train_test_spli
t(X_over, y_over,
                                                    test_size = .4,
                                                    stratify = y_over,
                                                    random_state = 1)
X_over_train.shape, y_over_train.shape, X_over_test.shape, y_over_tes
t.shape
```

Out[299]:
```
((3240, 5), (3240,), (2160, 5), (2160,))
```

In [300]:
```
lrm4 = LogisticRegression(solver = 'liblinear', random_state = 0)
model4_scores = f_build_classification_model(lrm4,
                                             'Model 4 - Logistic - Ove
r Sampling',
                                              X_over_train,
                                              y_over_train,
                                              X_over_test,
                                              y_over_test)
```

```
Logistic Model Precision Score :  0.8984
Logistic Model Recall Score :  0.95
Logistic Model F1 Score :  0.9235
```



In [301]:
```
df_model_results = df_model_results.append(model4_scores, ignore_index
= True)
```

9. Conclusion: Difference in performance of the various models.

In [302]:
```
df_model_results
```

Out[302]:

	model_name	accuracy_score	recall_score	precision_score	f1_score
0	Model1 - Logistic - No Optimization	0.9625	0.6196	0.8636	0.7215
1	Model2 - Logistic - Balanced Dataset	0.9138	0.9457	0.4754	0.6327
2	Model 3 - Logistic - Under Sampling	0.9293	0.9565	0.9072	0.9312
3	Model 4 - Logistic - Over Sampling	0.9213	0.9500	0.8984	0.9235

Above dataframe presents and compares the 4 models output. As advised for Imbalanced Datasets, most appropriate metric for comparison is F1 Score and it is apprent as well. For the first 2 models which are built without using imbalancing techniques, F1 scores are poor, where after applying the undersampling and oversampling techniques, the F1 Scores were improved greatly as approximately 92%.

Between these 2 imbalance resolution techniques, Undersampling performs far better but I will prefer to go with oversampling approach as it works with 5400 observations rather than 460 from undersampling approach.