IEE 520 REPORT

Instructor : Prof George Runger

By

Jay Bhanushali

1213436781

Data Description

```
In [100]: train.info()

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 2500 entries, 0 to 2499
Data columns (total 68 columns):
Row    2500 non-null int64
x1     2500 non-null int64
x2     2500 non-null int64
x3     2500 non-null int64
x4     2500 non-null int64
x5     2500 non-null object
x6     2500 non-null int64
x7     2500 non-null int64
x8     2500 non-null int64
x9     2500 non-null int64
x10    2500 non-null float64
x11    2500 non-null float64
x12    2500 non-null float64
x13    2500 non-null object
x14    2500 non-null int64
x15    2500 non-null int64
x16    2500 non-null int64
x17    2500 non-null int64
x18    2500 non-null int64
x19    2500 non-null int64
x20    2500 non-null int64
x21    2500 non-null int64
x22    2500 non-null int64
x23    2500 non-null int64
x24    2500 non-null int64
x25    2500 non-null int64
x26    2500 non-null int64
x27    2500 non-null int64
x28    2500 non-null int64
x29    2500 non-null int64
x30    2500 non-null int64
x31    2500 non-null int64
x32    2500 non-null int64
x33    2500 non-null int64
x34    2500 non-null int64
x35    2500 non-null int64
x36    2500 non-null int64
x37    2500 non-null int64
x38    2500 non-null int64
x39    2500 non-null int64
x40    2500 non-null int64
x41    2500 non-null int64
x42    2500 non-null int64
x43    2500 non-null int64
x44    2500 non-null int64
x45    2500 non-null int64
x46    2500 non-null int64
x47    2500 non-null int64
x48    2500 non-null int64
x49    2500 non-null int64
x50    2500 non-null int64
x51    2500 non-null int64
x52    2500 non-null int64
x53    2500 non-null int64
x54    2500 non-null int64
x55    2500 non-null int64
x56    2500 non-null int64
x57    2500 non-null int64
x58    2500 non-null int64
x59    2500 non-null int64
x60    2500 non-null int64
x61    2500 non-null int64
x62    2500 non-null int64
x63    2500 non-null int64
x64    2500 non-null object
x65    2500 non-null object
x66    2500 non-null int64
y      2500 non-null int64
dtypes: float64(3), int64(61), object(4)
memory usage: 1.3+ MB
```

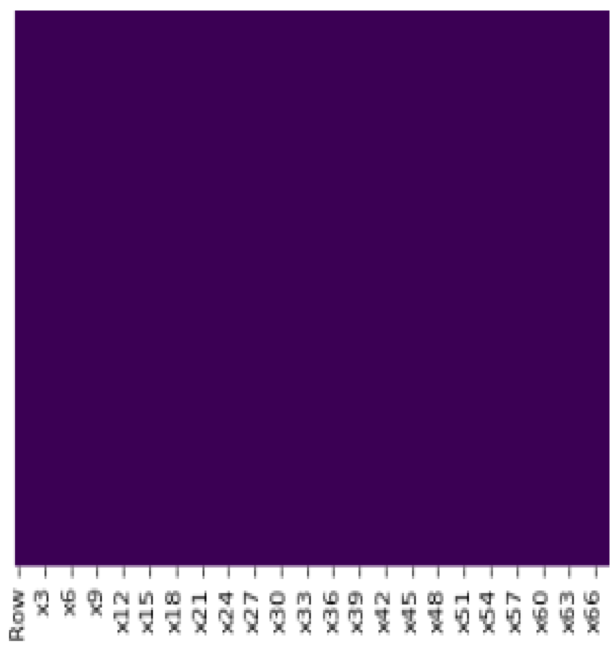
As we can see we have 66 independent variables and 1 dependent variable. Out of 66 dependent variables we have 4 categorical variables and therefore for analysis we have to replace them with dummy variables which we are doing with the help of command pd.get_dummies.

Exploratory Analysis

1)Checking for missing values

In [10]:
```
#checking missing data
sns.heatmap(train.isnull(),yticklabels=False,cbar=False,cmap='viridis')
#hence there is no missing data
```

Out[10]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x1b76de77b38>
```



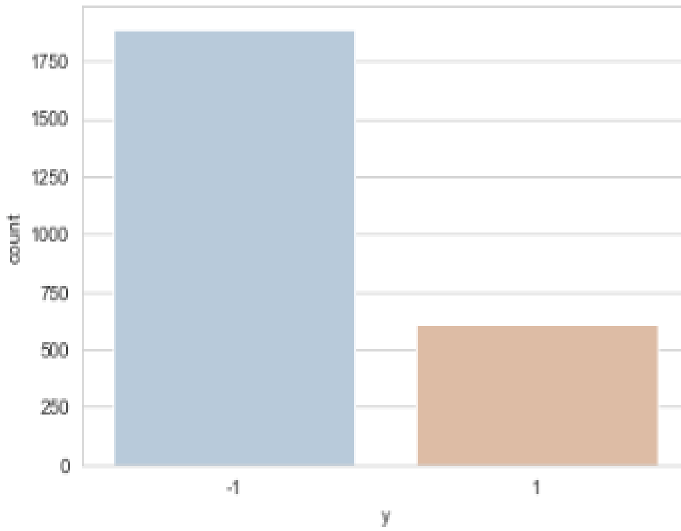
Hence there are no missing values.

2)Checking if the training data is balanced or not

It is observed that the model is unbalanced, not highly but the slightly as we can see

```
In [11]: sns.set_style('whitegrid')
         sns.countplot(x='y',data=train,palette='RdBu_r')
         #mapping y = 1 against -1

Out[11]: <matplotlib.axes._subplots.AxesSubplot at 0x1b76df9f908>
```



that the y = 1 class outcomes are much less that y = -1

Data Preprocessing

1)Now we will convert all the categorical variables into dummies

```
In [15]: #converting categorical to dummy and making dataset ready for analysis
         #train.info()

In [16]: x5 = pd.get_dummies(train['x5'],drop_first=True)
         x13 = pd.get_dummies(train['x13'],drop_first=True)
         x65 = pd.get_dummies(train['x65'],drop_first=True)
         x64 = pd.get_dummies(train['x66'],drop_first=True)

In [17]: train = pd.concat([train,x5,x13,x64,x65],axis=1)

In [18]: #train.info()

In [19]: train.drop(['x5','x13','x64','x65','Row'],axis=1,inplace=True)
```

In the code above we converted the categorical into dummies and concatenated the dummy variable column in our main dataframe followed by dropping the columns whose dummy variables we have created.

2)Since we know that the data we are using is unbalanced we try to balance it using 2 methods

1)Simple upsampling

2)Upsampling using SMOTE

Data Analysis

Sampling Method for training-testing data

We have used 2 types of sampling method

1)Simply using the train – test split by using train_test_split library from sklearn.model_selection

We have splitted training data in to 2 sets of training and testing data with training size as 70 % of the data and testing size as 30 %

After doing that we observe that our training set has 1322 instances of -1 and 428 instances of +1 , which is unbalanced and our test data has 569 instances of -1 and 181 instances of +1

(i)Applying logistic regression to this data

We get the following output

```
print(classification_report(y_test,predictions))

              precision    recall  f1-score   support

          -1       0.87      0.92      0.90       569
           1       0.70      0.59      0.64       181
```

```
print('balance_error_rate', balance_error_rate/2)

[[524  45]
 [ 75 106]]
balance_error_rate 0.246725378438474
```

As we can see that recall that we are getting for class 1 is 0.59 which is very low and means that our model is not able to predict the values of class 1(minority class correctly). Hence we have to do something in which we can train our model on data of class 1 properly.

I also tried applying other models but all did not give proper output.

2)Upsampling the whole Data using resample library from sklearn.utils and then using Train – split method

```
#RESAMPLING
#UPSAMPLING
from sklearn.utils import resample
```

```
# Separate majority and minority classes
train_majority = train[train.y == -1]
train_minority = train[train.y == 1]
```

```
# Upsample minority class
# Upsample minority class
train_minority_upsampled = resample(train_minority,
                                    replace=True,     # sample with replacement
                                    n_samples=1891,    # to match majority class train['y'].value_counts()
                                    random_state=123) # reproducible results
# Combine majority class with upsampled minority class
train_upsampled = pd.concat([train_majority, train_minority_upsampled])
```

Upsampling data

After upsampling now we have data with both y = 1 and -1 counts as 1891.In this method the resample library has duplicated various data with y = 1 output

Now we split the whole upsampled data into training and testing

And train our models on splitted upsampled train data and test on splitted upsampled test data

We also tried training model on the splitted upsampled train data and test on regular test data in the first part

```
#SPLITTING Upsampled data sets  into training and testing
y_train_upsampled = train_upsampled.y
X_train_upsampled = train_upsampled.drop('y', axis=1)
from sklearn.model_selection import train_test_split
X_train1, X_test1, y_train1, y_test1 = train_test_split(X_train_upsampled,y_train_upsampled,test_size=0.
                                                        random_state=101)
```

Splitting our up sampled data into train and test sets

Trying Random forest which is trained on splitted upsampled train data and tested on splitted upsampled test data we get this results and balanced error

```
#Training random on upscaled data
from sklearn.ensemble import RandomForestClassifier
rfc2 = RandomForestClassifier(n_estimators=100)
rfc_fitted_upsampled = rfc2.fit(X_train1, y_train1)
rfc_pred2 = rfc_fitted_upsampled.predict(X_test1)
print(confusion_matrix(y_test1,rfc_pred2))
print(classification_report(y_test1,rfc_pred2))
confusion_array8 = confusion_matrix(y_test1,rfc_pred2)

balance_error_rate8 = (confusion_array8[0][1]/sum(confusion_array8[0])) + ((confusion_array8[1][0])/sum

print('Balanced error :' , balance_error_rate8/2)
```

```
[[506  69]
 [ 16 544]]
              precision    recall  f1-score   support

          -1       0.97      0.88      0.92       575
           1       0.89      0.97      0.93       560

   micro avg       0.93      0.93      0.93      1135
   macro avg       0.93      0.93      0.93      1135
weighted avg       0.93      0.93      0.92      1135

Balanced error : 0.07428571428571429
```

We get balanced error rate of 0.074 percent but this may not be true because in the testing data is upsampled and therefore there might be many instances with the same inputs which might increase the accuracy of the model more.

3)We apply SMOTE(Synthetic Minority Over sampling technique)

Now we split data into 2 parts , training and testing and only apply SMOTE on Training data So that we can train over predictive model Equally good on both the classes of training data.

```
#APPLYING SMOTE
from imblearn.over_sampling import SMOTE
```

```
smt = SMOTE()
```

```
X_train_smote, y_train_smote = smt.fit_sample(X_train, y_train)
```

Now we try applying models and train on this upsampled data using SMOTE and test on the normal test data that was created while splitting

There are 2 models that are giving us good results

1)Logistic regression model

We do grid search to optimize the parameters and then fit the best model

Below is the model that we get after doing Grid search

```
#Trying grid search
from sklearn.model_selection import GridSearchCV
from sklearn.linear_model import LogisticRegression
logistic_gridsearch = LogisticRegression()
penalty = ['l1', 'l2']
C = [0.0001, 0.001, 0.01, 1, 100]
hyperparameters = dict(C=C, penalty=penalty)
best_clf = GridSearchCV(logistic_gridsearch, hyperparameters, cv =5 , verbose=0)
best_model = best_clf.fit(X_train_smote,y_train_smote)
print('Best Parameters',best_clf.best_params_)
```

```
predicted_logistic_best_model = best_model.predict(X_test)
```

```
print(classification_report(y_test,predicted_logistic_best_model))
print(confusion_matrix(y_test,predicted_logistic_best_model))
confusion_array99 = confusion_matrix(y_test,predicted_logistic_best_model)

balance_error_rate99 = (confusion_array99[0][1]/sum(confusion_array99[0])) + ((confusion_array99[1][0])

print('balance error rate',balance_error_rate99/2)
```

```
              precision    recall  f1-score   support

          -1       0.92      0.78      0.84       569
           1       0.53      0.80      0.64       181

   micro avg       0.78      0.78      0.78       750
   macro avg       0.73      0.79      0.74       750
weighted avg       0.83      0.78      0.79       750

[[441 128]
 [ 37 144]]
balance error rate 0.2146879763858276
```

Above picture shows us the balanced error on the training set.

2)Random forest with grid search

```
In [84]: rfc_smote_grid = RandomForestClassifier()
         param_grid_rfc = {
             'n_estimators': [1,100,200,500],
             'max_features': ['auto', 'sqrt', 'log2'],
             'max_depth' : [4,5,6,7],
             'criterion' :['gini', 'entropy']
         }
         CV_rfc = GridSearchCV(estimator=rfc_smote_grid, param_grid=param_grid_rfc, cv= 5)
         CV_rfc.fit(X_train_smote,y_train_smote)

Out[84]: GridSearchCV(cv=5, error_score='raise-deprecating',
                estimator=RandomForestClassifier(bootstrap=True, class_weight=None, criterion='gini',
                     max_depth=None, max_features='auto', max_leaf_nodes=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='warn', n_jobs=None,
                     oob_score=False, random_state=None, verbose=0,
                     warm_start=False),
                fit_params=None, iid='warn', n_jobs=None,
                param_grid={'n_estimators': [1, 100, 200, 500], 'max_features': ['auto', 'sqrt', 'log2'], 'max_
         depth': [4, 5, 6, 7], 'criterion': ['gini', 'entropy']},
                pre_dispatch='2*n_jobs', refit=True, return_train_score='warn',
                scoring=None, verbose=0)
```

And then use the random forest with best parameters to predict

Below is the results for random forest

```
In [87]: grid_predictions_rfc = CV_rfc.predict(X_test)

In [88]: print(confusion_matrix(y_test,grid_predictions_rfc))
         print(classification_report(y_test,grid_predictions_rfc))

[[434 135]
 [ 43 138]]
              precision    recall  f1-score   support

          -1       0.91      0.76      0.83       569
           1       0.51      0.76      0.61       181

   micro avg       0.76      0.76      0.76       750
   macro avg       0.71      0.76      0.72       750
weighted avg       0.81      0.76      0.78       750
```

FINAL Prediction

Now we will test our models on the test data provided sir and predict using the bets models(There were missing columns in the test so I just added 2 rows of the columns that were missing so that I can run my models on the testing data perfectly

```
[152]: Predited_values_final_rf  = CV_rfc.predict(Test_final_GivenbySir)

[156]: pd.value_counts(pd.Series(Predited_values_final_rf))

:[156]: -1    1128
         1     521
        dtype: int64

[157]: Predited_values_final_logistics = best_model.predict(Test_final_GivenbySir)

[158]: pd.value_counts(pd.Series(Predited_values_final_logistics))

:[158]: -1    1089
         1     560
        dtype: int64

ı [ ]: # import pandas as pd

       ## convert your array into a dataframe
       #df99 = pd.DataFrame (array)

       ## save to xlsx file

       #filepath = 'my_excel_file.xlsx'

       #df.to_excel(filepath, index=False
```