**House Loan Data Analysis**

DESCRIPTION

For safe and secure lending experience, it's important to analyze the past data. In this project, you have to build a deep learning model to predict the chance of default for future loans using the historical data. As you will see, this dataset is highly imbalanced and includes a lot of features that make this problem more challenging.  
  
**Objective:**Create a model that predicts whether or not an applicant will be able to repay a loan using historical data.

**Domain:**Finance

**Analysis to be done:**Perform data preprocessing and build a deep learning prediction model.

**Steps to be done:**

⦁    Load the dataset that is given to you  
⦁    Check for null values in the dataset  
⦁    Print percentage of default to payer of the dataset for the TARGET column  
⦁    Balance the dataset if the data is imbalanced  
⦁    Plot the balanced data or imbalanced data  
⦁    Encode the columns that is required for the model  
⦁    Calculate Sensitivity as a metrice  
⦁    Calculate area under receiver operating characteristics curve

Below is the code, along with output

|  |
| --- |
| #Import Librarires |
| import pandas as pd |
| import numpy as np |
| import matplotlib.pyplot as plt |
| import os |
| import warnings |
| import sklearn |
| import seaborn as sns |
| from sklearn import preprocessing |
| from sklearn.preprocessing import OneHotEncoder |
| from sklearn.datasets import make\_blobs |
| from sklearn.impute import SimpleImputer |
| from sklearn.pipeline import Pipeline |
| from sklearn.compose import ColumnTransformer |
| from sklearn.preprocessing import StandardScaler |
| from sklearn.svm import LinearSVC |
| from sklearn.metrics import roc\_auc\_score |
| from sklearn.linear\_model import LogisticRegression |
| from sklearn.metrics import roc\_auc\_score |
| from sklearn.calibration import CalibratedClassifierCV |
| from sklearn.metrics import confusion\_matrix |
| from sklearn.ensemble import RandomForestClassifier |
| from sklearn.metrics import accuracy\_score |
| from sklearn.linear\_model import SGDClassifier |
| import plotly.offline as py |
| import plotly.graph\_objs as go |
| from plotly.offline import init\_notebook\_mode, iplot |
| from sklearn.model\_selection import train\_test\_split |
| init\_notebook\_mode(connected=True) |
| #import cufflinks as cf |
| #cf.go\_offline() |
| import pickle |
| import gc |
| #import lightgbm as lgb |
| warnings.filterwarnings('ignore') |
| %matplotlib inline |

|  |
| --- |
| #1. Load the dataset that is given to you |
| house\_loan=pd.read\_csv('loan\_data.csv') |
| house\_loan.describe() |

Graphical user interface, text

Description automatically generated with medium confidence

Table

Description automatically generated

|  |
| --- |
| house\_loan1=pd.read\_csv('loan\_data.csv') |
| house\_loan1.describe() |

Graphical user interface, text

Description automatically generated with medium confidence

Table

Description automatically generated

|  |
| --- |
| #2.Check for null values in the dataset |
| house\_loan.columns |
| house\_loan.info() |
| house\_loan.isnull().sum() |
| house\_loan.head() |

Table

Description automatically generated

Table

Description automatically generated

|  |
| --- |
| #3.Print percentage of default to payer of the dataset for the TARGET column |
| defaulters=(house\_loan.TARGET==1).sum() |
| payers=(house\_loan.TARGET==0).sum() |
| print((defaulters/payers)\*100) |
| **#Output:** 8.781828601345662 |

|  |
| --- |
| #4.Balance the dataset if the data is imbalanced |
| without\_id=[column for column in house\_loan.columns if column!='SK\_ID\_CURR'] |
|  |
| #check for duplicate values |
| na=house\_loan[house\_loan.duplicated(subset=without\_id,keep=False)] |
| print("Duplicates are: ",na.shape[0]) |
| **#Output:** Duplicates are: 0 |

|  |
| --- |
| house\_loan.TARGET.value\_counts().plot(kind='pie'  autopct='%1.1f%%') |

Chart, pie chart

Description automatically generated

|  |
| --- |
| import matplotlib as plt |
| shuffled\_data=house\_loan.sample(frac=1,random\_state=3) |
| unpaid\_home\_loan=shuffled\_data.loc[shuffled\_data['TARGET']==1] |
| paid\_home\_loan=shuffled\_data.loc[shuffled\_data['TARGET']==0].sample(n=24825,random\_state=69) |
| normalised\_home\_loan=pd.concat([unpaid\_home\_loan,paid\_home\_loan]) |
| normalised\_home\_loan.TARGET.value\_counts().plot(kind='pie',autopct="%1.1f%%") |

Chart, pie chart

Description automatically generated

|  |
| --- |
| import tensorflow as tf |
| normalised\_home\_loan.info() |
| **#Output:** <class 'pandas.core.frame.DataFrame'>  Int64Index: 49650 entries, 207339 to 121862  Columns: 122 entries, SK\_ID\_CURR to AMT\_REQ\_CREDIT\_BUREAU\_YEAR  dtypes: float64(65), int64(41), object(16)  memory usage: 46.6+ MB |
| **normalised\_home\_loan.head** |
|  |
| normalised\_home\_loan.dropna(axis=0)  normalised\_home\_loan.info() |
| **Output:**  <class 'pandas.core.frame.DataFrame'>  Int64Index: 49650 entries, 207339 to 121862  Columns: 122 entries, SK\_ID\_CURR to AMT\_REQ\_CREDIT\_BUREAU\_YEAR  dtypes: float64(65), int64(41), object(16)  memory usage: 46.6+ MB |
| **normalised\_home\_loan.isnull().sum()** |
| **Output:** |
| #print(normalised\_home\_loan.apply())  print(pd.unique(normalised\_home\_loan.AMT\_REQ\_CREDIT\_BUREAU\_DAY))  print(pd.unique(normalised\_home\_loan.AMT\_REQ\_CREDIT\_BUREAU\_WEEK))  print(pd.unique(normalised\_home\_loan.AMT\_REQ\_CREDIT\_BUREAU\_MON))  print(pd.unique(normalised\_home\_loan.AMT\_REQ\_CREDIT\_BUREAU\_QRT))  print(pd.unique(normalised\_home\_loan.AMT\_REQ\_CREDIT\_BUREAU\_YEAR)) |
| **Output:**  [ 0. nan 1. 2. 4. 3. 9.]  [ 0. nan 1. 2. 4. 3. 5. 6.]  [ 0. nan 1. 3. 5. 9. 2. 6. 8. 4. 11. 12. 7. 13. 10. 17. 15. 14.  16. 18. 27.]  [ 0. nan 2. 3. 1. 4. 5. 6. 19. 7.]  [ 3. 0. nan 1. 5. 4. 2. 6. 7. 8. 9. 10. 14. 13. 12. 11. 22. 16.  23. 17.] |
| **normalised\_home\_loan.dropna(axis=0)** |
|  |
| print(normalised\_home\_loan.info())  print(normalised\_home\_loan.isnull().sum()) |
| **#Output:**  <class 'pandas.core.frame.DataFrame'>  Int64Index: 49650 entries, 207339 to 121862  Columns: 122 entries, SK\_ID\_CURR to AMT\_REQ\_CREDIT\_BUREAU\_YEAR  dtypes: float64(65), int64(41), object(16)  memory usage: 46.6+ MB  None  SK\_ID\_CURR 0  TARGET 0  NAME\_CONTRACT\_TYPE 0  CODE\_GENDER 0  FLAG\_OWN\_CAR 0  ...  AMT\_REQ\_CREDIT\_BUREAU\_DAY 7648  AMT\_REQ\_CREDIT\_BUREAU\_WEEK 7648  AMT\_REQ\_CREDIT\_BUREAU\_MON 7648  AMT\_REQ\_CREDIT\_BUREAU\_QRT 7648  AMT\_REQ\_CREDIT\_BUREAU\_YEAR 7648  Length: 122, dtype: int64 |
| **normalised\_home\_loan.TARGET.value\_counts().plot(kind='pie',autopct="%1.1f%%")** |
|  |
| normalised\_home\_loan.NAME\_CONTRACT\_TYPE.value\_counts().plot(kind='pie',autopct="%1.1f%%") |
|  |
| normalised\_home\_loan.CODE\_GENDER.value\_counts().plot(kind='pie',autopct="%1.1f%%") |
|  |
| normalised\_home\_loan.FLAG\_OWN\_CAR.value\_counts().plot(kind='pie',autopct="%1.1f%%") |
|  |
| normalised\_home\_loan.CNT\_CHILDREN.value\_counts().plot(kind='pie',autopct="%1.1f%%") |
|  |
| #!pip install chart\_studio  #!pip install cufflinks  import cufflinks as cf  cf.go\_offline()  cf.set\_config\_file(theme='polar')  normalised\_home\_loan[normalised\_home\_loan['AMT\_INCOME\_TOTAL'] < 2000000]['AMT\_INCOME\_TOTAL'].iplot(kind='histogram', bins=100,  xTitle = 'Total Income', yTitle ='Count of applicants',  title='Distribution of AMT\_INCOME\_TOTAL') |
| (normalised\_home\_loan[normalised\_home\_loan['AMT\_INCOME\_TOTAL']>1000000]['TARGET'].value\_counts())/len(normalised\_home\_loan[normalised\_home\_loan['AMT\_INCOME\_TOTAL'] > 1000000])\*100 |
| **#OutPut:**  0 64.864865  1 35.135135  Name: TARGET, dtype: float64 |
| #print((normalised\_home\_loan[normalised\_home\_loan['CNT\_CHILDREN']>1]['TARGET'].value\_counts())/len(normalised\_home\_loan[normalised\_home\_loan['CNT\_CHILDREN'] > 2])\*100)  print((normalised\_home\_loan[normalised\_home\_loan['CNT\_CHILDREN']>2]['TARGET'].value\_counts())/len(normalised\_home\_loan[normalised\_home\_loan['CNT\_CHILDREN'] > 2])\*100)  print((normalised\_home\_loan[normalised\_home\_loan['CNT\_CHILDREN']>5]['TARGET'].value\_counts())/len(normalised\_home\_loan[normalised\_home\_loan['CNT\_CHILDREN'] > 5])\*100)  #as number of children is increasing lone defaulters are increasing |
| **#OutPut:**  1 57.047872  0 42.952128  Name: TARGET, dtype: float64  1 81.818182  0 18.181818  Name: TARGET, dtype: float64 |
| print((normalised\_home\_loan[normalised\_home\_loan['FLAG\_OWN\_CAR']=='N']['TARGET'].value\_counts())/len(normalised\_home\_loan[normalised\_home\_loan['FLAG\_OWN\_CAR'] =='N'])\*100)  print((normalised\_home\_loan[normalised\_home\_loan['FLAG\_OWN\_CAR']=='Y']['TARGET'].value\_counts())/len(normalised\_home\_loan[normalised\_home\_loan['FLAG\_OWN\_CAR'] =='Y'])\*100)  #people with own cars are slighlty more likely to repay back the loan |
| **#OutPut:**  1 51.350064  0 48.649936  Name: TARGET, dtype: float64  0 52.823962  1 47.176038  Name: TARGET, dtype: float64 |
| print((normalised\_home\_loan[normalised\_home\_loan['CODE\_GENDER']=='M']['TARGET'].value\_counts())/len(normalised\_home\_loan[normalised\_home\_loan['CODE\_GENDER'] =='M'])\*100)  print((normalised\_home\_loan[normalised\_home\_loan['CODE\_GENDER']=='F']['TARGET'].value\_counts())/len(normalised\_home\_loan[normalised\_home\_loan['CODE\_GENDER'] =='F'])\*100)  #men more likely to default in payment of loans |
| **#OutPut:**  1 56.280372  0 43.719628  Name: TARGET, dtype: float64  0 53.867691  1 46.132309  Name: TARGET, dtype: float64 |
| print((normalised\_home\_loan[normalised\_home\_loan['NAME\_CONTRACT\_TYPE']=='Cash loans']['TARGET'].value\_counts())/len(normalised\_home\_loan[normalised\_home\_loan['NAME\_CONTRACT\_TYPE']=='Cash loans'])\*100)  print((normalised\_home\_loan[normalised\_home\_loan['NAME\_CONTRACT\_TYPE']=='Revolving loans']['TARGET'].value\_counts())/len(normalised\_home\_loan[normalised\_home\_loan['NAME\_CONTRACT\_TYPE']=='Revolving loans'])\*100)  #cash loans have a higher percent of defaulters |
| **#OutPut:**  1 50.802923  0 49.197077  Name: TARGET, dtype: float64  0 59.309995  1 40.690005  Name: TARGET, dtype: float64 |
| normalised\_home\_loan=normalised\_home\_loan.sample(frac=1,random\_state=5)  from sklearn.preprocessing import OrdinalEncoder  ordenc=OrdinalEncoder()  normalised\_home\_loan['NAME\_CONTRACT\_TYPE\_CODE']=ordenc.fit\_transform(normalised\_home\_loan[['NAME\_CONTRACT\_TYPE']])  print(normalised\_home\_loan[['NAME\_CONTRACT\_TYPE','NAME\_CONTRACT\_TYPE\_CODE']].head(20))  print(normalised\_home\_loan['NAME\_CONTRACT\_TYPE\_CODE'].value\_counts()) |
| **#outPut:** |
| normalised\_home\_loan['CODE\_GENDER\_CODE']=ordenc.fit\_transform(normalised\_home\_loan[['CODE\_GENDER']])  print(normalised\_home\_loan[['CODE\_GENDER','CODE\_GENDER\_CODE']].head(20))  print(normalised\_home\_loan['CODE\_GENDER\_CODE'].value\_counts()) |
| **#output:** |
| **#2 other values in code\_gender**  normalised\_home\_loan.loc[normalised\_home\_loan['CODE\_GENDER\_CODE']==2] |
| **#output:** |
| normalised\_home\_loan['FLAG\_OWN\_CAR\_CODE']=ordenc.fit\_transform(normalised\_home\_loan[['FLAG\_OWN\_CAR']])  print(normalised\_home\_loan[['FLAG\_OWN\_CAR','FLAG\_OWN\_CAR\_CODE']].head(20))  print(normalised\_home\_loan['FLAG\_OWN\_CAR\_CODE'].value\_counts()) |
| **#Output:** |
| normalised\_home\_loan['CNT\_CHILDREN\_CODE']=ordenc.fit\_transform(normalised\_home\_loan[['CNT\_CHILDREN']])  print(normalised\_home\_loan[['CNT\_CHILDREN\_CODE','CNT\_CHILDREN']].head(20))  print(normalised\_home\_loan['CNT\_CHILDREN\_CODE'].value\_counts()) |
| **#output:** |
| normalised\_home\_loan=normalised\_home\_loan.sample(frac=1,random\_state=45)  normalised\_home\_loan['TARGET'].value\_counts() |
| **#output:**  0 24825  1 24825  Name: TARGET, dtype: int64 |
| #y=y.sample(frac=1,random\_state=45)  normalised\_home\_loan\_features=['SK\_ID\_CURR','NAME\_CONTRACT\_TYPE\_CODE','CNT\_CHILDREN\_CODE','FLAG\_OWN\_CAR\_CODE','CODE\_GENDER\_CODE']  from sklearn.model\_selection import train\_test\_split  X=normalised\_home\_loan[normalised\_home\_loan\_features]  #X=X.sample(frac=1,random\_state=45)  blobs\_random\_seed = 42  centers = [(0,0), (5,5)]  cluster\_std = 1  frac\_test\_split = 0.33  num\_features\_for\_samples = 2  num\_samples\_total = 49650  # Generate data  inputs, targets = make\_blobs(n\_samples = num\_samples\_total, centers = centers, n\_features = num\_features\_for\_samples, cluster\_std = cluster\_std)  X\_train,X\_test,y\_train,y\_test=train\_test\_split(inputs,targets,test\_size=0.33,random\_state=45) |
| print(X\_train.shape, X\_test.shape, y\_train.shape, y\_test.shape) |
| **#output:**  (33265, 2) (16385, 2) (33265,) (16385,) |
| plt.pyplot.scatter(X\_train[:,0], X\_train[:,1])  plt.pyplot.title('Linearly separable data')  plt.pyplot.xlabel('X1')  plt.pyplot.ylabel('X2')  plt.pyplot.show() |
|  |
| **from sklearn import svm**  **from sklearn.metrics import plot\_confusion\_matrix**  **clf=svm.SVC(kernel='linear')**  **clf=clf.fit(X\_train,y\_train)**  **predictions = clf.predict(X\_test)**  **# Generate confusion matrix**  **matrix = plot\_confusion\_matrix(clf, X\_test, y\_test,**  **cmap=plt.cm.Blues,**  **normalize='true')**  **plt.pyplot.title('Confusion matrix for our classifier')**  **plt.pyplot.show(matrix)**  **plt.pyplot.show()** |
|  |
| from sklearn.metrics import precision\_score, recall\_score,f1\_score  print(precision\_score(y\_test, predictions))  print(recall\_score(y\_test, predictions))  print(f1\_score(y\_test,predictions,average=None)) |
| **#output:**  0.9998767562238107  0.9997535428219347  [0.99981863 0.99981515] |
| support\_vectors = clf.support\_vectors\_  # Visualize support vectors  plt.pyplot.scatter(X\_train[:,0], X\_train[:,1])  plt.pyplot.scatter(support\_vectors[:,0], support\_vectors[:,1], color='red')  plt.pyplot.title('Linearly separable data with support vectors')  plt.pyplot.xlabel('X1')  plt.pyplot.ylabel('X2')  plt.pyplot.show() |
|  |
| from mlxtend.plotting import plot\_decision\_regions  plot\_decision\_regions(X\_test, y\_test, clf=clf, legend=2)  plt.pyplot.show() |
|  |
|  |