**Lending Club Loan Data Analysis**

DESCRIPTION

Create a model that predicts whether or not a loan will be default using the historical data.

**Problem Statement:**

For companies like Lending Club correctly predicting whether or not a loan will be a default is very important. In this project, using the historical data from 2007 to 2015, you have to build a deep learning model to predict the chance of default for future loans. As you will see later this dataset is highly imbalanced and includes a lot of features that make this problem more challenging.

**Domain:** Finance

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

**Content:**

Dataset columns and definition:

* **credit.policy:** 1 if the customer meets the credit underwriting criteria of LendingClub.com, and 0 otherwise.
* **purpose:** The purpose of the loan (takes values "credit\_card", "debt\_consolidation", "educational", "major\_purchase", "small\_business", and "all\_other").
* **int.rate:** The interest rate of the loan, as a proportion (a rate of 11% would be stored as 0.11). Borrowers judged by LendingClub.com to be more risky are assigned higher interest rates.
* **installment:** The monthly installments owed by the borrower if the loan is funded.
* **log.annual.inc:** The natural log of the self-reported annual income of the borrower.
* **dti:** The debt-to-income ratio of the borrower (amount of debt divided by annual income).
* **fico:** The FICO credit score of the borrower.
* **days.with.cr.line:** The number of days the borrower has had a credit line.
* **revol.bal:** The borrower's revolving balance (amount unpaid at the end of the credit card billing cycle).
* **revol.util:**The borrower's revolving line utilization rate (the amount of the credit line used relative to total credit available).
* **inq.last.6mths:** The borrower's number of inquiries by creditors in the last 6 months.
* **delinq.2yrs:** The number of times the borrower had been 30+ days past due on a payment in the past 2 years.
* **pub.rec:**The borrower's number of derogatory public records (bankruptcy filings, tax liens, or judgments).

**Steps to perform:**

Perform exploratory data analysis and feature engineering and then apply feature engineering. Follow up with a deep learning model to predict whether or not the loan will be default using the historical data.

**Tasks:**

1.     Feature Transformation

* Transform categorical values into numerical values (discrete)

2.     Exploratory data analysis of different factors of the dataset.

3.     Additional Feature Engineering

* You will check the correlation between features and will drop those features which have a strong correlation
* This will help reduce the number of features and will leave you with the most relevant features

4.     Modeling

* After applying EDA and feature engineering, you are now ready to build the predictive models

**Code:**

|  |
| --- |
| # import libraries |
| import os |
| import pandas as pd |
| import numpy as np |
| import matplotlib.pyplot as plt |
| import seaborn as sns |
| from sklearn.model\_selection import train\_test\_split |
| from sklearn.preprocessing import MinMaxScaler |
| from tensorflow.keras.models import Sequential |
| from tensorflow.keras.layers import Dense,Dropout |
| from tensorflow.keras.callbacks import EarlyStopping |
| from tensorflow.keras.models import load\_model |
| from sklearn.metrics import confusion\_matrix, classification\_report |
| from pickle import dump, load |
|  |
| %matplotlib inline |

|  |
| --- |
| loan\_data\_Lendingclub = pd.read\_csv('loan\_data\_Copy.csv') |
| loan\_data\_Lendingclub.describe() |
| **#Output:** |
| loan\_data\_Lendingclub.shape |
| **#Output:**  (9578, 14) |
| loan\_data\_Lendingclub.head(10) |
| **#Output:** |
| loan\_data\_Lendingclub.dtypes |
| **@Output:**  credit.policy int64  purpose object  int.rate float64  installment float64  log.annual.inc float64  dti float64  fico int64  days.with.cr.line float64  revol.bal int64  revol.util float64  inq.last.6mths int64  delinq.2yrs int64  pub.rec int64  not.fully.paid int64  dtype: object |
| #Transform categorical values into numerical values  obj\_loan = loan\_data\_Lendingclub.select\_dtypes(include=['object']).copy()  obj\_loan.head() |
|  |
| obj\_loan["purpose"].value\_counts() |
| **#Output:** |
| obj\_loan = obj\_loan.fillna({"purpose" : "credit\_card"})  cleanup\_nums = {"purpose": {"credit\_card": 1,"debt\_consolidation": 2 }}  obj\_loan=obj\_loan.replace(cleanup\_nums)  obj\_loan.head() |
|  |
| plt.figure(figsize=(10,6))  loan\_data\_Lendingclub[loan\_data\_Lendingclub['credit.policy']==1]['fico'].hist(alpha=0.5,color='blue',bins=30,label='Credit.Policy=1')  loan\_data\_Lendingclub[loan\_data\_Lendingclub['credit.policy']==0]['fico'].hist(alpha=0.5,color='red',bins=30,label='Credit.Policy=0')  plt.legend()  plt.xlabel('FICO') |
|  |
| plt.figure(figsize=(10,6))  loan\_data\_Lendingclub[loan\_data\_Lendingclub['not.fully.paid']==1]['fico'].hist(alpha=0.5,color='blue',bins=30,label='not.fully.paid=1')  loan\_data\_Lendingclub[loan\_data\_Lendingclub['not.fully.paid']==0]['fico'].hist(alpha=0.5,color='red',bins=30,label='not.fully.paid=0')  plt.legend()  plt.xlabel('FICO') |
|  |
| plt.figure(figsize=(11,7))  sns.countplot(x='purpose',hue='not.fully.paid',data=loan\_data\_Lendingclub,palette='Set1') |
|  |
| sns.jointplot(x='fico',y='int.rate',data=loan\_data\_Lendingclub,color='purple') |
|  |
| plt.figure(figsize=(11,7))  sns.lmplot(y='int.rate',x='fico',data=loan\_data\_Lendingclub,hue='credit.policy',  col='not.fully.paid',palette='Set1') |
|  |
| loan\_num = loan\_data\_Lendingclub.select\_dtypes(include = ['float64','int64'])  loan\_num.head() |
|  |
| #correlation  cor\_matrix = loan\_data\_Lendingclub.corr().abs()  print(cor\_matrix) |
|  |
| upper\_tri = cor\_matrix.where(np.triu(np.ones(cor\_matrix.shape),k=1).astype(np.bool))  print(upper\_tri) |
|  |
| final\_data.corr()  plt.figure(  figsize=[16,12]  )  sns.heatmap(  data=final\_data.corr(),  cmap='viridis',  annot=False,  fmt='.2g'  ) |
|  |
| loan\_data\_Lendingclub.describe().transpose() |
|  |
| loan\_data\_Lendingclub['not.fully.paid'].isnull().mean()  loan\_data\_Lendingclub.groupby('not.fully.paid')['not.fully.paid'].count()/len(loan\_data\_Lendingclub) |
|  |
| count\_class\_0, count\_class\_1 = loan\_data\_Lendingclub['not.fully.paid'].value\_counts()  loan\_0 = loan\_data\_Lendingclub[loan\_data\_Lendingclub['not.fully.paid'] == 0]  loan\_1 = loan\_data\_Lendingclub[loan\_data\_Lendingclub['not.fully.paid'] == 1]  loan\_1\_over = loan\_1.sample(count\_class\_0, replace=True)  loan\_test\_over = pd.concat([loan\_0, loan\_1\_over], axis=0)  print('Random over-sampling:')  print(loan\_test\_over['not.fully.paid'].value\_counts())  sns.set\_style('darkgrid')  sns.countplot(x='not.fully.paid', data=loan\_test\_over) |
| Random over-sampling:  0 8045  1 8045  Name: not.fully.paid, dtype: int64 |
| col\_fea = ['purpose']  final\_data = pd.get\_dummies(loan\_test\_over,columns=col\_fea,drop\_first=True)  final\_data.info() |
|  |
| to\_train = final\_data[final\_data['not.fully.paid'].isin([0,1])]  to\_pred = final\_data[final\_data['not.fully.paid'] == 2]  X = to\_train.drop('not.fully.paid', axis=1).values  y = to\_train['not.fully.paid'].values  X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.3, random\_state = 101)  scaler = MinMaxScaler()  X\_train = scaler.fit\_transform(X\_train)  X\_test = scaler.transform(X\_test)  model = Sequential()  model.add(  Dense(19, activation='relu')  )  model.add(  Dense(10, activation='relu')  )  model.add(  Dense(5, activation='relu')  )  model.add(  Dense(1, activation='sigmoid')  )  model.compile(  optimizer='adam',  loss='binary\_crossentropy',  metrics=['accuracy']  )  early\_stop = EarlyStopping(  monitor='val\_loss',  mode='min',  verbose=1,  patience=25  )  model.fit(  X\_train,  y\_train,  epochs=200,  batch\_size=256,  validation\_data=(X\_test, y\_test),  callbacks=[early\_stop]  ) |
|  |
| pd.DataFrame(model.history.history)[['loss','val\_loss']].plot() |
|  |
| #predictions = model.predict\_classes(X\_test)  predict\_x=model.predict(X\_test)  predictions=np.argmax(predict\_x,axis=1)  print(  confusion\_matrix(y\_test,predictions),  '\n',  classification\_report(y\_test,predictions)  ) |
|  |
| model\_new = Sequential()  model\_new.add(  Dense(19, activation='relu')  )  model\_new.add(Dropout(0.2))  model\_new.add(  Dense(10, activation='relu')  )  model\_new.add(Dropout(0.2))  model\_new.add(  Dense(5, activation='relu')  )  model\_new.add(Dropout(0.2))  model\_new.add(  Dense(1, activation='sigmoid')  )  model\_new.compile(  optimizer='adam',  loss='binary\_crossentropy',  metrics=['binary\_accuracy']  )  model\_new.fit(  X\_train,  y\_train,  epochs=200,  batch\_size=256,  validation\_data=(X\_test, y\_test),  callbacks=[early\_stop]  ) |
|  |
| pd.DataFrame(model\_new.history.history)[['loss','val\_loss']].plot() |
|  |
| #predictions\_new = (model\_new.predict\_proba(X\_test) >= 0.2).astype('int')  predict\_prob=model.predict([X\_test])  predictions\_new=np.argmax(predict\_prob,axis=1)  print(  confusion\_matrix(y\_test,predictions\_new),  '\n',  classification\_report(y\_test,predictions\_new)  ) |
|  |
|  |