CreditRisk

March 17, 2022

[1]:
```
import numpy as np
import pandas as pd
from sklearn.model_selection import train_test_split
import tensorflow as tf
from tensorflow import keras
from tensorflow.keras import layers
from sklearn.preprocessing import MinMaxScaler
from matplotlib import pyplot as plt
from scipy import linalg
import matplotlib.pyplot as plt
from scipy import stats
import seaborn as sns
```

[2]:
```
Df = pd.read_csv("cs-training.csv", index_col = 0)
print(len(Df))
Df = Df.dropna()
print(len(Df))
X_Cols = list(Df.columns)
X_Cols.remove("SeriousDlqin2yrs")
X = Df[X_Cols].to_numpy()
Y = Df["SeriousDlqin2yrs"].to_numpy()
```

```
150000
120269
```

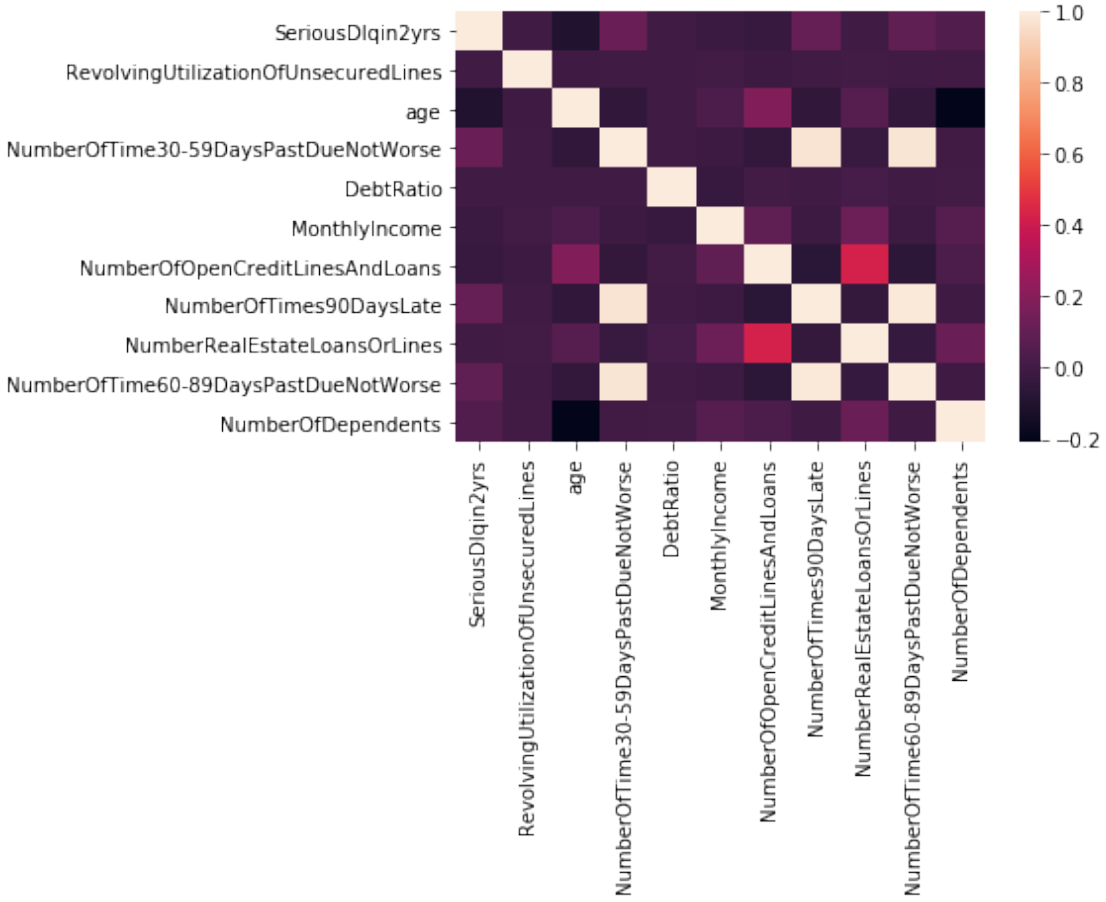
[3]:
```
X_Train, X_Test, Y_Train, Y_Test = train_test_split(X, Y, test_size = 0.33,␣
↪random_state = 42)
```

[4]:
```
len(X_Train)
```

[4]:
```
80580
```

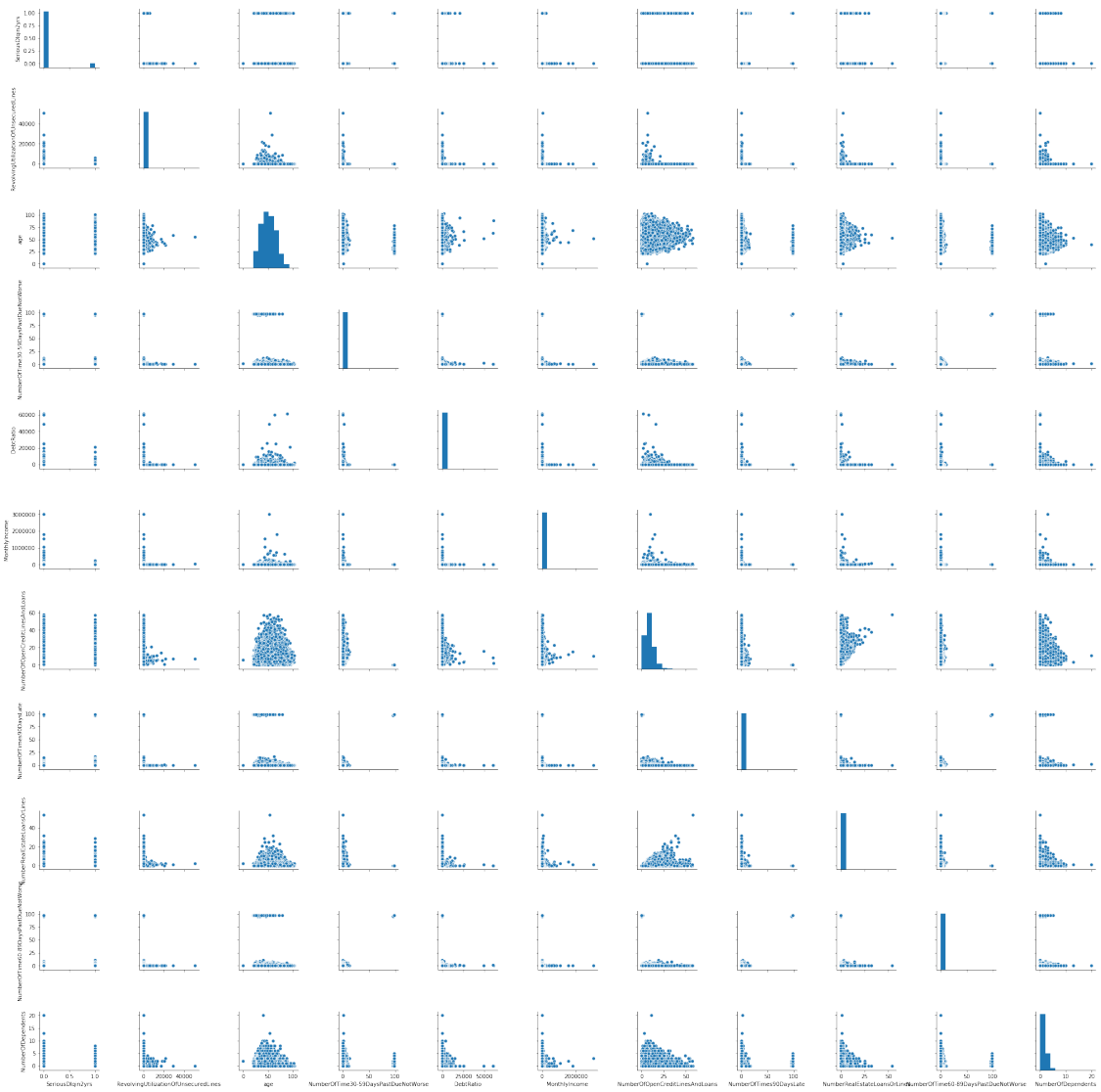
1 Premilinaries

[33]:
```
# Correlation Matrix
Corr_Matrix = Df.corr()
round(Corr_Matrix, 2)
Fig = sns.heatmap(Corr_Matrix)
Figure = Fig.get_figure()
Figure.savefig('CorrPlot.pdf', bbox_inches = "tight")
```



[31]:
```
### Two by Two Plots
Plt = sns.pairplot(Df)
Plt
```

[31]: <seaborn.axisgrid.PairGrid at 0x7fa64d7b6650>



2 Fisher LDA

[6]:
```
mu = np.mean(X_Train)
X_Train_Demeaned = (X_Train - mu).T
X_Test_Demeaned = (X_Test - mu).T
S_t = np.cov(X_Train_Demeaned)
S_w = np.zeros(S_t.shape)
for c in np.unique(Y_Train):
    S_w += np.cov(X_Train_Demeaned[:, Y_Train == c])

S_b = S_t - S_w
```

```
Vals, Vecs = linalg.eig(np.linalg.inv(S_w)@S_b)
Vecs = Vecs[:, np.argsort(Vals)]

W_lda = Vecs[:, -1:].real

X_Train_Lda = (W_lda.T@X_Train_Demeaned).T
X_Test_Lda = (W_lda.T@X_Test_Demeaned).T


print(np.mean(X_Train_Lda[Y_Train == 0]), np.mean(X_Train_Lda[Y_Train == 1]))
print(np.mean(X_Test_Lda[Y_Test == 0]), np.mean(X_Test_Lda[Y_Test == 1]))
```

```
85.44373421735314 83.43138114565562
85.49241455138895 83.98570707596363
```

[7]:
```
Y_Train
```

[7]:
```
array([0, 0, 0, …, 0, 1, 0])
```

3 Logistic Regression

[8]:
```
from sklearn.linear_model import LogisticRegression
Clf = LogisticRegression(random_state = 0, max_iter = 1000).fit(X_Train,␣
 ↪Y_Train)
Probs = Clf.predict_proba(X_Train)

Train_Predicted = np.zeros(len(Probs))
for i in range(len(Probs)):
    if (Probs[i, 0] > Probs[i, 1]):
        Train_Predicted[i] = 0
    else:
        Train_Predicted[i] = 1

Equal = 0
for i in range(len(Y_Train)):
    if (Y_Train[i] == Train_Predicted[i]):
        Equal = Equal + 1

Score = Probs[:, 1]/(1 - Probs[:, 1])
print("Scores Are:")
print(np.mean(Score[Y_Train == 0]), np.mean(Score[Y_Train == 1]))
```

```
Scores Are:
0.09199179721400955 0.3332044933838555
```

[9]:
```
print("Classification Accuracy on Training Set is:")
print(Equal/len(Y_Train))
```

```
Classification Accuracy on Training Set is:
0.9315711094564408
```

[10]:
```
Probs = Clf.predict_proba(X_Test)
Test_Predicted = np.zeros(len(Probs))

for i in range(len(Probs)):
    if (Probs[i, 0] > Probs[i, 1]):
        Test_Predicted[i] = 0
    else:
        Test_Predicted[i] = 1


Equal = 0
for i in range(len(Y_Test)):
    if (Y_Test[i] == Test_Predicted[i]):
        Equal = Equal + 1
```

[11]:
```
print("Classification Accuracy on Test Set is:")
Equal/len(Y_Test)
```

```
Classification Accuracy on Test Set is:
```

[11]:
```
0.9304845171206128
```

4 Random Forest

[12]:
```
from sklearn.ensemble import RandomForestClassifier
Clf = RandomForestClassifier(max_depth = 5, random_state = 0)
Clf.fit(X_Train, Y_Train)
Probs = Clf.predict_proba(X_Train)

Train_Predicted = np.zeros(len(Probs))
for i in range(len(Probs)):
    if (Probs[i, 0] > Probs[i, 1]):
        Train_Predicted[i] = 0
    else:
        Train_Predicted[i] = 1

Equal = 0
for i in range(len(Y_Train)):
    if (Y_Train[i] == Train_Predicted[i]):
        Equal = Equal + 1
```

```
Score = Probs[:, 1]/(1 - Probs[:, 1])
print("Scores Are:")
print(np.mean(Score[Y_Train == 0]), np.mean(Score[Y_Train == 1]))
```

```
Scores Are:
0.07023938518560487 0.39612069415283807
```

[13]:
```
print("Classification Accuracy on Training Set is:")
print(Equal/len(Y_Train))
```

```
Classification Accuracy on Training Set is:
0.9345619260362373
```

[14]:
```
Probs = Clf.predict_proba(X_Test)

Test_Predicted = np.zeros(len(Probs))

for i in range(len(Probs)):
    if (Probs[i, 0] > Probs[i, 1]):
        Test_Predicted[i] = 0
    else:
        Test_Predicted[i] = 1

Equal = 0
for i in range(len(Y_Test)):
    if (Y_Test[i] == Test_Predicted[i]):
        Equal = Equal + 1

Score = Probs[:, 1]/(1 - Probs[:, 1])
print("Scores Are:")
print(np.mean(Score[Y_Test == 0]), np.mean(Score[Y_Test == 1]))
```

```
Scores Are:
0.07135186453471327 0.3791356430665598
```

[15]:
```
print("Classification Accuracy on Test Set is:")
Equal/len(Y_Test)
```

```
Classification Accuracy on Test Set is:
```

[15]:
```
0.9323994053768047
```

[16]:
```
X_Train.shape
```

[16]:
```
(80580, 10)
```

[17]:
```
np.sum(Y_Train)
```

[17]: 5575

[18]:
```
Y_Train.shape
X_Train.shape
# len(Y_Train)
```

[18]: (80580, 10)

[19]:
```
Default = []
for i in range(len(X_Train)):
    if(Y_Train[i] == 1):
        Default.append(X_Train[i])
Default = np.asarray(Default)

Resampled = []
Resampled_y = []
Target = X_Train.shape[0]/2
Current = np.sum(Y_Train)

while(Current <= Target):
    Index = np.random.choice(Default.shape[0], 1)
    Resampled.append(Default[Index].squeeze())
    Resampled_y.append(1)
    Current += 1
    Target += .498

X_Train = np.concatenate((X_Train, np.asarray(Resampled)))
Y_Train = np.concatenate((Y_Train, np.asarray(Resampled_y)))
```

5 Dense NN

[25]:
```
# Class_Weight = {1: 0.95,0: 0.05}
import random
random.seed(1)
Model_in = keras.Input(shape = (10, ))
X = layers.Dense(10, activation = "relu")(Model_in)
X2 = layers.Dense(10, activation= "relu")(X)
X3 = layers.Dense(10, activation= "relu")(X2)
X4 = layers.Dense(10, activation= "relu")(X3)

Out = layers.Dense(1, activation= "sigmoid")(X4)

Model = keras.Model(Model_in, Out)
Model.compile(optimizer = 'adam', loss = 'binary_crossentropy')
Model.fit(X_Train, Y_Train, epochs = 100,
```

```
                batch_size = 128,
                shuffle = True)
```

```
Epoch 1/100
1170/1170 [==============================] - 2s 979us/step - loss: 3.9219
Epoch 2/100
1170/1170 [==============================] - 1s 1ms/step - loss: 1.4448
Epoch 3/100
1170/1170 [==============================] - 1s 972us/step - loss: 1.0019
Epoch 4/100
1170/1170 [==============================] - 1s 1ms/step - loss: 1.7301
Epoch 5/100
1170/1170 [==============================] - 1s 1ms/step - loss: 0.8861
Epoch 6/100
1170/1170 [==============================] - 1s 1ms/step - loss: 1.0590
Epoch 7/100
1170/1170 [==============================] - 1s 1ms/step - loss: 0.7866
Epoch 8/100
1170/1170 [==============================] - 1s 995us/step - loss: 1.1092
Epoch 9/100
1170/1170 [==============================] - 1s 1ms/step - loss: 1.1982
Epoch 10/100
1170/1170 [==============================] - 1s 1ms/step - loss: 0.6915
Epoch 11/100
1170/1170 [==============================] - 2s 1ms/step - loss: 0.7262
Epoch 12/100
1170/1170 [==============================] - 1s 978us/step - loss: 0.9853 0s -
los - ETA: 0s - loss:
Epoch 13/100
1170/1170 [==============================] - 1s 1ms/step - loss: 0.7690A: 0s - l
- ETA: 0s - loss: 0.78
Epoch 14/100
1170/1170 [==============================] - 1s 1ms/step - loss: 0.7644
Epoch 15/100
1170/1170 [==============================] - 1s 964us/step - loss: 0.5863
Epoch 16/100
1170/1170 [==============================] - 1s 1ms/step - loss: 0.6968
Epoch 17/100
1170/1170 [==============================] - 1s 1ms/step - loss: 0.5938
Epoch 18/100
1170/1170 [==============================] - ETA: 0s - loss: 0.633 - ETA: 0s -
loss: 0.632 - 1s 1ms/step - loss: 0.6293
Epoch 19/100
1170/1170 [==============================] - 1s 978us/step - loss: 0.6597
Epoch 20/100
1170/1170 [==============================] - 1s 977us/step - loss: 0.6044
Epoch 21/100
```

```
1170/1170 [==============================] - 1s 985us/step - loss: 0.6199
Epoch 22/100
1170/1170 [==============================] - 1s 984us/step - loss: 0.5601
Epoch 23/100
1170/1170 [==============================] - 1s 967us/step - loss: 0.5583
Epoch 24/100
1170/1170 [==============================] - 1s 980us/step - loss: 0.5560 0
Epoch 25/100
1170/1170 [==============================] - 1s 967us/step - loss: 0.5542
Epoch 26/100
1170/1170 [==============================] - 1s 966us/step - loss: 0.5811
Epoch 27/100
1170/1170 [==============================] - 1s 990us/step - loss: 0.5744
Epoch 28/100
1170/1170 [==============================] - 1s 983us/step - loss: 0.5443
Epoch 29/100
1170/1170 [==============================] - 1s 969us/step - loss: 0.5434
Epoch 30/100
1170/1170 [==============================] - 1s 990us/step - loss: 0.5419
Epoch 31/100
1170/1170 [==============================] - 1s 1ms/step - loss: 0.5413
Epoch 32/100
1170/1170 [==============================] - 1s 1ms/step - loss: 0.5366
Epoch 33/100
1170/1170 [==============================] - 1s 1ms/step - loss: 0.5397A: 0s -
Epoch 34/100
1170/1170 [==============================] - 1s 992us/step - loss: 0.5358
Epoch 35/100
1170/1170 [==============================] - 1s 1ms/step - loss: 0.5361
Epoch 36/100
1170/1170 [==============================] - 1s 1ms/step - loss: 0.5324
Epoch 37/100
1170/1170 [==============================] - 1s 1ms/step - loss: 0.5325
Epoch 38/100
1170/1170 [==============================] - 1s 1ms/step - loss: 0.5329
Epoch 39/100
1170/1170 [==============================] - 1s 1ms/step - loss: 0.5282
Epoch 40/100
1170/1170 [==============================] - 1s 1ms/step - loss: 0.5263
Epoch 41/100
1170/1170 [==============================] - 1s 1ms/step - loss: 0.5234
Epoch 42/100
1170/1170 [==============================] - 1s 1ms/step - loss: 0.5217
Epoch 43/100
1170/1170 [==============================] - 1s 1ms/step - loss: 0.5211
Epoch 44/100
1170/1170 [==============================] - 1s 989us/step - loss: 0.5212
Epoch 45/100
```

```
1170/1170 [==============================] - 1s 1ms/step - loss: 0.5218
Epoch 46/100
1170/1170 [==============================] - 1s 1ms/step - loss: 0.5160A: 0s -
Epoch 47/100
1170/1170 [==============================] - 1s 1ms/step - loss: 0.5162
Epoch 48/100
1170/1170 [==============================] - 1s 983us/step - loss: 0.5155
Epoch 49/100
1170/1170 [==============================] - 1s 1ms/step - loss: 0.5121
Epoch 50/100
1170/1170 [==============================] - 1s 1ms/step - loss: 0.5172
Epoch 51/100
1170/1170 [==============================] - 1s 987us/step - loss: 0.5089
Epoch 52/100
1170/1170 [==============================] - 1s 1ms/step - loss: 0.5081
Epoch 53/100
1170/1170 [==============================] - 1s 1ms/step - loss: 0.5049
Epoch 54/100
1170/1170 [==============================] - 1s 1ms/step - loss: 0.5069
Epoch 55/100
1170/1170 [==============================] - 1s 998us/step - loss: 0.5009
Epoch 56/100
1170/1170 [==============================] - 1s 1ms/step - loss: 0.5033
Epoch 57/100
1170/1170 [==============================] - 1s 1ms/step - loss: 0.5051
Epoch 58/100
1170/1170 [==============================] - 1s 1ms/step - loss: 0.5086
Epoch 59/100
1170/1170 [==============================] - 1s 989us/step - loss: 0.5024
Epoch 60/100
1170/1170 [==============================] - 1s 1ms/step - loss: 0.5014
Epoch 61/100
1170/1170 [==============================] - 1s 996us/step - loss: 0.5026
Epoch 62/100
1170/1170 [==============================] - 1s 1ms/step - loss: 0.5017
Epoch 63/100
1170/1170 [==============================] - 1s 1ms/step - loss: 0.5011
Epoch 64/100
1170/1170 [==============================] - 1s 1ms/step - loss: 0.5028
Epoch 65/100
1170/1170 [==============================] - 1s 1ms/step - loss: 0.4972
Epoch 66/100
1170/1170 [==============================] - 1s 1ms/step - loss: 0.5058
Epoch 67/100
1170/1170 [==============================] - 1s 1ms/step - loss: 0.5053
Epoch 68/100
1170/1170 [==============================] - 1s 973us/step - loss: 0.4977
Epoch 69/100
```

```
1170/1170 [==============================] - 1s 1ms/step - loss: 0.4984A: 0s -
loss: 0. - ETA: 0s
Epoch 70/100
1170/1170 [==============================] - 1s 995us/step - loss: 0.5009
Epoch 71/100
1170/1170 [==============================] - 1s 1ms/step - loss: 0.4963
Epoch 72/100
1170/1170 [==============================] - 1s 975us/step - loss: 0.4955
Epoch 73/100
1170/1170 [==============================] - 1s 962us/step - loss: 0.4978
Epoch 74/100
1170/1170 [==============================] - 1s 968us/step - loss: 0.4998
Epoch 75/100
1170/1170 [==============================] - 1s 990us/step - loss: 0.4948
Epoch 76/100
1170/1170 [==============================] - 1s 989us/step - loss: 0.6743
Epoch 77/100
1170/1170 [==============================] - 1s 978us/step - loss: 0.6691
Epoch 78/100
1170/1170 [==============================] - 1s 963us/step - loss: 0.5573
Epoch 79/100
1170/1170 [==============================] - 1s 1ms/step - loss: 0.5386
Epoch 80/100
1170/1170 [==============================] - 1s 989us/step - loss: 0.5334
Epoch 81/100
1170/1170 [==============================] - 1s 1ms/step - loss: 0.4990
Epoch 82/100
1170/1170 [==============================] - 1s 1ms/step - loss: 0.5009
Epoch 83/100
1170/1170 [==============================] - 2s 1ms/step - loss: 0.5015A: 1s -
loss: 0.50 - ETA: 1s - loss:  - ETA: 1
Epoch 84/100
1170/1170 [==============================] - 1s 1ms/step - loss: 0.5029A: 0s -
loss:
Epoch 85/100
1170/1170 [==============================] - 1s 1ms/step - loss: 0.5582
Epoch 86/100
1170/1170 [==============================] - 1s 1ms/step - loss: 0.5769A: 0
Epoch 87/100
1170/1170 [==============================] - 1s 1ms/step - loss: 0.5190
Epoch 88/100
1170/1170 [==============================] - 1s 1ms/step - loss: 0.5089
Epoch 89/100
1170/1170 [==============================] - 1s 1ms/step - loss: 0.4986A: 0s -
loss: 0.49
Epoch 90/100
1170/1170 [==============================] - 1s 988us/step - loss: 0.5007
Epoch 91/100
```

```
1170/1170 [==============================] - 1s 1ms/step - loss: 0.4979
Epoch 92/100
1170/1170 [==============================] - 1s 1ms/step - loss: 0.4951
Epoch 93/100
1170/1170 [==============================] - 1s 1ms/step - loss: 0.4960
Epoch 94/100
1170/1170 [==============================] - 1s 1ms/step - loss: 0.4960
Epoch 95/100
1170/1170 [==============================] - 1s 1ms/step - loss: 0.4929
Epoch 96/100
1170/1170 [==============================] - 1s 1ms/step - loss: 0.4928
Epoch 97/100
1170/1170 [==============================] - 1s 1ms/step - loss: 0.4935
Epoch 98/100
1170/1170 [==============================] - 1s 985us/step - loss: 0.4941
Epoch 99/100
1170/1170 [==============================] - 1s 983us/step - loss: 0.4916
Epoch 100/100
1170/1170 [==============================] - 1s 1ms/step - loss: 0.4912
```

[25]:
```
<tensorflow.python.keras.callbacks.History at 0x7fa654d32b90>
```

[26]:
```
Probs = Model.predict(X_Train)
Probs
```

[26]:
```
array([[0.14912373],
       [0.27558982],
       [0.21288124],
       …,
       [0.9269948 ],
       [0.84290063],
       [0.86961204]], dtype=float32)
```

[27]:
```
Probs.shape
```

[27]:
```
(149734, 1)
```

[28]:
```
Train_Predicted = np.zeros(len(Probs))
for i in range(len(Probs)):
    if (Probs[i] < 0.5):
        Train_Predicted[i] = 0
    else:
        Train_Predicted[i] = 1

Equal = 0
Default = 0
Non_Default = 0
```

```
for i in range(len(Y_Train)):
    if(Y_Train[i] == 1):
        if(Y_Train[i] == Train_Predicted[i]):
            Equal += 1
            Default += 1
    else:
        if(Y_Train[i] == Train_Predicted[i]):
            Equal += 1
            Non_Default += 1

Probs

Score = Probs/(1 - Probs)
print("Scores Are:")
print(np.mean(Score[Y_Train == 0]), np.mean(Score[Y_Train == 1]))
```

```
Scores Are:
0.7665077 4.3085194
```

[29]:
```
np.sum(Train_Predicted)
```

[29]:
```
64353.0
```

[30]:
```
print("Classification Accuracy on Training Set is:")
print(Equal/len(Y_Train))
print("Classification Accuracy on Non-Default Training Set is:")
print(Non_Default/(len(Y_Train) - np.sum(Y_Train)))
print("Classification Accuracy on Default Training Set is:")
print(Default/np.sum(Y_Train))
```

```
Classification Accuracy on Training Set is:
0.7652236632962454
Classification Accuracy on Non-Default Training Set is:
0.8348243450436638
Classification Accuracy on Default Training Set is:
0.6953659221988786
```