**CAP 5619 AI for FinTech**

**Drs. Mesut Ozdag & Ramya Akula**

**Spring 2023**

**Final - Group Research Project**

**Note: This task should include the deep learning model that you have developed after the midterm and it should produce better results.**

**BNPL Credit Worthiness ML application**

**Team 11**

**Brandon Doey, Pieter de Jong, Bunnary Say, Max Thedy, Michael Setzer**

**1 - Problem Statement**

BNPL is primarily geared toward populations with limited banking access and low credit scores. A common credit application could require a few days and the actual interaction could ward off possible borrowers. Our team wanted to investigate the feasibility of a more expeditious credit check for potential lenders and hypothesized the following: H1: Can a machine learning algorithm provide an instantaneous credit check with given user inputs? H2: Which of the applicable machine learning models has the highest accuracy?

At the time of the midterm submission, we found strong evidence in support of H1 and were able to establish that a XGBoost model (Chen & Guestrin, 2016) with given features could provide an almost instantaneous credit check for any potential borrower with 89.13% accuracy. Our goal for the final project was to find compelling evidence in support of H2 and we determined that the Keras Sequential model has an even higher accuracy score (on average 97.36%).

Ultimately, we will be able to predict a client's creditworthiness in real time through a Gradio (Gradio, 2023) application. The client will be presented with two distinct outcomes based on their inputs at the final stage. Either they will be given a loan, or they will be referred to a web-based financial education application.

**2 - Literature**

Buy Now Pay Later (BNPL) is a kind of short-term financing that lets people pay for goods and services over time without having to worry about interest. Splitting a transaction into short-term installments is possible because BNPL does not charge interest. It may also be beneficial to merchants by making it simpler and less expensive for customers to purchase their goods. This is in addition to providing customers with flexibility and accessibility. This is especially helpful for smaller businesses that want to compete with larger retailers that offer financing options like this. BNPL can be a good option for customers who have little or no credit history because some providers do not require a credit check or a high credit score to be approved.

We used a synthetic version of the very popular "All Lending Club loan data" dataset which is roughly 400MB in size (George, n.d.) to determine creditworthiness with a machine learning application. Since the data contained all the parameters and features necessary to determine creditworthiness, we decided to utilize the same data from the midterm in the final project. The goal for the project is to provide BNPL services for underbanked populations, i.e., Generation Z and millennials. Over the past few years, digital shopping websites have become more and more popular with millennials, indicating that these customers are very familiar with digital transactions. It will be affected by the small number of customers who have not yet adopted BNPL services. Current BNPL users are more likely to use mobile phone digital wallets to buy now and pay later. Mastercard, Visa, and American Express are just a few of the major credit card issuers investing in installment solutions and developing their own BNPL applications in response.

Even though banks and regulators need to step in to make this growing market segment even more sustainable, we are hoping that our BNPL application will positively impact it. There have been new developments in this field despite the absence of regulation in the United States. BNPL is becoming more and more popular in established online retail markets. In addition, they are forming alliances with established BNPL providers to broaden the scope of their credit options and meet the growing demands of customers. Virtual lease to-possess models, card-connected portion choices, incorporated shopping applications, and bigger ticket plays are instances of retail market plans of action (Garcia Alvarez, 2021). BNPL advances have expanded altogether and are expected to do so further from now on. More borrowers had paid a late fee, according to the most recent data from 2022, and more than 70% of applicants had been approved for credit (Buy Now, Pay Later: Trends in the market and how they affect people, 2022). They may also increase the likelihood of overextension because of their increased reliance on third-party data tracking. Due to loans, borrowers may obtain multiple loans from the same lender. Additionally, excessive use of the BNPL business model for months or years (Buy Now, Pay Later:) may have an impact on customers' ability to meet non-BNPL obligations. Trends in the market and how they affect people, 2022).

**3 - Modeling Type and Technique**

Our project focuses on using the Keras Sequential model, a powerful deep learning technique that can accomplish various tasks such as prediction, classification, and regression, to analyze the creditworthiness of potential borrowers. The model follows a linear stack of layers, where each layer carries out a specific function on the input data and passes the results to the following layer. This adaptable architecture allows for the addition or removal of layers as needed.

To start, we prepped the data to ensure that it was in the appropriate format for input into the model. We then partitioned the data into training and validation sets and used a portion of the data to train the model. During the training phase, the model adapts its parameters to reduce the difference between the anticipated and actual outputs, thereby minimizing the error between the predicted and actual outputs.

In order to find the best values for different model parameters, such as the learning rate, number of epochs, and batch size, we used techniques to adjust hyperparameters. Additionally, we utilized methods to prevent overfitting, which occurs when the model is too complex and only memorizes the training data instead of learning from it.

After completing the training process, we evaluated the model's performance by assessing it on the validation set and made necessary adjustments to enhance its accuracy. Once we were satisfied with the model's performance, we deployed it to predict the creditworthiness of potential borrowers in real-time using a Gradio application.

The Sequential model takes into account various borrower features, including employment status, income, credit history, and other pertinent details, to make predictions about the eligibility of a potential borrower for a BNPL loan. Each layer in the model carries out a unique function to process this information, resulting in a quick and almost instantaneous credit check.

We believe that the Keras Sequential model is a robust deep learning technique that can be used for multiple tasks, including creditworthiness prediction in our project. Through our training process and fine-tuning of hyperparameters and regularization methods, we were able to achieve high accuracy in real-time predictions of borrower eligibility for BNPL loans. Our project highlights the potential of deep learning techniques for credit analysis and decision-making in financial institutions.

**How does Keras Sequential Modeling work?**

The Keras Sequential model is an implementation of a feedforward neural network (FNN) built on top of TensorFlow, where each layer in the network has a single input and a single output. These networks are known as multilayer perceptrons (MLPs) or deep FFNs.

**Feedforward Neural Networks**

Like most neural networks, FNNs consist of layers, each containing multiple neurons. However, as a FNN, Keras Sequential’s implementation of the neurons in these layers differs from traditional NNs. Each layer decreases the total number of neurons until the final output layer, which has a single neuron corresponding to the output of the model. The following is a description of the Keras sequential layers:

* Input Layer
  + Receives the input features. The number of nodes in this layer corresponds directly to the dimensionality of the input data.
* Hidden (Dense) Layers
  + Our model has four hidden layers, which are responsible for extracting features from the input data. The number of neurons decreases successively, allowing the network to learn a hierarchy of features and representations (Keras Team).
  + Each hidden layer uses the Rectified Linear Unit (ReLU) activation function, which helps the network learn the non-linear relationships between the input and output.
  + Keras Sequential can use dense, convolutional, or recurrent layers as its hidden layers. Our approach uses dense layers, since we have no need for image, sound, or language processing.
* Output Layer
  + This is the last layer of the model, containing a single neuron, and produces the output with a sigmoid activation function.

A standard FNN is a type of NN in which the connections between nodes do not form a loop (DeepAI, 2019). Instead, the information in a FNN moves only in one direction, from the input layer, through the hidden layers, to the output layer. This linear flow of information is why they are referred to as feedforward networks.

One of the key challenges faced by our team in the implementation of Keras Sequential is that our implementation was required to be a deep neural network, but our original trimmed data set was relatively simple. The simplicity of our data set meant that added layers for deep learning were not able to produce an increase in the accuracy of the model. By adding features back to our data set, the deep Keras sequential model was able to learn more complex relationships in the data than our ensemble model was able to achieve.

**Activation Functions**

Our implementation makes use of two activation functions.

* Rectified Linear Unit (ReLU)
  + The neurons receive a weighted sum of the input data. Neurons in the dense layers of our model apply the ReLU activation function. This function will either pass the input value itself if it is positive, or pass 0 if it is negative.
  + ReLU introduces non-linearity, which helps the network model complex relationships in the data (Ramachandran et al., 2017).
* Sigmoid
  + The sigmoid activation function maps its input value to a value between 0 and 1. This function is used to produce the final output of the model.
  + Given that our problem was a binomial classification problem, sigmoid is well suited for our output.

**Loss Function**

Our model uses binary cross-entropy for its loss function. This loss function is commonly used for binomial classification problems such as our loan repayment problem (Godoy, 2022). Binary cross-entropy measures the difference between the predicted probability distribution and the true probability distribution of the binary outcome. Through this, we are able to account for the confidence of each prediction rather than simply accounting for the number of correct predictions.

**Optimizer**

Our model uses the Adam optimizer. Adam is a binary cross-entropy loss function and is an extension of the stochastic gradient descent (SGD) optimizer. Like SGD, Adam updates the weights of the neural network based on the gradients of the loss function with respect to the weights. However, Adam incorporates adaptive learning rates and momentum, which can improve the speed and stability of the training process when compared to traditional SGD.

**4 - Implementation**

4.1 Import all packages

4.2 Read data

4.3 Create an instance of the classifier

4.4 Train the classifier

4.5 Check on the training set and visualize performance

4.6 Compute the prediction according to the model

4.7 Test and Validate

4.8 Check on the test set and visualize performance

4.9 Compute the evaluation metrics - accuracy, precision, recall, F1-score

4.10 Compute the confusion matrix - sensitivity & specificity

**4.1 Import all packages**

*#import all necessary packages*

import os

import pandas as pd

import numpy as np

from sklearn.model\_selection import train\_test\_split

from sklearn.preprocessing import MinMaxScaler

from sklearn.metrics import classification\_report, confusion\_matrix

from sklearn.metrics import accuracy\_score, precision\_score, recall\_score, f1\_score, roc\_curve, roc\_auc\_score, confusion\_matrix

from sklearn import metrics

import tensorflow as tf

from tensorflow.keras.models import Sequential

from tensorflow.keras.layers import Dense, Dropout

from tensorflow.keras.metrics import Precision, Recall

import matplotlib.pyplot as plt

import seaborn as sns

import warnings

warnings.filterwarnings('ignore')

2023-04-23 12:57:18.978042: I tensorflow/core/platform/cpu\_feature\_guard.cc:193] This TensorFlow binary is optimized with oneAPI Deep Neural Network Library (oneDNN) to use the following CPU instructions in performance-critical operations: AVX2 AVX512F AVX512\_VNNI FMA

To enable them in other operations, rebuild TensorFlow with the appropriate compiler flags.

2023-04-23 12:57:19.083918: I tensorflow/core/util/util.cc:169] oneDNN custom operations are on. You may see slightly different numerical results due to floating-point round-off errors from different computation orders. To turn them off, set the environment variable `TF\_ENABLE\_ONEDNN\_OPTS=0`.

2023-04-23 12:57:19.089050: W tensorflow/stream\_executor/platform/default/dso\_loader.cc:64] Could not load dynamic library 'libcudart.so.11.0'; dlerror: libcudart.so.11.0: cannot open shared object file: No such file or directory

2023-04-23 12:57:19.089067: I tensorflow/stream\_executor/cuda/cudart\_stub.cc:29] Ignore above cudart dlerror if you do not have a GPU set up on your machine.

2023-04-23 12:57:19.113665: E tensorflow/stream\_executor/cuda/cuda\_blas.cc:2981] Unable to register cuBLAS factory: Attempting to register factory for plugin cuBLAS when one has already been registered

2023-04-23 12:57:19.621955: W tensorflow/stream\_executor/platform/default/dso\_loader.cc:64] Could not load dynamic library 'libnvinfer.so.7'; dlerror: libnvinfer.so.7: cannot open shared object file: No such file or directory

2023-04-23 12:57:19.622015: W tensorflow/stream\_executor/platform/default/dso\_loader.cc:64] Could not load dynamic library 'libnvinfer\_plugin.so.7'; dlerror: libnvinfer\_plugin.so.7: cannot open shared object file: No such file or directory

2023-04-23 12:57:19.622020: W tensorflow/compiler/tf2tensorrt/utils/py\_utils.cc:38] TF-TRT Warning: Cannot dlopen some TensorRT libraries. If you would like to use Nvidia GPU with TensorRT, please make sure the missing libraries mentioned above are installed properly.

**4.2 Read Data**

*#read in the clean csv file*

df = pd.read\_csv('/work/loan\_data\_clean\_dnn.csv')

*# turn all columns to int type*

loan\_data = df.astype(int)

*# drop loan\_status from loan\_data and assign to X*

X = loan\_data.drop('loan\_paid', axis=1)

y = loan\_data['loan\_paid']

*# split X and y into training and testing sets*

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, random\_state=42)

*# Split the training set into training and validation sets*

X\_train, X\_val, y\_train, y\_val = train\_test\_split(X\_train, y\_train, test\_size=0.2, random\_state=42)

*# Print the shapes of the datasets*

print('X\_train shape:', X\_train.shape)

print('y\_train shape:', y\_train.shape)

print('X\_test shape:', X\_test.shape)

print('y\_test shape:', y\_test.shape)

print('X\_val shape:', X\_val.shape)

print('y\_val shape:', y\_val.shape)

X\_train shape: (361891, 81)

y\_train shape: (361891,)

X\_test shape: (113091, 81)

y\_test shape: (113091,)

X\_val shape: (90473, 81)

y\_val shape: (90473,)

**4.3 Create an instance of the classifier**

*#add a scaler and verify the data*

scaler = MinMaxScaler()

X\_train= scaler.fit\_transform(X\_train)

X\_test = scaler.transform(X\_test)

X\_val = scaler.transform(X\_val)

print(X\_train.shape)

print(X\_test.shape)

print(X\_val.shape)

model = Sequential()

model.add(Dense(units=78,activation='relu'))

model.add(Dense(units=39,activation='relu'))

model.add(Dense(units=19,activation='relu'))

model.add(Dense(units=8,activation='relu'))

model.add(Dense(units=4,activation='relu'))

model.add(Dense(units=1,activation='sigmoid'))

model.compile(loss='binary\_crossentropy', optimizer='adam', metrics=['accuracy', Precision(name='precision'), Recall(name='recall')])

(361891, 81)

(113091, 81)

(90473, 81)

2023-04-23 12:57:23.724347: W tensorflow/stream\_executor/platform/default/dso\_loader.cc:64] Could not load dynamic library 'libcuda.so.1'; dlerror: libcuda.so.1: cannot open shared object file: No such file or directory

2023-04-23 12:57:23.724378: W tensorflow/stream\_executor/cuda/cuda\_driver.cc:263] failed call to cuInit: UNKNOWN ERROR (303)

2023-04-23 12:57:23.724396: I tensorflow/stream\_executor/cuda/cuda\_diagnostics.cc:156] kernel driver does not appear to be running on this host (p-15617cbb-2604-4e67-8d7c-8506cee32e96): /proc/driver/nvidia/version does not exist

2023-04-23 12:57:23.724643: I tensorflow/core/platform/cpu\_feature\_guard.cc:193] This TensorFlow binary is optimized with oneAPI Deep Neural Network Library (oneDNN) to use the following CPU instructions in performance-critical operations: AVX2 AVX512F AVX512\_VNNI FMA

To enable them in other operations, rebuild TensorFlow with the appropriate compiler flags.

**4.4 Train the classifier**

*#train the classifier*

keras\_seq = model.fit(x=X\_train,

y=y\_train,

epochs=40,

batch\_size=512,

validation\_data=(X\_val, y\_val), verbose=1)

Epoch 1/40

707/707 [==============================] - 2s 2ms/step - loss: 0.2247 - accuracy: 0.9066 - precision: 0.8847 - recall: 0.9213 - val\_loss: 0.1346 - val\_accuracy: 0.9474 - val\_precision: 0.9636 - val\_recall: 0.9234

Epoch 2/40

707/707 [==============================] - 1s 2ms/step - loss: 0.1203 - accuracy: 0.9545 - precision: 0.9489 - recall: 0.9546 - val\_loss: 0.1154 - val\_accuracy: 0.9556 - val\_precision: 0.9586 - val\_recall: 0.9469

Epoch 3/40

707/707 [==============================] - 1s 2ms/step - loss: 0.1115 - accuracy: 0.9579 - precision: 0.9530 - recall: 0.9575 - val\_loss: 0.1086 - val\_accuracy: 0.9589 - val\_precision: 0.9573 - val\_recall: 0.9556

Epoch 4/40

707/707 [==============================] - 1s 2ms/step - loss: 0.1085 - accuracy: 0.9589 - precision: 0.9544 - recall: 0.9583 - val\_loss: 0.1055 - val\_accuracy: 0.9603 - val\_precision: 0.9511 - val\_recall: 0.9657

Epoch 5/40

707/707 [==============================] - 1s 2ms/step - loss: 0.1055 - accuracy: 0.9597 - precision: 0.9554 - recall: 0.9590 - val\_loss: 0.1028 - val\_accuracy: 0.9607 - val\_precision: 0.9514 - val\_recall: 0.9660

Epoch 6/40

707/707 [==============================] - 1s 2ms/step - loss: 0.1030 - accuracy: 0.9605 - precision: 0.9564 - recall: 0.9597 - val\_loss: 0.0996 - val\_accuracy: 0.9626 - val\_precision: 0.9572 - val\_recall: 0.9639

Epoch 7/40

707/707 [==============================] - 1s 2ms/step - loss: 0.0988 - accuracy: 0.9623 - precision: 0.9581 - recall: 0.9618 - val\_loss: 0.1051 - val\_accuracy: 0.9584 - val\_precision: 0.9720 - val\_recall: 0.9390

Epoch 8/40

707/707 [==============================] - 1s 2ms/step - loss: 0.0957 - accuracy: 0.9632 - precision: 0.9590 - recall: 0.9627 - val\_loss: 0.0949 - val\_accuracy: 0.9632 - val\_precision: 0.9647 - val\_recall: 0.9570

Epoch 9/40

707/707 [==============================] - 1s 2ms/step - loss: 0.0932 - accuracy: 0.9642 - precision: 0.9606 - recall: 0.9633 - val\_loss: 0.0942 - val\_accuracy: 0.9639 - val\_precision: 0.9510 - val\_recall: 0.9738

Epoch 10/40

707/707 [==============================] - 1s 2ms/step - loss: 0.0900 - accuracy: 0.9651 - precision: 0.9617 - recall: 0.9642 - val\_loss: 0.0925 - val\_accuracy: 0.9643 - val\_precision: 0.9495 - val\_recall: 0.9763

Epoch 11/40

707/707 [==============================] - 1s 2ms/step - loss: 0.0883 - accuracy: 0.9656 - precision: 0.9622 - recall: 0.9647 - val\_loss: 0.0865 - val\_accuracy: 0.9661 - val\_precision: 0.9670 - val\_recall: 0.9609

Epoch 12/40

707/707 [==============================] - 1s 2ms/step - loss: 0.0862 - accuracy: 0.9665 - precision: 0.9635 - recall: 0.9653 - val\_loss: 0.0846 - val\_accuracy: 0.9672 - val\_precision: 0.9668 - val\_recall: 0.9636

Epoch 13/40

707/707 [==============================] - 1s 2ms/step - loss: 0.0844 - accuracy: 0.9668 - precision: 0.9639 - recall: 0.9656 - val\_loss: 0.0837 - val\_accuracy: 0.9673 - val\_precision: 0.9651 - val\_recall: 0.9657

Epoch 14/40

707/707 [==============================] - 1s 2ms/step - loss: 0.0823 - accuracy: 0.9681 - precision: 0.9656 - recall: 0.9665 - val\_loss: 0.0829 - val\_accuracy: 0.9674 - val\_precision: 0.9677 - val\_recall: 0.9631

Epoch 15/40

707/707 [==============================] - 1s 2ms/step - loss: 0.0815 - accuracy: 0.9683 - precision: 0.9660 - recall: 0.9664 - val\_loss: 0.0822 - val\_accuracy: 0.9689 - val\_precision: 0.9667 - val\_recall: 0.9674

Epoch 16/40

707/707 [==============================] - 1s 2ms/step - loss: 0.0801 - accuracy: 0.9689 - precision: 0.9670 - recall: 0.9667 - val\_loss: 0.0799 - val\_accuracy: 0.9689 - val\_precision: 0.9723 - val\_recall: 0.9615

Epoch 17/40

707/707 [==============================] - 1s 2ms/step - loss: 0.0782 - accuracy: 0.9694 - precision: 0.9679 - recall: 0.9669 - val\_loss: 0.0799 - val\_accuracy: 0.9689 - val\_precision: 0.9718 - val\_recall: 0.9619

Epoch 18/40

707/707 [==============================] - 1s 2ms/step - loss: 0.0777 - accuracy: 0.9696 - precision: 0.9683 - recall: 0.9670 - val\_loss: 0.0781 - val\_accuracy: 0.9695 - val\_precision: 0.9607 - val\_recall: 0.9752

Epoch 19/40

707/707 [==============================] - 1s 2ms/step - loss: 0.0768 - accuracy: 0.9702 - precision: 0.9691 - recall: 0.9675 - val\_loss: 0.0764 - val\_accuracy: 0.9706 - val\_precision: 0.9644 - val\_recall: 0.9736

Epoch 20/40

707/707 [==============================] - 1s 2ms/step - loss: 0.0757 - accuracy: 0.9707 - precision: 0.9698 - recall: 0.9677 - val\_loss: 0.0776 - val\_accuracy: 0.9696 - val\_precision: 0.9663 - val\_recall: 0.9694

Epoch 21/40

707/707 [==============================] - 1s 2ms/step - loss: 0.0749 - accuracy: 0.9709 - precision: 0.9699 - recall: 0.9681 - val\_loss: 0.0761 - val\_accuracy: 0.9708 - val\_precision: 0.9672 - val\_recall: 0.9712

Epoch 22/40

707/707 [==============================] - 1s 2ms/step - loss: 0.0737 - accuracy: 0.9714 - precision: 0.9704 - recall: 0.9687 - val\_loss: 0.0782 - val\_accuracy: 0.9698 - val\_precision: 0.9783 - val\_recall: 0.9572

Epoch 23/40

707/707 [==============================] - 1s 2ms/step - loss: 0.0731 - accuracy: 0.9715 - precision: 0.9707 - recall: 0.9685 - val\_loss: 0.0758 - val\_accuracy: 0.9702 - val\_precision: 0.9750 - val\_recall: 0.9615

Epoch 24/40

707/707 [==============================] - 1s 2ms/step - loss: 0.0725 - accuracy: 0.9719 - precision: 0.9712 - recall: 0.9689 - val\_loss: 0.0736 - val\_accuracy: 0.9717 - val\_precision: 0.9745 - val\_recall: 0.9653

Epoch 25/40

707/707 [==============================] - 1s 2ms/step - loss: 0.0720 - accuracy: 0.9719 - precision: 0.9712 - recall: 0.9689 - val\_loss: 0.0775 - val\_accuracy: 0.9704 - val\_precision: 0.9610 - val\_recall: 0.9770

Epoch 26/40

707/707 [==============================] - 1s 2ms/step - loss: 0.0713 - accuracy: 0.9720 - precision: 0.9716 - recall: 0.9688 - val\_loss: 0.0762 - val\_accuracy: 0.9702 - val\_precision: 0.9583 - val\_recall: 0.9796

Epoch 27/40

707/707 [==============================] - 1s 2ms/step - loss: 0.0709 - accuracy: 0.9724 - precision: 0.9721 - recall: 0.9691 - val\_loss: 0.0757 - val\_accuracy: 0.9704 - val\_precision: 0.9584 - val\_recall: 0.9798

Epoch 28/40

707/707 [==============================] - 1s 2ms/step - loss: 0.0711 - accuracy: 0.9725 - precision: 0.9720 - recall: 0.9694 - val\_loss: 0.0714 - val\_accuracy: 0.9726 - val\_precision: 0.9714 - val\_recall: 0.9705

Epoch 29/40

707/707 [==============================] - 1s 2ms/step - loss: 0.0702 - accuracy: 0.9728 - precision: 0.9725 - recall: 0.9695 - val\_loss: 0.0726 - val\_accuracy: 0.9719 - val\_precision: 0.9787 - val\_recall: 0.9615

Epoch 30/40

707/707 [==============================] - 1s 2ms/step - loss: 0.0688 - accuracy: 0.9736 - precision: 0.9733 - recall: 0.9704 - val\_loss: 0.0722 - val\_accuracy: 0.9727 - val\_precision: 0.9781 - val\_recall: 0.9637

Epoch 31/40

707/707 [==============================] - 1s 2ms/step - loss: 0.0690 - accuracy: 0.9733 - precision: 0.9734 - recall: 0.9698 - val\_loss: 0.0707 - val\_accuracy: 0.9725 - val\_precision: 0.9707 - val\_recall: 0.9710

Epoch 32/40

707/707 [==============================] - 1s 2ms/step - loss: 0.0679 - accuracy: 0.9737 - precision: 0.9734 - recall: 0.9704 - val\_loss: 0.0702 - val\_accuracy: 0.9731 - val\_precision: 0.9742 - val\_recall: 0.9687

Epoch 33/40

707/707 [==============================] - 1s 2ms/step - loss: 0.0669 - accuracy: 0.9741 - precision: 0.9739 - recall: 0.9709 - val\_loss: 0.0700 - val\_accuracy: 0.9734 - val\_precision: 0.9791 - val\_recall: 0.9643

Epoch 34/40

707/707 [==============================] - 1s 2ms/step - loss: 0.0673 - accuracy: 0.9741 - precision: 0.9741 - recall: 0.9706 - val\_loss: 0.0710 - val\_accuracy: 0.9730 - val\_precision: 0.9790 - val\_recall: 0.9634

Epoch 35/40

707/707 [==============================] - 1s 2ms/step - loss: 0.0668 - accuracy: 0.9744 - precision: 0.9745 - recall: 0.9709 - val\_loss: 0.0713 - val\_accuracy: 0.9729 - val\_precision: 0.9788 - val\_recall: 0.9635

Epoch 36/40

707/707 [==============================] - 1s 2ms/step - loss: 0.0658 - accuracy: 0.9747 - precision: 0.9748 - recall: 0.9713 - val\_loss: 0.0673 - val\_accuracy: 0.9742 - val\_precision: 0.9761 - val\_recall: 0.9692

Epoch 37/40

707/707 [==============================] - 1s 2ms/step - loss: 0.0651 - accuracy: 0.9747 - precision: 0.9749 - recall: 0.9711 - val\_loss: 0.0681 - val\_accuracy: 0.9737 - val\_precision: 0.9749 - val\_recall: 0.9693

Epoch 38/40

707/707 [==============================] - 1s 2ms/step - loss: 0.0640 - accuracy: 0.9754 - precision: 0.9758 - recall: 0.9717 - val\_loss: 0.0683 - val\_accuracy: 0.9737 - val\_precision: 0.9702 - val\_recall: 0.9742

Epoch 39/40

707/707 [==============================] - 1s 2ms/step - loss: 0.0634 - accuracy: 0.9757 - precision: 0.9762 - recall: 0.9720 - val\_loss: 0.0689 - val\_accuracy: 0.9733 - val\_precision: 0.9817 - val\_recall: 0.9613

Epoch 40/40

707/707 [==============================] - 1s 2ms/step - loss: 0.0627 - accuracy: 0.9759 - precision: 0.9764 - recall: 0.9722 - val\_loss: 0.0666 - val\_accuracy: 0.9746 - val\_precision: 0.9756 - val\_recall: 0.9705

**4.5 Check on the training set and visualize performance**

*# Evaluate the model on the training set*

train\_metrics = model.evaluate(X\_train, y\_train, verbose=1)

*# Extract the individual metrics*

train\_loss, train\_accuracy, train\_precision, train\_recall = train\_metrics

*# Print the metrics*

print("Train loss:", train\_loss)

print("Train accuracy:", train\_accuracy)

print("Train precision:", train\_precision)

print("Train recall:", train\_recall)

11310/11310 [==============================] - 9s 784us/step - loss: 0.0600 - accuracy: 0.9770 - precision: 0.9784 - recall: 0.9726

Train loss: 0.05999536067247391

Train accuracy: 0.9770427942276001

Train precision: 0.9784470200538635

Train recall: 0.9725534319877625

*# Plot the training and validation loss*

plt.plot(keras\_seq.history['loss'], label='Training loss')

plt.plot(keras\_seq.history['val\_loss'], label='Validation loss')

plt.xlabel('Epochs')

plt.ylabel('Loss')

plt.title('Training and Validation Loss')

plt.legend()

plt.show()

*# Plot the training and validation accuracy*

plt.plot(keras\_seq.history['accuracy'], label='Training accuracy')

plt.plot(keras\_seq.history['val\_accuracy'], label='Validation accuracy')

plt.xlabel('Epochs')

plt.ylabel('Accuracy')

plt.title('Training and Validation Accuracy')

plt.legend()

plt.show()

**4.6 Compute the prediction according to the model**

*# compute prediction on unseen (test) dataset*

y\_pred = (model.predict(X\_test) > 0.5).astype("int32")

print('Training Accuracy: ', round(accuracy\_score(y\_test, y\_pred)\*100, 2), '%')

print('Training Precision: ', round(precision\_score(y\_test, y\_pred)\*100, 2), '%')

print('Training Recall: ', round(recall\_score(y\_test, y\_pred)\*100, 2), '%')

print('Training F1: ', round(f1\_score(y\_test, y\_pred)\*100, 2), '%')

print('Training ROC AUC: ', round(roc\_auc\_score(y\_test, y\_pred)\*100, 2), '%')

print('Training Confusion Matrix: \n', confusion\_matrix(y\_test, y\_pred))

3535/3535 [==============================] - 2s 653us/step

Training Accuracy: 97.42 %

Training Precision: 97.57 %

Training Recall: 96.92 %

Training F1: 97.24 %

Training ROC AUC: 97.39 %

Training Confusion Matrix:

[[58644 1283]

[ 1640 51524]]

*# visualize the performance*

cm = metrics.confusion\_matrix(y\_test, y\_pred)

plt.figure(figsize=(9,9))

sns.heatmap(cm, annot=True, fmt=".3f", linewidths=.5, square = True, cmap = 'Blues\_r');

plt.ylabel('Actual label');

plt.xlabel('Predicted label');

all\_sample\_title = 'Accuracy Score: {0}'.format(accuracy\_score(y\_test, y\_pred))

plt.title(all\_sample\_title, size = 15);

*# make predictions on the testing set*

y\_pred\_class = (model.predict(X\_test) > 0.5).astype("int32")

pred=y\_pred\_class.tolist()

pred\_1 = [i[0] for i in pred]

pred\_2 = ''.join(str(pred\_1).split(','))

*# print true vs predicted*

print('True:', y\_test.values[0:25])

print('Pred:', pred\_2[0:50])

3535/3535 [==============================] - 2s 628us/step

True: [0 1 0 0 1 1 0 1 1 1 0 1 0 0 1 0 0 0 0 0 1 1 1 0 1]

Pred: [0 1 0 0 1 1 0 1 1 1 0 1 0 0 1 0 0 0 0 0 1 1 1 0 1

**4.7 Test and Validate**

*# Evaluate the model on the test set*

test\_metrics = model.evaluate(X\_test, y\_test, verbose=1)

*# Extract the individual metrics*

test\_loss, test\_accuracy, test\_precision, test\_recall = test\_metrics

*# Print the metrics*

print("Test loss:", test\_loss)

print("Test accuracy:", test\_accuracy)

print("Test precision:", test\_precision)

print("Test recall:", test\_recall)

3535/3535 [==============================] - 3s 763us/step - loss: 0.0672 - accuracy: 0.9742 - precision: 0.9757 - recall: 0.9692

Test loss: 0.06723771244287491

Test accuracy: 0.9741535782814026

Test precision: 0.9757039546966553

Test recall: 0.96915203332901

y\_pred\_class = (model.predict(X\_test) > 0.5).astype("int32")

*#predictions = (model.predict(X\_test) > 0.5).astype("int32")*

print('Testing Accuracy score: ', round(accuracy\_score(y\_test, y\_pred\_class)\*100, 2), '%')

print('Testing Precision score: ', round(precision\_score(y\_test, y\_pred\_class)\*100, 2), '%')

print('Testing Recall score: ', round(recall\_score(y\_test, y\_pred\_class)\*100, 2), '%')

print('Testing F1 score: ', round(f1\_score(y\_test, y\_pred\_class)\*100, 2), '%')

print('Testing ROC AUC score: ', round(roc\_auc\_score(y\_test, y\_pred\_class)\*100, 2), '%')

print('Testing Confusion matrix: \n', confusion\_matrix(y\_test, y\_pred\_class))

3535/3535 [==============================] - 2s 604us/step

Testing Accuracy score: 97.42 %

Testing Precision score: 97.57 %

Testing Recall score: 96.92 %

Testing F1 score: 97.24 %

Testing ROC AUC score: 97.39 %

Testing Confusion matrix:

[[58644 1283]

[ 1640 51524]]

**4.8 Check on the test set and visualize performance**

*# visualize the performance*

cm = metrics.confusion\_matrix(y\_test, y\_pred\_class)

plt.figure(figsize=(9,9))

sns.heatmap(cm, annot=True, fmt=".3f", linewidths=.5, square = True, cmap = 'Blues\_r');

plt.ylabel('Actual label');

plt.xlabel('Predicted label');

all\_sample\_title = 'Accuracy Score: {0}'.format(accuracy\_score(y\_test, y\_pred\_class))

plt.title(all\_sample\_title, size = 15);

*# plot ROC curve*

fpr, tpr, \_ = roc\_curve(y\_test, y\_pred\_class)

auc = roc\_auc\_score(y\_test, y\_pred\_class)

plt.figure()

plt.plot(fpr, tpr, label='ROC curve (AUC = %0.2f)' % auc)

plt.plot([0, 1], [0, 1], linestyle='--', color='r', label='Random classifier')

plt.xlabel('False Positive Rate (Specificity)')

plt.ylabel('True Positive Rate (Sensitivity)')

plt.title('ROC curve for loan classifier')

plt.legend(loc="lower right")

plt.grid(True)

plt.show()

**4.9 Compute the evaluation metrics - accuracy, precision, recall, F1-score**

y\_val\_pred = (model.predict(X\_val) > 0.5).astype("int32")

print('Validation accuracy score: ', round(accuracy\_score(y\_val, y\_val\_pred)\*100, 2), '%')

print('Validation precision score: ', round(precision\_score(y\_val, y\_val\_pred)\*100, 2), '%')

print('Validation recall score: ', round(recall\_score(y\_val, y\_val\_pred)\*100, 2), '%')

print('Validation F1 score: ', round(f1\_score(y\_val, y\_val\_pred)\*100, 2), '%')

print('Validation ROC AUC score: ', round(roc\_auc\_score(y\_val, y\_val\_pred)\*100, 2), '%')

2828/2828 [==============================] - 2s 582us/step

Validation accuracy score: 97.46 %

Validation precision score: 97.56 %

Validation recall score: 97.05 %

Validation F1 score: 97.3 %

Validation ROC AUC score: 97.44 %

**4.10 Compute the confusion matrix - sensitivity & specificity**

*# visualize the validation confusion matrix*

cm = confusion\_matrix(y\_val, y\_val\_pred)

sns.heatmap(cm, annot=True, fmt='d', cmap='Blues')

plt.title('Validation Confusion Matrix')

plt.xlabel('Predicted')

plt.ylabel('Actual')

plt.show()

*# plot ROC curve*

import matplotlib.pyplot as plt

fpr, tpr, thresholds = metrics.roc\_curve(y\_val, y\_val\_pred)

plt.plot(fpr, tpr)

plt.xlim([0.0, 1.0])

plt.ylim([0.0, 1.0])

plt.title('ROC curve for loan classifier')

plt.xlabel('False Positive Rate (Specificity)')

plt.ylabel('True Positive Rate (Sensitivity)')

plt.grid(True)

**5 - Observations**

5.1 Overview 5.2 Plot to show balanced dataset 5.3 Feature Importance 5.4 Plot Correlation Matrix 5.5 Scatterplot Open Accounts and Revolving Balance by Loan Status 5.6 Scatterplot loan size and associated late fees 5.7 Bar graph 5.8 Kernel Density Estimation Plot

**5.1 Overview**

We chose to use a synthetic dataset based on aggregated LendingClub loan data, as we did for the midterm project. This dataset was obtained from the Kaggle page located here (George, N., n.d). Unfortunately, due to the size of the dataset, we were unable to work with it easily in any online Jupyter style notebooks. As a result, we downloaded the dataset, uncompressed it, and removed a large number of columns that we deemed unnecessary. We also dropped any record whose "loan status" was anything other than "Fully Paid" (indicating the loan was paid until the balance was paid off) or "Charged Off" (indicating the borrower had failed to repay the loan). There were significantly more Fully Paid records than Charged Off records, resulting in an extremely imbalanced dataset. We knew this would have a negative impact on the accuracy of our model, so we randomly dropped Fully Paid records to bring this number down and closer to the Charged Off records (Brownlee, 2018). We were aware of other methods for manually balancing datasets, such as the Synthetic Minority Oversampling Technique (SMOTE), but due to the sheer size of the dataset, randomly removing the Fully Paid records seemed like a much simpler approach that still left us with plenty of data (over 500,000 records) to train our model on. Following the completion of these data engineering actions, a number of categorical columns were One-Hot Encoded and appended to the end of the dataset. The final step was to save the dataframe as a .csv file so that it could be saved to disk and uploaded to our Deepnote Jupyter notebook.

**5.2 Plot to show balanced dataset**

The following bar chart simply shows the balanced distribution of records based on our target variable of “loan\_paid”. In this case, a 1 represents a Fully Paid record and a 0 represents Charged Off.

*# visualize the target variable in the original dataset to establish balanced dataset*

sns.countplot(x='loan\_paid', data=df)

plt.show()

**5.3 Feature Importance**

This graph shows the feature importance of each column in the dataset as it relates to our target variable of loan\_paid. A negative correlation means that as one variable increases, the other decreases. A positive correlation means that as one variable increases, the other variable increases as well. In this example, the “last fico range high” column has the highest negative correlation with the “loan\_paid” column, suggesting that as FICO credit scores decrease, so does the chance of the borrower defaulting on their loan.

*# checking correlation of features to the new loan\_paid column*

plt.figure(figsize=(15,7))

new\_corr = df.corr().iloc[:-1,-1].sort\_values()

new\_corr.plot.bar(rot=90)

**5.4 Plot Correlation Matrix**

The following correlation matrix (Kumar, 2022) shows a strong negative correlation between the “loan\_paid” column and the “collections\_12\_mths\_ex\_med” column suggesting that if a borrower had high number of collections in the past year (excluding medical collections), then there is a much higher chance that they will default on their loan.

*# plot correlation matrix*

plt.figure(figsize=(12,10))

mask = np.zeros\_like(df.corr())

mask[np.triu\_indices\_from(mask)] = True

sns.heatmap(df.corr(), cmap='viridis', mask=mask, annot=False, square=True)

plt.show()

**5.5 Scatterplot Open Accounts and Revolving Balance by Loan Status**

The result of the scatter plot below shows that borrowers with fewer open accounts tend to have higher revolving balances on those accounts. This could indicate a lower credit score for these users since their total available credit would be smaller across all accounts. A high revolving balance could also mean these borrowers are overextended and may have trouble paying off their debts. The borrowers who ended up defaulting on their loans are highlighted in red below.

*# create a figure and an axis object*

fig, ax = plt.subplots()

colors = {1: "green", 0: "red"}

ax.scatter(df["open\_acc"], df["revol\_bal"], c=df["loan\_paid"].map(colors), alpha=0.1)

ax.set\_xlabel("Open Accounts")

ax.set\_ylabel("Revolving Balance")

ax.set\_title("Relationship between Open Accounts and Revolving Balance by Loan Status")

plt.show()

**5.6 Scatterplot loan size and associated late fees**

The graph below may help shed light on the correlation between loan size and associated late fees. It's reasonable to assume that late fees would increase in proportion to the size of the loan, but this isn't always the case. Indeed, as can be seen below, there was a spike in late fees when the loan amount reached the $35,000 mark.

*# scatter plot that compares loan\_amt and total\_rec\_late\_fee*

sns.scatterplot(x='loan\_amnt', y='total\_rec\_late\_fee', data=df)

plt.show()

**5.7 Bar graph**

The data below provides support for the hypothesis that loan\_paid is inversely related to collections\_12\_mths\_ex\_med. More collections in the past 12 months (excluding medical collections) correlates with a lower likelihood of loan repayment. This may be an indication that these borrowers are having more money problems than average or that their credit ratings are poorer.

*# visualize the relationship between loan\_paid and collections\_12\_mths\_ex\_med and group by loan\_paid*

sns.catplot(x='collections\_12\_mths\_ex\_med', y='loan\_paid', data=df, kind='bar')

plt.show()

**5.8 Kernel Density Estimation Plot**

The probability density function of a random variable can be estimated using the Kernel density estimation (KDE) graph (Malhotra, 2020). This KDE plot is used to help us visualize the distribution of FICO scores for borrowers with various loan statuses when applied to loan status and FICO score. The diagram shows FICO 450 score being extremely low and 850 being brilliant. By utilizing this KDE plot, we can perceive how the conveyance of credit ratings shifts across various advance situations with. In a nutshell, the KDE plots of the loan status and FICO score indicate that a higher FICO score is related to a lower likelihood of loan default and a higher likelihood of paying off the loan completely. This data can be utilized to assess the financial soundness of borrowers and settle on informed loaning choices. For instance, borrowers who have defaulted on their credits might have a top in the conveyance around a low credit rating of 450, demonstrating that borrowers with lower credit ratings are bound to default on their advances. Conversely, borrowers who are as of now making installments on their credits might have a top in the conveyance around a high credit rating of 850, demonstrating that borrowers with higher credit ratings are bound to make their installments on time. As a result, we are able to provide customized BNPL options to borrowers with varying degrees of creditworthiness by identifying the FICO score range associated with a higher risk of default.

*# Kernel Density Estimation Plots for loan\_status and last\_fico\_range\_high and show legend*

plt.figure(figsize=(12, 6))

sns.kdeplot(loan\_data[loan\_data['loan\_paid'] == 1]['last\_fico\_range\_high'], label='Charged Off', shade=True)

sns.kdeplot(loan\_data[loan\_data['loan\_paid'] == 0]['last\_fico\_range\_high'], label='Fully Paid', shade=True)

plt.xlabel('last\_fico\_range\_high')

plt.ylabel('Density')

plt.legend()

plt.show()

**6 - References**

Brownlee, J. (2018, April 25). Machine Learning Mastery. Retrieved from How to Remove Outliers for Machine Learning: <https://machinelearningmastery.com/how-to-use-statistics-to-identify-outliers-in-data/>

Buy Now, Pay Later: Market trends and consumer impacts. (2022, September 15). Retrieved from CFBP: <https://www.consumerfinance.gov/data-research/research-reports/buy-now-pay-later-market-trends-and-consumer-impacts/>

DeepAI. (2019, May 17). Feed Forward Neural Network. DeepAI. Retrieved April 22, 2023, from <https://deepai.org/machine-learning-glossary-and-terms/feed-forward-neural-network>

George, N. (n.d.). Kaggle. Retrieved from All Lending Club loan data: <https://www.kaggle.com/datasets/wordsforthewise/lending-club>

Godoy, D. (2022, July 10). Understanding binary cross-entropy / log loss: A visual explanation. Medium. Retrieved April 19, 2023, from <https://towardsdatascience.com/understanding-binary-cross-entropy-log-loss-a-visual-explanation-a3ac6025181a>

Gradio (2023). Build & Share Delightful Machine Learning Apps. Retrieved April 23, 2023, from <https://gradio.app/>

Kumar, A. (2022, April). Correlation Concepts, Matrix & Heatmap using Seaborn. Retrieved from Vitalflux: <https://vitalflux.com/correlation-heatmap-with-seaborn-pandas/amp/>

Malhotra, V. (2020, October 12). Medium. Retrieved from ML04: Kernel Density Estimation: <https://medium.com/analytics-vidhya/ml04-kernel-density-estimation-ee29a1578d0c>

Ramachandran, P., Zoph, B., & Le, Q. V. (2017). Searching for activation functions. ArXiv Preprint ArXiv:1710.05941.

Team, K. (n.d.). Simple. flexible. powerful. Keras. Retrieved April 20, 2023, from <https://keras.io/>