Artificial Neural Network Project Report

Ammar Saleem(i210327) , Wajeeh ul Hassan(i210337)

May 18, 2024

Contents

Chapter 1

Introduction

The purpose of this project is to develop an Artificial Neural Network (ANN) model to predict loan approval status based on various financial and personal attributes. The project involves preprocessing the dataset, defining the ANN architecture, training the model, and evaluating its performance. Additionally, the report discusses the challenges faced during the project and provides visual diagrams to better understand the data distribution and model performance.

Chapter 2

Data Preprocessing

Data preprocessing is a crucial step in building a machine learning model. It involves cleaning the data, handling missing values, encoding categorical variables, and scaling numerical features.

2.1 Loading the Dataset

The dataset used in this project is the loan approval dataset, which contains various features such as income, age, experience, marital status, house ownership, car ownership, profession, city, and state. The target variable is the '$Risk_Flag$', indicating whether the loan was approved (1) or not (0).

```
import pandas as pd
from sklearn.model_selection import train_test_split

df = pd.read_csv('loan_approval_dataset_preprocessed.csv')
X_train, X_test, y_train, y_test = train_test_split(
    df.drop("Risk_Flag", axis=1), df["Risk_Flag"], test_size
    =0.2, stratify=df["Risk_Flag"], random_state=42)
```

Listing 2.1: Loading the Dataset

2.2 Data Cleaning and Transformation

The data cleaning process includes handling missing values, encoding categorical variables, and scaling numerical features. Below is the code snippet for this preprocessing step.

```python
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns

df = pd.read_json('/content/loan_approval_dataset.json')
df = df.drop(['Id'], axis=1)

categorical_columns = ['Risk_Flag', 'Married/Single', 'House_Ownership', 'Car_Ownership', 'Profession', 'CITY', 'STATE']

# Encoding categorical variables
for column in categorical_columns:
    frequency_map = df[column].value_counts(normalize=True)
    df[f'{column}'] = df[column].map(frequency_map)

# Scaling numerical features
from sklearn.preprocessing import MinMaxScaler
numeric_columns = ['Income', 'Age', 'Experience', 'CURRENT_JOB_YRS', 'CURRENT_HOUSE_YRS']
scaler = MinMaxScaler()
df[numeric_columns] = scaler.fit_transform(df[numeric_columns])

df.to_csv('loan_approval_dataset_preprocessed.csv', index=False)
```

Listing 2.2: Data Cleaning and Transformation

Chapter 3

Model Training

The neural network model is defined using PyTorch, and it consists of three fully connected layers. The model is trained using the Binary Cross-Entropy Loss function and the Adam optimizer.

```
import torch
import torch.nn as nn
import torch.optim as optim
from torch.utils.data import DataLoader, TensorDataset

# Convert data to PyTorch tensors
device = torch.device("cuda" if torch.cuda.is_available() else "
    cpu")
X_train_tensor = torch.tensor(X_train.values, dtype=torch.
    float32).to(device)
y_train_tensor = torch.tensor(y_train.values, dtype=torch.
    float32).to(device)
X_test_tensor = torch.tensor(X_test.values, dtype=torch.float32)
    .to(device)
y_test_tensor = torch.tensor(y_test.values, dtype=torch.float32)
    .to(device)

# Create TensorDatasets
train_dataset = TensorDataset(X_train_tensor, y_train_tensor)
test_dataset = TensorDataset(X_test_tensor, y_test_tensor)

# Create DataLoaders
batch_size = 64
train_loader = DataLoader(dataset=train_dataset, batch_size=
    batch_size, shuffle=True)
```

```
test_loader = DataLoader(dataset=test_dataset, batch_size=
    batch_size, shuffle=False)

# Define the neural network architecture
class NeuralNetwork(nn.Module):
    def __init__(self, input_size):
        super(NeuralNetwork, self).__init__()
        self.fc1 = nn.Linear(input_size, 64)
        self.fc2 = nn.Linear(64, 32)
        self.fc3 = nn.Linear(32, 1)
        self.relu = nn.ReLU()
        self.sigmoid = nn.Sigmoid()

    def forward(self, x):
        x = self.relu(self.fc1(x))
        x = self.relu(self.fc2(x))
        x = self.sigmoid(self.fc3(x))
        return x

# Initialize the model and move it to GPU
input_size = X_train.shape[1]
model = NeuralNetwork(input_size).to(device)

# Define loss function and optimizer
criterion = nn.BCELoss()
optimizer = optim.Adam(model.parameters(), lr=0.001)

# Train the model
num_epochs = 100
for epoch in range(num_epochs):
    model.train()
    running_loss = 0.0
    for inputs, labels in train_loader:
        optimizer.zero_grad()
        outputs = model(inputs)
        loss = criterion(outputs, labels.view(-1, 1))
        loss.backward()
        optimizer.step()
        running_loss += loss.item() * inputs.size(0)
    epoch_loss = running_loss / len(train_loader.dataset)
    print(f"Epoch {epoch+1}/{num_epochs}, Loss: {epoch_loss:.4f}
    ")
```

Listing 3.1: Model Training

Chapter 4

Model Evaluation

After training the model, it is essential to evaluate its performance on the test set. The accuracy of the model is calculated as follows:

```
# Evaluate the model
model.eval()
correct = 0
total = 0
with torch.no_grad():
    for inputs, labels in test_loader:
        outputs = model(inputs)
        predicted = torch.round(outputs)
        total += labels.size(0)
        correct += (predicted == labels.view(-1, 1)).sum().item()

accuracy = correct / total
print(f"Test Accuracy: {accuracy:.4f}")
```

Listing 4.1: Model Evaluation

Chapter 5

Visual Diagrams

Visual diagrams are useful to understand the data distribution and the correlations between different features. Below are some visualizations created using Matplotlib and Seaborn.

5.1 Correlation Heatmap

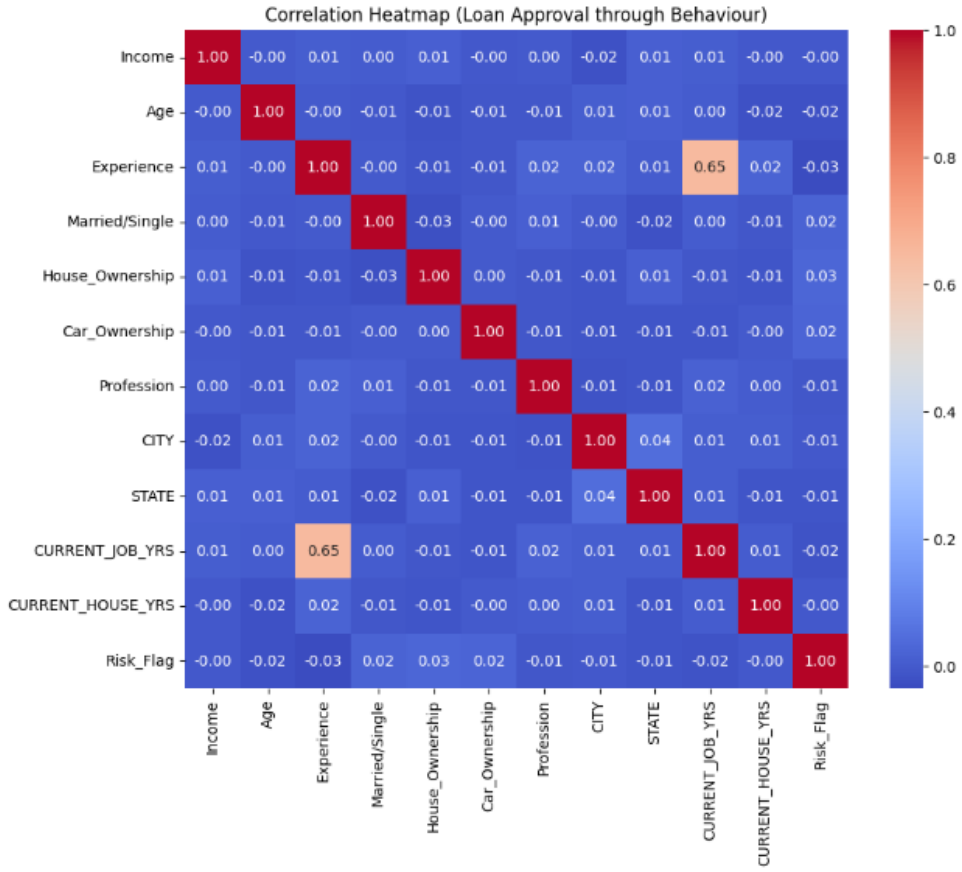

Figure 5.1: Correlation Heatmap (Loan Approval through Behaviour)

5.2 Feature Distributions

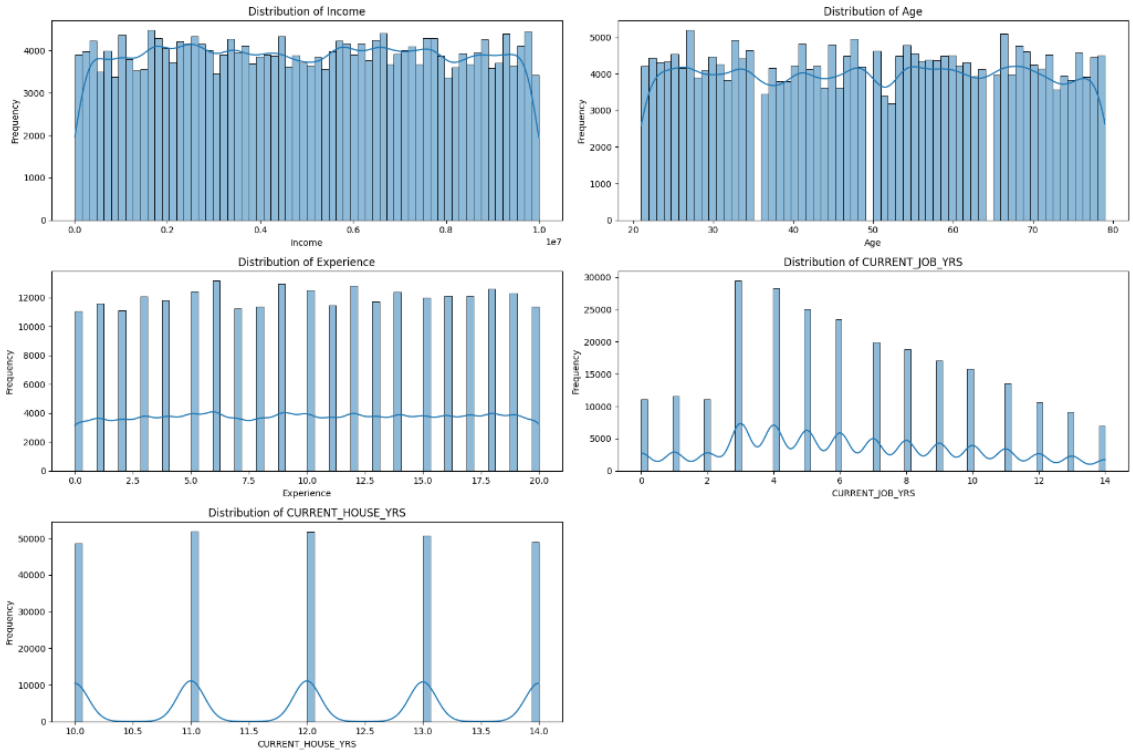

Figure 5.2: Distributions of Numeric Features

Chapter 6

Challenges Faced

6.1 Unbalanced Data

One of the significant challenges faced during this project was the unbalanced dataset. The target variable '$Risk_Flag$' had a disproportionate number of approved and unapproved loans, which could lead to a biased model.

6.2 Model Overfitting

Another common challenge was the potential for model overfitting. Given the complexity of neural networks, there was a risk of the model learning the training data too well and not generalizing effectively to new data.

Chapter 7

Conclusion

This project demonstrated the application of an Artificial Neural Network for predicting loan approval status. Despite challenges such as unbalanced data and potential overfitting, the model achieved a satisfactory level of accuracy. Future work could involve exploring more sophisticated models, addressing class imbalance through techniques such as SMOTE, and performing hyperparameter tuning to further enhance model performance.