# Financial Distress Risk Prediction

Rayyan Kazim

2025-10-10

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns



import sklearn
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score, confusion_matrix

import torch
import torch.nn as nn
from torch.autograd import Variable
import torchvision.transforms as transforms
import torchvision.datasets as dsets
from torch.utils.data import DataLoader, TensorDataset
```

### Introduction

In this project, we will consider a dataset from the UCI ML Repository. This dataset is called "Taiwanese Bankruptcy Prediction". This dataset incorporates the association between the Taiwan Stock Exchange business regulations, and bankruptcy. This data is collected from the Taiwan Economic Journal from the years 1999-2009. The purpose and goal of this project will be to successfuly build a model that can predict bankruptcy.

```
df = pd.read_csv("data.csv")
df
```

| | Bankrupt? | ROA(C) before interest and depreciation before interest | ROA(A) before interest and % after tax | ROA(B) b |
|---|---|---|---|---|
| 0 | 1 | 0.370594 | 0.424389 | 0.405750 |
| 1 | 1 | 0.464291 | 0.538214 | 0.516730 |
| 2 | 1 | 0.426071 | 0.499019 | 0.472295 |

| | Bankrupt? | ROA(C) before interest and depreciation before interest | ROA(A) before interest and % after tax | ROA(B) b |
|---|---|---|---|---|
| 3 | 1 | 0.399844 | 0.451265 | 0.457733 |
| 4 | 1 | 0.465022 | 0.538432 | 0.522298 |
| ... | ... | ... | ... | ... |
| 6814 | 0 | 0.493687 | 0.539468 | 0.543230 |
| 6815 | 0 | 0.475162 | 0.538269 | 0.524172 |
| 6816 | 0 | 0.472725 | 0.533744 | 0.520638 |
| 6817 | 0 | 0.506264 | 0.559911 | 0.554045 |
| 6818 | 0 | 0.493053 | 0.570105 | 0.549548 |

```
df.shape
```

```
(6819, 96)
```

In this dataset, the target variable is "Bankrupt?" which is being predicted by 95 variables.

### Data Splitting, Testing and Training

```
X = df.drop(["Bankrupt?"], axis=1)
Y = df["Bankrupt?"]
```

```
X_train, X_test, Y_train, Y_test = train_test_split(
    X, Y, test_size = 0.25, random_state=1, stratify=Y)
```

```
X_test_tensor = torch.from_numpy(X_test.to_numpy(dtype=np.float32))
Y_test_tensor = torch.from_numpy(Y_test.to_numpy(dtype=np.float32)).view(-1, 1)

X_train_tensor = torch.from_numpy(X_train.to_numpy(dtype=np.float32))
Y_train_tensor = torch.from_numpy(Y_train.to_numpy(dtype=np.float32)).view(-1, 1)
```

### Neural Network Construction

We will construct a Neural Network for out classification problem. Our input layer will consist of our 95 predictor variables, which will be passed on to a connected hidden layer with 128 neurons. This hidden layer will use ReLU activation, outputting 128 neurons before passing them on to the second hidden layer, which will convert them into 64 neurons. ReLU activation is applied again, passing on the 64 neurons to the output layer, which will output one value for each input sample.

```
train_dataset = TensorDataset(X_train_tensor, Y_train_tensor)
train_loader = DataLoader(train_dataset, batch_size = 128, shuffle = True)

class NeuralNetwork(nn.Module):
    def __init__(self):
        super().__init__()
        self.flatten = nn.Flatten()
        self.linear_relu_stack = nn.Sequential(
            nn.Linear(95, 128),
            nn.ReLU(),
            nn.Linear(128, 64),
            nn.ReLU(),
            nn.Linear(64, 1),
        )

    def forward(self, x):
        x = self.flatten(x)
        logits = self.linear_relu_stack(x)
        return logits

model = NeuralNetwork()
```

For this binary classification problem, we will use Binary Cross Entropy, as well as an Adam optimizer because of its fast training speed.

```
criterion = nn.BCEWithLogitsLoss()
optimizer = torch.optim.Adam(model.parameters(), lr=0.0001)
```

### Neural Network Training & Performance

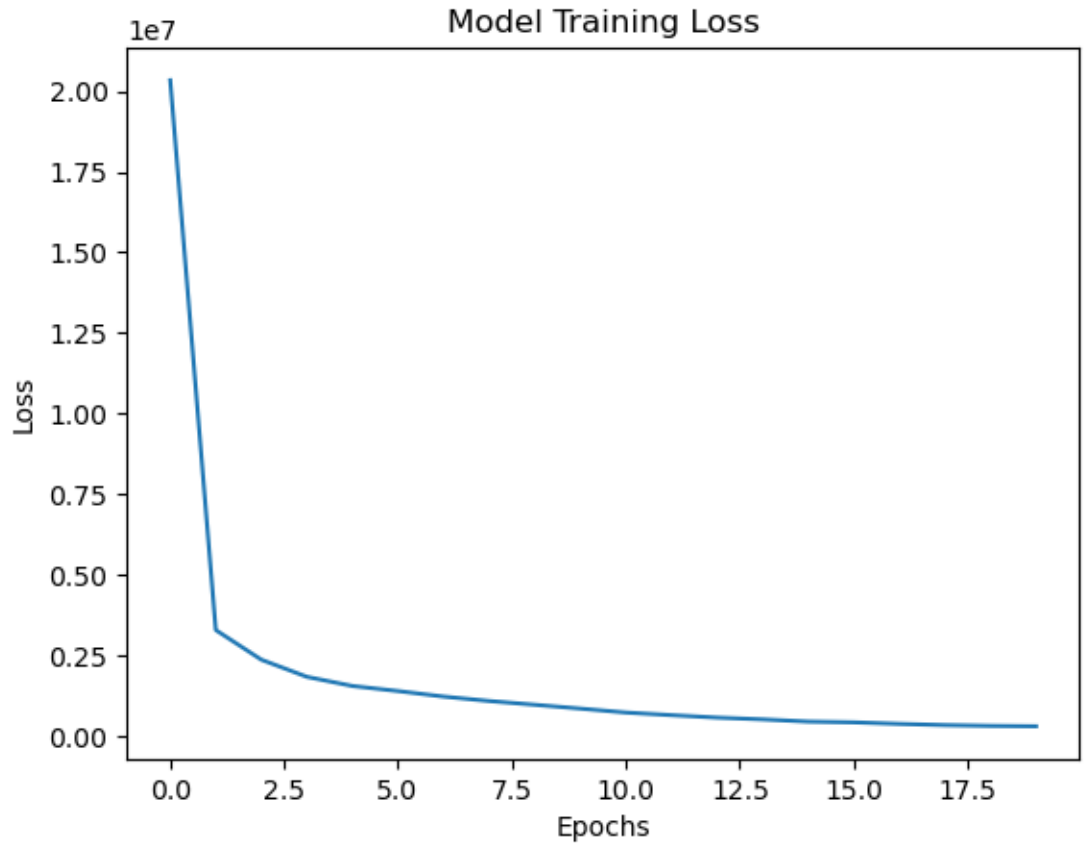
We will train our neural network on 20 epoch's with a learning rate of 0.0001 to allow the training process to remain stable and careful. We will also find the test accuracy of this model to ensure that it predicts the target variable efficiently.

```
epochs = 20
loss_values = []

for epoch in range(epochs):
    model.train()
    running_loss = 0.0
```

```
    for batch_X, batch_Y in train_loader:
        batch_Y = batch_Y.view(-1, 1).float()
        optimizer.zero_grad()
        prediction = model(batch_X)
        loss = criterion(prediction, batch_Y)

        if torch.isnan(loss) or torch.isinf(loss):
            print("NaN or Inf loss detected. Skipping batch.")
            continue

        loss.backward()
        optimizer.step()
        running_loss += loss.item()

    avg_loss = running_loss / len(train_loader)
    loss_values.append(avg_loss)
    print(f"Epoch {epoch+1}/{epochs}, Loss: {avg_loss:.4f}")

plt.plot(range(epochs), loss_values)
plt.title("Model Training Loss")
plt.xlabel("Epochs")
plt.ylabel("Loss")
plt.show()

################################################################################

model.eval()
with torch.no_grad():
    outcome = model(X_test_tensor)
    probabilities = torch.sigmoid(outcome)
    predictions = (probabilities >= 0.5).float()
    predictions = predictions.view(-1)

accuracy = accuracy_score(Y_test_tensor.cpu().numpy(), predictions.numpy())
print(f"Accuracy: {accuracy*100:.2f}%")
```

```
Epoch 1/20, Loss: 20338254.0691
Epoch 2/20, Loss: 3288552.3874
Epoch 3/20, Loss: 2369019.7437
```

```
Epoch 4/20, Loss: 1837804.0434
Epoch 5/20, Loss: 1554058.1938
Epoch 6/20, Loss: 1396611.4930
Epoch 7/20, Loss: 1225586.2598
Epoch 8/20, Loss: 1088473.9603
Epoch 9/20, Loss: 971945.6727
Epoch 10/20, Loss: 853389.5535
Epoch 11/20, Loss: 730779.3038
Epoch 12/20, Loss: 650020.7018
Epoch 13/20, Loss: 573789.1766
Epoch 14/20, Loss: 518874.3519
Epoch 15/20, Loss: 449769.8417
Epoch 16/20, Loss: 426757.0166
Epoch 17/20, Loss: 379271.1191
Epoch 18/20, Loss: 341149.9221
Epoch 19/20, Loss: 317807.4110
Epoch 20/20, Loss: 305748.1428
Accuracy: 94.90%
```



:

## Conclusion

Our model performed well with predicting Bankruptcy. Based on the training loss plot, we can see a very large decrease in the loss, indicating that the model was well trained and was able to learn. Our model had a test accuracy of 94.9%, implying that majority of the predictions that it made (for Bankruptcy) were correct.