University of Tehran

School of Electrical and Computer Engineering

Machine Learning

Homework 3: Neural Network and Decision Tree

Author:
Alireza Javid

Student Number:
810198375

Contents

1 Problem 1: Basics of Neural Network

(a) Cost functions and activation functions are essential elements of a neural network. The cost function measures the difference between predicted and actual output, and it helps to adjust the weights and biases during training. The activation function is applied to the output of each neuron, and it decides whether the neuron should be activated or not. The activation function is responsible for adding non-linearity to the neural network, enabling it to solve complex problems. The selection of the cost function and activation function is based on the problem and network architecture. Different types of problems require different cost functions, and different activation functions can perform better on different data types. It is crucial to choose the right cost and activation functions for optimal neural network performance.

(b) The initial values of the weights and biases in a neural network can impact the final results by affecting the connections between neurons. If the initial values are set too high or too low, the network may not be able to converge to the optimal solution during training. Conversely, if the initial values are set close to the optimal solution, the network may converge quickly, reducing the training time. Random initialization can lead to suboptimal solutions or slow convergence. Techniques such as Xavier and He initialization can be used to set appropriate initial values. Regularization can also help prevent overfitting and improve generalization performance. The impact can be controlled and mitigated through proper initialization and regularization techniques.

(c) Forward propagation is the process of generating an output from the input data, while backward propagation is the process of updating the weights of the network to reduce the error between the predicted and actual output. During forward propagation, each neuron in the network receives input from the neurons in the previous layer applies its activation function, and passes its output to the neurons in the next layer. The final output of the network is generated by the output layer. Conversely, backward propagation starts with computing the loss or error between the output generated by the network and the actual output that we want the network to produce. Then, the error is propagated backward through the network layer by layer using the chain rule of calculus. At each layer, the error is used to compute the gradients of the weights and biases, which are used to update the parameters using an optimization algorithm such as gradient descent.

Both forward and backward propagation are essential for training a neural network and improving its performance.

(d) The learning rate is a hyperparameter that controls the step size of weight updates during training. A learning rate that is too high may cause the network to overshoot the optimal solution and become unstable, while a learning rate that is too low may cause the network to converge too slowly. It is important to choose an appropriate learning rate for the specific task and network architecture being used. One approach is to use a learning rate schedule or adaptive learning rate algorithms such as Adam, RMSProp, or Adagrad, which can be more effective than using a fixed learning rate.

(e) The depth of a neural network refers to the number of layers it has, and a deeper network is generally better than a shallower one due to increased representational power, hierarchical feature learning, better performance on large datasets, and improved regularization. However, increasing the depth also increases the risk of overfitting and requires more computational resources and training data. The appropriate depth should be chosen based on the task, dataset, and available resources, and techniques such as dropout, early stopping, and regularization can be used to mitigate overfitting.

(f) The biggest drawback with the training process of neural networks is ***overfitting***. Overfitting is a problem that occurs when a neural network becomes too complex and fits the training data too closely, leading to a poor generalization of new data. To address this problem, several techniques can be used, including regularization, dropout, early stopping, data augmentation, and reducing the complexity of the model. These techniques can help prevent the model from memorizing noise in the training data and encourage it to learn more meaningful and generalizable patterns.

2 Problem 2: MLP Network

(a) The input layer should contain two neurons for x_1 and x_2. The first three neurons of the first hidden layer are used to specify triangular area and the other five are used for pentagon area. The first neuron for the second hidden layer is used to concatenate the previous layers that had been used for specifying the triangular area and the second neuron is for the pentagon area. In the output layer we "OR" two polygons.

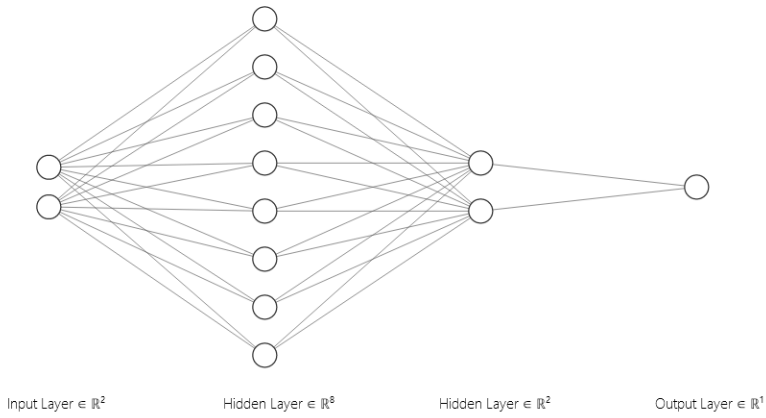


Figure 1: Neural Network Architecture for Specifying Selected Area in Part 1

(b) Again, the input layer should contain two neurons for x_1 and x_2. The first hidden layer is used to specify the given hexagon shape(we have six neurons for this task) and in the output layer we "And" selected areas.

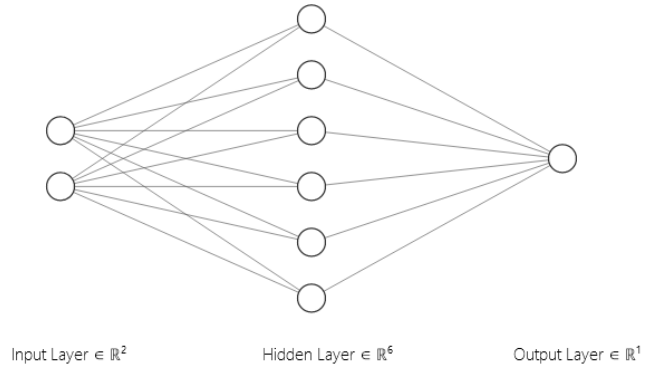


Figure 2: Neural Network Architecture for Specifying Selected Area in Part 2

(c) This is the classic "XOR" problem. Once Again, the input layer should contain two neurons for x_1 and x_2. The first hidden layer separates the middle area (blue class) from the other two (red class) and in the output layer we "OR" red class areas.

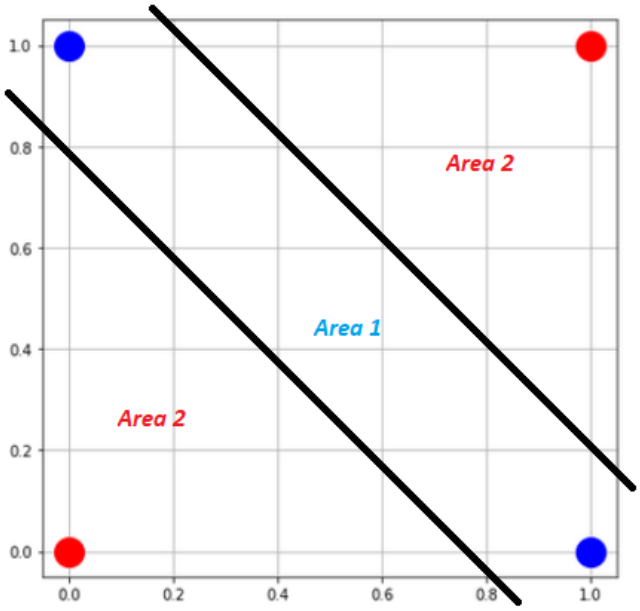


Figure 3: Area Separation in Part 3

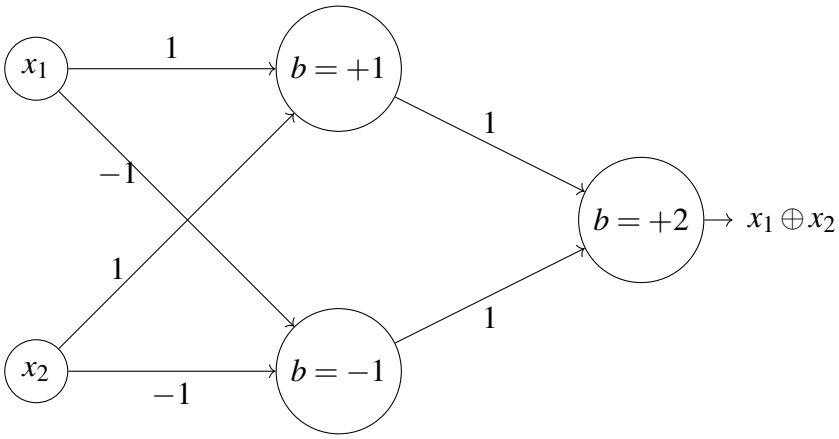


Figure 4: Neural network architecture for XOR problem

In Figure 3 we depict the separation area and Figure 4 indicates the proposed network.

3 Problem 3: CNN Network

(a) We can observe that the first class is smother than the second class. Another thing, the second class is comparatively sparser than the first, indicating a greater number of zero values and lower density.

(b) The below picture shows the result of convolution with the given filter.

1	3
2	6

(a) Convolution of Figure 1

5	1
5	1

(b) Convolution of Figure 2

1	2
2	2

(c) Convolution of Figure 3

1	2
1	1

(d) Convolution of Figure 4

Figure 5: Convolution of 4 Figures

After Maxpooling we have:

6	5
2	2

Figure 6: Final Result of 4 Figures

(c) We can normalize the final value by dividing it by the maximum value achievable with the given filter (in this case, 6). The resulting normalized value, denoted as x, can be used to determine the class label as follows:

$$\begin{cases} x > 0.5 \rightarrow class\,1 \\ otherwise \rightarrow class\,2 \end{cases}$$

From this decision rule, we correctly distinguish the four figures.

4 Problem 4: Activation Functions

(a) The RELU function can describe as follows:

$$f(x) = \max(0, x)$$

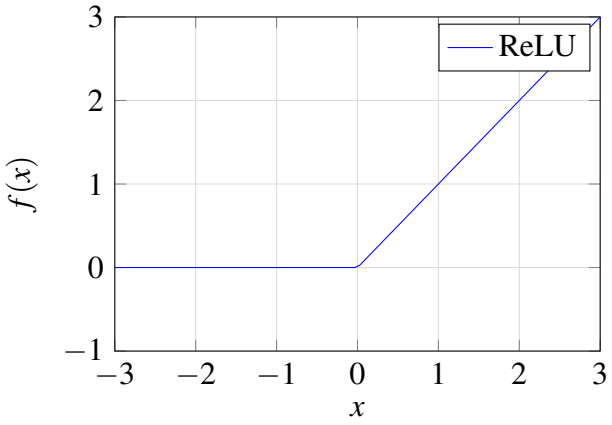


Figure 7: ReLU Plot

As we can see it is simpler and computationally cheaper than the sigmoid and tanh activation functions because it only involves a comparison and a single multiplication operation. Its derivative is also easy to compute, making it better for gradient-based optimization algorithms during neural network training. The sigmoid and tanh functions are more complex and computationally expensive, which can slow down training, and they also have vanishing gradients, which makes training difficult. Therefore, ReLU is preferred in neural networks due to its simplicity, computational efficiency, and ease of optimization during training.

(b) The vanishing gradient problem is when gradients become very small, slowing down or preventing neural network training. As the gradients are backpropagated through the layers, they are multiplied by the derivative of the activation function at each layer, which can cause the gradient to become very small. Activation functions like sigmoid and tanh are prone to this issue due to the saturation of their outputs at extreme input values, resulting in vanishing gradients. ReLU avoids this problem because its derivative is either 0 or 1 and does not depend on the input value. This ensures that gradients do not become too small, allowing for effective learning. As a result, ReLU is a popular choice for deep neural networks that require many layers to capture complex patterns in data.

(c) Sigmoid function with binary cross entropy loss. In classification problems, the sigmoid function is used in the final layer of a neural network to obtain a probability value between 0 and 1, representing the likelihood of the input belonging to the positive class. For multi-class classification, the softmax activation function is commonly used in the final layer to ensure that the outputs sum up to 1 and represent the probabilities of each class. In contrast, the ReLU activation function is typically used in the hidden layers of the network to introduce non-linearity and enhance the model's expressive power.

(d) The ReLU activation function is simple and effective but suffers from the *"dying ReLU"* problem, where the output is always zero, which can happen during training if the weights are not initialized properly or if the learning rate is too high. Leaky ReLU is a variation that returns a small, non-zero value for negative inputs, preventing this issue and defined as below:

$$f(x) = \begin{cases} x, & \text{if } x > 0 \\ \alpha x, & \text{otherwise} \end{cases}$$

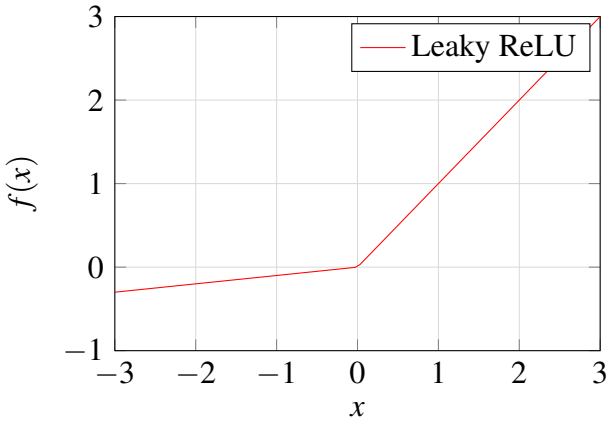


Figure 8: Leaky ReLU Plot

where α is a small constant (usually around 0.01).

It can improve neural network performance and prevent overfitting. However, its effectiveness can depend on the specific problem being solved.

Overall, Leaky ReLU is preferred over regular ReLU in situations where the "dying ReLU" problem occurs during training, or when there is a concern about overfitting. However, the effectiveness of Leaky ReLU depends on the specific problem and may not always be the best choice.

5 Problem 5: Decision Tree

Table 1: Employment, Education, Gender, and Work Experience

Employment (X)	University Degree (U)	Gender (S)	Work Experience (W)
Rejected	B.Sc.	Male	1
Accepted	B.Sc.	Female	2
Rejected	Ph.D.	Male	0
Accepted	B.Sc.	Male	2
Rejected	M.Sc.	Male	1
Rejected	Ph.D.	Female	1
Accepted	M.Sc.	Female	0
Accepted	B.Sc.	Female	1
Rejected	M.Sc.	Male	0
Rejected	M.Sc.	Male	2

(a)

$$\begin{cases} rejected = 6 \\ accepted = 4 \end{cases}$$

If X indicates hiring, we have:

$$H(X) = H(\frac{6}{10}, \frac{4}{10}) = -\sum_i p(x_i) log(p(x_i)) = -\frac{6}{10} log(\frac{6}{10}) - \frac{4}{10} log(\frac{4}{10}) = 0.971$$

(b) If U indicates university degree and W indicates work experience, we have:

$$I(X;U) = H(X) - H(X|U) \quad I(X;W) = H(X) - H(X|W)$$

For conditional entropy, we have:

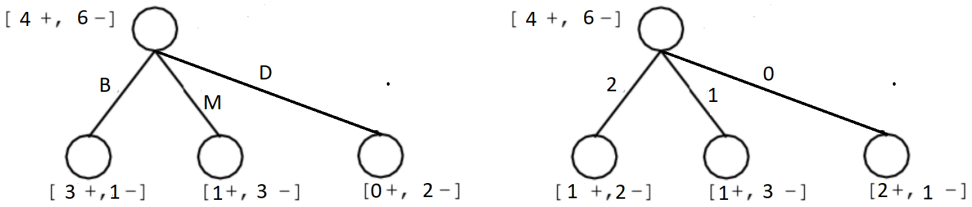
$$H(Y|X) = \sum_i H(Y|X = x_i) P(X = x_i)$$

Thus, for calculating mutual information we have:

$$H(X|U) = \frac{4}{10} H(\frac{3}{4}, \frac{1}{4}) + \frac{4}{10} H(\frac{1}{4}, \frac{3}{4}) + \frac{2}{10} H(1,0) = 0.649$$

$$H(X|W) = \frac{3}{10} H(\frac{2}{3}, \frac{1}{3}) + \frac{4}{10} H(\frac{1}{4}, \frac{3}{4}) + \frac{3}{10} H(\frac{1}{3}, \frac{2}{3}) = 0.875$$

$$I(X;U) = H(X) - H(X|U) = 0.322 \quad I(X;W) = H(X) - H(X|W) = 0.096$$



(a) Decision Tree Base on U (b) Decision Tree Base on W

Figure 9: Decision Trees

(c) By applying the majority rule we have this confusion matrix:

Predicted Values	Actual Values: Positive (1)	Actual Values: Negative (0)
Positive (1)	TP =2	FP =1
Negative (0)	FN =2	TN =5

Figure 10: Confusion Matrix for Decision Tree Base on W

Now, we calculate questioned variables:

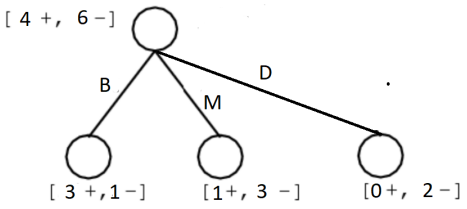
$$\text{Precision} = \frac{\sum \text{TP}}{\sum \text{TP} + \text{FP}} = \frac{2}{3}$$

$$\text{Recall} = \frac{\sum \text{TP}}{\sum \text{TP} + \text{FN}} = \frac{2}{4}$$

$$\text{Accuracy} = \frac{\sum \text{TP} + \text{TN}}{\sum \text{TP} + \text{FP} + \text{FN} + \text{TN}} = \frac{7}{10}$$

$$\text{Specificity} = \frac{\sum \text{TN}}{\sum \text{FP} + \text{TN}} = \frac{5}{6}$$

(d) Here we repeat our calculation for a decision tree based on sex.



(a) Decision Tree Base on Sex

		Actual Values: Positive (1)	Actual Values: Negative (0)
Predicted Values	Positive (1)	TP = 3	FP =1
Predicted Values	Negative (0)	FN = 1	TN =5

(b) Confusion Matrix for Decision Tree Base on Sex

Figure 11: Decision Tree and Confusion Matrix Base on Sex

$$\text{Precision} = \frac{\sum \text{TP}}{\sum \text{TP} + \text{FP}} = \frac{3}{4}$$

$$\text{Recall} = \frac{\sum \text{TP}}{\sum \text{TP} + \text{FN}} = \frac{3}{4}$$

$$\text{Accuracy} = \frac{\sum \text{TP} + \text{TN}}{\sum \text{TP} + \text{FP} + \text{FN} + \text{TN}} = \frac{8}{10}$$

$$\text{Specificity} = \frac{\sum \text{TN}}{\sum \text{FP} + \text{TN}} = \frac{5}{6}$$

6 Problem 6: ECG Classification

(a) We can use the *value_counts*() function for this purpose.

```
N    5992
O    3151
A     923
~     187
Name: label, dtype: int64
```

Figure 12: The Number of Data Related to All 4 Classes

(b) Figure 11 depicts the histogram plot for each class.

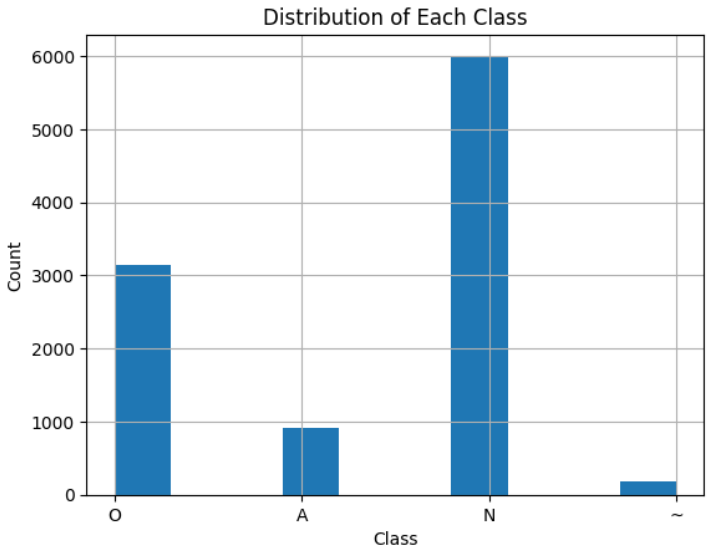


Figure 13: The Distribution of 4 Classes

It is evident that the dataset contains limited data for the $\sim$ and A classes. Therefore, the reliability of our decisions regarding these classes is not substantial. On the other hand, we have a significant amount of data for the O and N classes, and our decisions regarding them hold more validity. The presence of asymmetric data distribution can bias our model to certain classes and reduce accuracy.

(c) We have designed a 4 layers model for the classification and our results are:

```
33/33 [==============================] - 0s 2ms/step
Classification Report:
              precision    recall  f1-score   support

           O       0.82      0.79      0.80       590
           A       0.55      0.70      0.62       321
           N       0.52      0.29      0.37        98
           ~       0.00      0.00      0.00        17

    accuracy                           0.70      1026
   macro avg       0.47      0.44      0.45      1026
weighted avg       0.69      0.70      0.69      1026
```

Figure 14: Classification Report

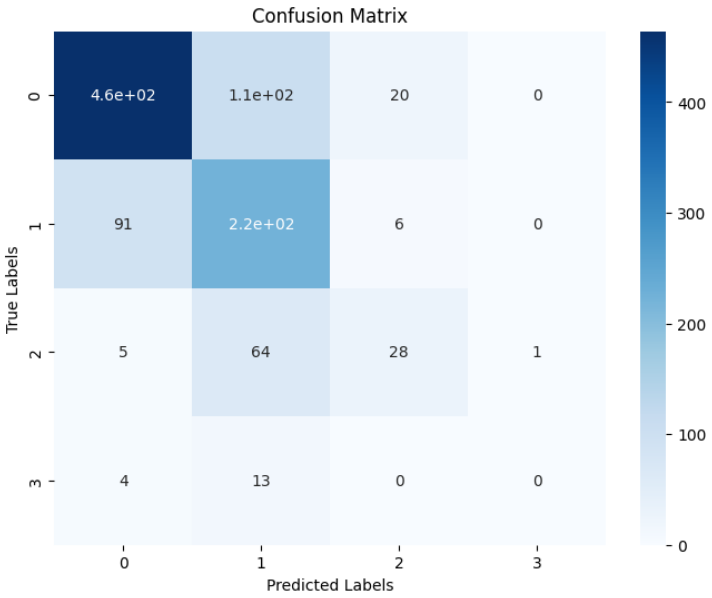


Figure 15: Confusion Matrix

The network performance demonstrates a clear distinction between the classification of the O and N classes, which yields the highest accuracy, and the classification of the $\sim$ and A classes, which yield the lowest accuracy. This discrepancy can be attributed to the asymmetric distribution of data, which introduces a bias towards the O and N classes. Consequently, the accuracy for the $\sim$ and A classes is adversely affected and remains relatively low.

(d) We can use the *StandardScaler*() function for this purpose.

Classification Report:

	precision	recall	f1-score	support
O	0.89	0.91	0.90	590
A	0.81	0.79	0.80	321
N	0.88	0.85	0.86	98
~	0.44	0.41	0.42	17
accuracy			0.85	1026
macro avg	0.75	0.74	0.75	1026
weighted avg	0.85	0.85	0.85	1026

Figure 16: Classification Report After Normalization

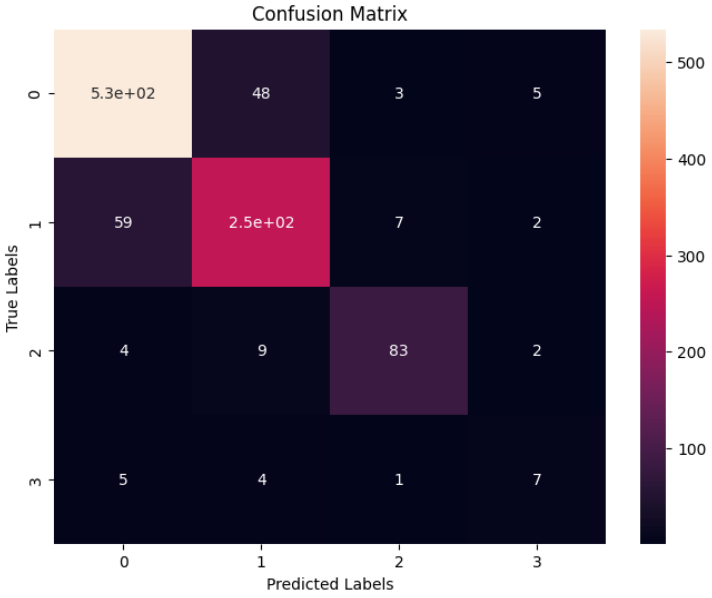


Figure 17: Confusion Matrix After Normalization

(e) Normalization improves the performance of the network for several reasons:

- **Equalizing the scales**: Different features in the dataset may have different scales or ranges. By normalizing the data, we bring all features to a similar scale, ensuring that no single feature dominates the learning process. This helps the network to learn from all features more effectively.
- **Facilitating convergence**: Normalization can help the network converge faster during training. When the input data is normalized, the optimization process can reach the optimal solution more quickly, as the gradient descent algorithm can navigate the parameter space more efficiently.
- **Preventing numerical instability**: In some cases, large input values can cause numerical instability in the network, leading to issues such as exploding or vanishing gradients. Normalization helps to mitigate these problems by keeping the values within a manageable range.
- **Improving generalization**: Normalization can aid in improving the generalization performance of the network. By reducing the impact of outliers

and extreme values, the network becomes more robust to variations in the input data and can better generalize to unseen samples.

Overall, normalization ensures that the network receives well-scaled and stable inputs, enabling it to learn more effectively, converge faster, and make better generalizations.

(f) In this part, we change our dataset as below:

```
N     5992
AN    4074
Name: label, dtype: int64
```

Figure 18: The Number of Data Related to 2 Classes

In the output layer of this part, we should have only 2 neurons. The results are:

```
Classification Report:
              precision    recall  f1-score   support

          AN       0.90      0.91      0.91       608
           N       0.87      0.85      0.86       399

    accuracy                           0.89      1007
   macro avg       0.89      0.88      0.88      1007
weighted avg       0.89      0.89      0.89      1007
```

Figure 19: Model Architecture

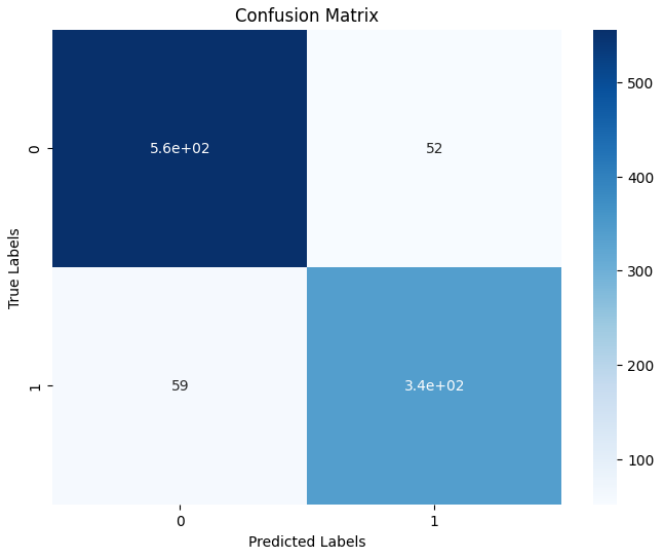


Figure 20: Confusion Matrix

Our model demonstrates enhanced performance and improved accuracy in this particular section.

You can see the code details in the attached file.

7 Problem 7: MADALINE Network

(a) We make a dataset and its columns are X coordination, Y coordination, and the data label of each data. Using *scatterplot* we can depict the distribution of the data:

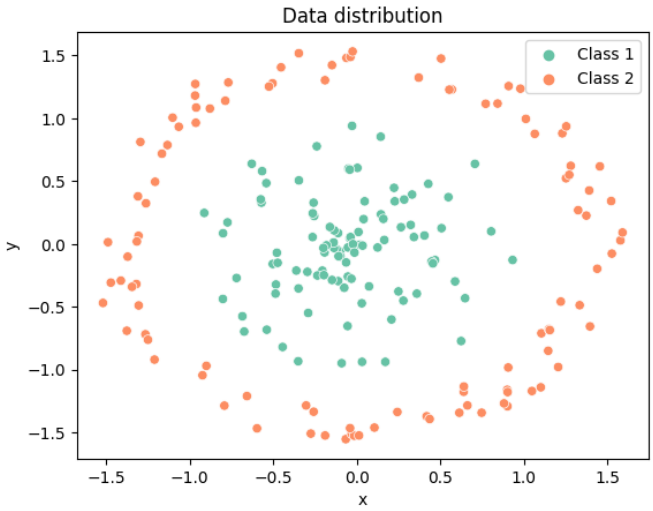


Figure 21: Distribution of the Data

(b) The MadaLine algorithm is an enhanced version of the AdaLine algorithm. It addresses the issue of the linear inseparability of classes by utilizing multiple AdaLine classifiers combined through logical OR or AND operations. It can classify classes that form a single space or convex hyperspace.

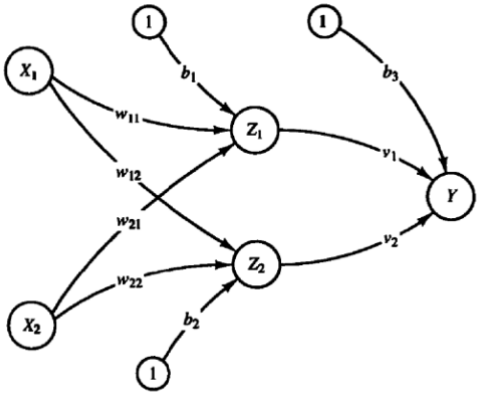


Figure 22: MadaLine Network

We have used the MRI algorithm for the MadaLine network implementation. The activation function we have used is:

$$f(x) = \begin{cases} 1 & if\, x \geq 0 \\ -1 & if\, x < 0 \end{cases}$$

You can see details of the implementation of the Madaline network in [1].

After implementation of this network, our result with 3 neurons is:

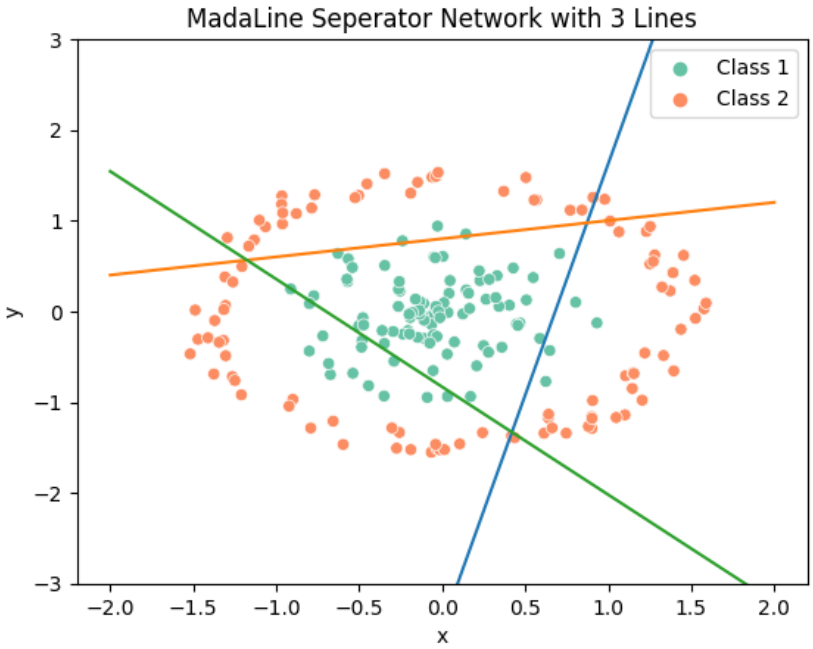


Figure 23: Dataset and Decision Hyperplane with 3 Neurons

Our model achieved an accuracy of 89.5% after completing 100 epochs.

(c) We repeat this method with 6 neurons:

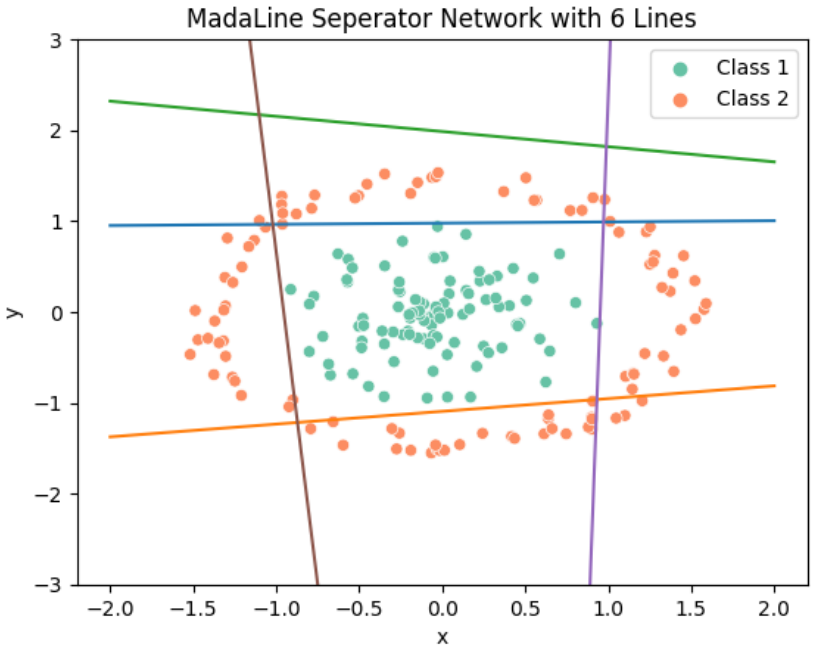


Figure 24: Dataset and Decision Hyperplane with 6 Neurons

Our model achieved an impressive accuracy of 100% after completing 35 epochs. We can see our algorithm is much faster and more accurate with 6 neurons.

(d) Increasing the number of neurons in the Madaline network's hidden layer improved performance and accuracy. This expansion enhanced the network's ability to capture complex patterns, leading to faster convergence and improved speed and accuracy. The increased parameters allowed for a better representation of input features and improved discrimination between classes. The network optimized its weights and biases more efficiently, resulting in faster learning and improved generalization. In this problem with using 4 neurons we would separate two classes and as we can see in figure 23, two neurons are surplus. Overall, increasing the number of neurons in the hidden layer proved beneficial in enhancing the Madaline network's performance and accuracy.

You can see the code details in the attached file.

8 Problem 8: Classification of Cifar10 Images Using CNN Network

(a) We can randomly select some of the pictures and display them.

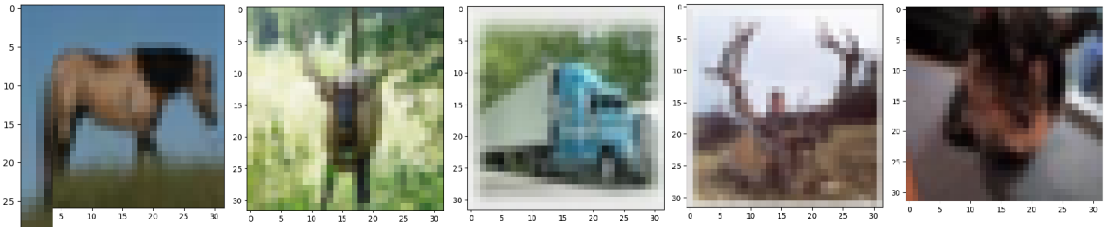


Figure 25: Random Pictures

(b) We divide both train and test data by 255 (the maximum value) and then convert class vectors to binary class matrices which are called one hot encoding.

(c) We design the following network:

```
Model: "sequential_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_6 (Conv2D)            (None, 30, 30, 32)        896

conv2d_7 (Conv2D)            (None, 28, 28, 32)        9248

conv2d_8 (Conv2D)            (None, 26, 26, 32)        9248

max_pooling2d_2 (MaxPooling  (None, 13, 13, 32)        0
2D)

dropout_3 (Dropout)          (None, 13, 13, 32)        0

conv2d_9 (Conv2D)            (None, 11, 11, 64)        18496

conv2d_10 (Conv2D)           (None, 9, 9, 64)          36928

conv2d_11 (Conv2D)           (None, 7, 7, 64)          36928

max_pooling2d_3 (MaxPooling  (None, 3, 3, 64)          0
2D)

dropout_4 (Dropout)          (None, 3, 3, 64)          0

flatten_1 (Flatten)          (None, 576)               0

dense_2 (Dense)              (None, 512)               295424

dropout_5 (Dropout)          (None, 512)               0

dense_3 (Dense)              (None, 10)                5130

=================================================================
Total params: 412,298
Trainable params: 412,298
Non-trainable params: 0
```

Figure 26: Model Architecture

We compile this network with the following optimizers:

- **SGD** : Using this optimizer, we get 77% accuracy for train data, 73.3% accuracy for validation data, and 72.5% accuracy for test data. We plot the change in loss and accuracy during training.

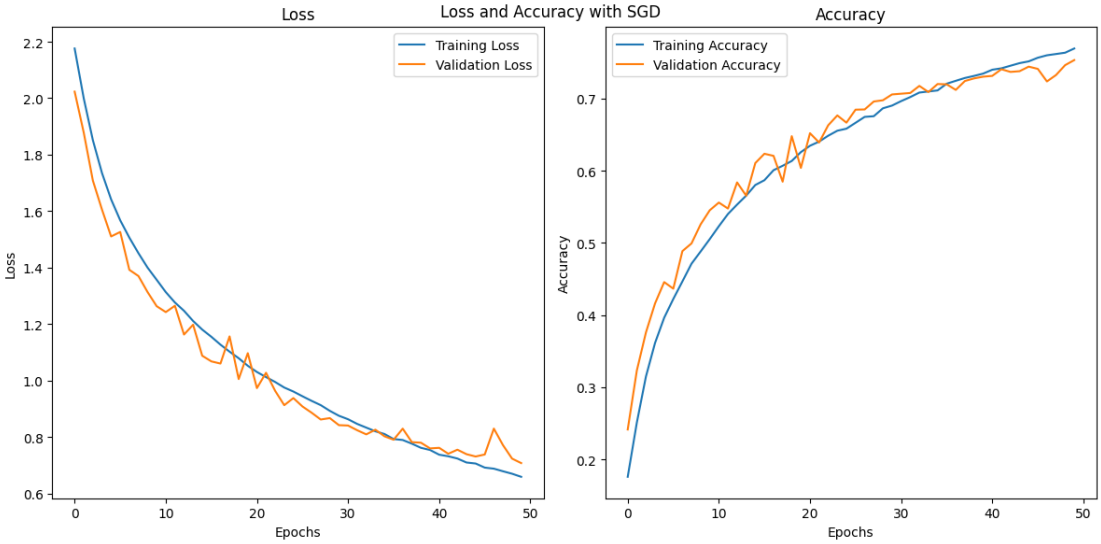


Figure 27: Change in Loss and Accuracy with SGD with epoch

- **Adam** : Using this optimizer, we get 74.5% accuracy for train data, 73.2% accuracy for validation data, and 72.5% accuracy for test data. We plot the change in loss and accuracy during training.

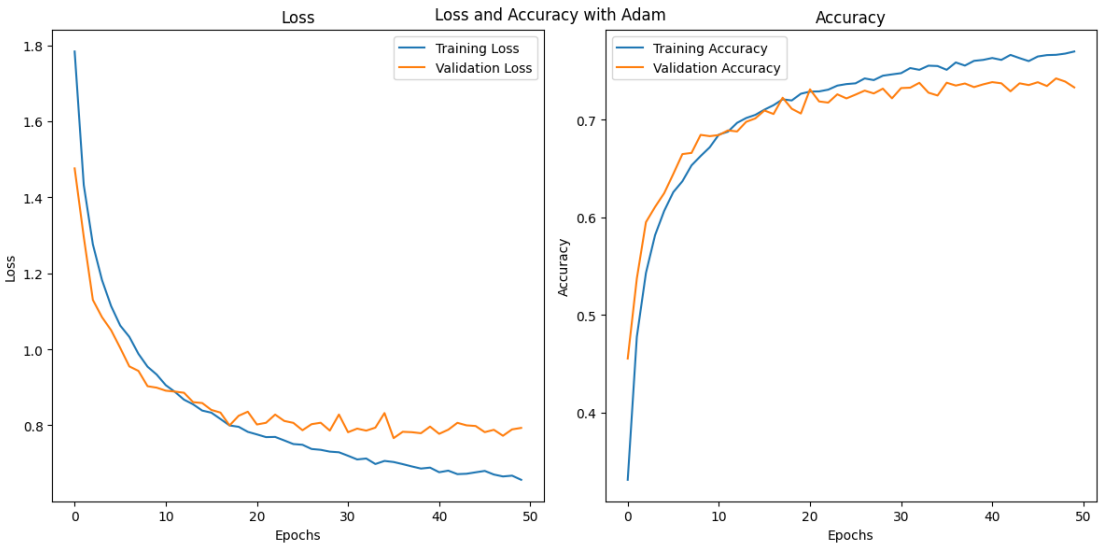


Figure 28: Change in Loss and Accuracy with Adam with epoch

- **RMSProp** : Using this optimizer, we get 76.7% accuracy for train data, 73.9% accuracy for validation data, and 73.2% accuracy for test data. We plot the change in loss and accuracy during training.

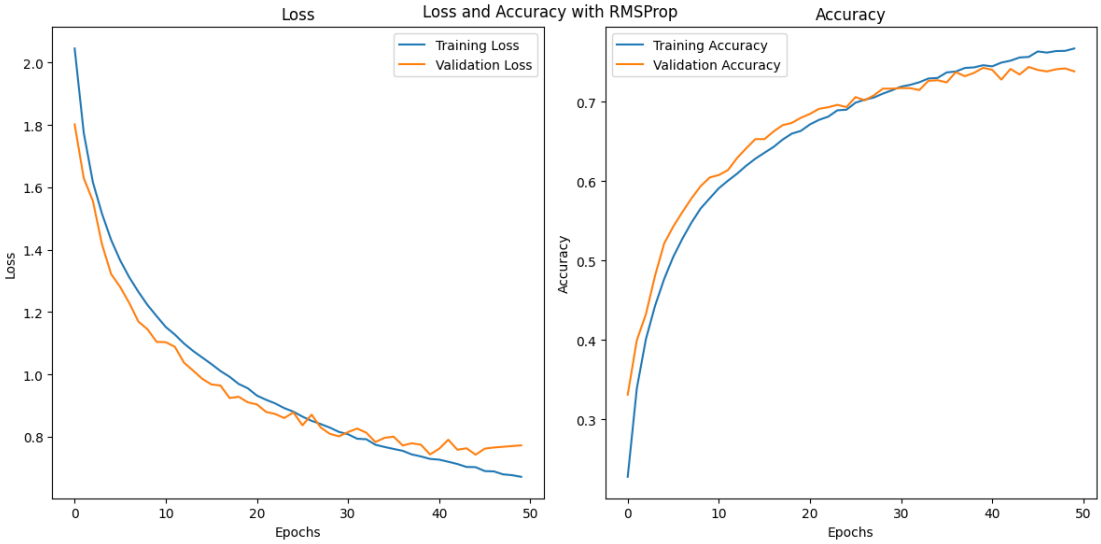


Figure 29: Change in Loss and Accuracy with RMSProp with epoch

It is noticeable, the convergence rate is slower in SGD and faster in Adam. The final accuracy is about the same, however, Adam reveals a faster solution but a little less accurate.

(d) The classification report for each solver, for training and test data, can be demonstrated below:

Classification Report (Test Data):

	precision	recall	f1-score	support
airplane	0.81	0.73	0.76	1000
automobile	0.88	0.86	0.87	1000
bird	0.67	0.61	0.64	1000
cat	0.55	0.60	0.57	1000
deer	0.71	0.69	0.70	1000
dog	0.64	0.69	0.66	1000
frog	0.78	0.84	0.81	1000
horse	0.79	0.76	0.77	1000
ship	0.83	0.84	0.84	1000
truck	0.84	0.85	0.84	1000
accuracy			0.75	10000
macro avg	0.75	0.75	0.75	10000
weighted avg	0.75	0.75	0.75	10000

Figure 30: Classification Report with SGD

Classification Report (Test Data):

	precision	recall	f1-score	support
airplane	0.78	0.71	0.74	1000
automobile	0.93	0.78	0.85	1000
bird	0.67	0.61	0.64	1000
cat	0.52	0.57	0.54	1000
deer	0.65	0.74	0.70	1000
dog	0.68	0.58	0.63	1000
frog	0.71	0.85	0.78	1000
horse	0.84	0.69	0.76	1000
ship	0.71	0.89	0.79	1000
truck	0.83	0.82	0.82	1000
accuracy			0.73	10000
macro avg	0.73	0.73	0.73	10000
weighted avg	0.73	0.73	0.73	10000

Figure 31: Classification Report with Adam

Classification Report (Test Data):

	precision	recall	f1-score	support
airplane	0.80	0.73	0.76	1000
automobile	0.91	0.83	0.87	1000
bird	0.69	0.54	0.61	1000
cat	0.55	0.50	0.52	1000
deer	0.72	0.72	0.72	1000
dog	0.54	0.79	0.64	1000
frog	0.75	0.84	0.79	1000
horse	0.78	0.76	0.77	1000
ship	0.89	0.76	0.82	1000
truck	0.79	0.85	0.82	1000
accuracy			0.73	10000
macro avg	0.74	0.73	0.73	10000
weighted avg	0.74	0.73	0.73	10000

Figure 32: Classification Report with RMSProp

9 Problem 9: Classification of Diabetes Dataset Using Decision Tree

(a) We can use the $DecisionTreeClassifier()$ and $train\_test\_split()$ functions for this purpose.

```
Accuracy on training data: 1.0
Accuracy on test data: 0.7012987012987013
```

Figure 33: Accuracy for Test and Train Data

```
Classification Report (Test Data):
              precision    recall  f1-score   support

           0       0.80      0.72      0.76       151
           1       0.56      0.66      0.61        80

    accuracy                           0.70       231
   macro avg       0.68      0.69      0.68       231
weighted avg       0.72      0.70      0.71       231
```

Figure 34: Classification Report

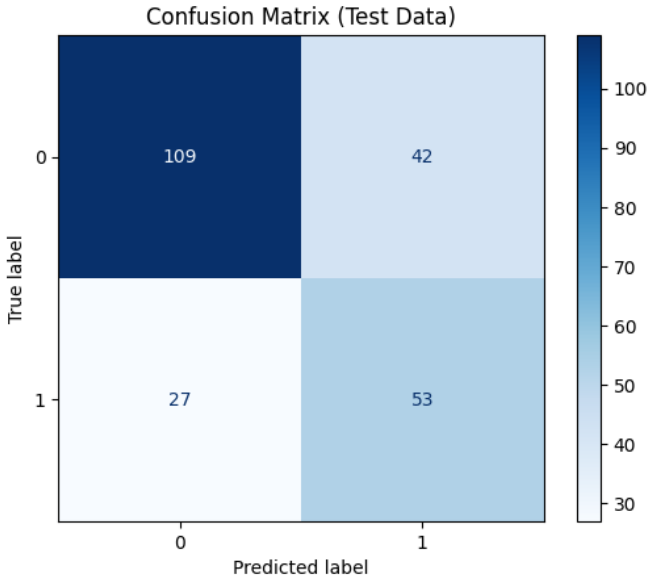


Figure 35: Confusion Matrix

It's obvious the classifier has over-fitted on train data and depicts high differences in train and test accuracy.

(b) We use the *graphviz* library to visualize the designed tree.

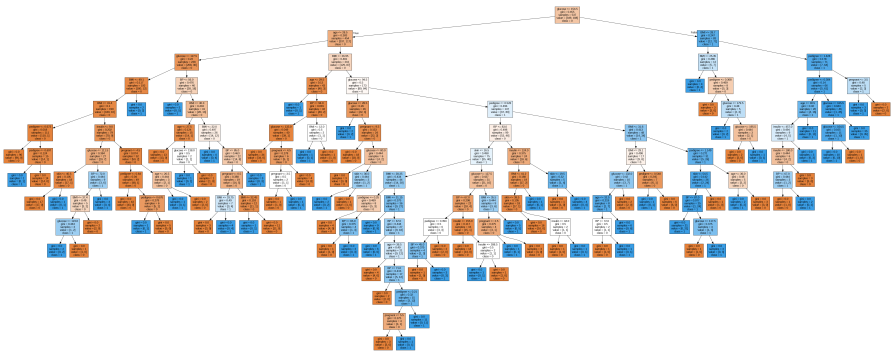


Figure 36: Decision Tree

Here, it's worth noting that the decision tree generated in its current form is unpruned. An unpruned tree tends to be complex and may not provide a clear and easily interpretable representation of the underlying decision-making process. This can make it challenging for users to understand and explain the decision tree's predictions.

(c) Pre-pruning, also known as early stopping, is a technique used in decision tree algorithms to prevent overfitting and improve generalization. It involves stopping the tree construction process before it becomes fully grown by introducing conditions to halt further splitting. Common pre-pruning techniques include setting a maximum depth for the tree, specifying a minimum number of samples per leaf, limiting the number of leaf nodes, requiring a minimum impurity decrease for splits, and using a validation set to monitor performance and stop growth. By limiting the complexity of the tree, pre-pruning helps control overfitting and improves the model's ability to generalize to unseen data. The choice of pre-pruning techniques and hyperparameter values should be based on experimentation and cross-validation.

(d) In this part, we prevent over-fitting and set a maximum value for the depth of the tree.

```
Accuracy on training data: 0.7597765363128491
Accuracy on test data: 0.7186147186147186
```

Figure 37: Accuracy for Test and Train Data

```
Classification Report (Test Data):
              precision    recall  f1-score   support

           0       0.72      0.92      0.81       151
           1       0.69      0.34      0.45        80

    accuracy                           0.72       231
   macro avg       0.71      0.63      0.63       231
weighted avg       0.71      0.72      0.69       231
```

Figure 38: Classification Report

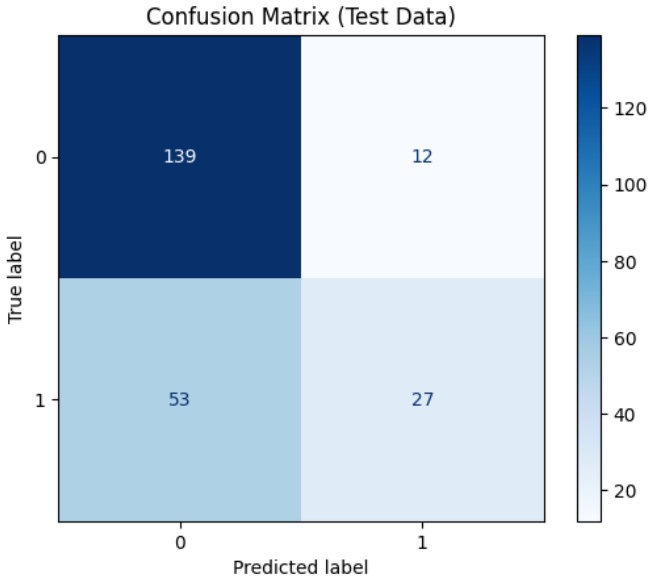


Figure 39: Confusion Matrix

The decrease in training accuracy suggests that our model has become simpler and more general, striking a balance between bias and variance. This trade-off is a fundamental concept in machine learning. While reducing model complexity introduces some bias, resulting in lower training accuracy, it often leads to improved performance on unseen data. This indicates a reduction in overfitting and better generalization. Consequently, despite the decrease in training accuracy, the overall outcome is positive, as the model becomes less prone to overfitting and more capable of performing well on new, unseen data.

(e) The pruned model is simpler, more explainable, and easier to interpret compared to the original decision tree model plot. By eliminating irrelevant branches and nodes, the pruned model captures the essential patterns and features in the

data more effectively. This simplification enhances the understandability of the model's decision-making process. Overall, the pruned model strikes a balance between complexity and simplicity, offering a clearer and more meaningful representation of the underlying data.

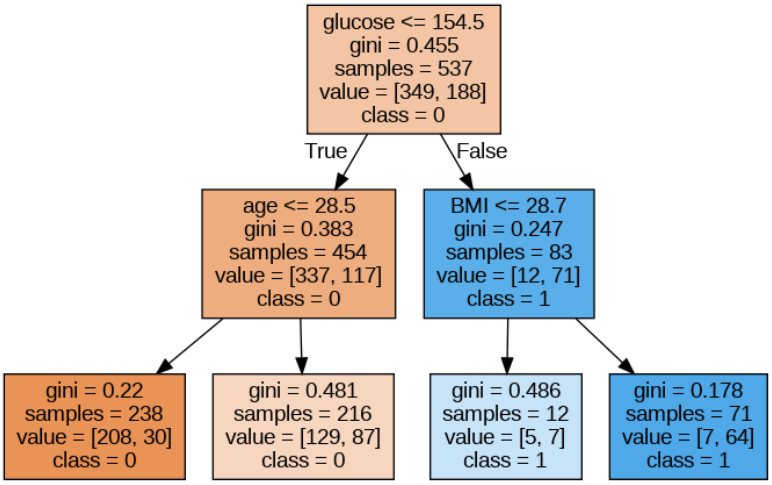


Figure 40: Pruned Decision Tree

References

[1] S. Haykin, *Neural Networks: A Comprehensive Foundation*. Upper Saddle River, NJ: Prentice Hall, 1999.