# CSI 5387: Data Mining and Concept Learning

## Winter 2020

## Assignment 2

**Submitted By: Deepshi Mediratta (300086013)**

## 1. Data Preprocessing

### 1.1 Missing Data

A dataset can have missing values due to corruption in the way it was collected. It is important to handle these missing values to achieve successful results. After analyzing the dataset for missing information, significant amount of missing values was found in the columns- Glucose (0.6%), BloodPressure (4.4%), SkinThickness (29.5%), Insulin (48.3%) and BMI (1.4%).

The missing data is handled in the dataset with SciKit library's Simple Imputer model. Wherever we have a '?' in the dataset, the Imputer will replace it with a new value.

I used the strategy 'mean' to replace the value that replaces the null values in the dataset. For example, in the feature 'BloodPressure' the values are replaced with 72.30654, in the feature 'SkinThickness' the values are replaced with 29.1. Similarly, other values are also replaced by the mean.

The following snippet shows a sample of the dataset before replacing missing data-

| | Pregnancies | Glucose | BloodPressure | SkinThickness | Insulin | BMI | DiabetesPedigreeFunction | Age | Outcome |
|---|---|---|---|---|---|---|---|---|---|
| **0** | 6 | 148 | 72 | 35 | ? | 33.6 | 0.627 | 50 | 1 |
| **1** | 1 | 85 | 66 | 29 | ? | 26.6 | 0.351 | 31 | 0 |
| **2** | 8 | 183 | 64 | ? | ? | 23.3 | 0.672 | 32 | 1 |
| **3** | 1 | 89 | 66 | 23 | 94 | 28.1 | 0.167 | 21 | 0 |
| **4** | 0 | 137 | 40 | 35 | 168 | 43.1 | 2.288 | 33 | 1 |
| **5** | 5 | 116 | 74 | ? | ? | 25.6 | 0.201 | 30 | 0 |
| **6** | 3 | 78 | 50 | 32 | 88 | 31 | 0.248 | 26 | 1 |
| **7** | 10 | 115 | ? | ? | ? | 35.3 | 0.134 | 29 | 0 |
| **8** | 2 | 197 | 70 | 45 | 543 | 30.5 | 0.158 | 53 | 1 |
| **9** | 8 | 125 | 96 | ? | ? | ? | 0.232 | 54 | 1 |

The following snippet shows the same sample of the dataset after applying SimpleImputer and replacing the missing values with mean.

| | Pregnancies | Glucose | BloodPressure | SkinThickness | Insulin | BMI | DiabetesPedigreeFunction | Age | Outcome |
|---|---|---|---|---|---|---|---|---|---|
| 0 | 6.0 | 148.0 | 72.00000 | 35.00000 | 154.372796 | 33.600000 | 0.627 | 50.0 | 1 |
| 1 | 1.0 | 85.0 | 66.00000 | 29.00000 | 154.372796 | 26.600000 | 0.351 | 31.0 | 0 |
| 2 | 8.0 | 183.0 | 64.00000 | 29.15342 | 154.372796 | 23.300000 | 0.672 | 32.0 | 1 |
| 3 | 1.0 | 89.0 | 66.00000 | 23.00000 | 94.000000 | 28.100000 | 0.167 | 21.0 | 0 |
| 4 | 0.0 | 137.0 | 40.00000 | 35.00000 | 168.000000 | 43.100000 | 2.288 | 33.0 | 1 |
| 5 | 5.0 | 116.0 | 74.00000 | 29.15342 | 154.372796 | 25.600000 | 0.201 | 30.0 | 0 |
| 6 | 3.0 | 78.0 | 50.00000 | 32.00000 | 88.000000 | 31.000000 | 0.248 | 26.0 | 1 |
| 7 | 10.0 | 115.0 | 72.30654 | 29.15342 | 154.372796 | 35.300000 | 0.134 | 29.0 | 0 |
| 8 | 2.0 | 197.0 | 70.00000 | 45.00000 | 543.000000 | 30.500000 | 0.158 | 53.0 | 1 |
| 9 | 8.0 | 125.0 | 96.00000 | 29.15342 | 154.372796 | 32.457464 | 0.232 | 54.0 | 1 |

## 1.2 Normalization/Standardization

The features in the dataset are numeric, but they are not on the same scale. To achieve this, min-max scaling technique is used to scale the data to a fixed range- 0 to 1. I have used sklearn.preprocessing.MinMaxScaler to achieve this.

$$X_{norm} = \frac{X - X_{min}}{X_{max} - X_{min}}(new_{max} - new_{min}) + new_{min}$$

The following snippet shows a sample of the dataset after normalizing to range 0 to 1(Only Features).

| | Pregnancies | Glucose | BloodPressure | SkinThickness | Insulin | BMI | DiabetesPedigreeFunction | Age |
|---|---|---|---|---|---|---|---|---|
| 0 | 0.352941 | 0.670968 | 0.590164 | 0.304348 | 0.182474 | 0.314928 | 0.234415 | 0.483333 |
| 1 | 0.058824 | 0.264516 | 0.540984 | 0.239130 | 0.182474 | 0.171779 | 0.116567 | 0.166667 |
| 2 | 0.470588 | 0.896774 | 0.524590 | 0.240798 | 0.182474 | 0.104294 | 0.253629 | 0.183333 |
| 3 | 0.058824 | 0.290323 | 0.540984 | 0.173913 | 0.111111 | 0.202454 | 0.038002 | 0.000000 |
| 4 | 0.000000 | 0.600000 | 0.327869 | 0.304348 | 0.198582 | 0.509202 | 0.943638 | 0.200000 |
| 5 | 0.294118 | 0.464516 | 0.606557 | 0.240798 | 0.182474 | 0.151329 | 0.052519 | 0.150000 |
| 6 | 0.176471 | 0.219355 | 0.409836 | 0.271739 | 0.104019 | 0.261759 | 0.072588 | 0.083333 |
| 7 | 0.588235 | 0.458065 | 0.592677 | 0.240798 | 0.182474 | 0.349693 | 0.023911 | 0.133333 |
| 8 | 0.117647 | 0.987097 | 0.573770 | 0.413043 | 0.641844 | 0.251534 | 0.034159 | 0.533333 |
| 9 | 0.470588 | 0.522581 | 0.786885 | 0.240798 | 0.182474 | 0.291564 | 0.065756 | 0.550000 |

### 1.3 Train-Test split

The dataset is partitioned into training dataset and test dataset. The training set contains the classification label and is used for learning. The test set is used to test the model's accuracy. This is done using the method train_test_split provided by the SciKit learn library. The following is the split configuration.

| | % | Count |
|---|---|---|
| **Training Data** | 75% | 576 |
| **Test Data** | 25% | 192 |

## 2. Model Development

### 2.1 Single Layer

#### 2.1.1 Training Model

Loss = Binary Crossentropy

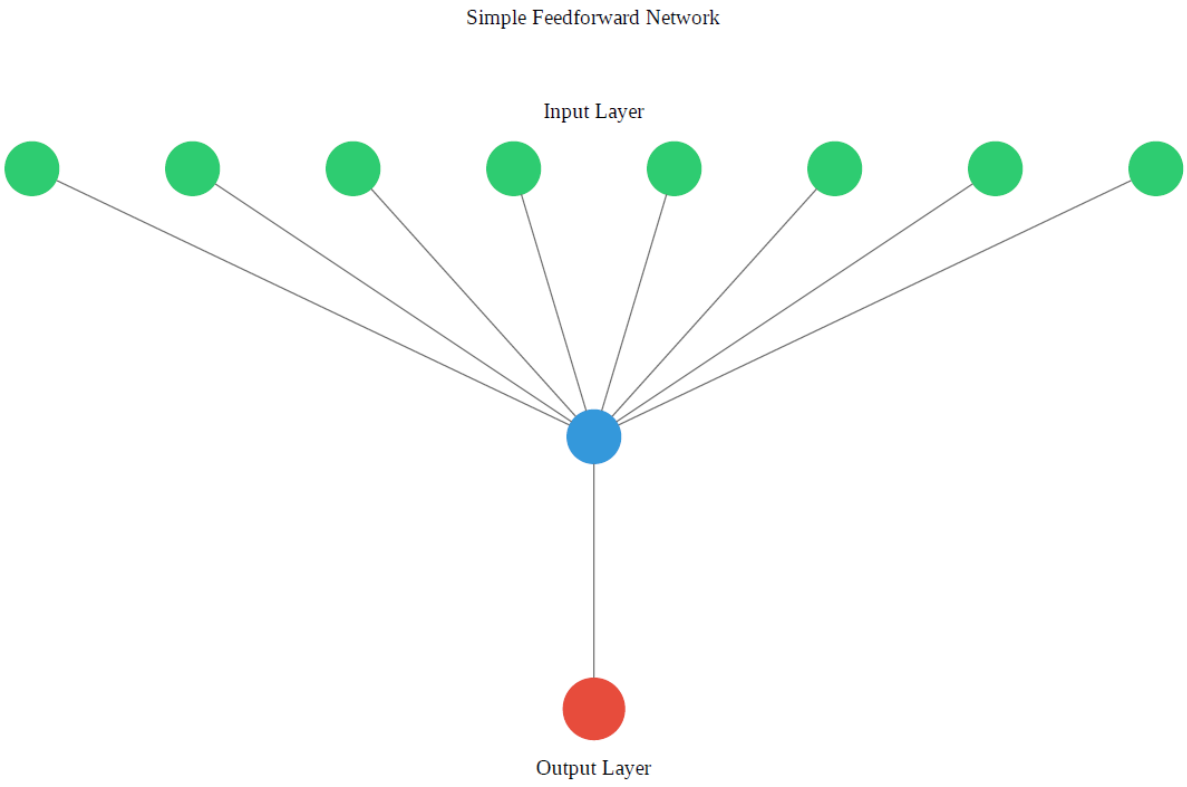
Optimizer = Adam

Activation of first layer = linear

Activation of output layer = sigmoid

Epochs = 150

#### 2.1.2 Plotting the Neural Network



```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_1 (Dense)              (None, 1)                 9
_________________________________________________________________
dense_2 (Dense)              (None, 1)                 2
=================================================================
```

### 2.1.3 Printing the weights and bias

The following are the weights and bias of between the input layer and the hidden node-

| | |
|---|---|
| $W_0$ (Bias) | 1.0719262 |
| $W_1$ | -0.79294497 |
| $W_2$ | -1.5398409 |
| $W_3$ | 1.2884599 |
| $W_4$ | -0.09530403 |
| $W_5$ | -0.51220816 |
| $W_6$ | -1.2032514 |
| $W_7$ | -1.0087095 |
| $W_8$ | -1.2391063 |

### 2.1.4 Generating predictions on test set

Threshold = 0.40

If the output from the neural network is greater than 0.40, then the label is 1, otherwise the output is 0.

Accuracy on test set(y_test) is 68.75%

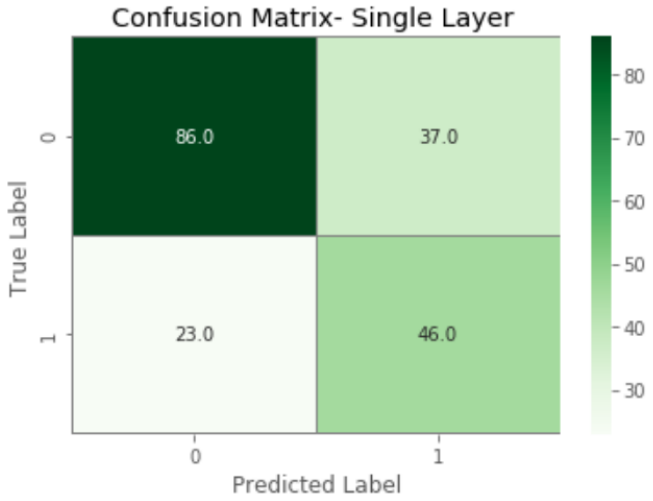
**Sample A** = [0.176471, 0.219355, 0.409836, 0.271739, 0.104019, 0.261759, 0.072588, 0.083333]
Actual Label = 1
Predicted Label = 0

**Sample B** = [0.352941, 0.967742, 0.639344, 0.240798, 0.182474, 0.108384, 0.0217768, 0.633333]
Actual Label = 1
Predicted Label = 1

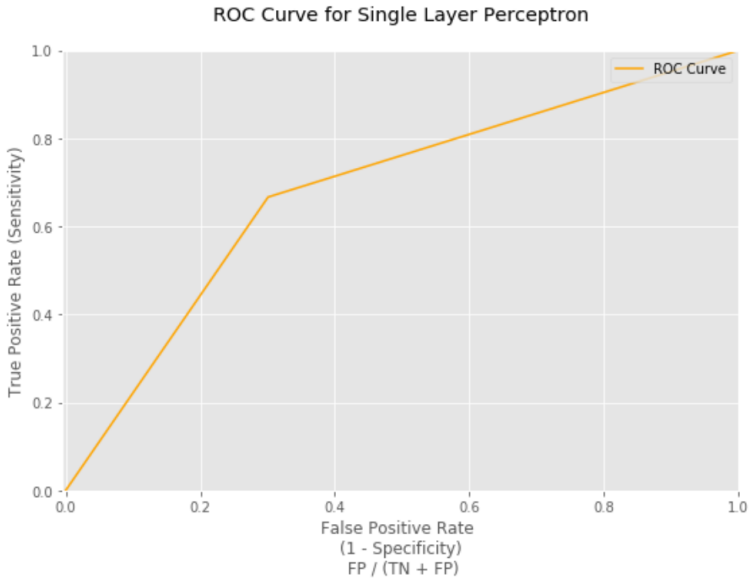
### 2.1.5 Confusion Matrix

Sensitivity: 66.66

Specificity: 69.91




### 2.1.6 ROC Curve




### 2.1.7 Results

| Single Layer Perceptron | |
|---|---|
| Training Accuracy | 73.44% |
| Test Accuracy | 68.75% |
| Sensitivity | 66.66 |
| Specificity | 69.91 |
| Area Under the ROC curve | 68.29 |

## 2.2 Multi Layer

### 2.2.1 Training Model

Layer 1, activation = Linear

Layer 2, activation = Linear

Layer 3, activation = Sigmoid

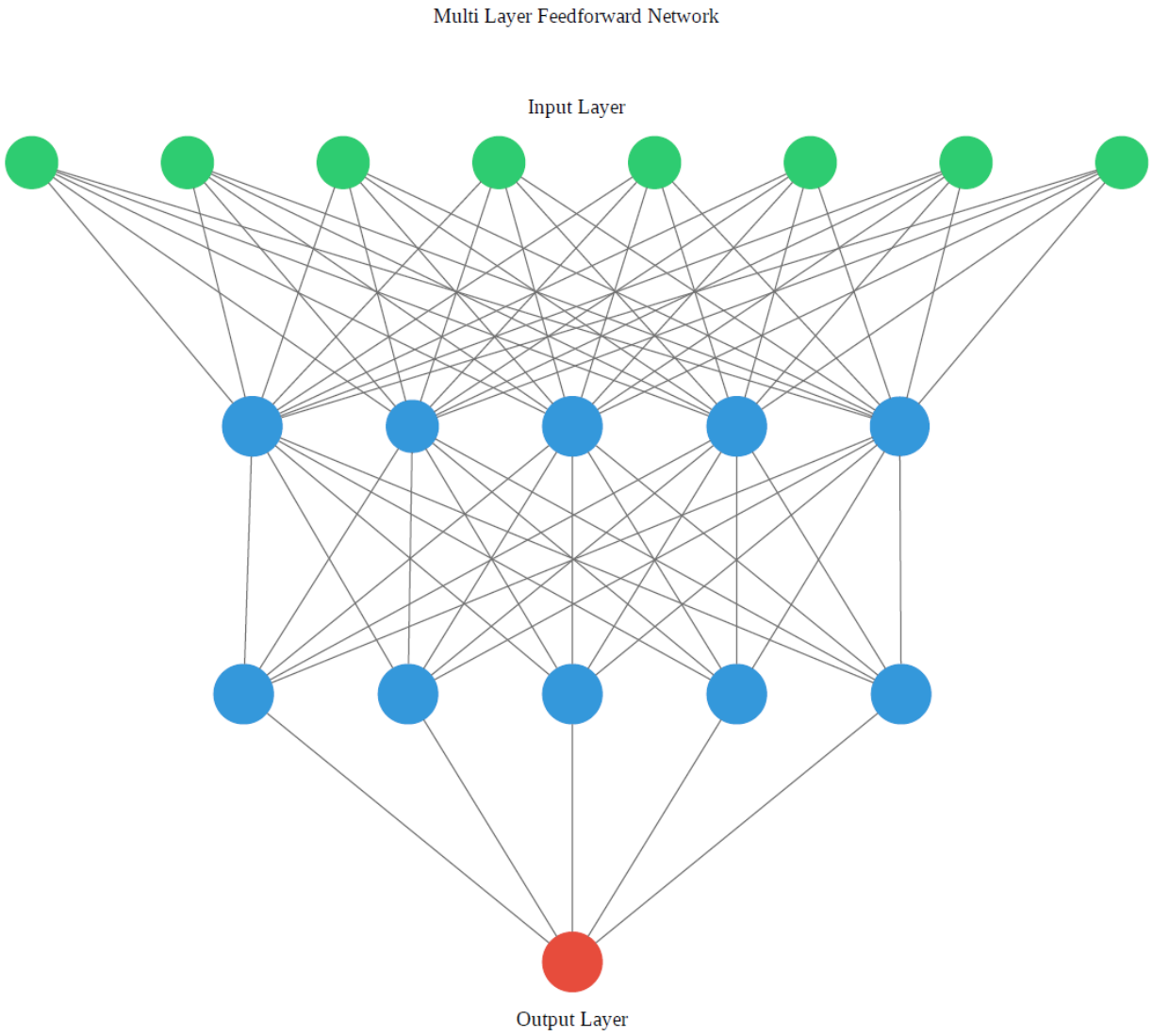
Loss = Binary Cross entropy

Optimizer = Adam

Epoch = 150

Batch size = 10

### 2.2.2 Plotting the Neural Network



```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_3 (Dense)              (None, 5)                 45
_________________________________________________________________
dense_4 (Dense)              (None, 5)                 30
_________________________________________________________________
dense_5 (Dense)              (None, 1)                 6
=================================================================
```

### 2.2.3 Printing Weights and bias

**Layer 1:**

```
Weights = [[-0.8563855   0.07834488  0.38206294 -0.2364078   0.61581093]
 [-0.19751102 -0.9974625  -0.6622703  -0.06009922  0.36121586]
 [ 0.71295696 -0.13580357  0.14800173  0.4176822  -0.1685529 ]
 [-0.4659252   0.13717239 -0.3445492   0.77804404  0.34458876]
 [-0.02041052 -0.3151512   0.18509208  0.5075335  -0.6215947 ]
 [-0.8696279  -0.7558522  -0.3686173  -0.5291634  -0.45839503]
 [ 0.05204249 -0.2361069  -0.21189974 -0.3607502  -0.5237303 ]
 [ 0.16519026 -0.256725   -0.5466635  -0.30268672 -0.24682966]]
Biases = [ 0.34506914  0.39925662  0.322645    0.24237785 -0.26647672]
```

**Layer 2:**

```
Weights = [[-0.18346056  0.98389846  1.010243    0.22751442  0.18854874]
 [-0.9517719   0.99827373  0.05728059  0.24928275 -0.7320455 ]
 [-1.0846953   0.67326283  0.9004637   0.06737702 -0.09174947]
 [-0.23712419 -0.49336272  0.35393205 -0.35010117 -0.4031896 ]
 [ 0.41292834 -0.6846582   0.11550495  0.1630262   0.05692496]]
Biases = [-0.27112487  0.3074697   0.26036382 -0.23260203 -0.28498244]
```

**Layer 3:**

```
Weights = [[ 1.5657389 ]
 [-0.88816166]
 [-0.86672384]
 [ 0.7379403 ]
 [ 1.2507896 ]]
Biases = [-0.24494104]
```

### 2.2.4 Generating predictions on test set

Threshold = 0.6

If the output from the neural network is greater than 0.6, then the label is 1, otherwise the output is 0.

Accuracy on test set(y_test) is 75.52%

**Sample A** = [0.176471, 0.219355, 0.409836, 0.271739, 0.104019, 0.261759, 0.072588, 0.083333]

Actual Label = 1

Predicted Label = 0

**Sample B** = [0.352941, 0.967742, 0.639344, 0.240798, 0.182474, 0.108384, 0.0217768, 0.633333]

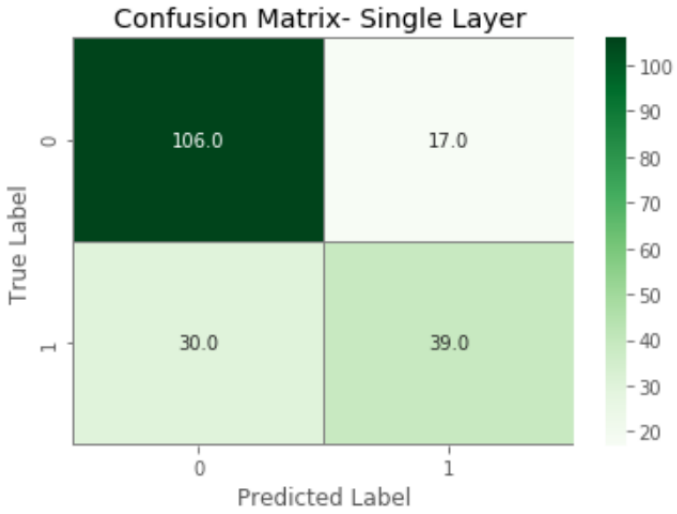
Actual Label = 1

Predicted Label = 1

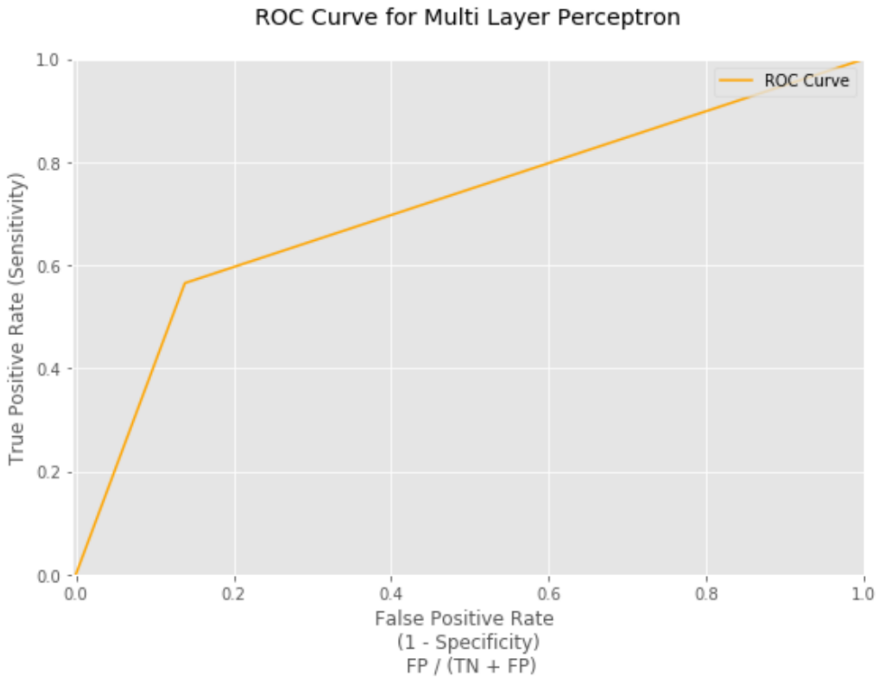
### 2.2.5 Confusion Matrix

Sensitivity: 56.52

Specificity: 86.17




### 2.2.6 ROC Curve



### 2.2.7 Results – Multi Layer Perceptron

| Multi Layer Perceptron | |
|---|---|
| Training Accuracy | 72.92% |
| Test Accuracy | 75.52% |
| Sensitivity | 56.52 |
| Specificity | 86.17 |
| Area Under the ROC curve | 71.35 |

## 2.3 Experiments on Multi Layer Perceptron

I performed some experiments in multi layer perceptron by varying the optimizer, the learning rate and the number of epochs. It was observed that the test accuracy was better with Stochastic Gradient decent as compared to Adam optimizer. The model performed best when the learning rate was 0.01 As the number of epochs increased, the test accuracy was better, but after 100 epoch the test accuracy was almost the same. The following are the results from these experiments.

| **Optimizer** | |
|---|---|
| | **Test Accuracy** |
| **Adam** | 75.52% |
| **SGD** | 76.04% |
| | |
| **Learning Rate(SGD)** | |
| **0.1** | 75.52% |
| **0.01** | 76.04% |
| **0.001** | 63.02 |
| | |
| **Number of epochs(and SGD=0.01)** | |
| **50** | 71.87% |
| **100** | 76.04% |
| **150** | 76.04% |
| | |
| | |

### 2.4 Single Layer VS Multi layer Perceptron

It can be observed that the overall accuracy is better for multi layer perceptron. This can be verified by looking at the test accuracy and area under the ROC curve.

However, the true positive rate for single layer perceptron is better and the false positive rate for multi layer is better.

| | **Multi Layer Perceptron** | **Single Layer Perceptron** |
|---|---|---|
| Training Accuracy | 72.92% | 73.44% |
| Test Accuracy | 75.52% | 68.75% |
| Sensitivity | 56.52 | 66.66 |
| Specificity | 86.17 | 69.91 |
| Area Under the ROC curve | 71.35 | 68.29 |
| Number of hidden Layers | 2 | 0 |
| Number of nodes in hidden layers | 5 | 1 |
| Number of epochs | 150 | 150 |

## 3. Model Comparison

### 3.1 Support Vector Machine

C = 1.0

Kernel = Linear

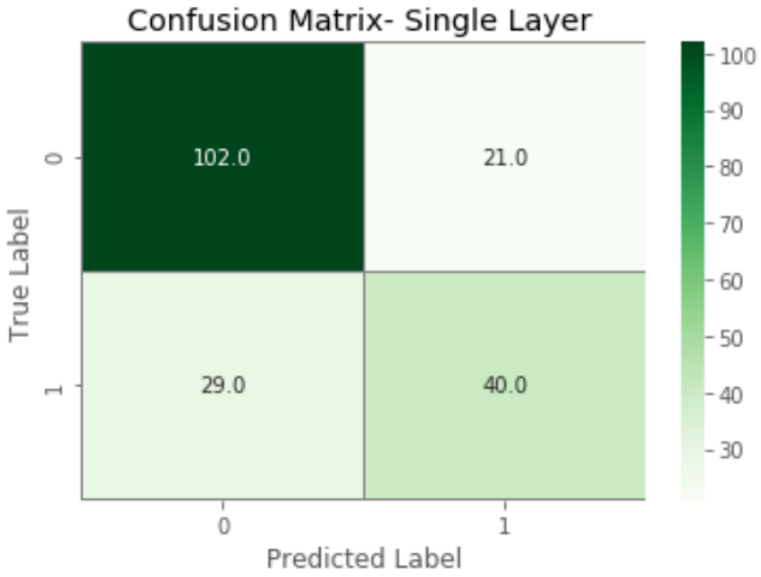
Degree = 3

Gamma = auto

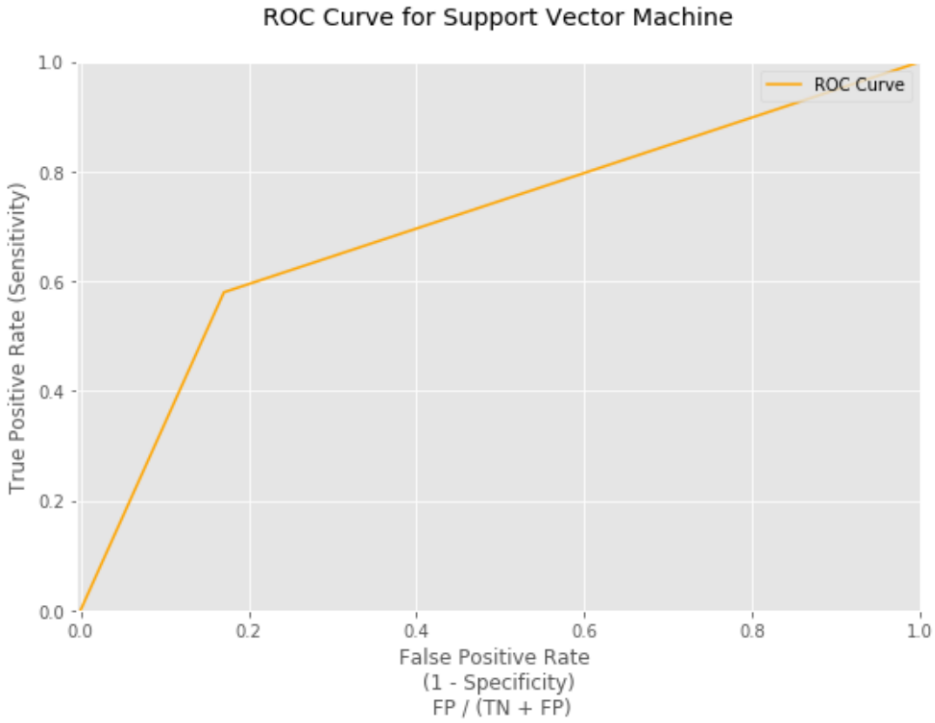
### 3.2 Confusion Matrix

Sensitivity: 57.97

Specificity: 82.92




### 3.3 ROC Curve




### 3.4 Results

| Support Vector Machine | |
|---|---|
| Test Accuracy | 73.95% |
| Sensitivity | 57.97 |
| Specificity | 82.92 |
| Area Under the ROC curve | 70.44 |

## 4. Model Evaluation

### 4.1 Multi-Layer Perceptron VS Support Vector Machine

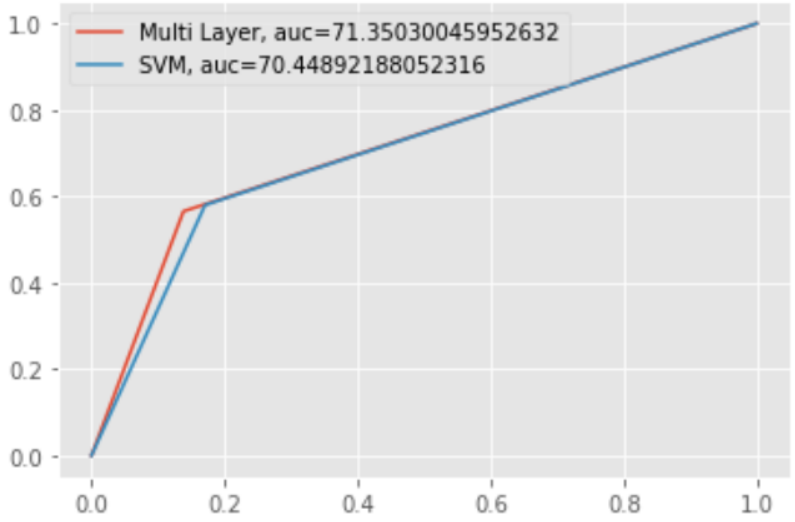
It can be observed that the overall accuracy is better for multi layer perceptron. This can be verified by looking at the test accuracy and area under the ROC curve.

However, the true positive rate for SVM is better and the false positive rate for multi layer is better.

| | Support Vector Machine | Multi-Layer Perceptron |
|---|---|---|
| Test Accuracy | 73.95% | 75.52% |
| Sensitivity | 57.97 | 56.52 |
| Specificity | 82.92 | 86.17 |
| Area Under the ROC curve | 70.44 | 71.35 |

**ROC Analysis- SVM VS Multi-Layer Perceptron**

From the ROC curve, we can clearly see that multi layer performed has slightly higher accuracy than SVM because the area under the curve is higher.




### 4.2 Best Model

| Criterion | Model Name | Value |
|---|---|---|
| **Test Accuracy** | Multi Layer Perceptron | 75.52% |
| **Sensitivity** | SVM | 57.97 |
| **Specificity** | Multi Layer Perceptron | 86.17 |