Rohini Vishwanathan
Knowledge Discovery and Data Mining
IST 340

CE6- Neural Network Evaluation Report

1. Evaluation Approach

MEASURE	DESCRIPTION	DEFINITION OF VALUE FUNCTION	WEIGHT	THRESHOLD
ACCURACY	Accuracy on test set (from model.evaluate( )	1 if accuracy >= 0.75 else 0	0.5	0.75
LIFT (Top 10%)	Lift at Top 10% decile	1 if lift >= 2.0 else 0	0.3	2.0
STABILITY	Difference in BAD rate between Top and Bottom decile	1 if stability >= 0.5 else 0	0.2	0.5

The composite performance score is calculated as - Overall Score = (Accuracy Score * 0.5) + (Lift Score * 0.3) + (Stability Score * 0.2)

2. Summary of Results- Note on Initializers Used All models used he_uniform initialization for hidden layers (ideal for ReLU activations), and glorot_uniform (Xavier) for the sigmoid output layer, Lab 6 reference code.

NN Label/ Description	Accuracy		Lift (Top 10%)		Stability	Overall Score	Justification
	Value	**Score**	**Value**	**Score**	**Score**		
Model 1	0.7533	1	2.00	1	0	0.80	Misses stability threshold
Model 2	0.7533	1	2.67	1	1	1.00	Meets all threshold

NN Label/ Description	Accuracy		Lift (Top 10%)		Stability	Overall Score	Justification
Model 3	0.7867	1	2.44	1	1	1.00	Strong Accuracy, high lift and stable model
Model 4	0.7800	1	0.00	0	0	0.50	High accuracy but fails lift and stability

3. Description of the Best Model

Based on the composite score and individual metric performance, Model 3 emerges as the best performing neural network. It uses two hidden layers (32 and 16 neurons respectively) with ReLU activation, Dropout regularization, and a sigmoid output layer.

Model 3 achieve the highest test accuracy (0.7867) and meets the lift and stability criteria with a lift of 2.44 and stability of 0.73. The balanced performance across all three measures results in a composite score of 1.00, the highest possible. Compared to other models, Model 3 maintains a high generalization while avoiding overfitting, making it a robust choice for deployment.

In addition to its scoring performance, Model 3's ROC curve demonstrates strong separation between true positives and false positives, and its classification report reflects high precision and recall for both classes. This confirms that the model is well-calibrated, not just in overall accuracy, but also in its sensitivity to class imbalance, as emphasized in the lecture notes. The use of Dropout also helps prevent overfitting by randomly deactivating neurons during training, which contributes to the model's stability and generalizability.

Additionally, Model 3 demonstrates solid class-wise performance with an FI score of 0.60 of the minority class (bad), and 0.85 for the majority class (good), indicating string balance between precision and recall- particularly important given the dataset’s class imbalance.

4. Evidence of Experimentation

Key evidence include-

- **Confusion Matrices and classification Report for all models, showing class-level performance.**

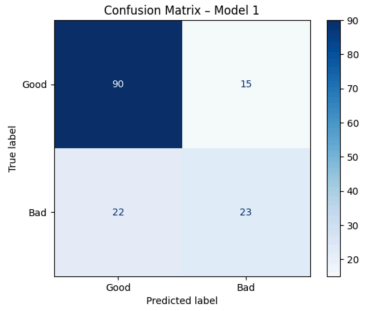
Model 1-

Classification Report Model 1

```
from sklearn.metrics import classification_report

# Example: after predictions for model_1
print("Classification Report - Model 1")
print(classification_report(y_test, y_pred_1))
```

```
Classification Report - Model 1
              precision    recall  f1-score   support

           0       0.80      0.86      0.83       105
           1       0.61      0.51      0.55        45

    accuracy                           0.75       150
   macro avg       0.70      0.68      0.69       150
weighted avg       0.74      0.75      0.75       150
```

```
5/5 ━━━━━━━━━━━━━━━━━━━━ 0s 14ms/step
Test Accuracy (Model 1): 0.7533
```



	actual	score
0	0	0.184271
1	0	0.102270
2	1	0.562175
3	0	0.582466
4	0	0.094892

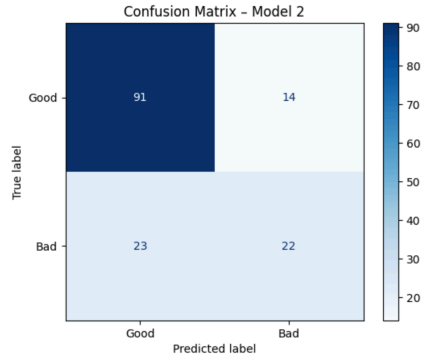
Model 2-

Classification Report - Model 2

```
print("Classification Report - Model 2")
print(classification_report(y_test, y_pred_2))
```

```
Classification Report - Model 2
              precision    recall  f1-score   support

           0       0.80      0.87      0.83       105
           1       0.61      0.49      0.54        45

    accuracy                           0.75       150
   macro avg       0.70      0.68      0.69       150
weighted avg       0.74      0.75      0.74       150
```

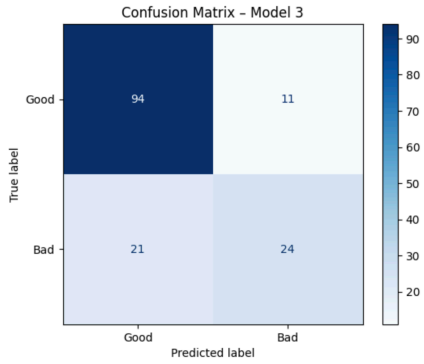
```
5/5 ━━━━━━━━━━━━━━━━━━━━ 0s 24ms/step
Test Accuracy (Model 2): 0.7533
```



	actual	score
0	0	0.141231
1	0	0.249051
2	1	0.764282
3	0	0.662964
4	0	0.066325

Model 3-

```
WARNING:tensorflow:5 out of the last 11 calls to <function TensorFlowTrainer.make_predict_function.<locals>.one_step_on_data_distributed at 0x7f446a2514e0> triggered tf.function
1/5 ━━━━━━━━━━━━━━━━━━━━ 0s 111ms/stepWARNING:tensorflow:5 out of the last 11 calls to <function TensorFlowTrainer.make_predict_function.<locals>.one_step_on_data_distributed at
5/5 ━━━━━━━━━━━━━━━━━━━━ 0s 56ms/step
Test Accuracy (Model 3): 0.7867
```



	actual	score
0	0	0.075421
1	0	0.136663
2	1	0.498096
3	0	0.608188
4	0	0.124865

Classification Report- Model 3

```
[24] print("Classification Report – Model 3")
     print(classification_report(y_test, y_pred_3))
```

```
Classification Report – Model 3
              precision    recall  f1-score   support

           0       0.82      0.90      0.85       105
           1       0.69      0.53      0.60        45

    accuracy                           0.79       150
   macro avg       0.75      0.71      0.73       150
weighted avg       0.78      0.79      0.78       150
```

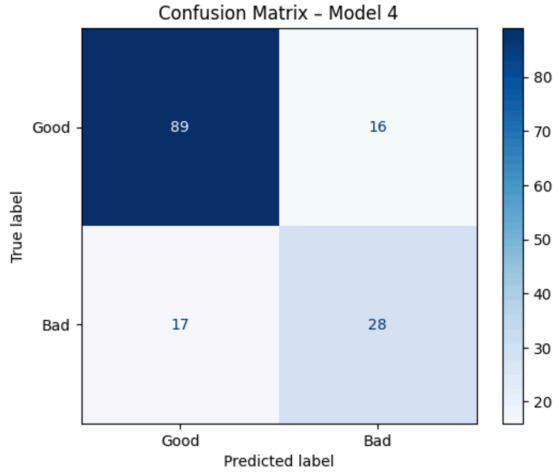
Model 4-

Classification Report - Model 4

```
[25] print("Classification Report — Model 4")
     print(classification_report(y_test, y_pred_4))
```

```
Classification Report — Model 4
              precision    recall  f1-score   support

           0       0.84      0.85      0.84       105
           1       0.64      0.62      0.63        45

    accuracy                           0.78       150
   macro avg       0.74      0.73      0.74       150
weighted avg       0.78      0.78      0.78       150
```

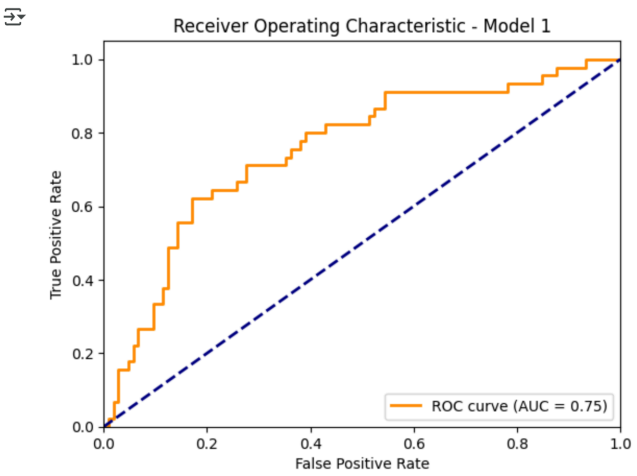
```
5/5 ━━━━━━━━━━━━━━━━━━━━ 0s 14ms/step
Test Accuracy (Model 4): 0.7800
```



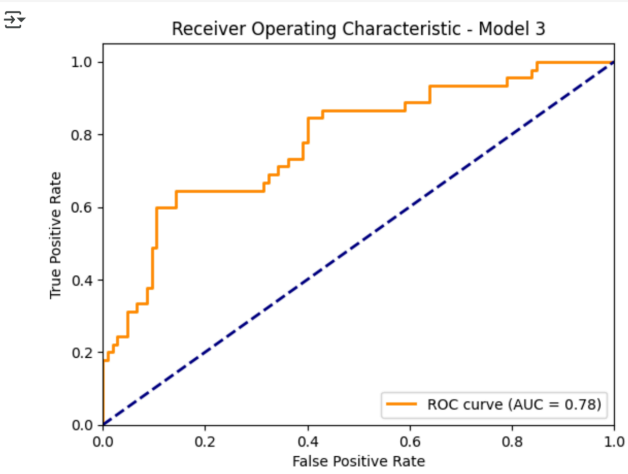
	actual	score
0	0	0.041274
1	0	0.194978
2	1	0.579227
3	0	0.583921
4	0	0.204044

• ROC Curves with AUC values that demonstrate model discrimination ability

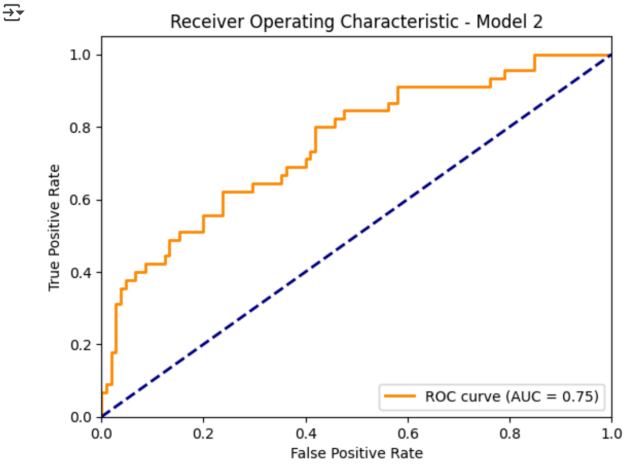
Model 1 -



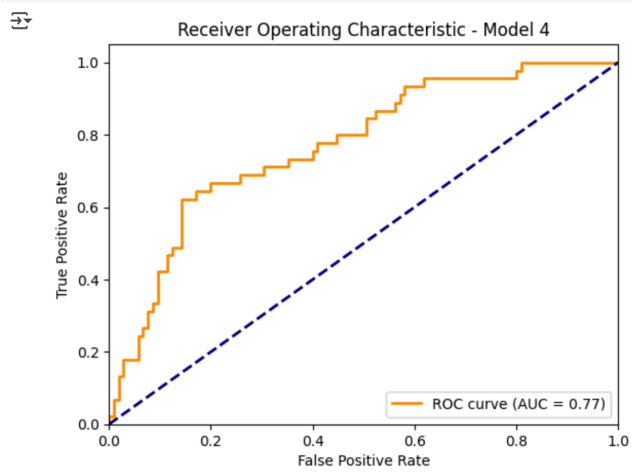
Model 2-



Model 3-



Model 4-



• **Lift and Stability Tables, calculated from top and bottom deciles of predicted scores, used for scoring fairness and robustness.**

Model 1 -

```
Overall BAD rate: 0.3000
Top Decile BAD rate: 0.6000
Bottom Decile BAD rate: 0.1333
Lift (Top 10%): 2.00
Stability (Top vs Bottom): 0.47
```

decile	count	bads	bad_rate
0	15	9	0.600000
1	15	8	0.533333
2	15	10	0.666667
3	15	4	0.266667
4	15	4	0.266667
5	15	2	0.133333
6	15	4	0.266667
7	15	0	0.000000
8	15	2	0.133333
9	15	2	0.133333

Model 2-

```
Overall BAD rate: 0.3000
Top Decile BAD rate: 0.8000
Bottom Decile BAD rate: 0.0000
Lift (Top 10%): 2.67
Stability (Top vs Bottom): 0.80
```

decile	count	bads	bad_rate
0	15	12	0.800000
1	15	7	0.466667
2	15	5	0.333333
3	15	5	0.333333
4	15	3	0.200000
5	15	6	0.400000
6	15	3	0.200000
7	15	0	0.000000
8	15	4	0.266667
9	15	0	0.000000

Model 3-

```
Overall BAD rate: 0.3000
Top Decile BAD rate: 0.7333
Bottom Decile BAD rate: 0.0000
Lift (Top 10%): 2.44
Stability (Top vs Bottom): 0.73
```

decile	count	bads	bad_rate
0	15	11	0.733333
1	15	9	0.600000
2	15	9	0.600000
3	15	0	0.000000
4	15	5	0.333333
5	15	5	0.333333
6	15	1	0.066667
7	15	2	0.133333
8	15	3	0.200000
9	15	0	0.000000

Model 4-

```
Overall BAD rate: 0.3000
Top Decile BAD rate: 0.0000
Bottom Decile BAD rate: 0.6000
Lift (Top 10%): 0.00
Stability (Top vs Bottom): -0.60
```

decile	count	bads	bad_rate
0	15	0	0.000000
1	15	2	0.133333
2	15	1	0.066667
3	15	5	0.333333
4	15	4	0.266667
5	15	2	0.133333
6	15	3	0.200000
7	15	9	0.600000
8	15	10	0.666667
9	15	9	0.600000

• Final Test Accuracy values using model.evaluate( ) for consistency.

Model 1-

Model Evaluation using evaluate()

```
[33] # for Model 1
     test_loss_1, test_accuracy_1 = model_1.evaluate(X_test_scaled, y_test, verbose=0)
     print(f"Model 1 Final Test Accuracy: {test_accuracy_1:.4f}, Loss: {test_loss_1:.4f}")
```

```
Model 1 Final Test Accuracy: 0.7533, Loss: 0.5903
```

Model 2-

Model Evaluation using evaluate()

```
[32] # Example for Model 2
     test_loss_1, test_accuracy_1 = model_2.evaluate(X_test_scaled, y_test, verbose=0)
     print(f"Model 2 Final Test Accuracy: {test_accuracy_1:.4f}, Loss: {test_loss_1:.4f}")
```

Model 3-

Model Evaluation using evaluate()

```
# Example for Model 3
test_loss_1, test_accuracy_1 = model_3.evaluate(X_test_scaled, y_test, verbose=0)
print(f"Model 3 Final Test Accuracy: {test_accuracy_1:.4f}, Loss: {test_loss_1:.4f}")
```

```
Model 3 Final Test Accuracy: 0.7867, Loss: 0.5362
```

Model 4-

Model Evaluation using evaluate()

```
[30] # Example for Model 4
     test_loss_1, test_accuracy_1 = model_4.evaluate(X_test_scaled, y_test, verbose=0)
     print(f"Model 4 Final Test Accuracy: {test_accuracy_1:.4f}, Loss: {test_loss_1:.4f}")
```

```
Model 4 Final Test Accuracy: 0.7800, Loss: 0.5864
```

• Overall Score calculations, based on rubric thresholds and weights, showing exactly how the best model was determined.

Model 1-

```
overall_score_1 = (acc_score_1 * 0.5) + (lift_score_1 * 0.3) + (stab_score_1 * 0.2)
print(f"Model 1 Overall Score: {overall_score_1:.2f}")
```

```
Model 1 Overall Score: 0.80
```

Model 2-

```
overall_score_2 = (acc_score_2 * 0.5) + (lift_score_2 * 0.3) + (stab_score_2 * 0.2)
print(f"Model 2 Overall Score: {overall_score_2:.2f}")
```

```
Model 2 Overall Score: 1.00
```

Model 3-

```
overall_score_3 = (acc_score_3 * 0.5) + (lift_score_3 * 0.3) + (stab_score_3 * 0.2)
print(f"Model 3 Overall Score: {overall_score_3:.2f}")
```

```
Model 3 Overall Score: 1.00
```

Model 4-

```
overall_score_4 = (acc_score_4 * 0.5) + (lift_score_4 * 0.3) + (stab_score_4 * 0.2)
print(f"Model 4 Overall Score: {overall_score_4:.2f}")
```

```
Model 4 Overall Score: 0.50
```

Each output section is clearly labeled and grouped by model for transparency and ease of comparison.

Google Colab Link -

https://colab.research.google.com/drive/1qOcZQ64pebh4HD5b5GmA_f85Vnt0Nm5A?usp=sharing