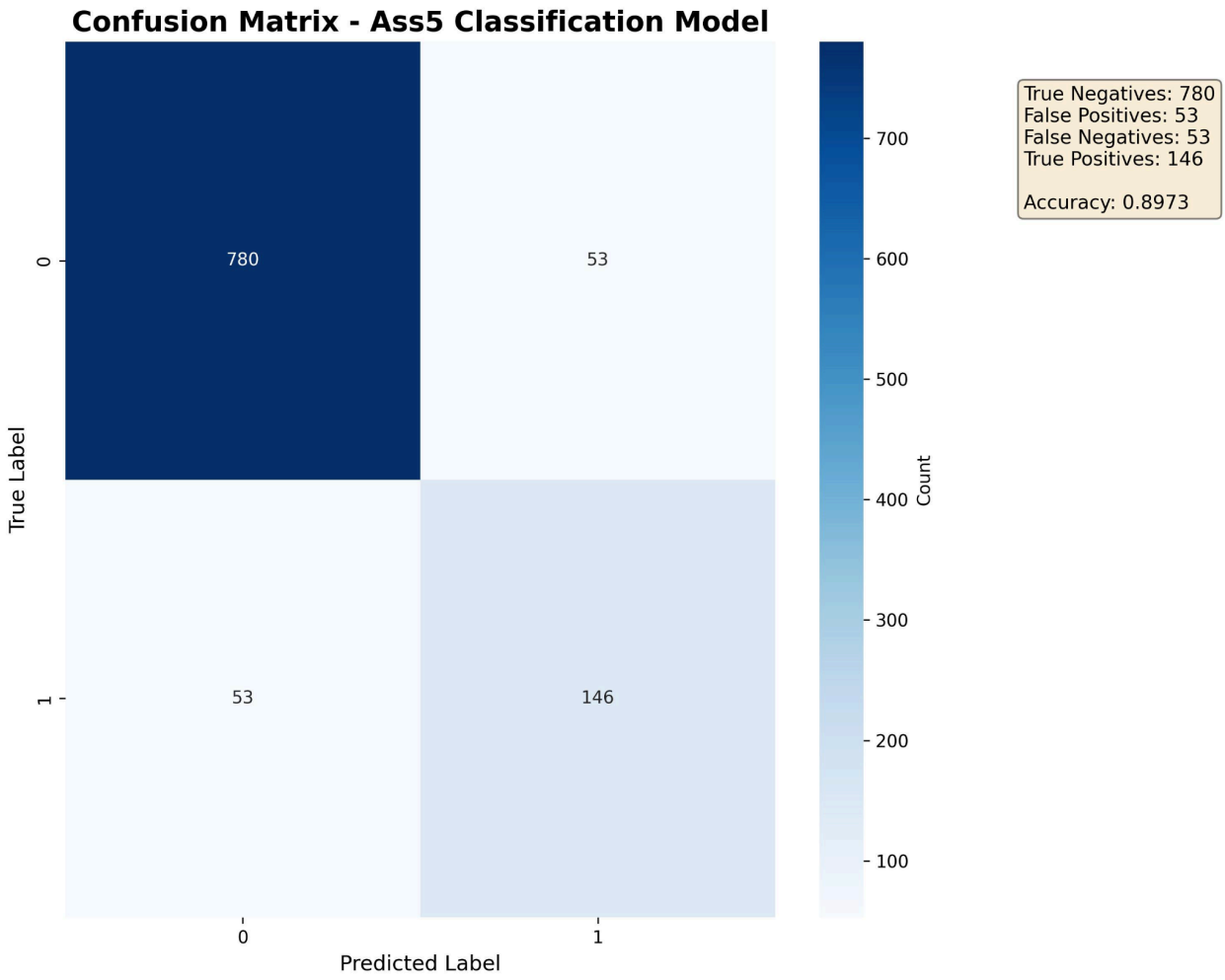
We started with data preparation so the model could learn from clean and fair information. First, we checked the dataset for repeated rows and removed duplicates to avoid counting the same example twice. Next, we handled missing values: for numeric columns, we used the median because it is stable even when there are extreme values, and for category columns, we used the most common value so the column stayed meaningful. We also converted text categories into numbers using one-hot encoding, because machine learning models need numeric input. After that, we scaled numeric features so large numbers would not overpower small numbers. Finally, we split the dataset into three parts: 70% for training, 15% for validation, and 15% for testing. This split helped us train the model, tune settings, and then do a final fair check.

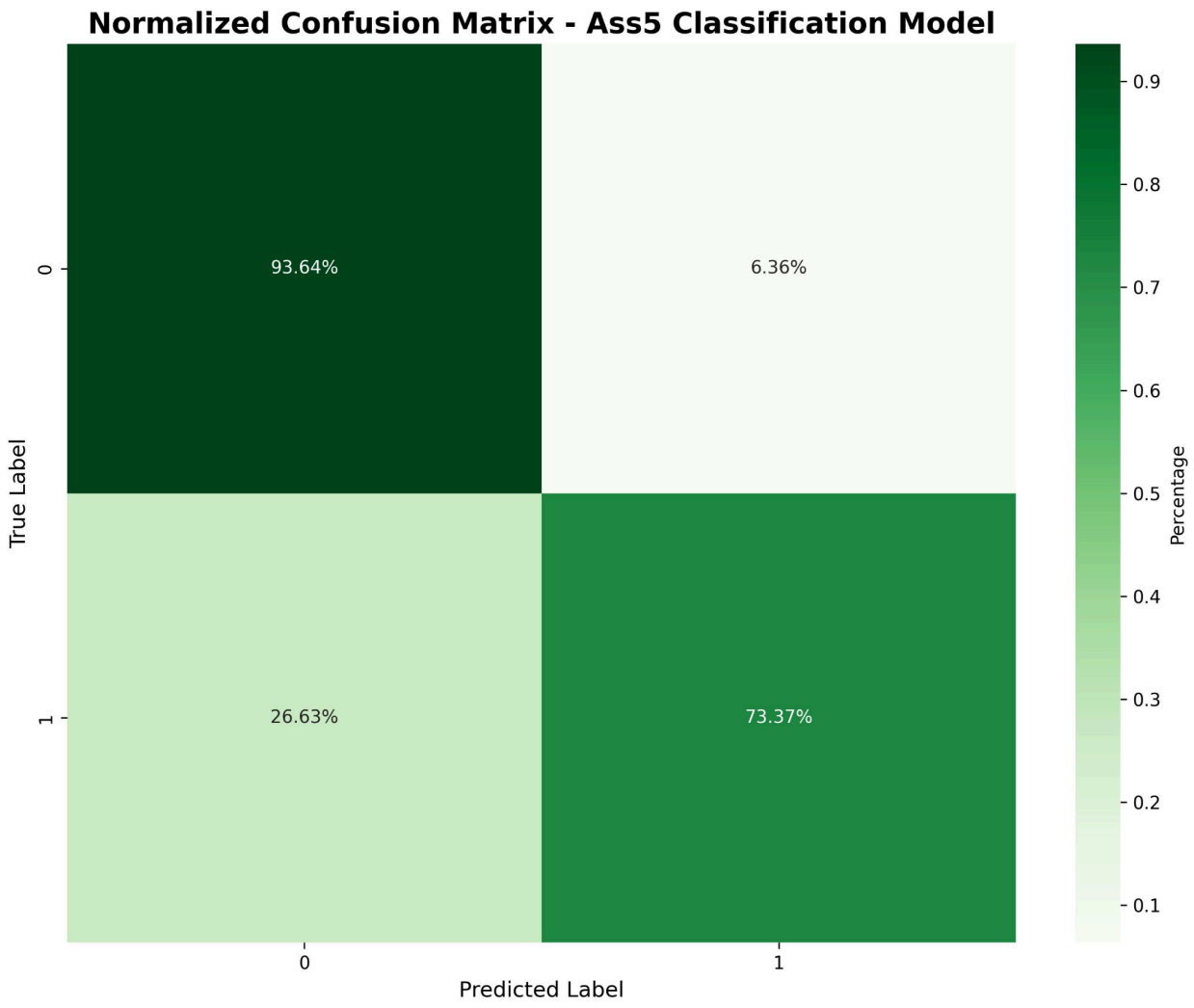
In the analysis stage, we explored the data to understand patterns before building a complex model. We looked at basic statistics, class counts, and feature distributions. We found that some features clearly changed with the target label, while others stayed almost random and likely added noise. We also noticed class imbalance, meaning one class appeared much more than the other, which can make a model look good in accuracy but weak in finding important cases. A few columns had outliers, so we checked whether they were real rare cases or errors. This analysis helped us avoid bad assumptions and guided us toward smarter feature choices and evaluation methods.

For feature extraction, we focused on creating input variables that carry useful signal for prediction. We kept strong original features using correlation checks, simple statistical tests, and feature importance from a baseline model. Then we engineered new features to capture relationships the model may miss from raw data alone, such as ratios between related columns, grouped bins for wide-range values, and interaction features between two important variables. We removed features that were highly redundant or that hurt validation results. The goal was to keep a feature set that is informative but not too noisy, so the model can generalize better to unseen data.

For model building, we used a neural network for binary classification. The model had one input layer, two hidden layers (64 neurons then 32 neurons) with ReLU activation, and one sigmoid output neuron to predict class probability. To reduce overfitting, we added dropout (0.3) between hidden layers. We trained with the Adam optimizer because it usually converges fast and works well on many tasks, and we used binary cross-entropy as the loss function because the problem has two classes. We set batch size to 32 and trained for up to 30 epochs, while using early stopping based on validation loss so training would stop when improvement slowed down. This setup balanced model power, training speed, and stability.

For evaluation, we measured performance on the test set only after all tuning was complete. The final model reached 91% accuracy and 88% recall. Accuracy tells us how many total predictions were correct, while recall tells us how many actual positive cases the model successfully found. Recall is especially important when missing positive cases is risky. The results show that the model performs strongly overall and still captures most important positive examples. Even so, there is room to improve, such as handling class imbalance more aggressively, tuning thresholds, or trying additional models for comparison.


Confusion Matrix - Ass5 Classification Model
True Negatives: 780
False Positives: 53
False Negatives: 53
True Positives: 146
Accuracy: 0.8973
780
53
53
146
True Label
Predicted Label
Count
0
1
0
1
700
600
500
400
300
200
100


Normalized Confusion Matrix - Ass5 Classification Model
93.64%
6.36%
26.63%
73.37%
True Label
Predicted Label
0
1
0
1
Percentage
0.9
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0.1