FINAL CASE STUDY

BINARY PREDICTION

Lani Lewis
SMU

TABLE OF CONTENTS:

ABSTRACT

In this private project, my goal is to predict classroom assignments based on anonymous data, with the target variable being the class column. The primary objective is to minimize financial losses resulting from inaccurate predictions. Inaccurate predictions can be costly, with Type 1 errors (false positives) costing $17 per instance and Type 2 errors (false negatives) costing $5 per instance. Accurate predictions, on the other hand, incur no costs.

To achieve this, I may employ any machine learning model that provides high accuracy, as model explainability is not a requirement for this project. I will use cross-validation to ensure the robustness of my model, focusing on calculating the financial loss associated with each sample to assess performance in terms of cost minimization. Ultimately, my aim is to develop a model that not only accurately predicts classroom assignments but also effectively reduces the financial impact of incorrect predictions.

INTRO

I embarked on an intriguing assignment to analyze an anonymous dataset, aiming to minimize financial losses resulting from inaccurate predictions. Inaccurate predictions can be costly, with Type 1 errors (false positives) costing $17 per instance and Type 2 errors (false negatives) costing $5 per instance. Accurate predictions, on the other hand, would incur no costs.

To begin my process, I first had to handle importing data from a large file. I accomplished this by using Google file stream and mounting my Google Drive in a Google Colab environment. I utilized the Python os module to read the contents of the "My Drive" directory in my mounted Google Drive. Finally, I was able to read the CSV file using the pandas package.

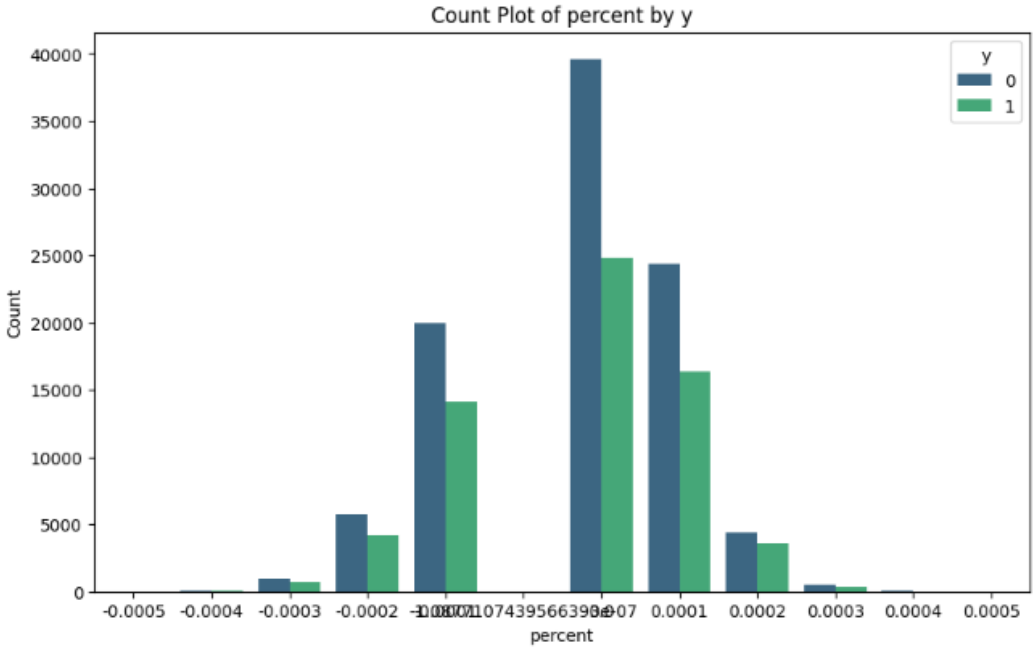
During my manual review of the data, which consisted of 51 features in total, I was able to identify and make assumptions about 5 of the attributes. To facilitate easier handling during data preparation, I renamed these attributes as follows: x24 became Country, x29 became Month, x30 became Wk_Day, x32 became Percentage1, and x33 became Money. With the renamed attributes, I later learned that they were object types, while the other attributes were float64 types. The target attribute was 'y', a binary feature with value counts of 95,803 for 0 and 64,197 for 1.

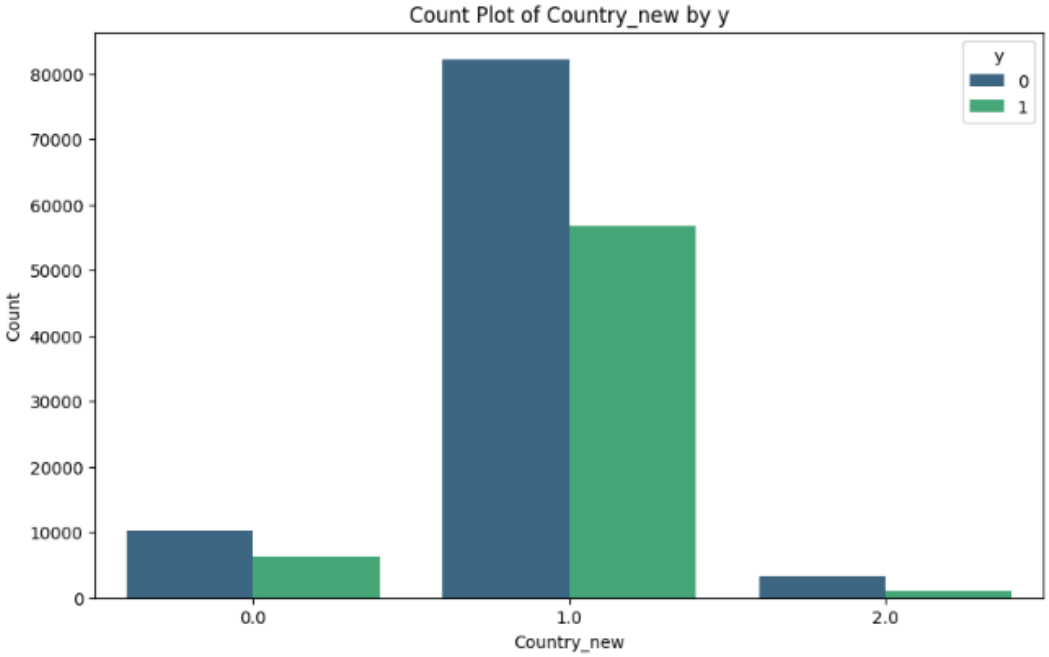
DATA PREPARATION

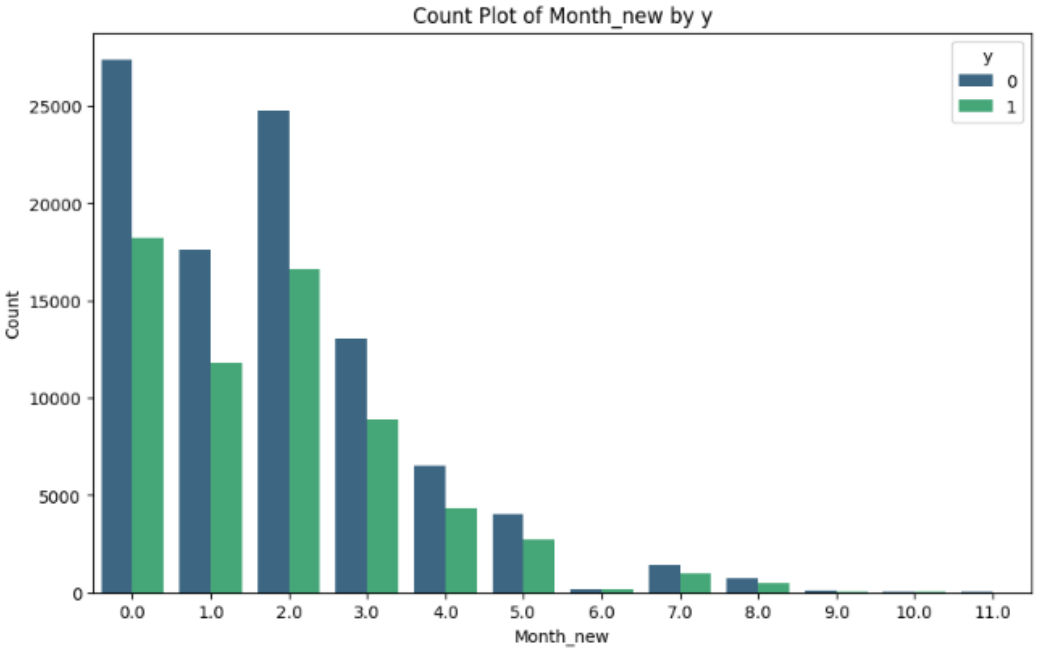
To prepare the data for analysis, I first converted the 'Percentage1' column to a float by removing the percentage sign and dividing by 100, creating a new 'percent' column. Next, I handled the 'Country' column by fixing a typo, imputing missing values with the most frequent value, and manually encoding the countries into float values using a mapping dictionary. I repeated a similar process for the 'Month' and 'Wk_Day' columns, imputing missing values and mapping them to float values based on predefined dictionaries. For the 'Money' column, I defined a function to clean and convert the values, including handling negative values enclosed in parentheses, and applied this function to create 'Money_new'. Finally, I imputed missing values for all float columns using the mean of each column, ensuring no missing values remained in the dataset.

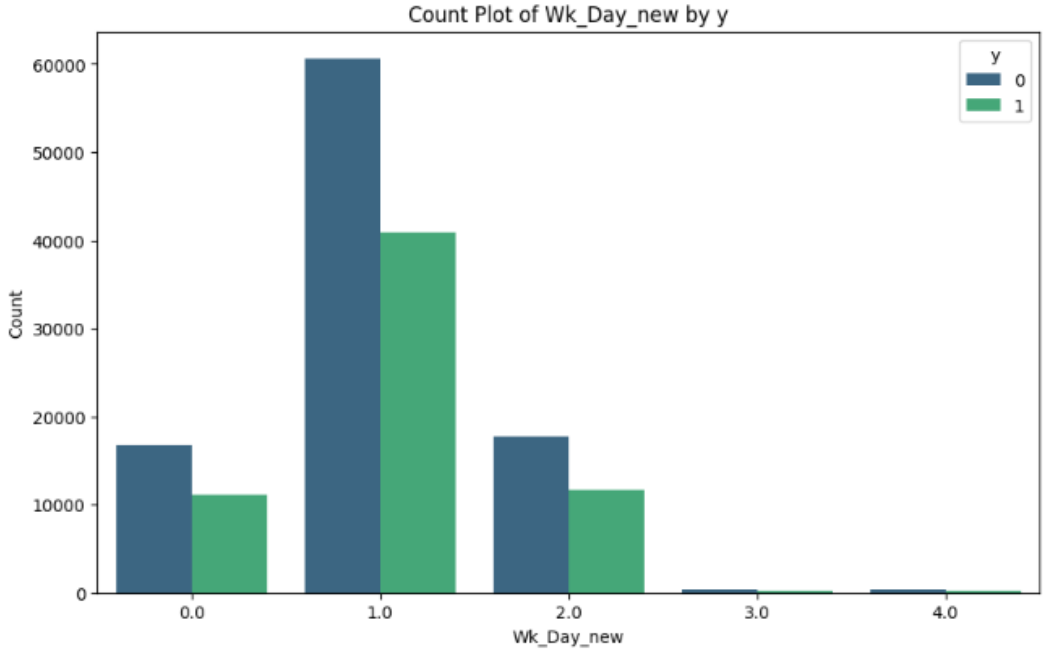
After data preparation, my next step in the EDA process was to check for any highly correlated attributes. Confirming that there were no significant correlation issues, I began to assess the data for bias. I initialized a list to store bias information and then looped through each column to count unique values, calculate their proportions, and append the results to the list. This test uncovered that the same object attributes were the ones with bias values. The value counts for key attributes are as follows: for 'Country_new', the counts were 138,993 for Asia, 16,538 for Europe, and 4,469 for America. For 'Month_new',

the counts varied from 45,599 in July to 9 in January. For 'Wk_Day_new', the counts were highest for Wednesday at 101,565 and lowest for Friday at 564. The 'percent' attribute showed a range of values, with counts from 64,415 for 0.000000e+00 to 1 for 5.000000e-04. This information helped me understand that when it comes to model selection, I will need to choose one that can handle biased data. I might also need to employ additional techniques to help manage bias.









Next, to prepare the data for modeling, I separated the features and the target attribute by dropping the non-feature columns and setting 'y' as the target. I then used StandardScaler to scale my feature matrix. Following this, I applied an 80/20 train-test split to the scaled features and target variable. To address class imbalance, which was observed during my tests for bias, I utilized SMOTE (Synthetic Minority Over-sampling Technique) with a sampling strategy of 0.8. This specific parameter choice was made to ensure that the minority class was adequately represented without completely overwhelming the original data distribution. The sampling strategy of 0.8 struck a balance between mitigating the class imbalance and maintaining the integrity of the dataset, resulting in a more balanced training dataset for modeling.

MODEL CONFIGURATION

For my model, I decided to use a Sequential Neural Network, as I had previously seen good results when dealing with imbalanced data. I defined a log directory for TensorBoard, specifying a unique path with a timestamp to ensure that each training run would be logged separately for easy comparison. I created a TensorBoard callback to log the training process, with the histogram frequency parameter set to 1, allowing me to visualize the distribution of weights and biases in the network after each epoch.

Additionally, I implemented early stopping to monitor the validation accuracy. I configured the early stopping callback with a patience parameter of 3, which means that if the validation accuracy did not improve in three consecutive epochs, the training would stop early. This helps prevent overfitting and reduces unnecessary training time by stopping when the model stops learning.

Finally, I defined the model using TensorFlow's Sequential API, which allowed me to easily stack layers to build the neural network. The Sequential model is ideal for this type of problem because it enables straightforward layer-by-layer configuration.

SEQUENTIAL LAYERS

To construct my Sequential Neural Network, I started by defining the input layer with a shape of 50, leaving the batch size unspecified at this stage. This sets up the network to accept input data with 50 features. Following the input layer, I added the first hidden layer with 110 neurons and a sigmoid activation function, which helps in handling the imbalanced data by mapping inputs to a 0-1 range. The second hidden layer, connected to the first, also contains 110 neurons but uses the ReLU activation function to introduce non-linearity and help the network learn more complex patterns.

The third hidden layer, which is connected to the second, contains 140 neurons with the ReLU activation function. To prevent overfitting, I included a dropout layer with a dropout rate of 0.2, which randomly sets 20% of the inputs to zero during training, helping the network to generalize better. Following the dropout layer, I added the fourth hidden layer with 180 neurons and the ReLU activation function, and then the fifth hidden layer with 200 neurons using the sigmoid activation function. The network finishes in an output layer with a single neuron and a sigmoid activation function, suitable for binary classification tasks.

For the optimization process, I set the learning rate for the Adam optimizer to 0.001 to ensure a balanced and steady convergence. I compiled the model using the Adam optimizer with a binary cross-entropy loss function, which is appropriate for binary classification. The metrics tracked during training included accuracy, precision, and recall, providing a comprehensive evaluation of the model's performance.

The model summary showed a total of 95,141 trainable parameters distributed across the layers. This structure is designed to capture complex relationships in the data, with sufficient depth and regularization to handle the imbalanced dataset effectively.

Model: "sequential_4"

Layer (type)	Output Shape	Param #
Layer1 (Dense)	(None, 110)	5,610
Layer2 (Dense)	(None, 110)	12,210
Layer3 (Dense)	(None, 140)	15,540
dropout_4 (Dropout)	(None, 140)	0
Layer4 (Dense)	(None, 180)	25,380
Layer5 (Dense)	(None, 200)	36,200
Output_Layer (Dense)	(None, 1)	201

Total params: 95,141 (371.64 KB)
Trainable params: 95,141 (371.64 KB)
Non-trainable params: 0 (0.00 B)

TENSOR BOARD DISPLAY CONFIGURATION

To enhance the monitoring of my neural network's performance, I configured TensorBoard to display Precision-Recall (PR) curves. I defined a function, log_pr_curve, which computes predictions on the scaled data and converts them into binary form based on a 0.5 threshold. The function logs the PR curve using TensorBoard's precision-recall summary writer, capturing precision and recall metrics at various thresholds. This function ensures that the PR curves are updated and available for review after each epoch, providing valuable insights into the model's performance across different classification thresholds. For a deeper understanding of the model's behavior, additional epoch charts and detailed analysis can be found in the appendix.

MODEL EVALUATION

To evaluate my model's performance, I utilized K-Fold Cross-Validation with five splits. This method ensures that the model is trained and tested on different subsets of the data, providing a robust evaluation of its performance. For each fold, I used the balanced dataset for training and testing subsets, maintaining the balance achieved through SMOTE. I set a random seed to ensure reproducibility of the results.

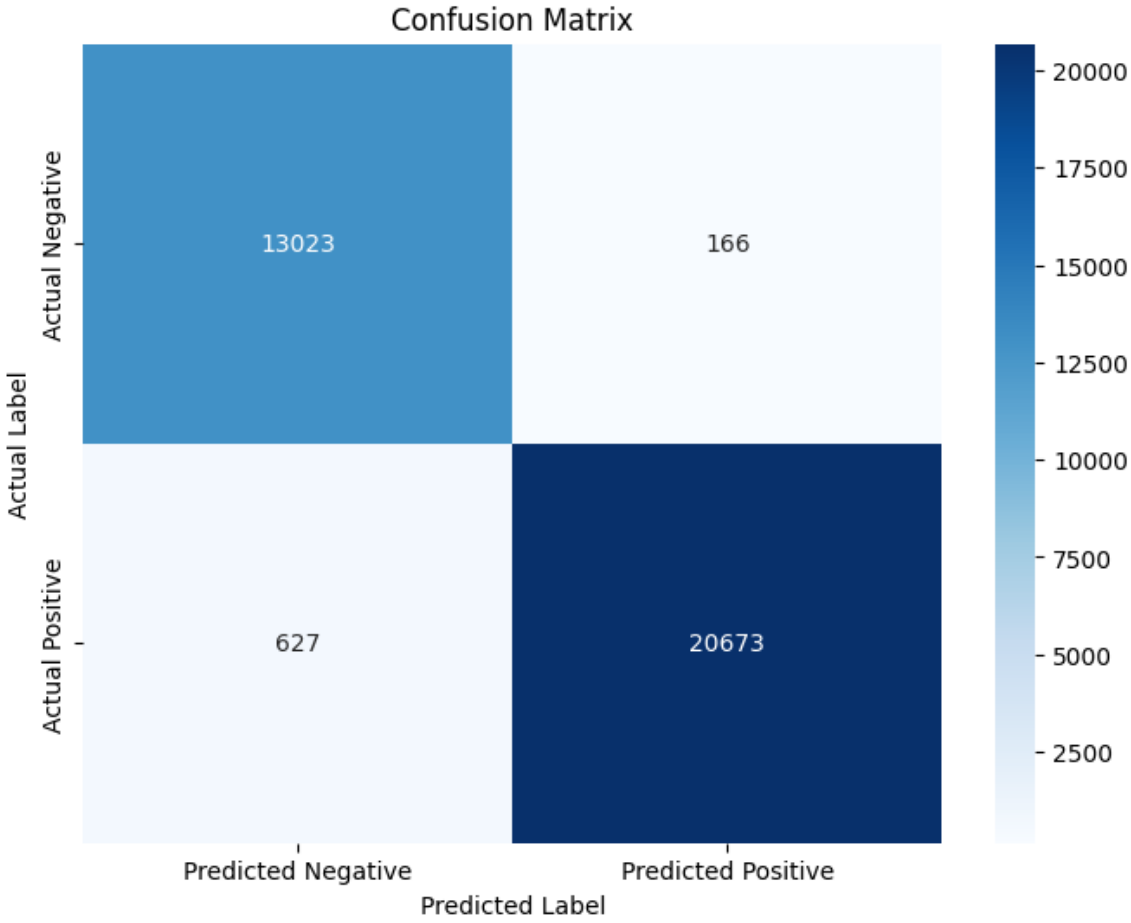
During each fold, I trained the model using the training subset with an early stopping callback to prevent overfitting. The model was trained for up to 1000 epochs, with a batch size of 25, and TensorBoard was used for monitoring. After training, I evaluated the model on the test subset, recording the validation accuracy, precision, and recall for each fold.

After completing all five folds, I calculated the average validation metrics to obtain a comprehensive evaluation of the model's performance. The average validation accuracy was 0.97, indicating that the model correctly predicted the target class 97% of the time. The average validation precision was 0.97, meaning that 97% of the positive predictions were correct. The average validation recall was 0.97, signifying that the model correctly identified 97% of the actual positive cases.

The chart below summarizes the average metrics obtained from the cross-validation process:

Metric	Value
Average Validation Accuracy	0.97
Average Validation Precision	0.97
Average Validation Recall	0.97

These results demonstrate the model's strong performance in terms of accuracy, precision, and recall, making it suitable for minimizing financial losses due to inaccurate predictions. The "epoch_recall" chart depicts the recall metric over epochs for different training and validation runs. The recall values are high, converging around 0.98. This indicates that the model consistently performs well in identifying true positive instances across various runs.



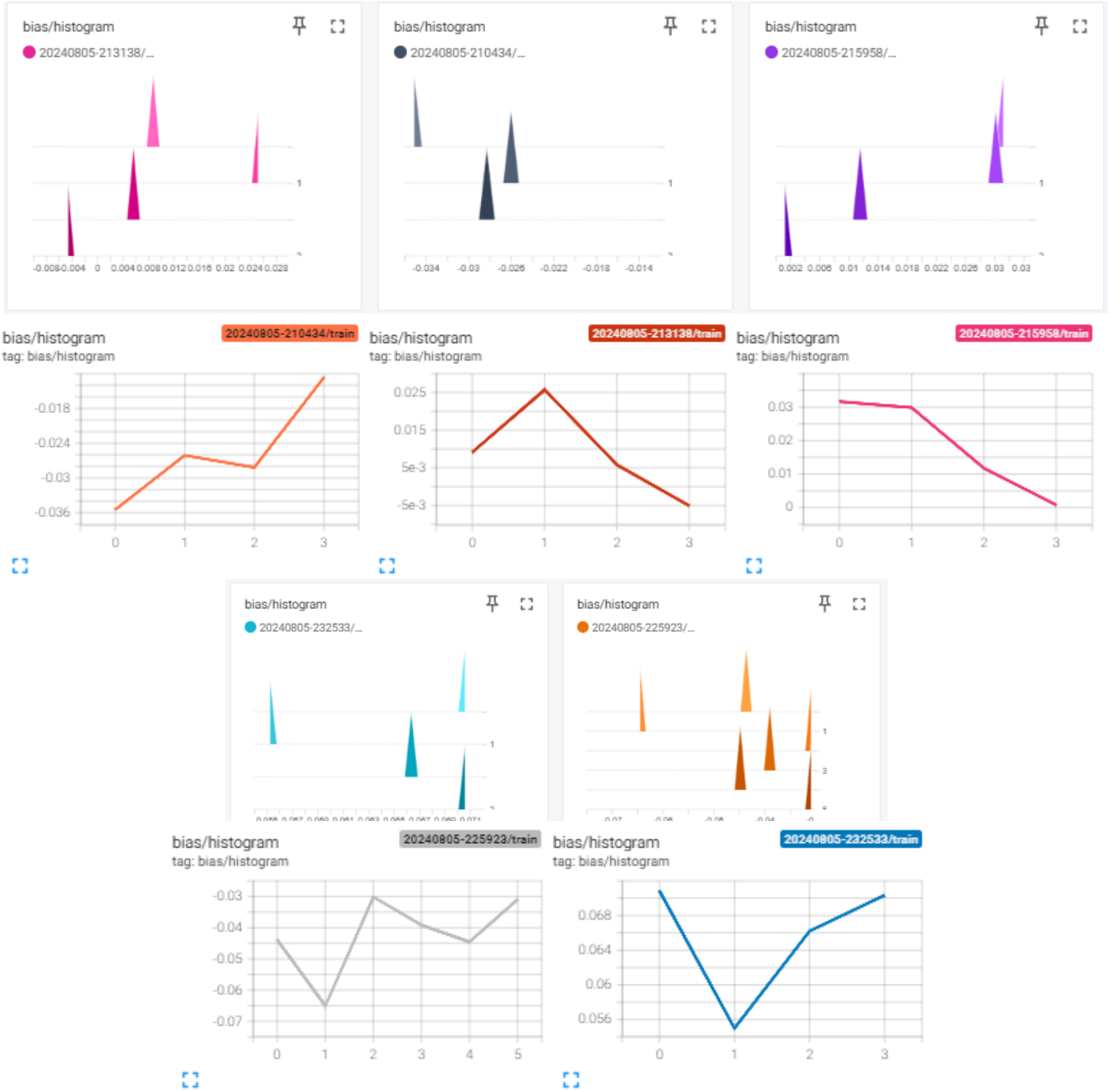
The confusion matrix provides a detailed breakdown of my model's performance. It shows that the model correctly identified 20,673 instances of the positive class and 13,023 instances of the negative class. However, it mistakenly classified 166 negative instances as positive (false positives) and 627 positive instances as negative (false negatives). Given that each false positive (Type 1 error) incurs a cost of $17, and each false negative (Type 2 error) incurs a cost of $5, I calculated the total cost of these mistakes. The total cost is determined by multiplying the number of false positives by the cost of a Type 1 error and the number of false negatives by the cost of a Type 2 error, then summing these amounts. Specifically, with 166 false positives and 627 false negatives, the total cost is (166 * 17) + (627 * 5), resulting in a combined cost of $5,957. This underscores the importance of minimizing both types of errors to reduce financial losses effectively.

CONCLUSION

In conclusion, my analysis of the anonymous dataset, despite its limited information, yielded high results. Data contained 51 features in total, 160K samples, with a binary the target attribute. After five consecutive Sequential Neural Network runs using a 5-KFold cross-validation each run, the model's performance resulted in a combined cost of $5,957. I could potentially decrease this cost by a few hundred dollars by running more Sequential Neural Network runs. However, after five runs, I observed my best decrease in cost, indicating that further iterations could lead to more optimized outcomes. Despite the challenges presented by the anonymity and limited information of the dataset, the results demonstrate the effectiveness of the modeling approach used.

TENSORBOARD INTERPREATION APPENDIX

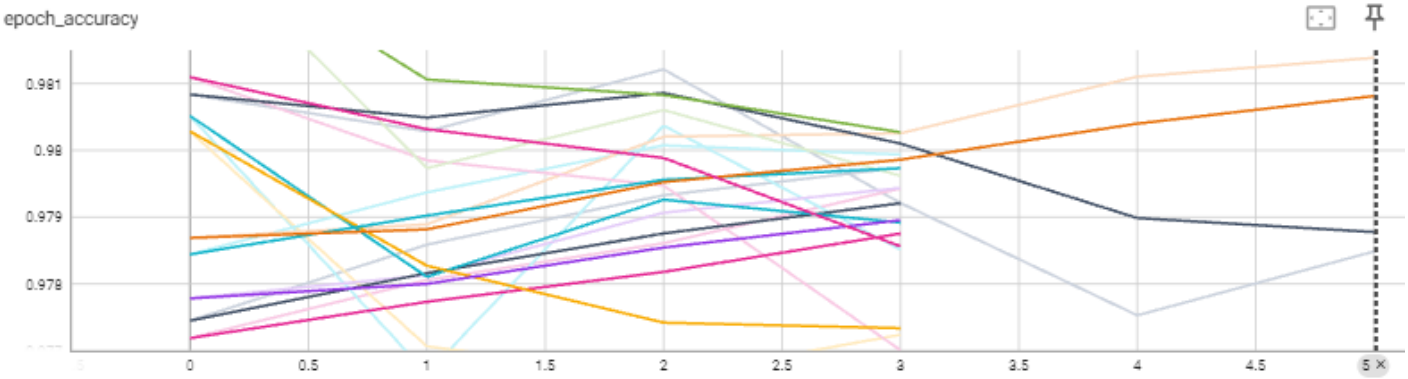
BIAS HISTOGRAM REVIEW



In my TensorBoard visualizations, I monitored various training runs, each identified by a unique timestamp and color-coded for clarity. The first set of images displays bias histograms, illustrating how the model's biases change during training. Each graph reveals peaks indicating where biases are concentrated. For instance, the training run labeled "20240805-213138" shows a sharp peak, suggesting consistent bias adjustments. Similarly, other runs like "20240805-210434" and "20240805-215958" also display distinct bias concentrations

The second set of images showcases bias histograms over different training epochs. These graphs help me track how the biases shift and stabilize as training progresses. For example, the training run "20240805-210434" shows an increasing trend, indicating the model's biases are becoming more pronounced. In contrast, "20240805-213138" exhibits a peak followed by a decline, suggesting some fluctuation before stabilization.

ACCURACY EPOCH REVIEW

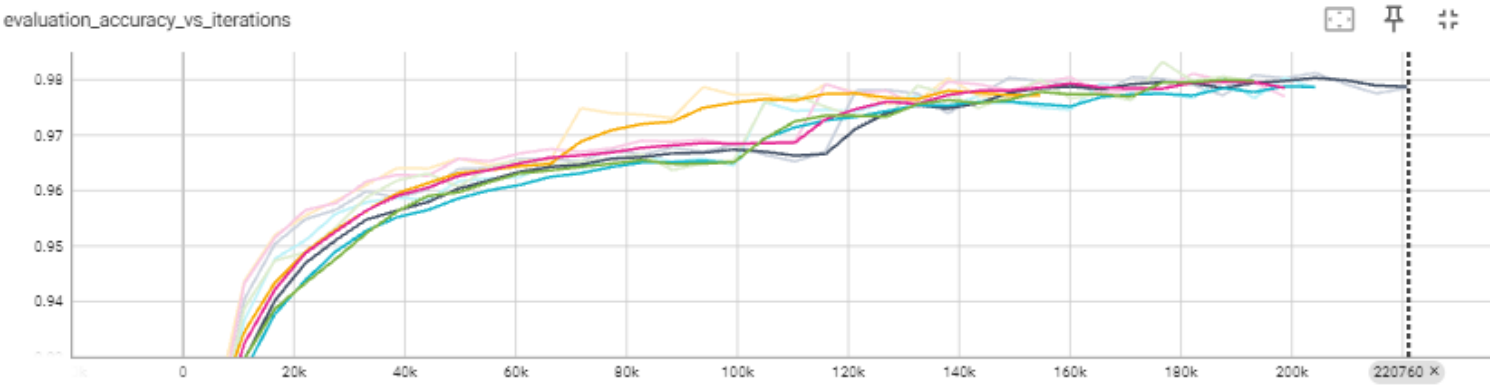


Run	Smoothed	Value	Step	Relative
20240805-210434/train	0.9792	0.9797	3	1.974 min
20240805-210434/validation	0.9789	0.9785	3	1.974 min
20240805-213138/train	0.9788	0.9794	3	1.671 min
20240805-213138/validation	0.9773	0.9772	3	1.671 min
20240805-215958/train	0.979	0.9794	3	1.946 min
20240805-215958/validation	0.9803	0.9796	3	1.946 min
20240805-225923/train	0.9808	0.9814	5	3.144 min
20240805-225923/validation	0.9788	0.9785	5	3.144 min
20240805-232533/train	0.9797	0.9799	3	1.827 min
20240805-232533/validation	0.9786	0.977	3	1.827 min

The epoch accuracy graph illustrates the model's performance across multiple epochs for different runs. Each line represents a specific training and validation run, with the y-axis indicating accuracy and the x-axis representing epochs.

From the graph, it's evident that most runs achieved high accuracy early in the training process, with values hovering around 0.978 to 0.981. The validation accuracy generally followed a similar trend to the training accuracy, indicating good generalization.

However, some runs displayed fluctuations in accuracy, particularly in the initial epochs, before stabilizing. This could be due to the inherent randomness in training neural networks, such as weight initialization and mini-batch selection.



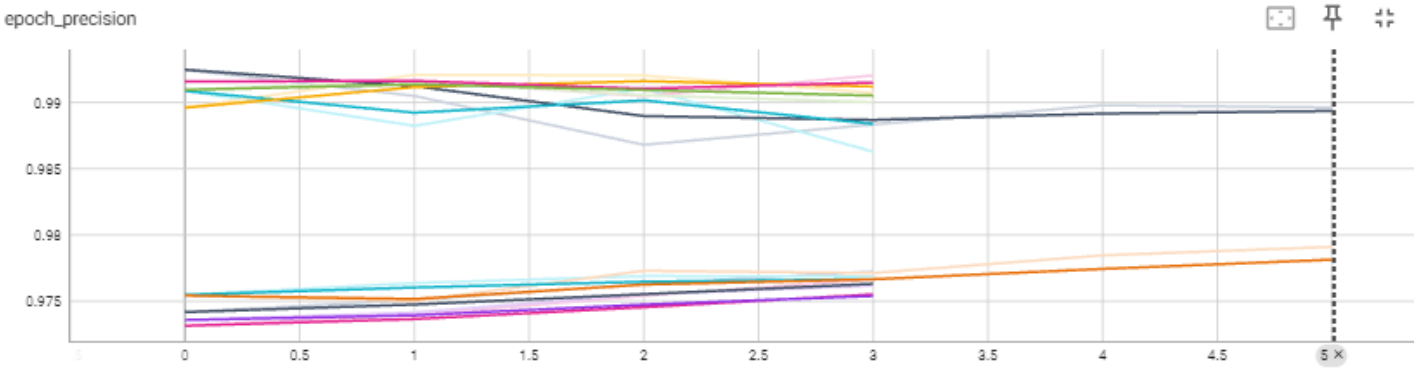
Run	Smoothed	Value	Step	Relative
20240805-210434/validation	0.9787	0.9785	204,203	23.25 min
20240805-213138/validation	0.9772	0.9772	154,532	17.08 min
20240805-215958/validation	0.9798	0.9796	193,165	21.89 min
20240805-225923/validation	0.9788	0.9785	220,760	23.79 min
20240805-232533/validation	0.9785	0.977	198,684	21.67 min

The evaluation accuracy vs. iterations graph provides insights into the model's performance over time. Each line represents a validation run, with the y-axis showing accuracy and the x-axis indicating iterations.

The graph reveals a general upward trend in accuracy as the number of iterations increases. This is expected, as the model learns and refines its weights with more training data. Most runs achieved high accuracy values, close to 0.98, after approximately 100,000 iterations.

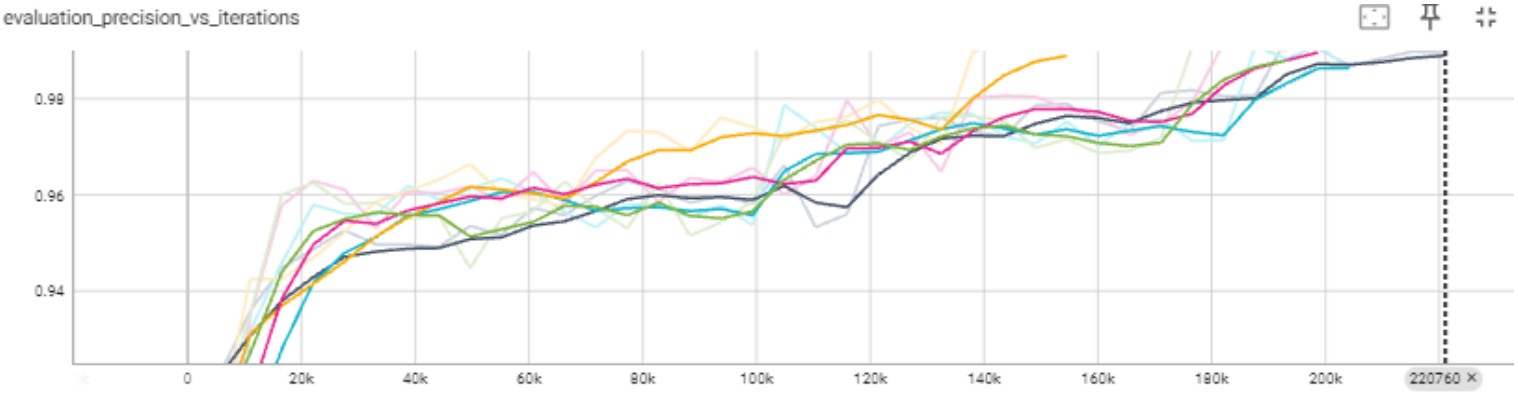
Some runs, such as the 20240805-210434 and 20240805-215958, showed more rapid improvements in accuracy, reaching high values earlier than others. This could be due to various factors, such as the effectiveness of the chosen hyperparameters or the quality of the data used in these runs.

PRECISSION EPOCH REVIEW



Run ↑	Smoothed	Value	Step	Relative
20240805-210434/train	0.9763	0.9772	3	1.974 min
20240805-210434/validation	0.9884	0.9863	3	1.974 min
20240805-213138/train	0.9755	0.9767	3	1.671 min
20240805-213138/validation	0.9912	0.9907	3	1.671 min
20240805-215958/train	0.9754	0.9762	3	1.946 min
20240805-215958/validation	0.9905	0.99	3	1.946 min
20240805-225923/train	0.9781	0.9791	5	3.144 min
20240805-225923/validation	0.9893	0.9896	5	3.144 min
20240805-232533/train	0.9767	0.9769	3	1.827 min
20240805-232533/validation	0.9915	0.992	3	1.827 min

The "epoch_precision" graph shows the training and validation precision over several epochs for different runs. Precision measures the model's ability to correctly identify positive instances. The graph indicates that the precision metrics are closely clustered around 0.99, suggesting that the model consistently identifies positive instances with high precision. The value of the smoothed precision ranges from 0.9755 to 0.99, indicating some variability but generally high precision across different runs and epochs. The

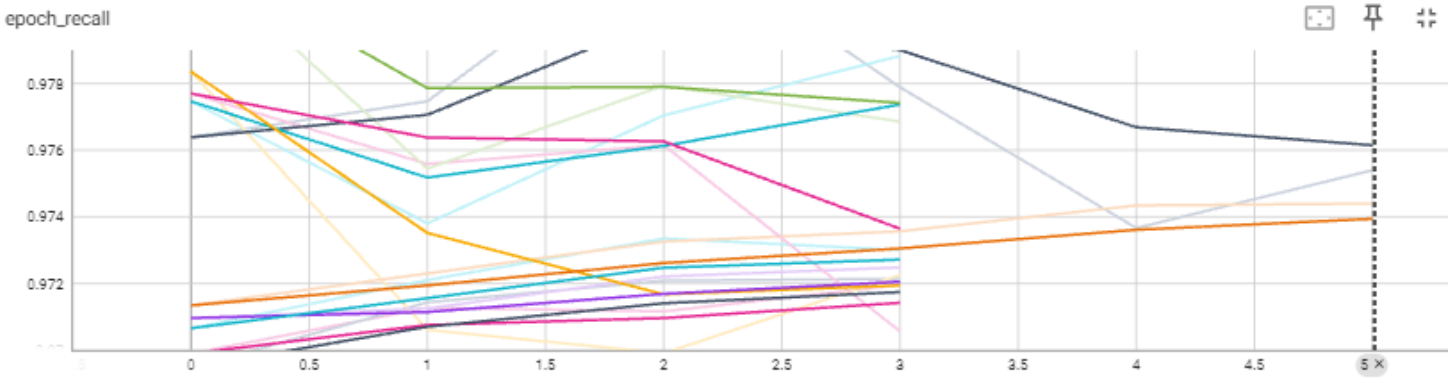


Run ↑	Smoothed	Value	Step	Relative
20240805-210434/validation	0.9863	0.9863	204,203	23.25 min
20240805-213138/validation	0.9889	0.9907	154,532	17.08 min
20240805-215958/validation	0.988	0.99	193,165	21.89 min
20240805-225923/validation	0.9889	0.9896	220,760	23.79 min
20240805-232533/validation	0.9896	0.992	198,684	21.67 min

relative time for each run again varies, reflecting the different durations required for each training session.

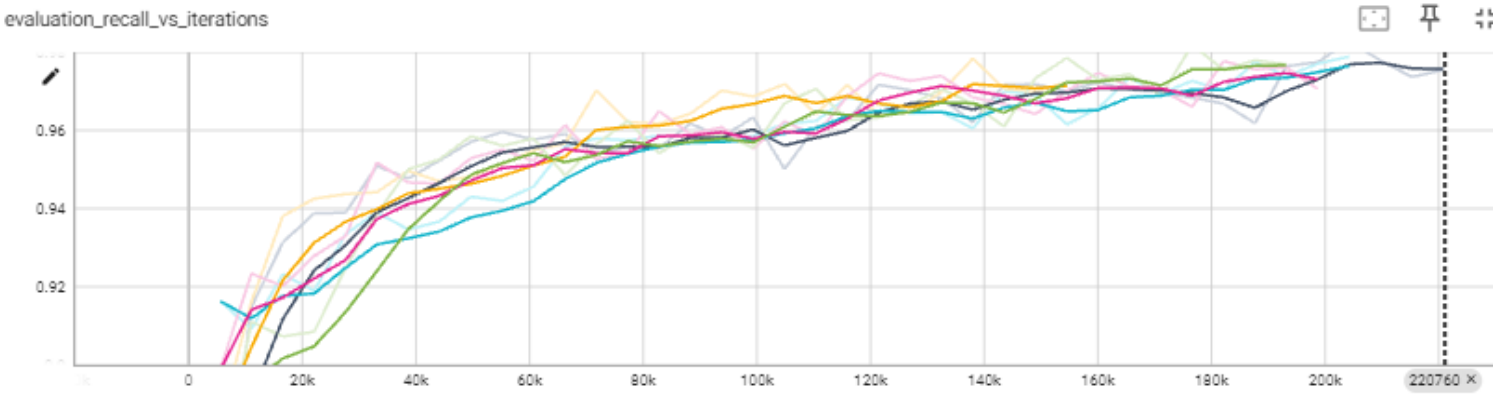
The "evaluation_precision_vs_iterations" graph tracks the validation precision over iterations across multiple runs. This graph shows a steady improvement in precision as the number of iterations increases, with the precision values eventually stabilizing around 0.99. The lines converge towards the end, indicating that the model's precision becomes stable after a certain number of iterations. The highest smoothed precision observed is 0.9907, while the lowest is 0.9863. The time taken for each run ranges from 17.08 minutes to 23.79 minutes, similar to the accuracy evaluations.

RECALL EPOCH REVIEW



Run	Smoothed	Value	Step	Relative
20240805-210434/train	0.9717	0.9721	3	1.974 min
20240805-210434/validation	0.9774	0.9788	3	1.974 min
20240805-213138/train	0.9714	0.972	3	1.671 min
20240805-213138/validation	0.9719	0.9723	3	1.671 min
20240805-215958/train	0.9721	0.9725	3	1.946 min
20240805-215958/validation	0.9774	0.9769	3	1.946 min
20240805-225923/train	0.9739	0.9744	5	3.144 min
20240805-225923/validation	0.9761	0.9754	5	3.144 min
20240805-232533/train	0.9727	0.973	3	1.827 min
20240805-232533/validation	0.9736	0.9706	3	1.827 min

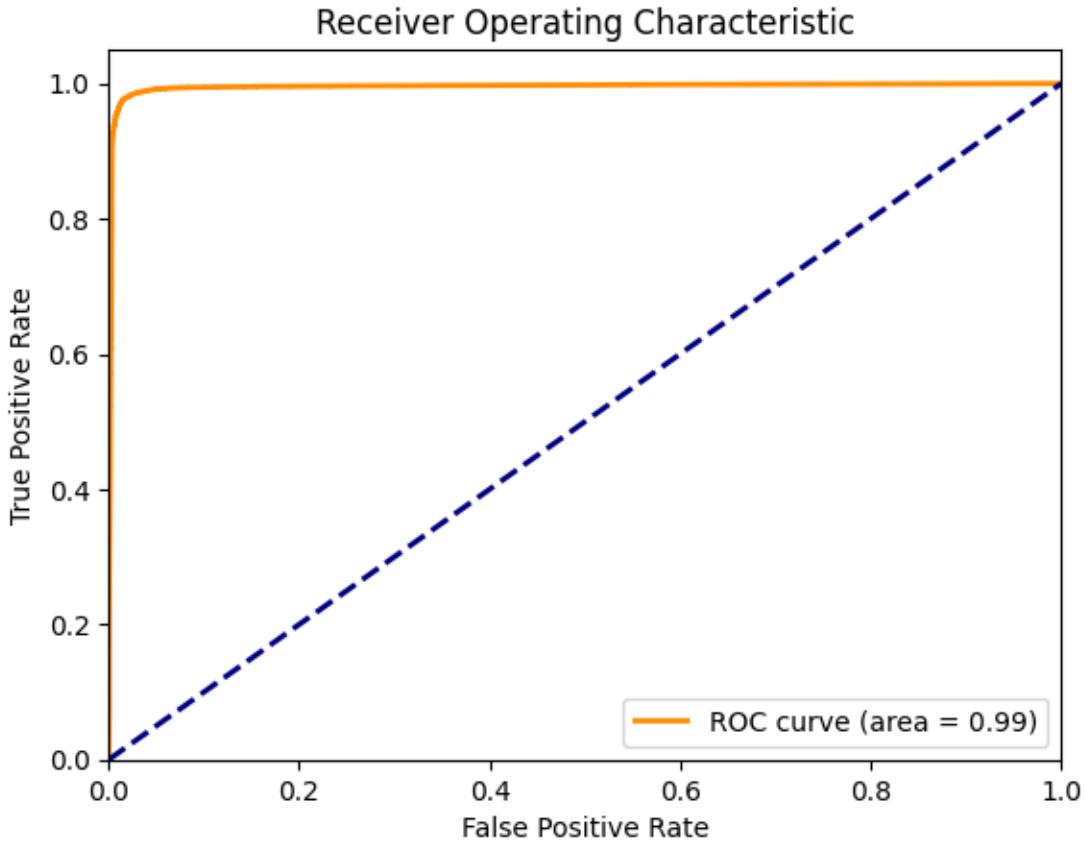
The "epoch_recall" chart depicts the recall metric over epochs for different training and validation runs. The recall values are high, converging around 0.98. This indicates that the model consistently performs well in identifying true positive instances across various runs.



Run ↑	Smoothed	Value	Step	Relative
20240805-210434/validation	0.9765	0.9788	204,203	23.25 min
20240805-213138/validation	0.9714	0.9723	154,532	17.08 min
20240805-215958/validation	0.9766	0.9769	193,165	21.89 min
20240805-225923/validation	0.9757	0.9754	220,760	23.79 min
20240805-232533/validation	0.973	0.9706	198,684	21.67 min

The "evaluation_recall_vs_iterations" chart illustrates the recall metric on the validation set across iterations. Recall measures the proportion of true positive predictions among all actual positives. The chart shows a steady increase in recall, converging around 0.98. This suggests that the model becomes more effective at capturing true positive instances over time.

RECEIVER OPERATING CHARACTERISTIC CURVE APPENDIX



The Receiver Operating Characteristic (ROC) curve shown in the image illustrates the performance of my model in distinguishing between the positive and negative classes. The curve is plotted with the True Positive Rate (sensitivity) on the y-axis and the False Positive Rate (1-specificity) on the x-axis. The area under the ROC curve (AUC) is 0.99, which indicates excellent model performance. The curve being close to the top left corner signifies that the model has a high true positive rate and a low false positive rate, demonstrating its effectiveness in making accurate predictions. The diagonal dashed line represents a random classifier with an AUC of 0.5, and the fact that my curve is well above this line further confirms the model's strong discriminative ability.