Introduction to Machine Learning

FINAL PROJECT WRITTEN REPORT

Gaurang Ruparelia, Remington Vaughan | CS-UY 4563 | 26th April, 2023

Introduction:

Connect-4 is a two-player strategy game where the objective is to connect four pieces of the same color vertically, horizontally, or diagonally on a 6x7 grid. In the game of Connect-4, the board is a 6x7 grid, and each player can place a piece in one of seven columns, resulting in a branching factor of 7. This means that at each turn, there are up to 7 possible moves that a player can make. The high branching factor in Connect-4 makes it challenging to solve optimally using traditional search algorithms. An optimal solution in Connect-4 would involve examining all possible moves from the current game state, evaluating each resulting game state, and selecting the best move based on the evaluation. However, with a branching factor of 7, the number of possible game states increases exponentially as the game progresses, quickly becoming too large to explore exhaustively.

The dataset used for this machine learning project is the Connect-4 dataset from the UCI Machine Learning Repository. The author of this data set is John Tromp and has 67,557 samples. This dataset contains all legal 8-ply positions in the game of Connect-4 in which neither player has won yet, and in which the next move is not forced. An 8-ply position refers to a board configuration in which eight moves have been made, four by each player, but neither player has won yet, and the next move is not forced. It represents a game state in the middle of the game where both players have made several moves, but the outcome is still uncertain.

The dataset has 42 features that represent board positions on a 7x6 board, where 0 represents a blank position, 1 represents a position taken by player 1, and 2 represents a position taken by player 2.

The outcome class is the game-theoretical value for the first player (a numerical value that represents the outcome of the game from the perspective of the first player, assuming optimal play by both players). The outcome class can take three values:

- 2: win, which means that the first player has won the game.
- 1: loss, which means that the first player has lost the game.
- 0: draw, which means that the game ended in a draw.

The problem that we are trying to solve with this dataset is a classification model to predict the game-theoretical value for the first player based on the current board position.

We use three machine learning techniques to evaluate this: multiclass logistic regression, multiclass Support Vector Machines (SVM), and Neural Networks. We evaluate the performance of each model on the test set using the accuracy metric and confusion matrix. We also visualize the feature weights and errors for each model using different feature transformations, regularizations, and normalization methods.

Unsupervised Analysis

The dataset consists of 43 attributes, which include 42 board positions (6 rows by 7 columns board) and one truth label column. Given the nature of our dataset, we used heat maps to recreate the connect-4 games in our dataset to better understand player 1 and player 2 strategies.

The board is arranged in 6 rows (labeled from 1 to 6) and 7 columns (labeled from a to g). A sample display of the heat map is shown in figure 1 below.

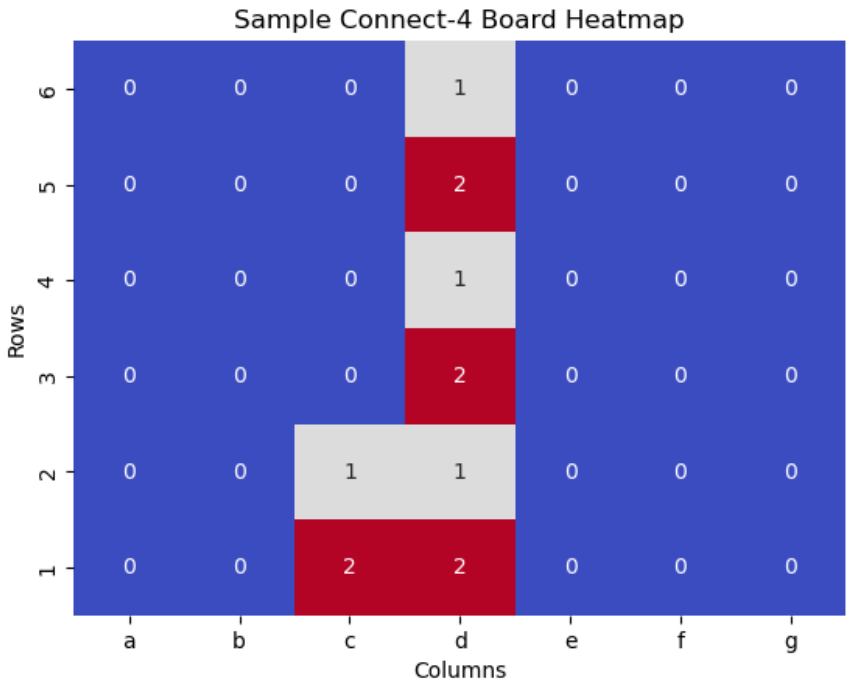

Figure 1: Sample board (heatmap) from the dataset

To conduct further analysis, we examined the net number of times that player 1 played each board position. We computed this by subtracting the number of instances of player 2 playing a position from the number of instances of player 1 playing a position for each position on the board. The heat map for this is shown below:

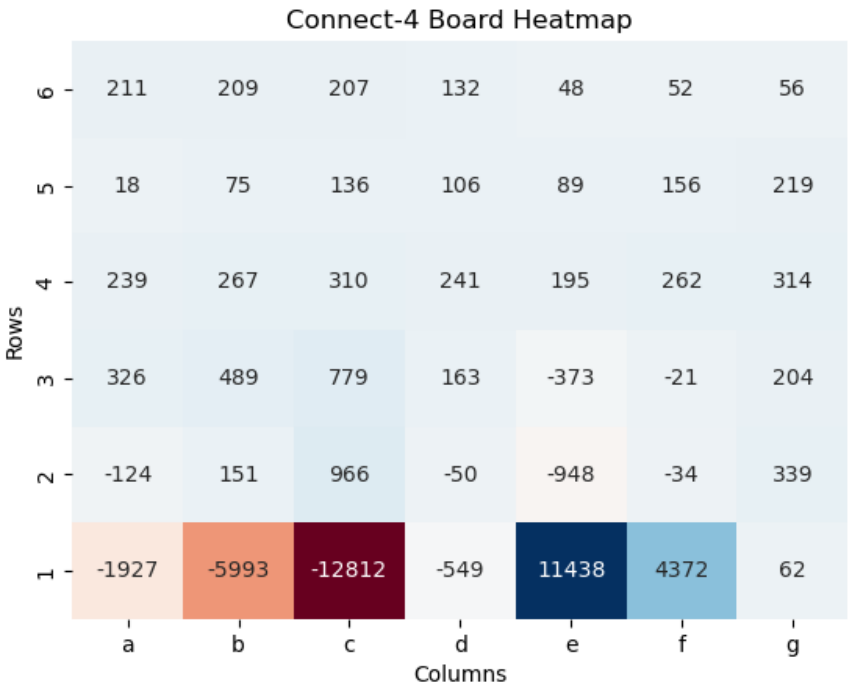

Figure 2: Net board positions from the datset

The shading of red cells in the above diagram indicates the net positions where player 1 made more moves than player 2, while the shading of blue cells indicates the net positions where player 2 made more moves than player 1. Negative values in cells represent player 1's net moves, while positive values in cells represent player 2's net moves.

The board positions at the bottom had greater polarization. This is likely due to the creator of the dataset removing boards that are mirror images of each other. Were this not the case, the heat map would necessarily be symmetrical. Additionally, the positions farther above saw much less activity. This is probably because the dataset only consisted of 8-ply boards; since in this game positions near the bottom must be played before positions near the top, the ones farther up had less chances to be played.

Lastly, we plotted a bar graph to examine the distribution of

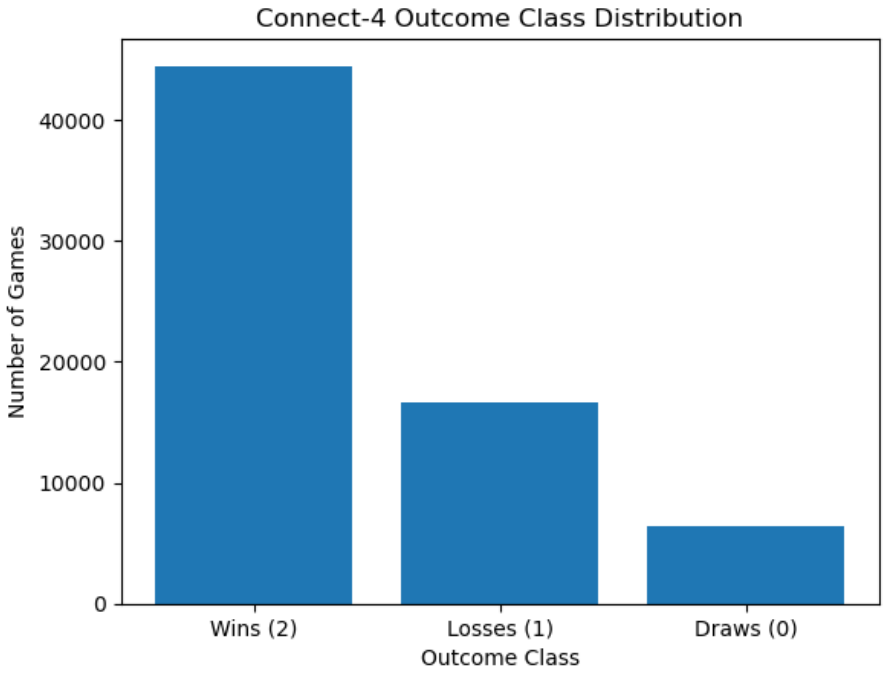

Figure 3: Distribution of outcome classes

Our main finding, however, was the fact that the dataset was highly biased in terms of the class outcomes. The dataset was roughly 65% positions where player 1 would win, 25% positions where player 1 would lose, and 10% positions where the game would draw. In the preprocessing stage, we addressed this bias.

Supervised Analysis

Preprocessing

Because our input data was already normalized, the model inputs were not significantly transformed. The target data, which initially came in ordinal coding, was transformed into one-hot encoding, since the classes do not have a natural ordering. This meant that, for Logistic Regression and SVM, three models needed to be trained to determine the class membership of a board.

Additionally, after some rudimentary training, we noticed that, when trained on the data as described above, the models would often be highly biased towards predicting class 2 as the outcome, even sometimes guessing class 2 exclusively. This makes sense as with 65% of the examples being class 2, the model would be training on class 2 much more frequently and would get a minimum ~65% accuracy when only guessing 2.

To combat this, we discarded extra data like was suggested in class. We took all of the examples from class 0, and then took an equal amount of examples from class 1 and class 2. This resulted in a smaller dataset, but with an even distribution of classes. An added benefit of this was faster training and testing, which allowed us to

experiment further with the hyperparameters. Furthermore, we ensured that all training sets, validation sets, and test sets were composed of exactly one third of each class. The training set was 80% of the remaining examples, with the validation and test sets receiving 10% each.

Method

Since the outcome column could take any one of three values, we performed multiclass logistic regression and multiclass SVM. After training the three binary classifiers, we calculate the accuracy on the training set by checking for each example, the classifier with the highest prediction and comparing that with the ground truth label. We then repeat the same process for the validation set to find the accuracy on the validation set. We adjust many hyperparameters depending on the technique. Finally, we combine the training set with the validation set, pick the hyperparameters where we get the highest accuracy on the validations set, train the model on the combined set, and evaluate it on the test set to get the final performance.

Logistic regression

The first model we picked was logistic regression. We used the sklearn implementation of logistic regression. We began by adding a bias term to the preprocessed X and y values. With 100 iterations, no feature transformations, no regularization or penalty we trained our logistic regression model and measured training set accuracy of 43% and validation set accuracy of 42%. The confusion matrix and classification report for the three labels is shown below:

Figure 4: Training set confusion matrix before hyperparameters are set

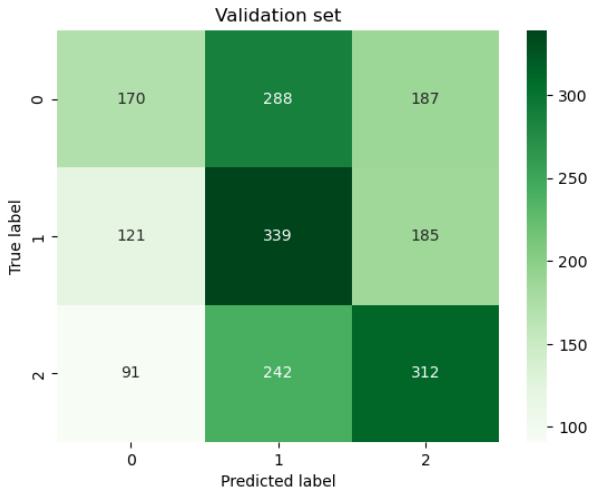

Figure 5: Validation set confusion matrix before hyperparameters are set

In this case, the confusion matrix is a table with three rows and three columns, where each row represents the true label, and each column represents the predicted label. The cells of the table contain the number of instances that were classified according to the combination of the true and predicted labels.

The diagonal of the matrix (from top left to bottom right) represents the instances that were correctly classified, while the off-diagonal cells represent the instances that were misclassified. Darker shades indicate a higher number of instances, while lighter shades indicate a lower number.

Based on the confusion matrix, we created a classification report for both the training set and validation set shown below

	precision	recall	f1-score	support
0.0	0.46	0.26	0.33	5159
1.0	0.40	0.54	0.46	5159
2.0	0.44	0.48	0.46	5159
accuracy			0.43	15477
macro avg	0.43	0.43	0.42	15477
weighted avg	0.43	0.43	0.42	15477

Figure 6: Training set classification report before hyperparameters are set

	precision	recall	f1-score	support
0.0	0.45	0.26	0.33	645
1.0	0.39	0.53	0.45	645
2.0	0.46	0.48	0.47	645
accuracy			0.42	1935
macro avg	0.43	0.42	0.42	1935
weighted avg	0.43	0.42	0.42	1935

Figure 7: Validation set classification report before hyperparameters are set

For the training set classification report, looking at the precision, recall, and F1-score for each class, we can see that the model performs similarly across all three classes, with precision ranging from 0.40 to 0.46, recall ranging

from 0.26 to 0.54, and F1-score ranging from 0.33 to 0.46. This suggests that the model is not performing significantly better or worse for any particular class. The macro average of precision, recall, and F1-score is 0.43, which is the same as the weighted average since the support for each class is the same. This means that each class is given equal importance in the calculation of the overall metrics.

For the validation set classification report, the conclusions are similar with precision ranging from 0.39 to 0.46, recall ranging from 0.26 to 0.53, and F1-score ranging from 0.33 to 0.47, and because the macro average of precision, recall, and F1-score is 0.42.

Knowing this information, we then changed our hyperparameters to improve performance on the validation set. We trained the model with L1 and L2 regularizations, with hyperparameters of C = [0, 0.0001, 0.001, 0.01, 0.1, 1, 10, 100, 1000] for three cases: data set with no features transformed, dataset with polynomial degree two feature transformations applied, and dataset with polynomial degree three feature transformations applied. For all these models, we use 100 iterations. We experimented with 500 iterations for certain hyperparameter configurations but this did not show any significant improvements, increased training time drastically, and stressed our computing resources.

Table 1: Accuracy table for different feature transformations, regularization techniques, and lambda (regularization cost)

	0	10^{-4}	10^{-3}	10^{-2}	10^{-1}	10^{0}	10^{1}	10^{2}	10^{3}
X: L1 train	0.4353	0.3333	0.3632	0.4220	0.3921	0.4359	0.4339	0.4361	0.4346
X: L1 validation	0.4393	0.3333	0.3638	0.4248	0.3933	0.4295	0.4295	0.4331	0.4362
X: L2 train	0.4209	0.3972	0.3888	0.3845	0.3788	0.4209	0.4250	0.4255	0.4269
X: L2 validation	0.4274	0.3907	0.3798	0.3773	0.3814	0.4274	0.4243	0.4233	0.4238
X^2: L1 train	0.6377	0.3333	0.3788	0.5468	0.6186	0.6372	0.6336	0.6305	0.6305
X^2: L1 validation	0.6377	0.3333	0.3788	0.5468	0.6186	0.6372	0.6336	0.6305	0.6279
X^2: L2 train	0.6403	0.4176	0.4227	0.6109	0.6393	0.6403	0.6408	0.6413	0.6382
X^2: L2 validation	0.6403	0.4176	0.4227	0.6109	0.6393	0.6403	0.6408	0.6413	0.6382
X^3: L1 train	0.8429	0.3303	0.4472	0.6110	0.7417	0.8430	0.8836	0.8922	0.8946
X^3: L1 validation	0.6651	0.3385	0.4543	0.5984	0.6729	0.6636	0.6393	0.6217	0.6217
X^3: L2 train	0.7891	0.4865	0.6745	0.7555	0.7837	0.7891	0.7865	0.7859	0.7883
X^3: L2 validation	0.6770	0.4708	0.6388	0.6656	0.6749	0.6770	0.6780	0.6718	0.6796

Looking at the trend across rows, we see that for X: L1 and X: L2, the train and validation accuracies are relatively close to each other, indicating that these models are not overfitting or underfitting too severely. However, for X^2 and X^3, we see a larger discrepancy between train and validation accuracies, especially for X^3. This suggests that these models may be overfitting to the training data and may not generalize well to new data.

Looking at the trend across columns, for X: L1 and X^2: L2, we see a gradual decrease or no change in accuracy as the regularization parameter increases, which is expected. However, for X^3: L1 and X^3: L2, we see a peak in validation accuracy at regularization parameter=1 before decreasing at higher values of the regularization parameter. This suggests that these models may be sensitive to the choice of regularization parameter and may require more fine-tuning for the X^3 feature transformations.

It is important to note that the X^3 model seems to perform the best, but also has the highest potential for overfitting. The model seems to be learning the noise or random fluctuations in the training data, rather than the underlying pattern or trend.

We picked the model with best performance on the validation set, which was the one with 79% training set accuracy and 68% validation set accuracy with hyperparameters X^3 feature transformation, L2 regularization, and 10^3 regularization parameters. When retrained on the combined training + validation set and evaluated on the test set, we got accuracy of 67% along with the following results:

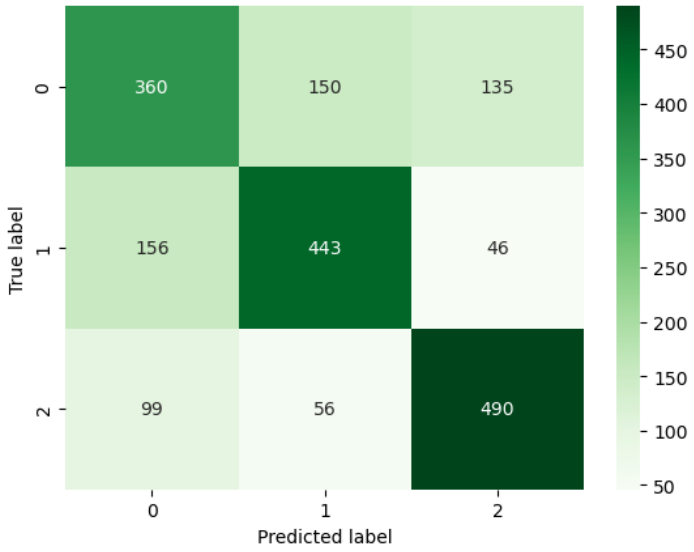

Figure 8: Confusion matrix for test set

	precision	recall	f1-score	support
0.0	0.59	0.56	0.57	645
1.0	0.68	0.69	0.68	645
2.0	0.73	0.76	0.74	645
accuracy			0.67	1935
macro avg	0.67	0.67	0.67	1935
weighted avg	0.67	0.67	0.67	1935

Figure 9: Classification report for test set

Looking at the precision, recall, and F1-score of each class, we can see that the model performs relatively well on classes 1 and 2, with precision and recall scores above 0.6 and F1-scores above 0.6. However, the model performs relatively poorly on class 0, with a precision score of 0.59 and a recall score of 0.56, resulting in an F1-score of only 0.57. This suggests that the model has a relatively high false negative rate for class 0, meaning that it is failing to correctly identify instances of class 0. The model may not be sensitive enough to class 0 or may be biased towards predicting other classes. As a result, the model may need to be adjusted or tuned to better recognize instances of class 0, which could potentially improve its overall performance.

SVM

Our second model was SVM. We chose to experiment with a linear kernel and a polynomial kernel of degree 2, as different kernel functions would take too long to train in reasonable periods of time. Each model was trained using the pegasos algorithm with L2 regularization using different values of lambda and different thresholds. We initially trained the models performance after 20 iterations and we observed the following results.

Table 2: Accuracy transform with no transform

	0	10^{-3}	10^{-2}	10^{-1}	10^{0}	10^{1}	10^{2}	10^{3}	10^{4}
0.1	0.552	0.557	0.539	0.514	0.510	0.510	0.510	0.510	0.510
0.25	0.552	0.557	0.539	0.514	0.510	0.510	0.510	0.510	0.510
0.5	0.552	0.557	0.539	0.514	0.510	0.510	0.510	0.510	0.510
0.75	0.552	0.557	0.539	0.514	0.510	0.510	0.510	0.510	0.510
0.9	0.552	0.557	0.539	0.514	0.510	0.510	0.510	0.510	0.510

Table 1: Accuracy table for different lambda and threshold values for SVM with no transform

Table 3: Accuracy transform with second degree polynomial transform

	0	10^{-3}	10^{-2}	10^{-1}	10^{0}	10^{1}	10^{2}	10^{3}	10^{4}
0.1	0.617	0.644	0.644	0.607	0.584	0.554	0.554	0.554	0.554
0.25	0.617	0.644	0.644	0.607	0.584	0.554	0.554	0.554	0.554
0.5	0.617	0.644	0.644	0.607	0.584	0.554	0.554	0.554	0.554
0.75	0.617	0.644	0.644	0.607	0.584	0.554	0.554	0.554	0.554
0.9	0.617	0.644	0.644	0.607	0.584	0.554	0.554	0.554	0.554

Table 1: Accuracy table for different lambda and threshold values for SVM with degree 2 polynomial transform

The accuracy was computed based on how frequently the trained model guessed the correct class, not how frequently each class surpassed the threshold. This is because only one class can be predicted for each input. The predicted class is the class that has the highest score for the given example, and the accuracy will be determined based on this rather than on the individual accuracies for each class.

The best performing model was the polynomial degree 2 with lambda 0.001. The threshold selected didn't seem to impact performance for any of the models we trained, so we selected 0.5. The following are the graphs for the cost and accuracy of the model for every class.

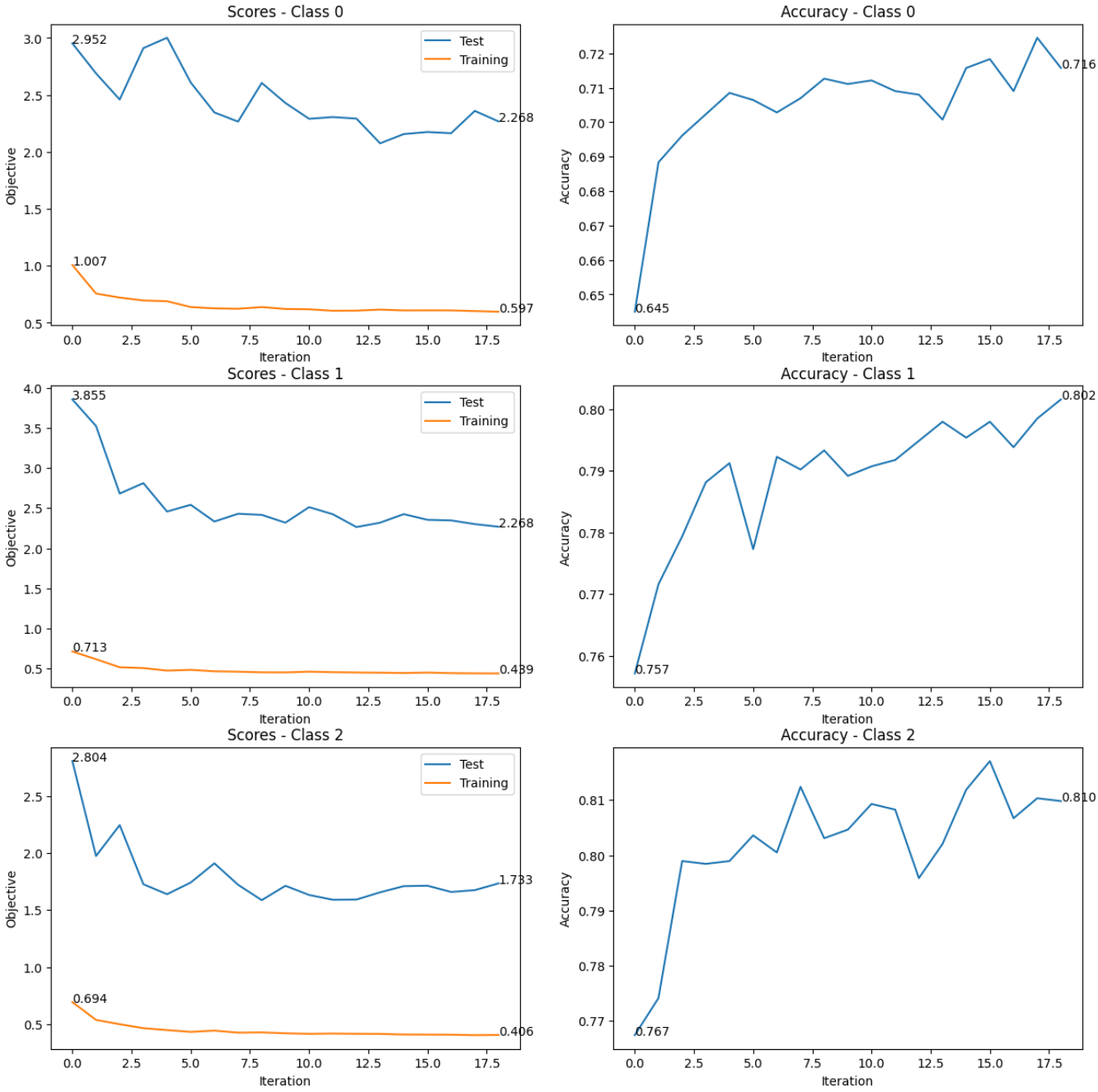

Figure 10: Cost function and accuracy for test vs training

We then trained this model on the combined train and validation data for 500 iterations and obtained a final test accuracy of 65.8%. The following are the graphs for each class like before.

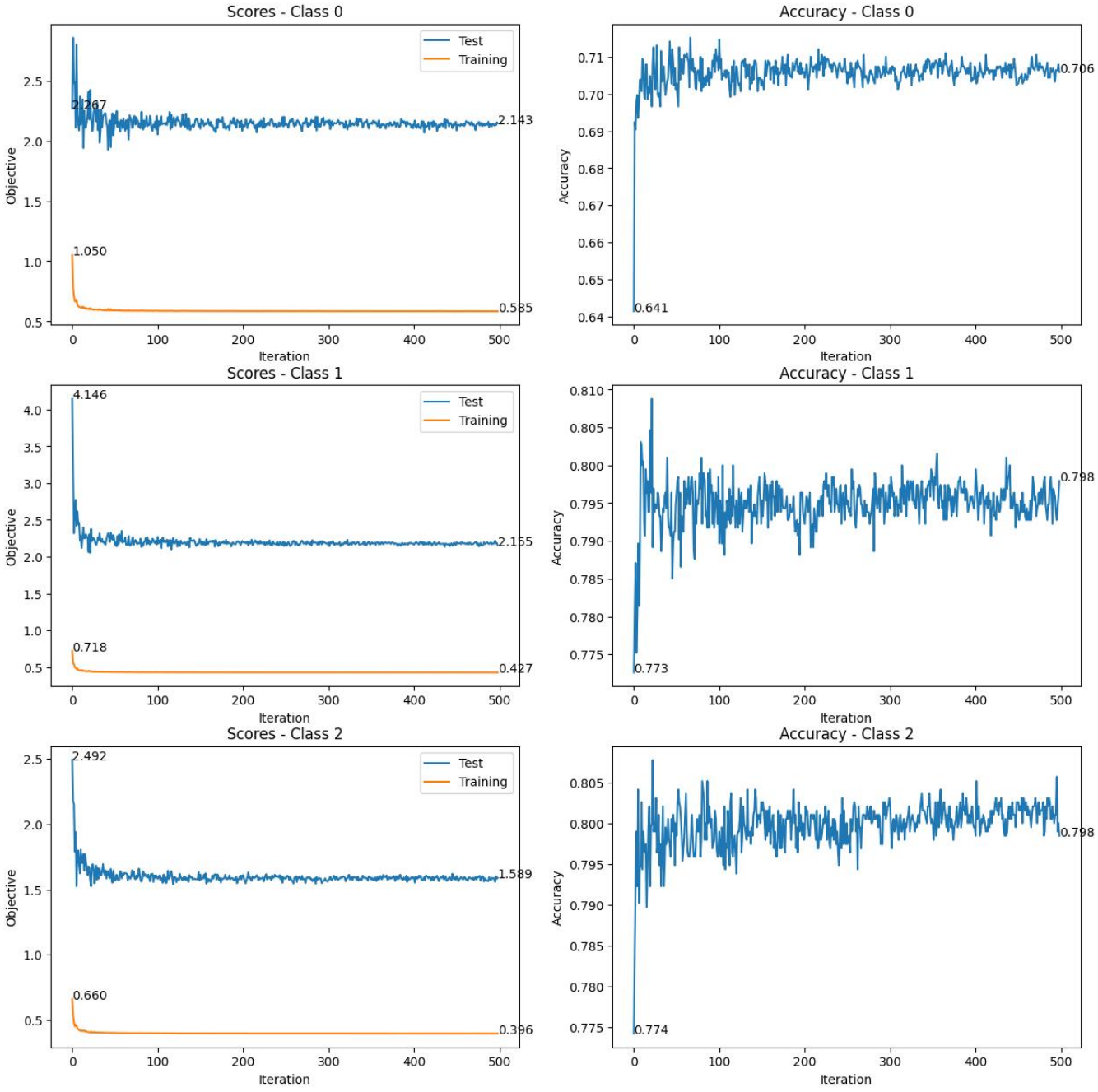

Figure 11: Cost function and accuracy for test vs training

The following is: a confusion matrix indicating how it guessed for each class; and a histogram comparing how frequently each class was guessed compared to the frequency that class showed up in the test set.

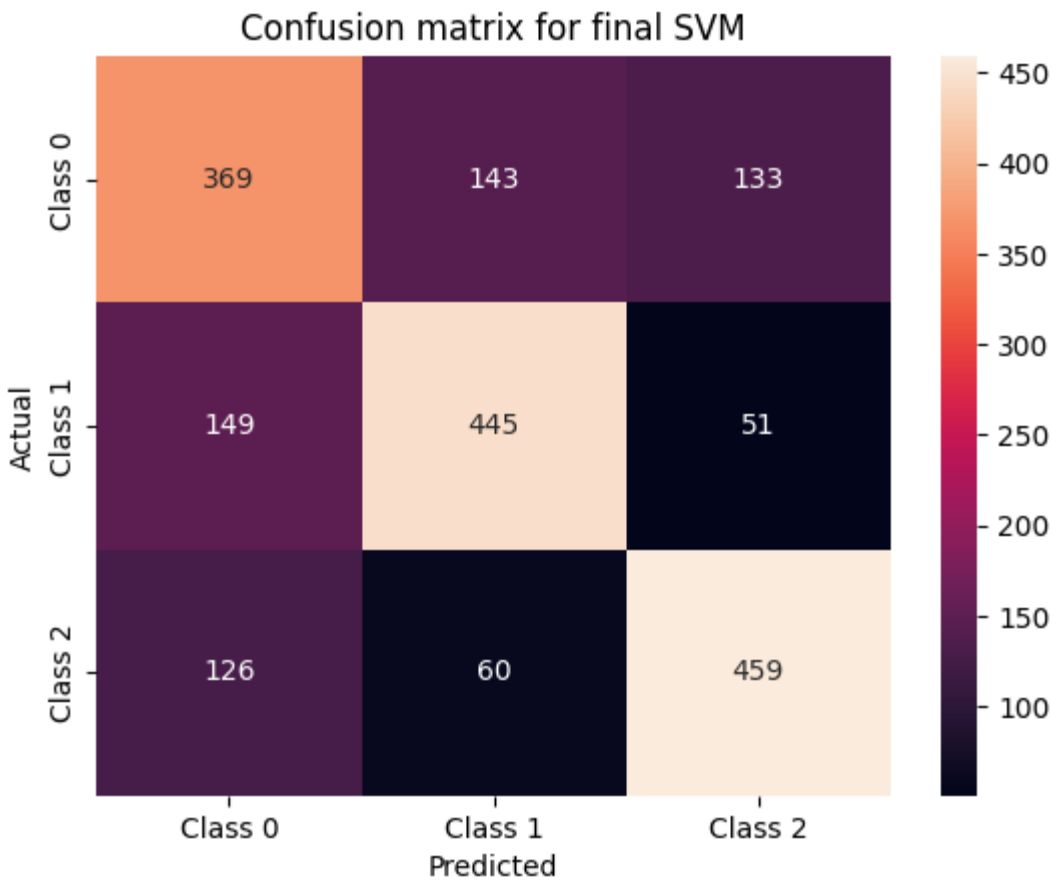

Figure 12: Confusion matrix for test set

	precision	recall	f1-score	support
0	0.57	0.57	0.57	645
1	0.69	0.69	0.69	645
2	0.71	0.71	0.71	645
accuracy			0.66	1935
macro avg	0.66	0.66	0.66	1935
weighted avg	0.66	0.66	0.66	1935

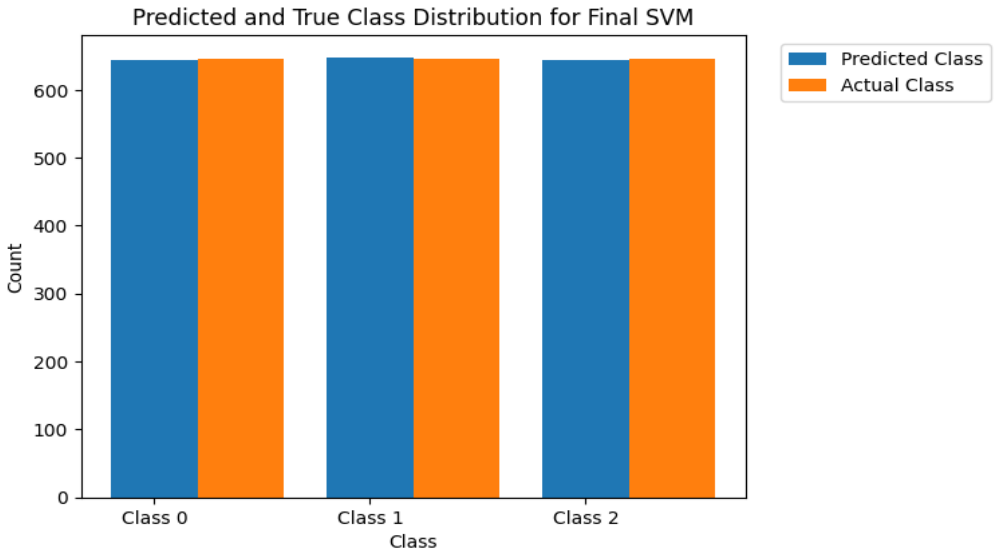

Figure 13: Histogram for test set

As seen in the confusion matrix diagram above, the diagonal (from left top to right bottom) has the largest values and the lightest color (which indicates high values). When compared to class 1 and class 2, the model is more likely to incorrectly predict class 0 (false positive), when the actual label was otherwise. There seems to also be the case that the model predicts class 1 or class 2 when the actual label was class 0 (false negative). This suggests that the model may have difficulty distinguishing class 0 from the other classes and needs more training examples on this class.

Neural Networks

Our third model type was Neural Networks. The activation functions we used were ReLU, Sigmoid, and Tanh. These were used on three different architectures: a 5-layer model [42, 30, 15, 15, 3]; a 6-layer model [42, 30, 15, 15, 15, 3]; and a 7-layer model [42, 30, 15, 15, 15, 15, 3]. All three were initially observed under L2 regularization with the values of lambda being {0, 1e-3, 1e-2, 1e-1, 1, 1e1, 1e2, 1e3] for 1000 iterations with a learning rate of 5e-6. The cost and accuracy of the training and validation sets were noted for each model after each iteration of training. The following table shows the accuracy values for each model/lambda combination, with the highest accuracy for each model highlighted.

Table 5: Accuracy for different NN architecture

	0	10^{-3}	10^{-2}	10^{-1}	10^{0}	10^{1}	10^{2}	10^{3}
5-layer ReLU	0.4786	0.4786	0.4786	0.4786	0.4786	0.4791	0.4703	0.3333
5-layer Sigmoid	0.3463	0.3463	0.3457	0.3457	0.3499	0.3488	0.3333	0.3333
5-layer Tanh	0.6057	0.6057	0.6052	0.6057	0.6062	0.6083	0.5886	0.3333
6-layer ReLU	0.3602	0.3602	0.3602	0.3602	0.3612	0.3587	0.3463	0.3333
6-layer Sigmoid	0.3359	0.3359	0.3359	0.3349	0.3344	0.3442	0.3333	0.3333
6-layer Tanh	0.6047	0.6047	0.6047	0.6047	0.6047	0.6036	0.5897	0.3333
7-layer ReLU	0.5442	0.5442	0.5442	0.5442	0.5442	0.5437	0.5406	0.3333
7-layer Sigmoid	0.3468	0.3468	0.3468	0.3488	0.3416	0.3514	0.3333	0.3333
7-layer Tanh	0.5990	0.5990	0.5990	0.5990	0.5990	0.6005	0.5752	0.3333

Table 1: Accuracy for each Model-Lambda combination

The graphs of each of these models indicated very clearly that all of these models could be left training for longer and continue to improve, and that the learning rates could be adjusted to allow faster descent for each of these models. To reduce the time of training, the lambdas were narrowed down to those between 0.1 and 100, as the best accuracy for all the models in the initial training was found within those lambda values.

Comparing the graphs of these results, however, indicated that the models using the Sigmoid activation function hadn't really been run long enough to accurately approximate how well it performs.

We increased its learning rate to 1e-5 and trained it again for 2000 iterations to see how it would compare. We only used values of Lambda between 0.1 and 100, as the best accuracy for all the models in the initial training was found within those lambda values. We observed the following results:

Table 6: Accuracy for different NN architecture and lambda

	10^{-1}	10^{0}	10^{1}	10^{2}
5-layer Sigmoid	0.4362	0.4388	0.4176	0.3333
6-layer Sigmoid	0.3488	0.3390	0.3364	0.3333
7-layer Sigmoid	0.3649	0.3623	0.3333	0.3333

Table 1: Sample table

This is the graph obtained from the 5-layer Sigmoid

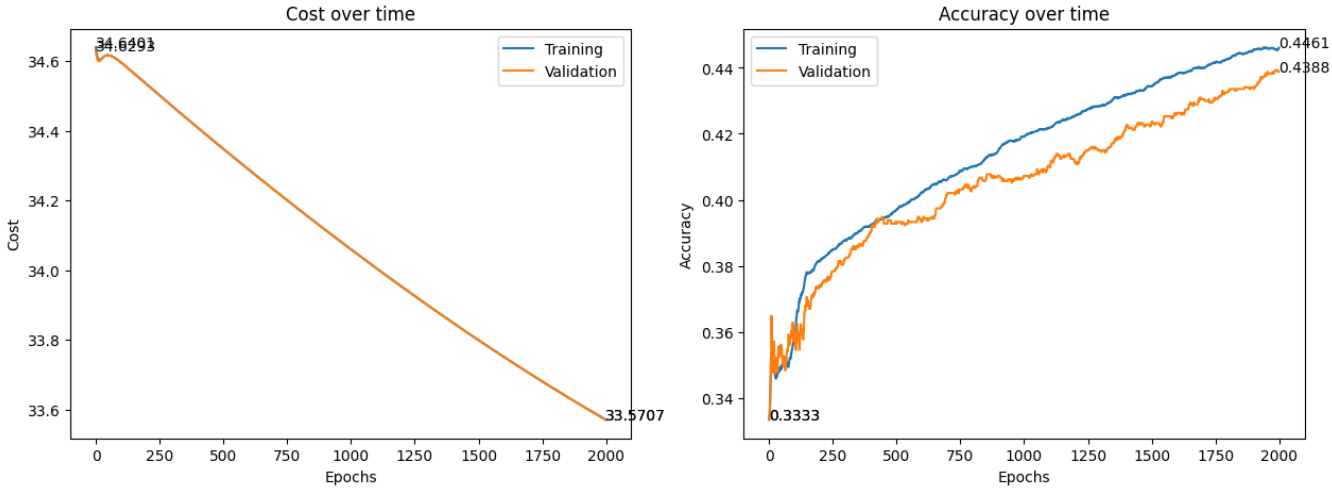

Figure 14: Cost function for NN

Even with this change, we concluded that the best performing architecture and lambda was the 5-layer Tanh. We retrained this model using both the training and validation data to obtain our final Neural Network model. The learning rate used was 5e-6, the lambda used was 10, and the iterations trained for was 25,000. The final test accuracy obtained was 72.14%. Looking at the accuracy diagram, we find that the training set accuracy, 44.6%, is very similar to the validation set accuracy, 43.8%, which suggests that the model is fitting well to the

underlying patterns in the data.

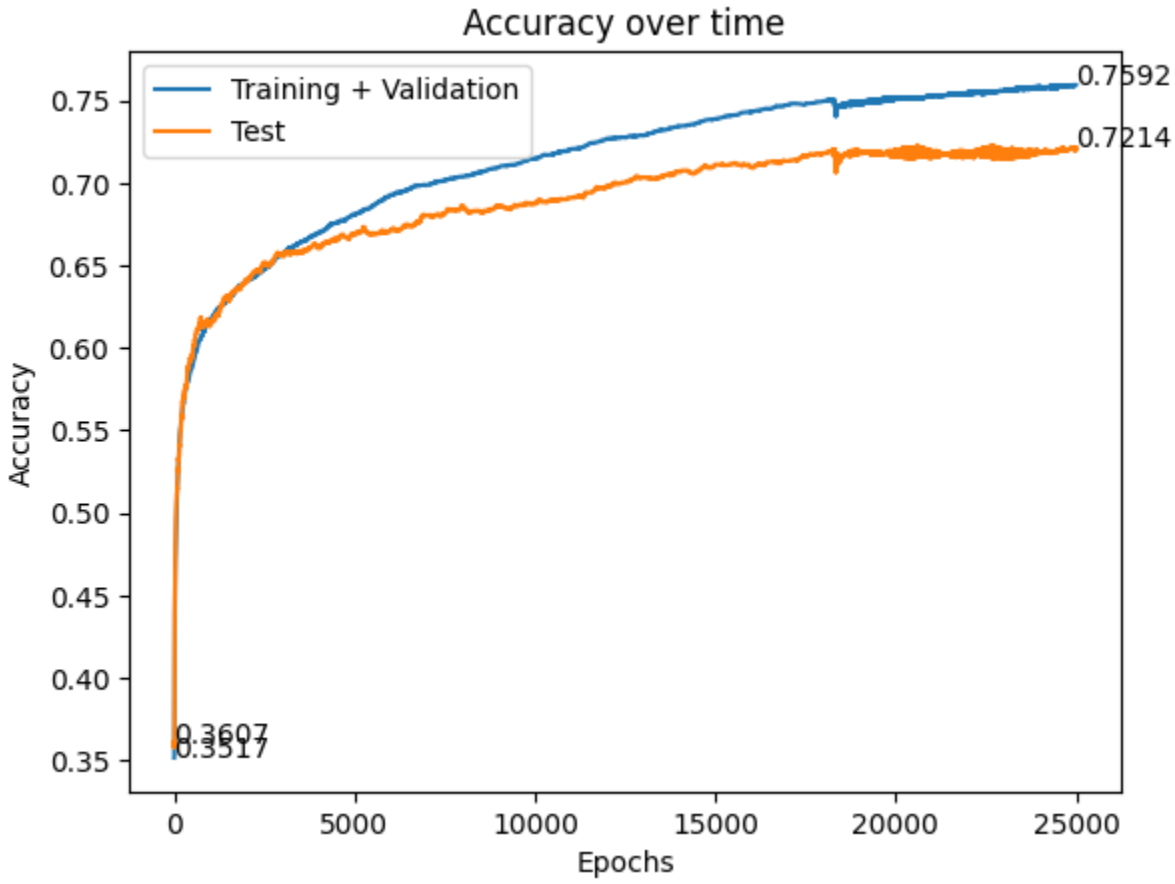

Figure 15: Accuracy for NN

As seen in the diagram above, the test set accuracy and training + validation set accuracy differ by 4% (72% and 76% respectively). This again suggests that the model fits well.

The following is: a confusion matrix indicating how it guessed for each class; and a histogram comparing how frequently each class was guessed compared to the frequency that class showed up in the test set

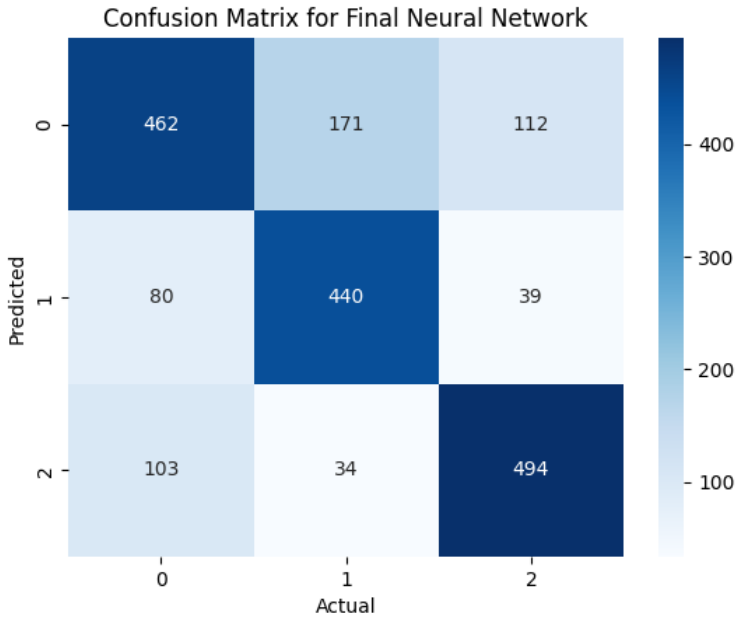

Figure 6: Confusion Matrix for NN

	precision	recall	f1-score	support
0	0.62	0.72	0.66	645
1	0.79	0.68	0.73	645
2	0.78	0.77	0.77	645
accuracy			0.72	1935
macro avg	0.73	0.72	0.72	1935
weighted avg	0.73	0.72	0.72	1935

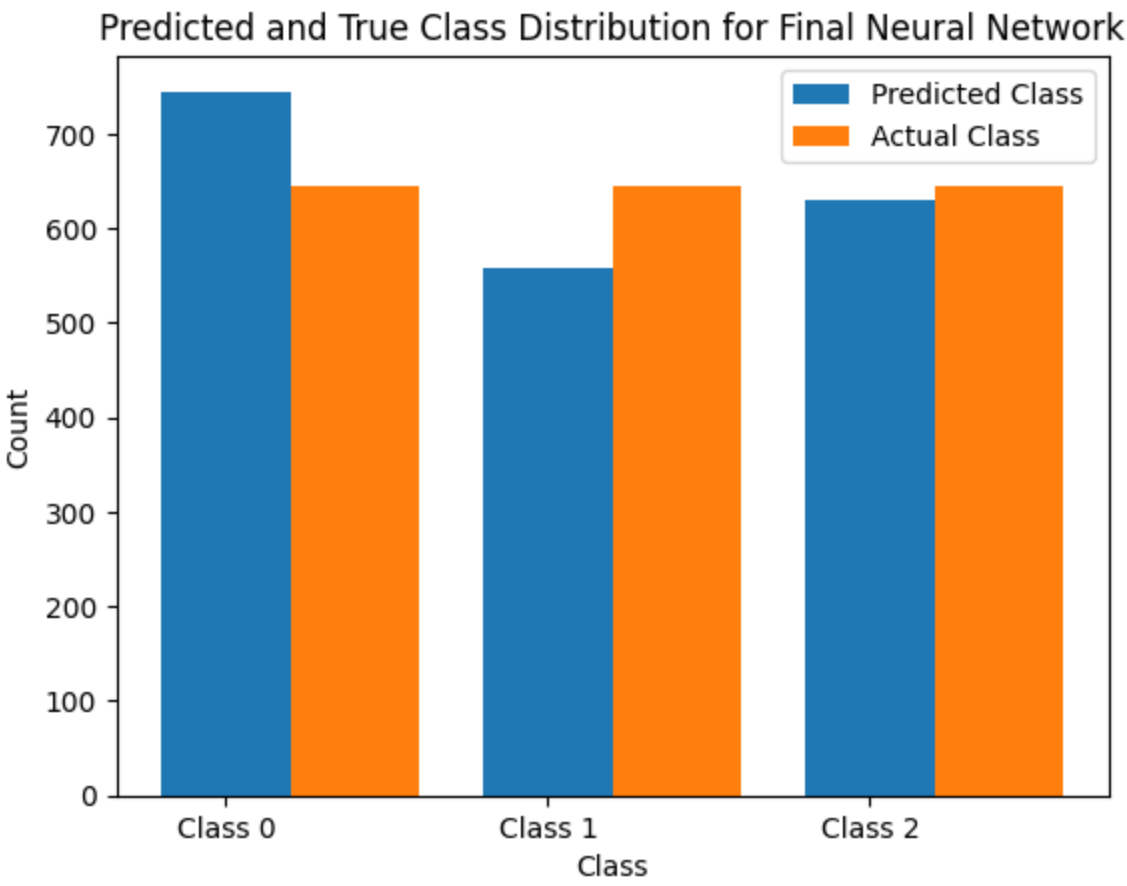

Figure 7: Histogram for NN

The confusion matrix illustrates that class 2 seems to have the fewest errors made. The histograms make a noteworthy point regarding the performance of the Neural network across classes. The model has a slight bias for class 0, meaning it guesses class 0 more frequently than other classes. An increase in the training set for the other classes might be able to correct this problem.

Conclusion

An algorithm that randomly guesses the outcome class value would achieve 33% accuracy on the test set. All three algorithms we used in this project exceeded this performance by a factor of two or more, showing that it is possible to perform better on the problem described using learning algorithms.

The best performance using the logistic regression algorithm was one in which we achieved 79% training set accuracy, 68% validation set accuracy and 67% test set accuracy with hyperparameters: X^3 feature transformation, L2 regularization, and 10^3 regularization parameters. However, it is important to note that this model was overfitting the data given the discrepancy in performance between the training set and validation set. Some improvements to deal with overfitting are data augmentation (generating new training data by applying transformations to the existing data or by playing more games), increasing the size of the validation set (helps to reduce overfitting by providing a better estimate of the model's performance on unseen data), and

higher regularization. It is also important to acknowledge that increasing C, feature transformations, and using L2 technique continued to improve performance, which implies that higher values in all three hyperparameters could further improve performance. This is a future scope of this project given our findings. A criticism of our approach could also be the limited number of iterations (100 in this case), but this decision was made given that we had to train three binary classifiers per model, increasing iterations did not show substantial performance improvement, and our computation resources were limited. The tradeoffs made it so that our decision was more feasible for this project.

The best performance using the SVM algorithm was one in which we achieved 66% test set accuracy with hyperparameters: X^2 feature transformation, and 10^3 regularization parameter. However, we discovered that the model performed poorly on class 0 despite our undersampling efforts. This can be corrected by rebalancing the class weights or using a different class weighting strategy (provided in sklearn). This will adjust the contribution of each class to the overall loss function during training and can help the model to better handle imbalanced data. Another approach is to use alternative methods for handling class imbalance, such as oversampling the minority. Additionally, it may be worth exploring other kernel functions or hyperparameters that could better capture the patterns in the imbalanced data. Finally, it may be worth considering alternative algorithms that are specifically designed for handling imbalanced datasets, such as random forests, or deep neural networks with attention mechanisms.

The best performance using the Neural Network algorithm was one in which we achieved 76% training+validation set accuracy and 72% test set accuracy with a 3-hidden layer architecture [42, 30, 15, 15, 3], tanh activation function, L2 regularization with lambda 10.

As discussed before, the original dataset had far more samples classified as class 2 than any other class. Because of this, we used undersampling and reduced the size of the dataset so that it could be more representative of each class. This substantially reduced our classification errors on class 0 and we realized that the accuracy produced from a more balanced dataset is better overall.

All things considered, our project focuses on the scenario where 8-ply positions were played, with the objective of predicting the future positions. It is important to note that this project is just a small part of a larger effort to develop an agent that can play connect 4 games with high accuracy and efficiency. By exploring the nuances of predicting future positions in this specific context, we hope to develop more robust and effective strategies for the connect 4 game. Another idea would be to use evolutionary models that train as they play as opposed to training on abstract positions before play and no change after that. The idea behind training models as they play is to enable them to learn and evolve in real-time based on feedback from the environment. This can lead to more adaptive and responsive agents that are better able to handle dynamic environments like games. One approach to implementing evolutionary models in gaming is through reinforcement learning, which involves training an agent to make decisions based on rewards or penalties received during gameplay. The agent learns to optimize its actions to maximize the total reward over time, and its performance improves as it gains more experience playing the game.

Citations

1. Tromp, John. “Connect-4.” *OpenML*, 4 June 2017, www.openml.org/search?type=data&status=active&id=40668.