CAP5404 Deep Learning for Computer Graphics
Dr. Corey Toler-Franklin

Course Project Part I Report
Simple Learning Models: Classifiers and Regressors

Submitted by:
Sandeep Jalui, Prateek Behera, Akash Kumar Kondaparthi.

1. Tic Tac Toe

The Tic Tac Toe game consists of a 3x3 square board for which a player must occupy three consecutive spots on the board to win.

There are two players, and each player takes turns occupying a spot on the board. The input features for this game include nine arguments with possible values {-1,1,0}. The nine arguments represent nine squares on the board and 1 at that square means that player X has occupied the square, -1 means that player O has occupied the square, and 0 means an unoccupied square. The output for the final game has three possibilities, 2: win, 1: loss, and 0: draw.

1.1. Tic Tac Toe Final Boards – Classification

This dataset is about the game winning or losing and we need to predict win or lose using classification algorithm. The first 9 features represent the states of each Tic Tac Toe square, and the final argument is the output feature y, the winning player. The input features include from x0, x1, …, x8 and output y. There is total 958 possible Tic Tac Toe game board configurations.

Three models were used to classify the dataset: Linear SVM, k-nearest Neighbours and Multilayer Perceptron.

1.1.1. Linear Support Vector Machine (final board)

For the test set, the accuracy scores were

Metrics	**Scores (%)**
Accuracy Score	96.875
Recall Score	95.522
Precision Score	97.710
F1 Score	96.484
Accuracy after Cross Validation	98.70
Standard Deviation after Cross Validation	1.01

Table 1: Linear SVM Evaluation Metrics (final board)

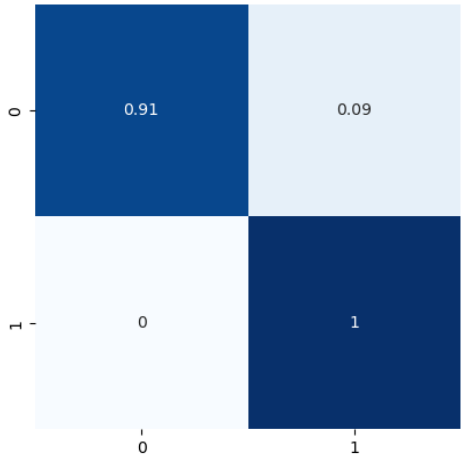

Figure 1: Linear SVM confusion matrix (final board)

1.1.2. k-Nearest Neighbors (final board)

For the test set, the accuracy scores were

Metrics	**Scores (%)**
Accuracy Score	99.47
Recall Score	99.25
Precision Score	99.60
F1 Score	99.42
Accuracy after Cross Validation	99.74
Standard Deviation after Cross Validation	0.53

Table 2: KNN Evaluation Metrics (final board)

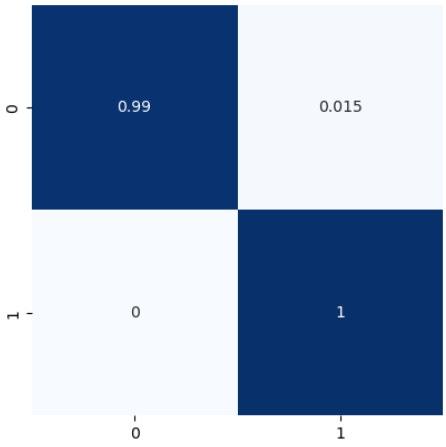

Figure 2: KNN Confusion Matrix (final board)

1.1.3. Multilayer Perceptron (final board)

For the test set, the accuracy scores were

Metrics	Scores (%)
Accuracy Score	92.19
Recall Score	89.50
Precision Score	93.4
F1 Score	91.03
Accuracy after Cross Validation	93.32
Standard Deviation after Cross Validation	6.78

Table 3: Multilayer Perceptron (final board)

The number of hidden layers selected from MLP was 5 and their respective sizes were (200, 150, 100, 100, 50). The activation function chosen was ReLU function. The default learning rate was 0.001. The maximum number of iterations allowed was 200. Tolerance for the optimization was 0.5. The rest of the values were set to default. The performances scores obtained were

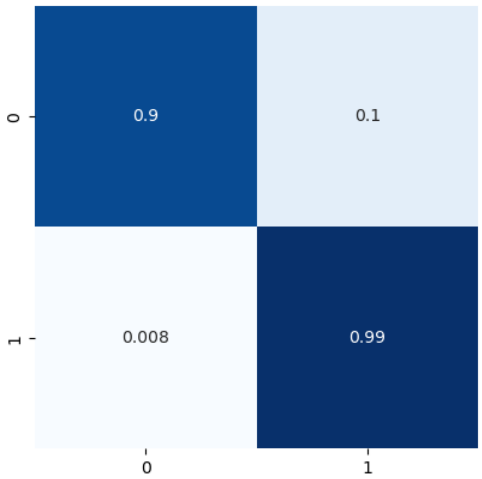

Figure 3: Multilayer Perceptron confusion matrix (final board)

Run code command: - python tictactoe.py --dataset final

1.2. Tic Tac Toe Intermediate boards optimal play (single label) - Classification
In this board, it is the player -1's turn and the goal is to calculate the optimal move for this player. We have the intermediate board here as the input states data. This is a multi-class classification task where the inputs are x0, x1, ..., x8, the states of the Tic Tac Toe squares, and the output y is the index of the best move. The output label is single labeled with values ranging from 0 to 8 representing the 9 possible moves on the tic-tac-toe board.

We divide the dataset into train and test data with a split ratio of 80%: 20% and train on three classifiers namely Linear SVM, KNN and Multilayer perceptron and evaluate them based on the following evaluation metrics:

1. Accuracy Score

2. Precision Score

3. Recall Score

4. F1 Score

We then perform 10-fold cross validation to minimize any overfitting and calculate the corresponding Accuracy and Std deviation scores.

1.2.1. Linear Support Vector Machine (Intermediate Board – single label)

After training the Linear SVM model on train dataset we evaluate the learned model on the test dataset,

For the test set, the accuracy scores were

Metrics	**Scores (%)**
Accuracy Score	13.27
Recall Score	12.99
Precision Score	14.80
F1 Score	11.17
Accuracy after Cross Validation	13.07
Standard Deviation after Cross Validation	1.86

Table 4: Linear SVM Evaluation Metrics (single label)

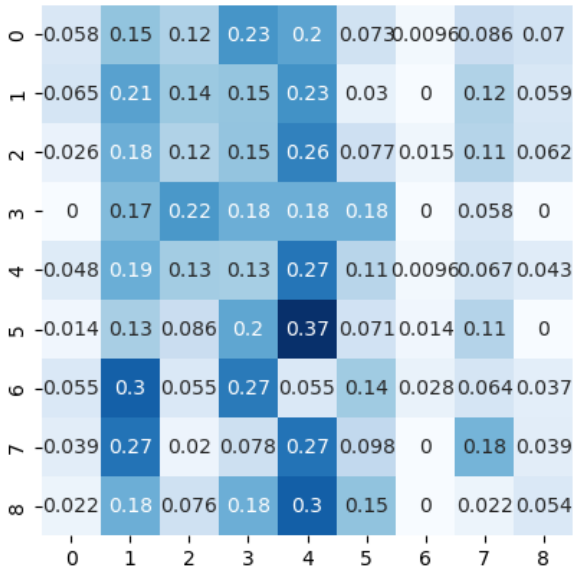

Figure 4: Linear SVM Confusion matrix (single label)

We can see that the evaluation metrics on the Linear SVM gives pretty poor values. This is because the classes are not divided in a linear way. The classification decision surface in non-linear and classifying it with a linear kernel will result in poor performance. We can verify this

by using a non-linear kernel and expect far better output. For example, here we use a Support Vector Machine with kernel RBF as the classifier and can see from the evaluation metrics how the performance improves.

For the test set, the accuracy scores were

Metrics	**Scores (%)**
Accuracy Score	87.56
Recall Score	86.73
Precision Score	87.87
F1 Score	87.10
Accuracy after Cross Validation	82.77
Standard Deviation after Cross Validation	1.66

Table 5: SVM with RBF Kernel Evaluation Metrics (single label)

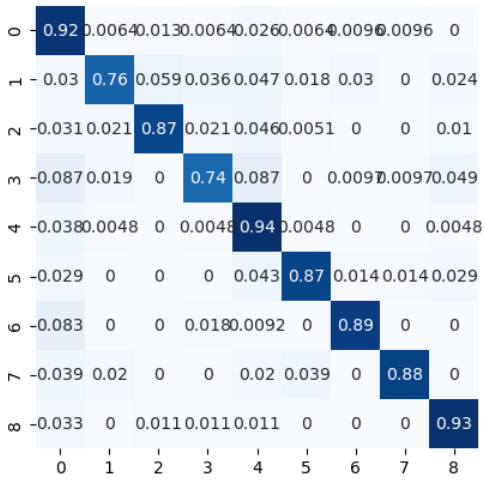

Figure 5: SVM with RBF Kernel Confusion Matrix (single label)

1.2.2. k-Nearest Neighbors (Intermediate Board – single label)

After training the KNN model on train dataset we evaluate the learned model on the test dataset. The number of neighbors is chosen as 3 after grid searching through several values. The model performs as follows:

For the test set, the accuracy scores were

Metrics	**Scores (%)**
Accuracy Score	74.75
Recall Score	70.85
Precision Score	75.15
F1 Score	72.65
Accuracy after Cross Validation	74.58
Standard Deviation after Cross Validation	1.72

Table 6: KNN Evaluation Metrics (single label)

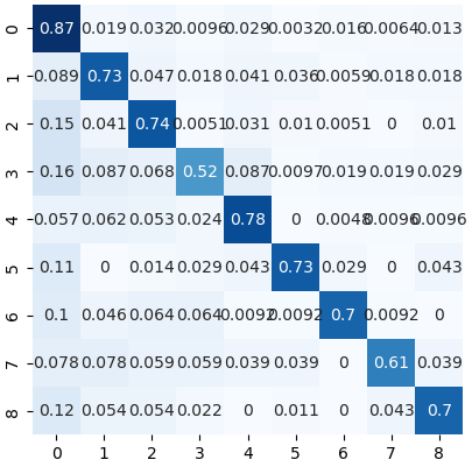

Figure 6: KNN Confusion matrix (single label)

We can see that KNN being a nonlinear model performs far better than a Linear SVM.

1.2.3. Multilayer Perceptron (Intermediate Board – single label)

After training the MLP model on the train dataset we evaluate the learned model on the test dataset. The number of hidden layers selected for the MLP was 5 and their respective sizes were (2000,1000,500,100,50,). The activation function chosen was ReLU function by default. The learning rate is chosen as 0.001. The maximum number of iterations allowed was 200. These values were chosen after running a grid search on the learning rate and hidden layer depth. The rest of the values were set to default.

For the test set, the accuracy scores were,

Metrics	**Scores (%)**
Accuracy Score	85.58
Recall Score	84.75
Precision Score	85.02
F1 Score	84.63
Accuracy after Cross Validation	86.32
Standard Deviation after Cross Validation	1.23

Table 7: MLP Evaluation Metrics (single label)

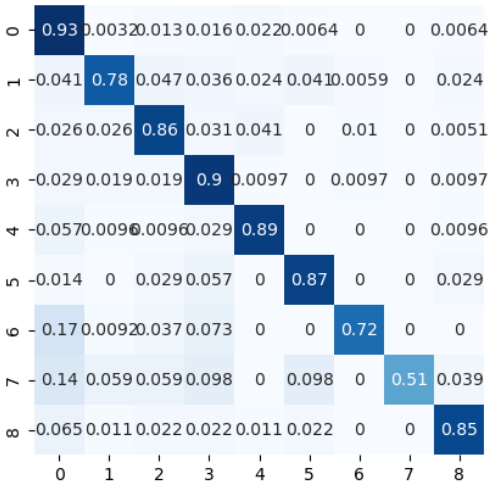

Figure 7: MLP Confusion matrix (single label)

The accuracy scores for the single label classifier for the 3 models were as follows:

1. Linear SVM – 13.27%
2. KNearestNeighbours – 76.58%
3. Multi-layer Perceptron – 86.32%

Conclusion for classifiers: -

In conclusion, we can see that the Multilayer perceptron classifier performs the best compared to Linear SVM and KNN. Linear SVM classifier shows poor performance due to the non-linearity of the dataset, it severely underfits.

Only using 1/10th of the dataset:

Now we train the classifiers using only 1/10th of the dataset.

The accuracy scores for the single label classifier for the 3 models were as follows:

1. Linear SVM – 13.72%
2. KNearestNeighbours – 53.12%
3. Multi-layer Perceptron – 51.95%

We can see KNN drops in performance, but in case of MLP the performance nose-dives from 86.32% to 51.95%. Linear SVM performs marginally better.

So, increasing the training dataset from small (1/10th of the dataset) to large scale (80% of the dataset) leads to marginally worse performance in SVM, but substantially better performance in KNN and MLP classifiers.

Thus, Linear SVM scales badly for large datasets. Whereas KNN and MLP classifiers scale pretty well.

Run code command: - python tictactoe.py --dataset single

1.2. Tic Tac Toe Intermediate boards optimal play (multi-label) – Regression

This dataset lists all the possible optimal moves for the player -1. For this dataset, the inputs are x0, x1, ..., x8 and the outputs are y0, y1, ..., y8, where the corresponding y is 1 if the given square is an optimal move for player, otherwise 0. We evaluate across three regressors namely – Linear Regressor, KNN and Multilayer Perceptron. We then use these regressors as if they were classifiers.

1.3.1. Linear Regression (Intermediate Board – Multi label)

The model implemented was linear regression with Ridge Regularization with default regularization parameter 1.0. After normalizing the data, we evaluate the performance metrics by averaging the scores across all 9 labels. We obtain the following average scores and the confusion matrix.

Metrics	Avg Scores (%)
Accuracy Score	52.08
Recall Score	23.66
Precision Score	54.52
F1 Score	32.75

Table 8: Linear SVM Evaluation Metrics (Multi label)

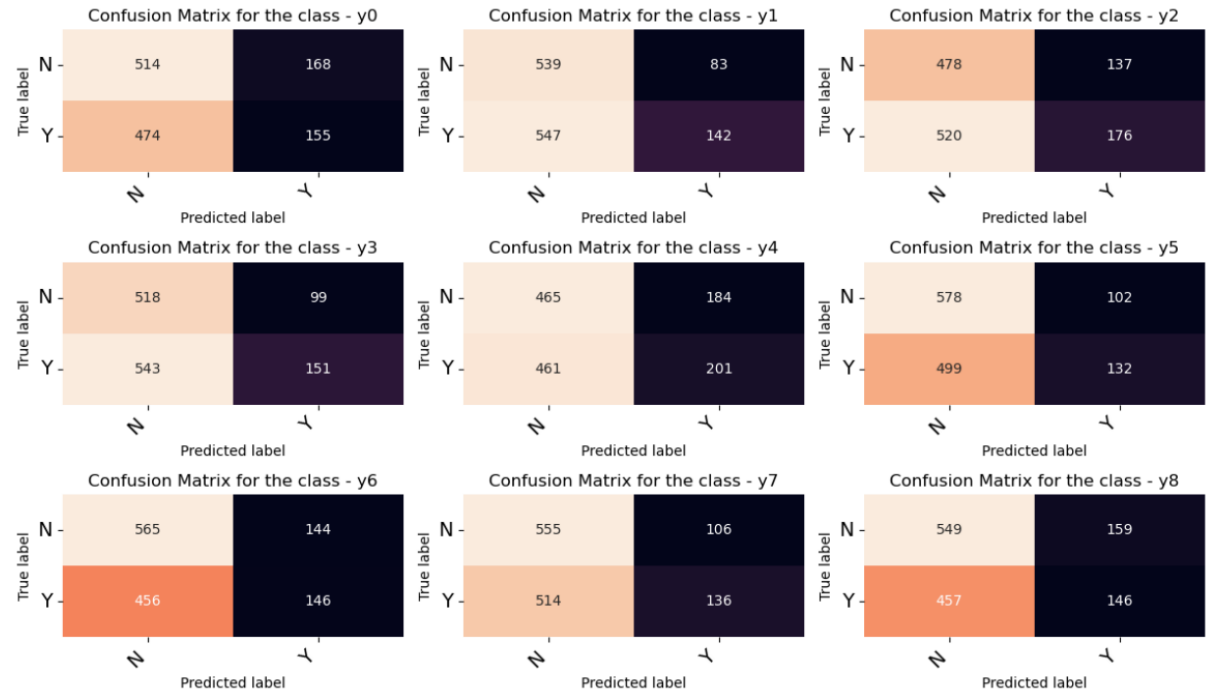

Figure 8: LinearSVM Confusion Matrix (Multi label)

Again, we can see the Linear SVM model performs badly due to non-linearity in the dataset.

1.3.2. k-Nearest Neighbors (Intermediate Board – Multi label)

The number of neighbors is chosen as 3 for the model and the rest of the parameters are chosen as default. Again, these parameters are chosen after running a grid search on the number of neighbors. After normalizing the data, we average the performance scores across all 9 labels. The model performs as follows:

Metrics	Avg Scores (%)
Accuracy Score	90.56
Recall Score	79.59
Precision Score	72.90
F1 Score	76.02

Table 9: KNN Evaluation Metrics (Multi label)

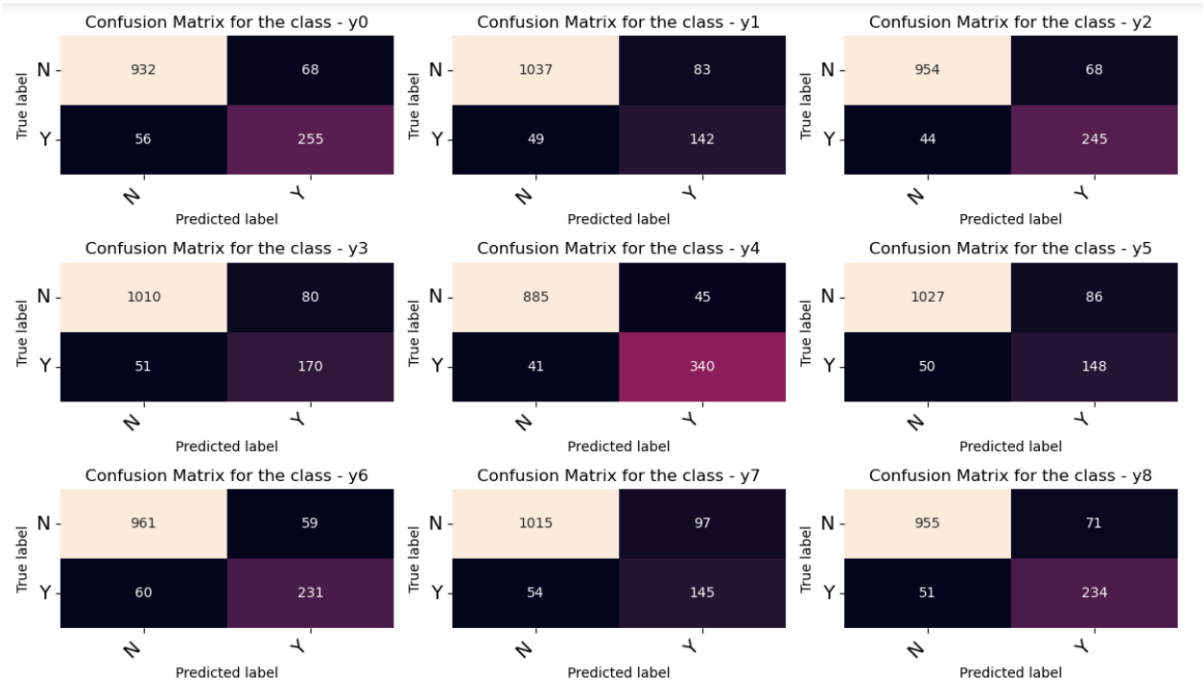

Figure 9: KNN Confusion Matrix (Multi label)

1.3.3. Multilayer Perceptron (Intermediate Board – Multi label)

The number of hidden layers selected for the MLP was 5 and their respective sizes were (200, 150, 100, 100, 50). The activation function chosen was the ReLU function. The default learning rate was 0.001. The maximum number of iterations allowed was 200. Tolerance for the optimization was 0.5. The rest of the values were set to default. After normalizing the data, we average the evaluation metrics calculated across all 9 labels.

The average performances scores were:

Metrics	Avg Scores (%)
Accuracy Score	95.27
Recall Score	88.99
Precision Score	88.70
F1 Score	88.52

Table 10: MLP Evaluation Metrics (Multi label)

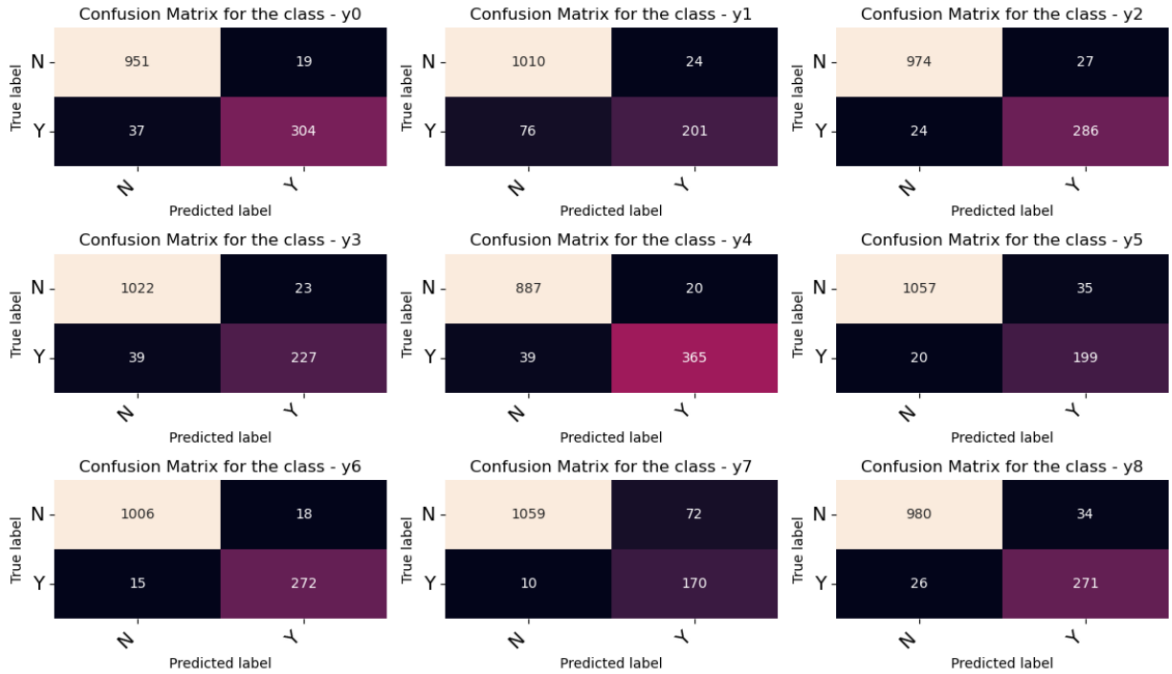

Figure 10: MLP Confusion Matrix (Multi label)

The accuracy scores for the multi label regressor for the 3 models were as follows:

1. Linear Regression– 52.08%
2. KNearestNeighbours – 90.56%
3. Multi-layer Perceptron – 95.27%

Conclusion for regressors: -

Clearly Multi-layer perceptron regressor performs the best while KNN performs respectively. Linear Regression on the other hand shows poor performance due to the non-linearity of the dataset, it severely underfits.

Run code command: - python tictactoe.py --dataset multi

2. Connect four

Like Tic Tac Toe, a player wins in Connect Four when the respective player occupies four consecutive positions in a 6 X 7 board. The given database has all 67557 instances in the game of Connect Four in which neither player has won yet. The attributes had three instances: x meaning player x has taken the spot, o meaning player o has taken, and b meaning blank. The output is the game-theoretical value for the first player which is 2: win, 1: loss, or 0: draw. This game has been implemented using game theory algorithms like minimax algorithm. In this project, we will discuss the implementation of connect four game using AI classification models.

Architecture: -

A) Data Generation and AI training.

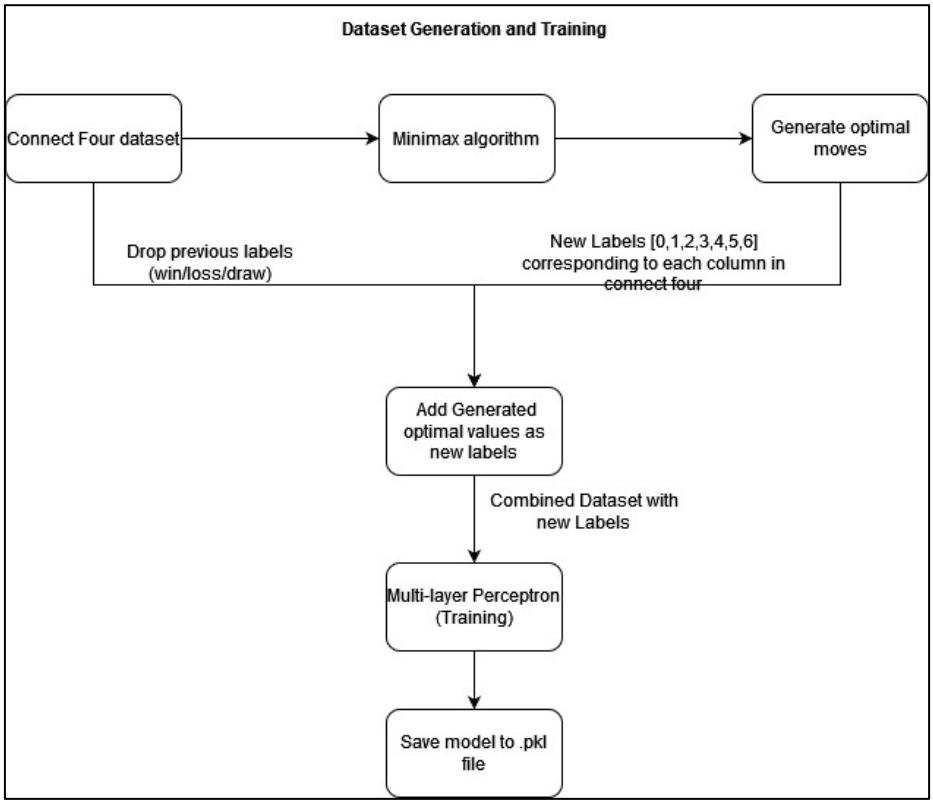

Figure 11: Data Generation & Training

For the game to continue, the AI needs to predict the optimal move so that the computer can play the game. So, the given connect four datasets with 42 attributes is passed through the minimax algorithm for creating optimal moves. Minimax is a type of backtracking algorithm that is used for game theory and decision-making. The aim of this algorithm is to find the optimal move of the player in two-player games like tic tac toe, connect four, chess, and mancala. The two players in the minimax algorithm are called the maximizer and the minimizer. The maximizer aims for the highest score possible while the minimizer aims for the lowest score possible [1][2]. Once the optimal moves are generated, the combined dataset is generated with 42 input features. Multi-layer perceptron algorithm is applied on the combined dataset to predict the optimal move. The average accuracy of the model is 64%. This model has five hidden layers with dimensions (200, 150, 100, 100, 50). A pkl version of the model is saved to be used in the final game code.

B) Final Gameplay

The first chance is played by user. Then the board array gets updated and the next chance is for the computer to play. The final_game.py loads .pkl file and predicts the next optimal move based on the given board array state. Based on the predicted value, the computer plays the next step. In each iteration, it checks whether the four coins are connected or not. This iteration goes on until the player or the computer joins the four coins early.

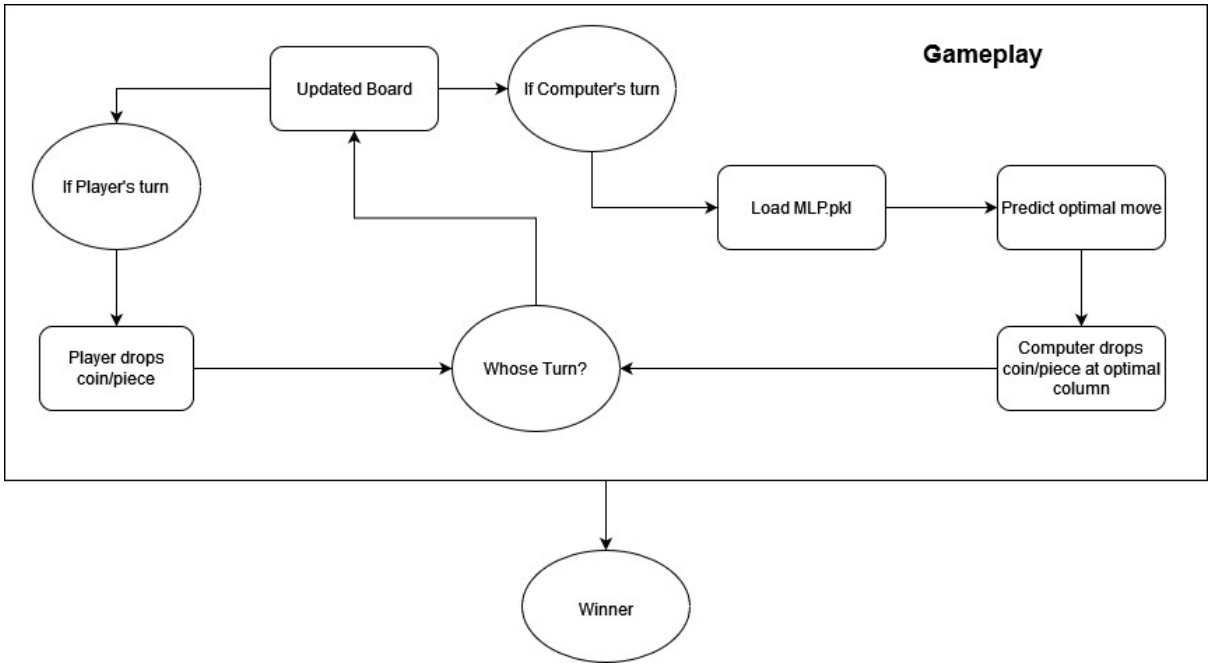

Figure 12: Final Gameplay

C) Evaluation

Metrics	Scores (%)
Accuracy Score	64.10
Recall Score	44.36
Precision Score	48.17
F1 Score	44.72

Figure 13: Evaluation for Multilayer Perceptron in Connect Four Human Gameplay for optimal moves

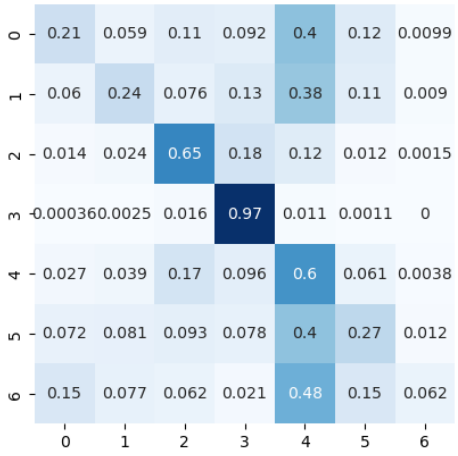

Figure 14: MLP Confusion Matrix for Connect four

Multiple types of classification algorithms were experimented on the above dataset generated which is X input states and Y optimal moves. The average accuracy of the model was around 63-64%. Due to this issue, this AI code was not efficient enough to beat the user in the game. Although the architecture of this entire game is correct, if we can generate the optimal moves

through other techniques, it is possible to increase the accuracy of the model and hence efficient AI code for Connect Four game.

D) Output from Game Screen

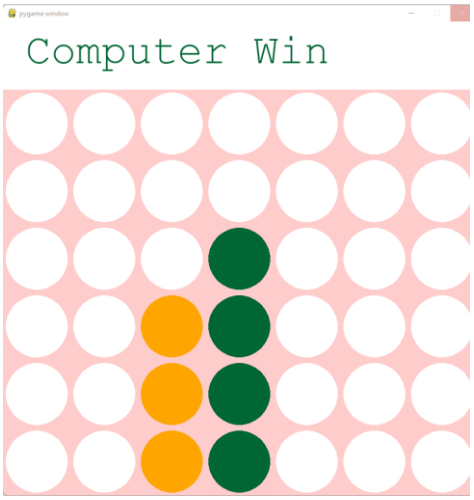

Figure 15: Game UI Screen

Run code command: - python final_game.py

Conclusion for Human Game Play: -

Multiple classification and regression algorithms were implemented on tic tac toe datasets. Out of all the models, the MLP model has performed best on all datasets. For connect four game, the architecture was build using minimax algorithm and MLP models. The accuracy of the model was low and hence user was able to beat the AI. However, if we generate the optimal move dataset using different techniques, then we can achieve the good performance on MLP model and hence the connect four game be further optimized.

References:

1) https://www.geeksforgeeks.org/minimax-algorithm-in-game-theory-set-1-introduction/

2) https://medium.com/analytics-vidhya/artificial-intelligence-at-play-connect-four-minimax-algorithm-explained-3b5fc32e4a4f