Course Project Part I
Simple Learning Models: Classifiers and Regressors

Akshay Sharma (akshaysharma@ufl.edu)
Malvika Ranjitsinh Jadhav (jadhav.m@ufl.edu)
Tanvi Jain (tjain@ufl.edu)

October 2022

1 Overview

This project consists of two main parts. One is to evaluate the performance of classification and regression models on a *Tic Tac Toe* game. Second is to use the knowledge from above models and build a command line *Connect Four* game where a human can play against a machine learning model.

2 Instruction To Run The Code

In order to get the results for classifiers and regressors, run the file **tic_tac_toe.ipynb** on jupyter notebook. File **requirement.txt** contains all dependencies that need to be installled in order to execute this code. This file contains the preprocessing, training and testing for classification and regression models for following three datasets: 1) tictac_final.txt 2) tictac_single.txt 3) tictac_multi.txt.

For running the interactive game, go to "connect_four" folder and run the command **python3 game.py** in your terminal. Make sure to have all the packages installed that are mentioned in the file requirements.txt.
For playing the game, click on the column you want to drop the dot into and for model's turn just click anywhere on the window for model to make a move.
Note: In file **tic_tac_toe.ipynb**, code to get the optimal value using GridSearchCV for each classification and regression model have been commented and models have been trained on optimal values.

3 Classifiers and Regressors on *Tic Tac Toe*

In this section, multiple classification and regression models are discussed that have been implemented on provided datasets for *Tic Tac Toe*. There are few functions defined in the file **tic_tac_toe.ipynb** that are commonly used for classification and regression. Few of them have been discussed below.

1. *split_data*: This function will shuffling the records of the dataframe and return the split for train and test data.
2. *kfold_split*: This function will set the k-value for cross validation and will make the splits on X_train and y_train.
3. *preprocess*: This function checks the filter for $\frac{1}{10}^{th}$ data and calls the split_data function for selected dataframe.
4. *accuracyScore*: This function predicts the test output(y_pred) for y_test and if regressor model is not called, it calls accuracy_score using y_test and y_pred to capture the test accuracy of the model.

3.1 Classifiers

Classification models have been implemented on two datasets i.e., tictac_final.txt and tictac_single.txt. We have implemented linear SVM, k-nearest neighbors, and multilayer perceptron on the datasets containing final boards classification dataset and intermediate boards optimal play (single label). We have defined following functions that have been used by classification models.

1. *confusionMatrix*: This function creates a normalized confusion matrix (that sums to one for each row) using y_test and y_pred values and plots a heatmap for better visualization of the confusion matrix.
2. *model*: This function calls *split_data* and *kfold_split*. Next the model with selected hyperparameters is fitted and validated using k-fold cross-validation. Next the training accuracy is calculated and then *accuracyScore* and *confusionMatrix* functions are called to calculate the testing accuracy and confusion matrix.

3.1.1 Implementation

Before moving to training the model, finding hyperparameters and evaluation results, the datasets containing final boards classification dataset and intermediate boards optimal play (single label) have been shuffled and randomly split for train and test data with 2:1 ratio respectively. K-value for k-fold cross-validation is also set to 10. For each model, to find one of the optimal solution, GridSeachCV has been used to find the best hyperparameters from the provided parameters grid on the training dataset.

1. **Linear SVM**
   Linear SVM is used for linearly separable data, which means if a dataset can be classified into two classes by using a single straight line, then such data is termed as linearly separable data, and classifier is used called as Linear SVM classifier.[2].

   (a) Tic Tac Toe Final boards classification dataset
   For this dataset, SVM classifier function is trained using k-fold cross validation and the parameters used to train has { kernel = 'linear', C = 1.0, gamma = 10, decision_function_shape = 'ovo'}

   (b) Tic Tac Toe Intermediate boards optimal play (single label)
   For this dataset, SVM classifier function is trained using k-fold cross validation and the parameters used to train has { kernel = 'rbf', C = 1.5, gamma = 1.5, decision_function_shape = 'ovo'}

2. **K-Nearest Neighbors (KNN)**
   The k-nearest neighbors technique employs proximity to classify or anticipate how a single data point will be grouped. It is a non-parametric, supervised learning classifier.

   (a) Tic Tac Toe Final boards classification dataset
   For this dataset, KNeighborsClassifier function is trained using k-fold cross validation and the parameters used to train has { n_neighbors = 3, metric = 'euclidean', weights = 'uniform' }

   (b) Tic Tac Toe Intermediate boards optimal play (single label)
   For this dataset, KNeighborsClassifier function is trained using k-fold cross validation and the parameters used to train has { n_neighbors = 27, metric = 'cosine', weights = 'distance' }

3. **Multilayer Perceptron (MLP)**
   MLP is a feed-forward artificial neural network which consist of one input, one output and multiple hidden layers in between. MLP classifier generates a nominal variables as output. MLPClassifier by Scikit-learn has multiple parameters that can be define number of hidden layers, maximum iteration, activation function etc.

   (a) Tic Tac Toe Final boards classification dataset
   For this dataset, MLPClassifier function is trained using k-fold cross validation and the parameters used to train has { hidden_layer_sizes = (256, 256, 128), max_iter = 200, activation = 'relu', solver = 'adam', random_state = 60, alpha = 1e-5}

   (b) Tic Tac Toe Intermediate boards optimal play (single label)
   For this dataset, MLPClassifier function is trained using k-fold cross validation and the parameters used to train has { hidden_layer_sizes = (256, 256, 128), max_iter = 300, activation = 'relu', solver = 'adam', random_state = 50, alpha = 1e-5}

Train and test accuracies can be observed in **Table 1** and confusion matrix for final and intermediate board (single label) for all the following classification models can be observed in **Figure 1** and **Figure 2** respectively.

3.1.2 Evaluation Results

1. Accuracy of classifiers trained on final and intermediate (single label) data is reported in **Table 1** and and confusion matrix in **Figure 1** and **Figure 2**

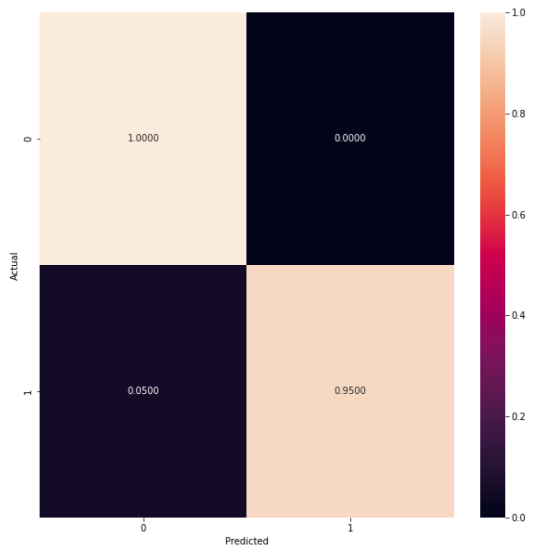
2. Accuracy of classifiers trained on $\frac{1}{10}^{th}$ of the original data can be observed in **Table 2** and and confusion matrix in **Figure 3** and **Figure 4** for final and intermediate (single label) dataset respectively.

	Linear SVM		KNN		MLP	
Dataset	Train (%)	Test (%)	Train (%)	Test (%)	Train (%)	Test (%)
tictac_final	98.4	98.1	99.8	99.7	99.1	98.7
tictac_single	85.3	84.7	87.6	86.8	91.1	90.9

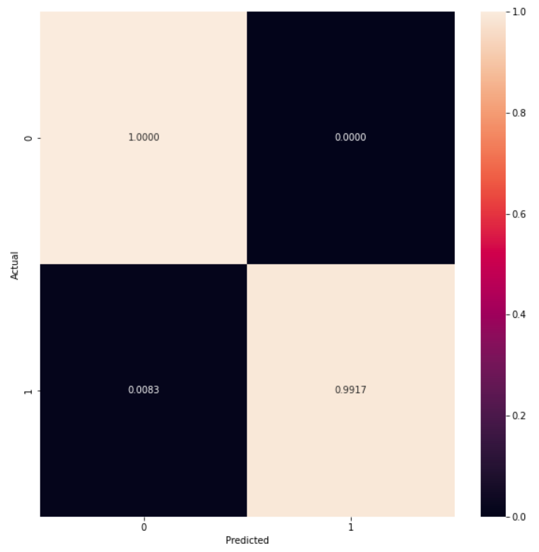
Table 1: Accuracy for classifiers on Final and Intermediate board (single label)

	Linear SVM		KNN		MLP	
Dataset	Train	Test	Train	Test	Train	Test
tictac_final	98.3%	96.9%	76.4%	71.9%	76.9%	75.0%
tictac_single	37.5%	30.9%	54.6%	50.2%	69.2%	68.2%

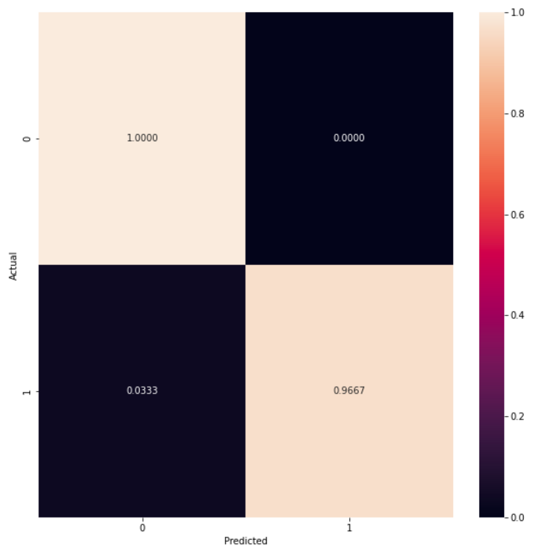
Table 2: Accuracy for classifiers on Final and Intermediate board (single label) on $\frac{1}{10}^{th}$ of the original data



(a) Linear SVM

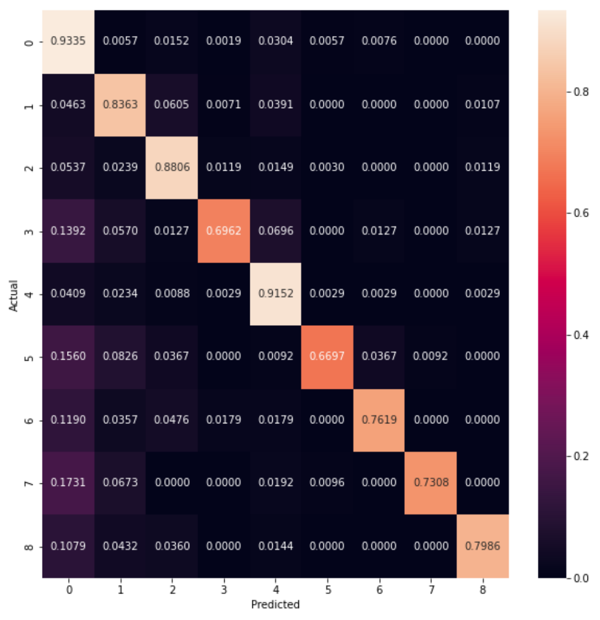


(b) KNN

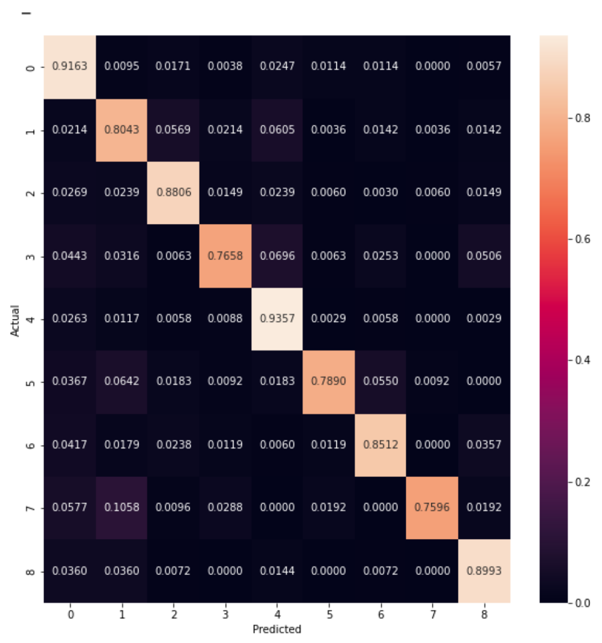


(c) MLP

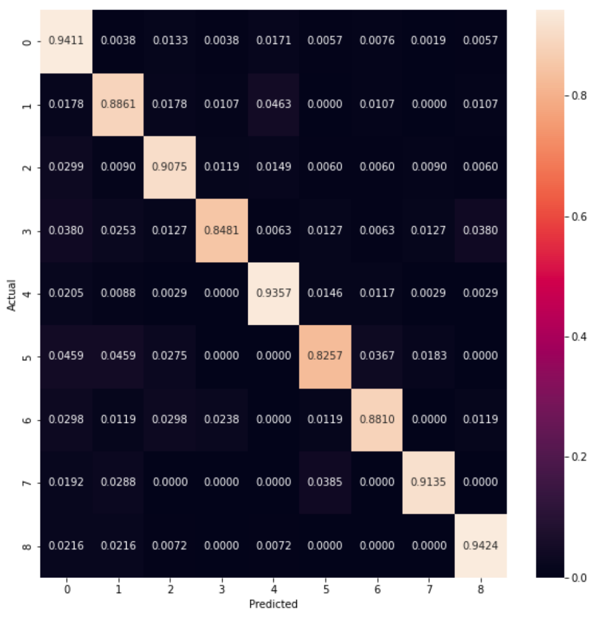
Figure 1: Normalized Confusion Matrix for "Final boards classification dataset"



(a) Linear SVM

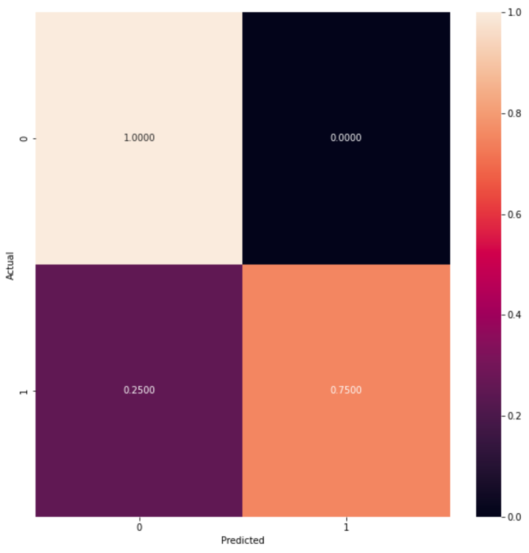


(b) KNN

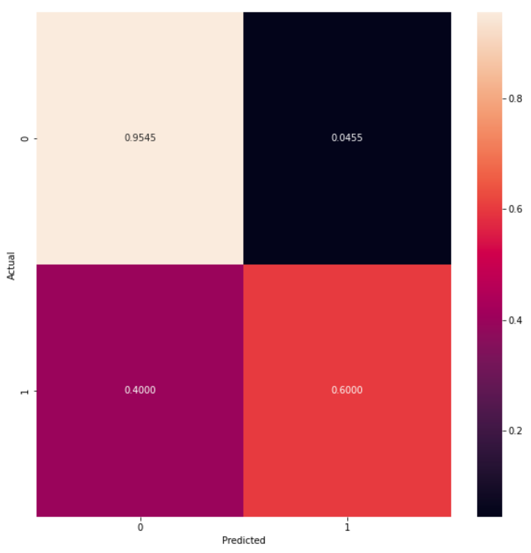


(c) MLP

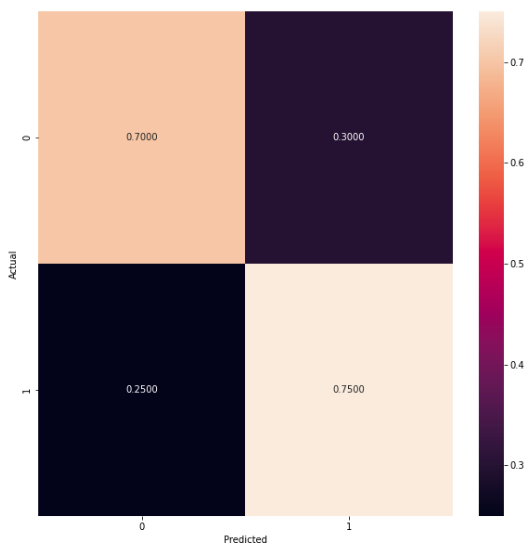
Figure 2: Normalized Confusion Matrix for ”Intermediate boards optimal play (single label)”



(a) Linear SVM

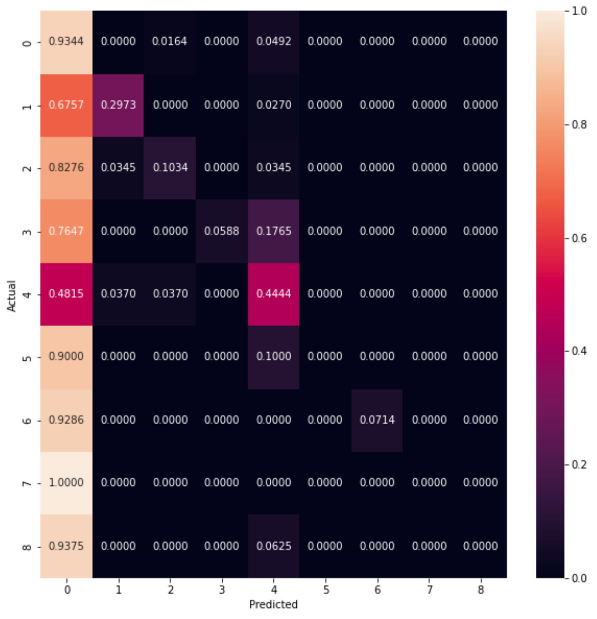


(b) KNN

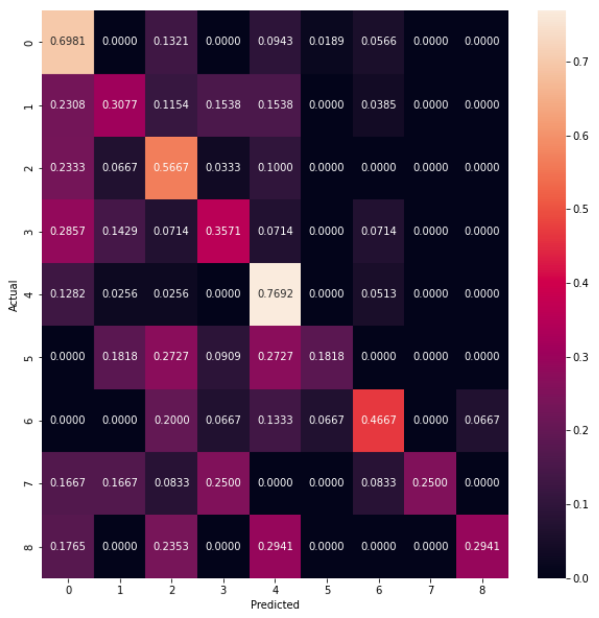


(c) MLP

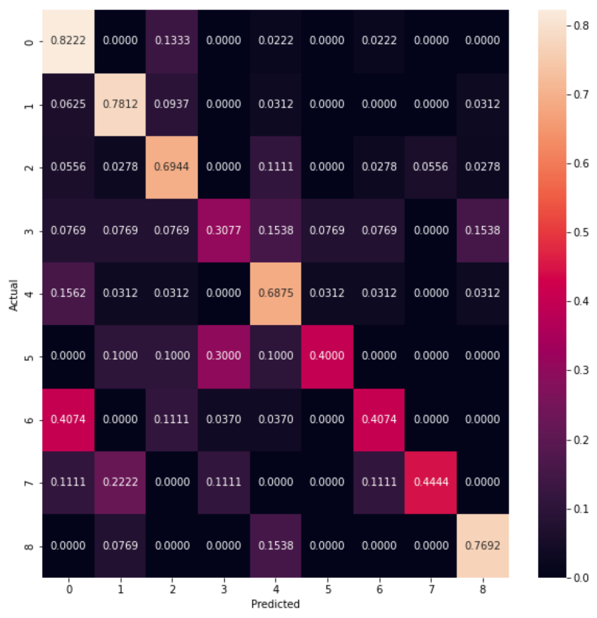
Figure 3: Normalized Confusion Matrix for ”Final boards classification dataset” on $\frac{1}{10}^{th}$ of the original data



(a) Linear SVM



(b) KNN



(c) MLP

Figure 4: Normalized Confusion Matrix for "Intermediate boards optimal play (single label)" on $\frac{1}{10}^{th}$ of the original data

3.2 Regressors

Regression models have been implemented on one dataset i.e., tictac_multi.txt. We have implemented linear regression using normal equation, k-nearest neighbors regressor, and multilayer perceptron on the dataset containing intermediate boards optimal play (multi label). We have defined following functions that are common for most of the regression models except linear regression.

1. *outputRounding*: This function round of the predicted values to a binary value i.e., 0 or 1
2. *accuracyMulti*, *precisionMulti*, *recallMulti*, *f1Measure* : These function calculates the accuracy, precision, recall and F1 score by iterating (on the range of labels) over all the predicted and actual labels and calculate the corresponding score for the multi-label classifier.

3. *regressorModel*: This function calls *split_data* and *kfold_split* to shuffle and split the provided multi-label dataset. Next the model with selected hyperparameters is fitted and validated using k-fold cross-validation. To calculate the validation score of each k-fold, predicted outputs are passed *outputRounding* to alter the outputs as the multi-label classification and score for each fold is calculated using *accuracyMulti*. Next the testing accuracy is calculated by using *accuracyMulti* for the altered predicted values.

3.2.1 Implementation

Before moving to training the model, finding hyperparameters and evaluation results, the datasets containing intermediate boards optimal play (multi label) have been shuffled and randomly split for train and test data with 2:1 ratio respectively. K-value for k-fold cross-validation is also set to 10. For KNN and MLP, to find one of the optimal solution, GridSeachCV has been used to find the best hyperparameters from the provided parameters grid on the training dataset

1. **Linear Regression (Using Normal Equation)**
   Linear regression model fits a best plane/line by computing parameters that minimize a cost function on the provided dataset. For this report, cost function used is mean squared error (MSE). We have implemented the Normal Equation which is the closed-form solution for the Linear Regression algorithm where we can find the optimal value of theta using the derivative of MSE loss function. To calculate theta , we take the partial derivative of the MSE loss function with respect to theta and set it equal to zero to obtain the optimal equation for calculate theta.

   $$\theta = (X^T X)^{-1} X^T \vec{y}.$$

   To calculate the loss function, we have used Mean Squared Error loss using

   $$\Sigma_{i=1}^{m} (y^{(i)} - \theta^T x^{(i)})^2$$

   For making predictions, we have used following equation to predict values using theta.

   $$\hat{y} = \theta^T . x$$

   X $\rightarrow$ Input data (Training Data)
   y $\rightarrow$ Target variable
   theta $\rightarrow$ The parameter
   y_hat $\rightarrow$ Prediction (dot product of theta and X)

2. **K-Nearest Neighbors (KNN)**
   The k-nearest neighbors regression technique tries to predict the value of the output variable by using a local average.

   (a) Tic Tac Toe Intermediate boards optimal play (multi label)
   For this dataset, KNeighborsRegressor function is trained using k-fold cross validation and the parameters used to train has { n_neighbors = 19, metric = 'cosine', weights = 'distance', algorithm = 'brute' }

3. **Multilayer Perceptron (MLP)**
   MLP regressor also consist of one input, one output and multiple hidden layers in between. Unlike MLP classifier, MLP regressor generates continuous value output. MLPRegressor by Scikit-learn has multiple parameters that can be define number of hidden layers, maximum iteration, activation function etc.

   (a) Tic Tac Toe Intermediate boards optimal play (multi label)
   For this dataset, MLPRegressor function is trained using k-fold cross validation and the parameters used to train has { hidden_layer_sizes = (300, 216, 200), max_iter = 500, activation = 'relu', solver = 'adam', random_state = 20}

Train and test accuracies for above mentioned models and hyperparameters can be observed in **Table 3** .

3.2.2 Evaluation Results

Accuracy of classifiers trained on final and intermediate (multi label) data is reported in **Table 3**

	Linear Regression		KNN		MLP	
Dataset	Train (%)	Test (%)	Train (%)	Test (%)	Train (%)	Test (%)
tictac_multi	78.2	78.1	84.1	84.0	89.0	88.7

Table 3: Accuracy for regressors on Intermediate board (multi label)

4 Human Game Play: *Connect Four*

4.1 Model Architecture

To decide the most suitable architecture we have analysed the performance of multiple classification models on the given dataset for connect four. the following table summarizes the performances of those models.

Architecture	Accuracy	Precision	Recall	F1 score
K Nearest Neighbour	0.81	0.70	0.76	0.72
Multi Layer Perceptron	0.92	0.89	0.87	0.88
Decision tree	0.8	0.75	0.72	0.71
Random Forest	0.75	0.51	0.55	0.52
Extra trees Classifier	0.9	0.85	0.92	0.87
AdaBoost Classifier	0.6	0.50	0.44	0.39
Naive bayes	0.25	0.11	0.18	0.14

Table 4: Comparative performance of different architectures on random sample of Connect Four dataset

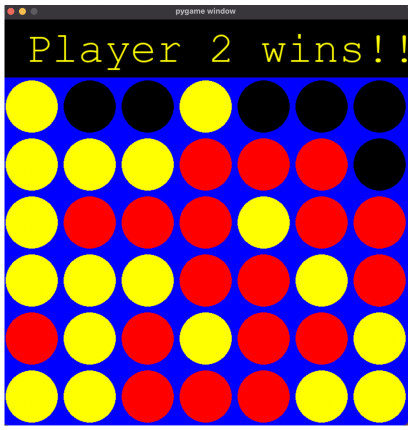
Based on the above table we chose the Multi Layer Perceptron model for the Human Game Play application. As we can see the chosen model has the highest performance followed by the Extra Trees Classifier. The worst performance is of the Naive Bayes classifier. Here all the models were tested on same data.
For this dataset, we have trained and instance of MLPClassifier using 10-fold cross validation and the parameters of the model are:
{ hidden_layer_sizes = (256, 256, 128), max_iter = 500, activation = 'relu', solver = 'sgd', random_state = 20}

4.2 Results

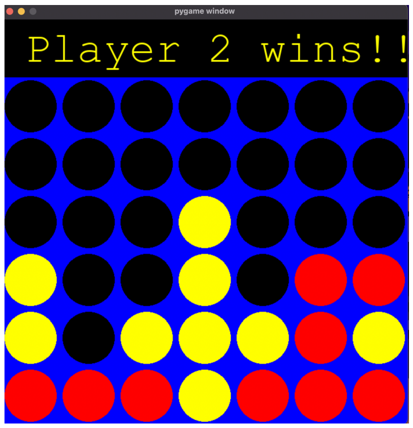
We are using Multi Layer Perceptron for the Human Game Play where Player 1 is Human with RED dots and Player 2 is our Ml model with YELLOW dots. Each player gets alternative turns to play and player will makes the starting move. The goal of the game is to connect 4 dots with same color horizontally, vertically, or diagonally.

We have trained the model to predict a move against player 1. The model predicts 3 classes as ['draw' 'loss' 'win']. The preferred move for our model will be the one that make the opponent loss. So, while making our model move, we test 7 possiblities of moves for 7 columns of the connect four board and whichever move will make the oppenent loss, our model will go with that move.

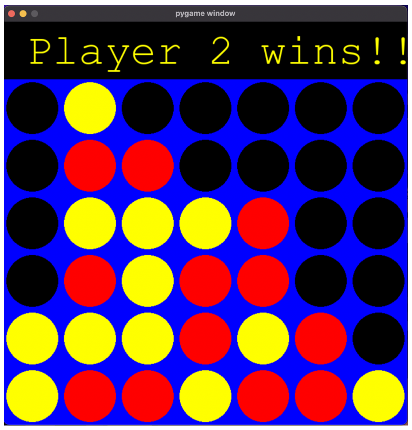
We have played and tested the interactive connect four game around 10 times and the MLP model is able to win 6 times out of 10 and loss 4 times while playing against human. Please refer to the figures below to see in which scenarios the model wins or losses.



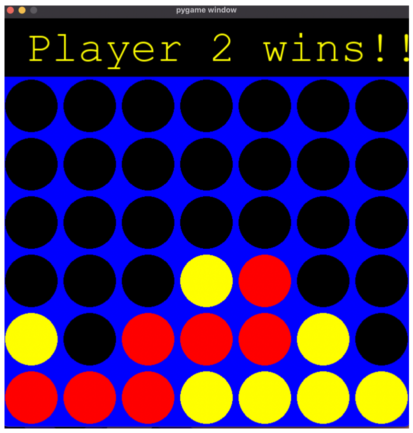
(a) P2 Winning Scenario 1



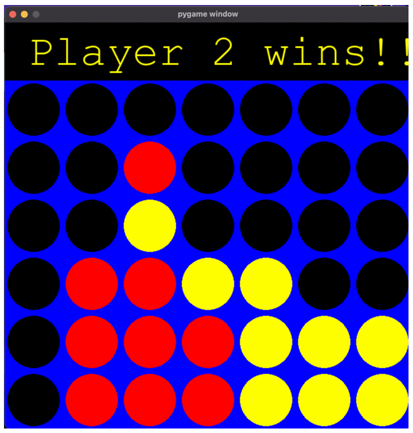
(b) P2 Winning Scenario 2



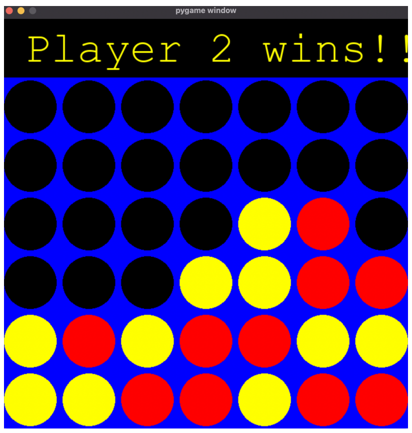
(c) P2 Winning Scenario 3



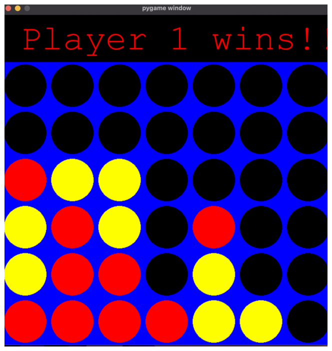
(d) P2 Winning Scenario 4



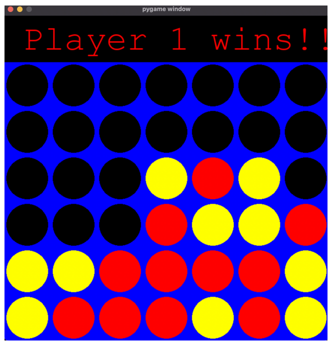
(e) P2 Winning Scenario 5



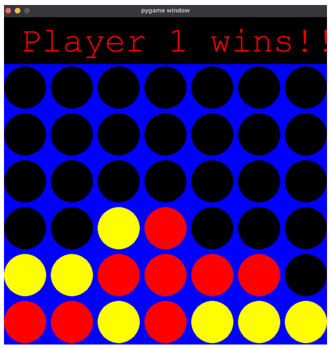
(f) P2 Winning Scenario 6



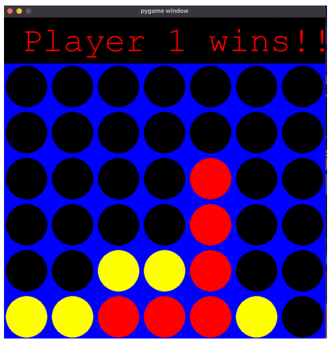
(a) P1 Winning Scenario 1



(b) P1 Winning Scenario 2



(c) P1 Winning Scenario 3



(d) P1 Winning Scenario 4

5 Inferences

1. What method works best for classification and Why?
   For the final board classification dataset, K-nearest neighbour classifier has the best test (99.7%) and train (99.8%) accuracy as compared to others based on the observations in **Table 1**. Also it has the better generalization as well (based on the difference between test and train accuracy of models). This could be because of the well defined data and the small size of the dataset. Also since the dataset is small and no complex classification is required, KNN works well.
   For the intermediate boards optimal play (single label), Multilayer Perceptron works best compared to the other two model based on the test (91.1%) and train (90.0%) accuracy as mentioned in **Table 1**. MLP can handle considerably large dataset better. Linear SVM and KNN didn't work very well as compared to MLP because data is not linearly separable and is not well defined.

2. What method works best for regression and Why?
For the intermediate boards optimal play (multi label), Multilayer Preceptron regressor works best out of Linear Regression (using Normal Equation), KNN and MLP based on the test and train accuracies observed in **Table 3**. MLP works best because of the large size of dataset and the hidden layes that provide depth to the model. KNN regression is not the best because it uses lazy learning approach and also for larger datasets, it takes more time to predict the values. Linear regression tries to fit a plane/line and predicts a continuous value as output but this dataset as fixed set of values as label. This is one reason why linear regression is not working well on this dataset. KNN and linear regression are comparatively more dependent on well defined data compared to MLP.

3. What happens to the accuracy of the classifiers if they are trained on $\frac{1}{10}^{th}$ as much data?
Based on the results observed in **Table 1** and **Table 2** , accuracy of SVM remains unaffected for binary classification task irrespective of the size of dataset but shows a drastic decrease in accuracy for the multi-class classification problem. As expected accuracy of k-NN and MLP models reduces significantly as the models face a data famine during training and cannot generalize well.

4. Explain why certain methods scale better to larger datasets than the others.
As the number dimensions of the feature space increase the distribution distance between data points grows narrower. So for high-dimensions there can be multiple data point with same distance from neighbours, this reduces the precision of k-NN as dataset gets larger in terms of features. As we can see when the number of features increases from 9 to 42 for multi-class classification precision of k-NN model reduces from 0.82 to 0.70. Whereas for the multi layer perceptron model with increased feature set the precision remains unaffected from 0.9 to 0.89. So the reason for better scaling of the algorithms is subjective and dependent on the models being compared.

References

[1] https://mmuratarat.github.io/2020-01-25/multilabel_classification_metrics.

[2] https://www.javatpoint.com/machine-learning-support-vector-machine-algorithm.