**CIS 6930/4930 Deep Learning for Computer Graphics**

**Part I. Classifiers and Regressors**

**Kalpak Seal Shantanu Ghosh**

# **Introduction**

This project evaluates the performance of classification and regression algorithms on a game of Tic Tac Toe.

The project lists statistical accuracy and confusion matrices for several classifiers and regressors like linear regression, linear SVM, k-nearest neighbors, and multilayer perceptron using the provided datasets. The datasets have been synthetically generated.

The AI learns from these games to produce reasonable moves for one of the players in the game.

A picture containing diagram

Description automatically generated

# **Program Execution**

1. To train the models

python3 buildModels.py

1. To play the game

python3 tictactoe.py

# **Implementation**

1. **Classifiers**

There are 2 classification datasets on which the models were from.

The algorithms that have been used are:

* linear SVM
* k-nearest neighbors
* multilayer perceptron.

**Datasets**

1. **Final boards classification dataset**The final boards classification dataset is a binary classification task for boards where the game is over.   
   The input features are x0, x1 ···, x8, the states of each Tic Tac Toe square

**A screenshot of a cell phone

Description automatically generated**

The output feature *y* is the winning player.   
Each line of the input file lists the input features followed by a single output feature, as: **x0, x1 ···, x8, y.**

1. **Intermediate boards optimal play (single label)**This is a multi-class classification task where the board is set up so that it is **‘O’** player’s move, and the ***goal is to predict the next move that is optimal for the O player (i.e. player −1).***

Inputs are *x*0, *x*1 ···,*x*8, the states of the *Tic Tac Toe* squares for a given board, and the output *y* is the index of the best move for the O player (player −1).

**Performance**

A **10-fold cross validation** has been used.

1. **On the final board classification dataset**

|  |  |  |
| --- | --- | --- |
| **Algorithm** | **Details** | **Accuracy (%)** |
| K Nearest Neighbors | Optimal K value - 1 | 100 |
| Linear SVM | L2 penalty parameter (‘C’) – 10,  Kernel coefficient (‘gamma’) - 1 | 98.95 |
| Multi-Layer Perceptron | Activation function - relu,  Hidden Layer configuraiton: (120, 120) | 99.47 |

Here, all the three algorithms perform really good, with K Nearest Neighbors getting the best of them all. The reason for KNN doing the best can be attributed to the smaller size of the dataset and the fact that the data is very well defined. Also, since no complicated classification is involved in this case, KNN gets to perform the best of them all.

1. **Intermediate boards optimal play (single label)**

|  |  |  |
| --- | --- | --- |
| **Algorithm** | **Details** | **Accuracy (%)** |
| K Nearest Neighbors | Optimal K value - 1 | 84.6 |
| Linear SVM | L2 penalty parameter (‘C’) – 10,  Kernel coefficient (‘gamma’) - 1 | 31.8 |
| Multi-Layer Perceptron | Activation function - relu,  Hidden Layer configuraiton: (200, 200) | 93.82 |

In this scenario, Multi-layer Perceptron gives the best accuracy amongst all the algorithm. The dataset is substantially large and with a hidden layer configuration of (200, 200) the MLP algorithms learns the best out of all the algorithms

Linear SVM performs the worst because the data is not linearly separable. We have also tried the RBF Kernel which gives an accuracy of 93%, and this is because RBF is a non-linear (Gaussian) kernel.

1. **If the models are trained on 10% of the data**
   1. **On the final board classification dataset**

|  |  |  |
| --- | --- | --- |
| **Algorithm** | **Details** | **Accuracy (%)** |
| K Nearest Neighbors | Optimal K value - 1 | 85 |
| Linear SVM | L2 penalty parameter (‘C’) – 10,  Kernel coefficient (‘gamma’) - 1 | 100 |
| Multi-Layer Perceptron | Activation function - relu,  Hidden Layer configuraiton: (200, 200) | 100 |

* 1. **Intermediate boards optimal play (single label)**

|  |  |  |
| --- | --- | --- |
| **Algorithm** | **Details** | **Accuracy (%)** |
| K Nearest Neighbors | Optimal K value - 5 | 60 |
| Linear SVM | L2 penalty parameter (‘C’) – 12,  Kernel coefficient (‘gamma’) - 1 | 54.19 |
| Multi-Layer Perceptron | Activation function - relu,  Hidden Layer configuraiton: (150, 150) | 74.04 |

1. **Regression**

There is 1 classification datasets on which the models were from.

The algorithms that have been used are:

* linear SVM
* k-nearest neighbors
* multilayer perceptron.

**Datasets**

**Intermediate boards optimal play (multi label) dataset.**

Since this is really a multi-label classification problem that is being treated as a regression problem, we report the accuracy of the multi-label regressor as if it were a classifier, by rounding the output of each regression output to either 0 or 1 and comparing with the testing data.

|  |  |  |
| --- | --- | --- |
| **Algorithm** | **Details** | **Accuracy** |
| KNN | Optimal K value - 3 | 90.64 |
| Multi-Layer Perceptron | Activation function: tanh  alpha (L2 penalty (regularization term): 0.0001  hidden layer configuration: (200, 100, 100, 50)  solver: 'lbfgs' (optimizer in the family of quasi-Newton methods)  max\_iter: 1000  learning\_rate - constant | 97.61 |
| Linear Regression | N/A | 72.62 |

In the regression problem, where the problem is ideally a multi-label classification problem, but treated here as a regression one, we find after repeated iterations and hyperparameter tunings that the Multi-Layer Perceptron algorithm performs the best out of all the three.

The reason for MLP to perform the best can be attributed to the availability of large dataset and also the fact that we had made the network quite deep. Also, the activation function that gets selected is tanh and this fits our dataset perfectly since our data ranges from [-1, 1].

The KNN algorithms performs well, but not the best and this is because of the simple nature of the algorithm by itself. KNN is a lazy learning model with local approximation having commonly the distance function and k value as its hyperparameters. Thus, the simple nature of the algorithm is mainly a reason to perform poorly in comparison to MLP but much better than Linear Regression.

Linear Regression on other hand, predicts a continuous value ranging from negative infinity to infinity. The algorithm tries to fit a straight line in this problem which demands a classification. Whereas, Logistic Regression gives a probability of the positions ranging from 0 to 1.

Also, Linear Regression is highly sensitive to imbalance data and such can be seen from a comparison between the R2 and RMSE errors between the two.

**Thresholds in Regressors**

* In the implementation of Linear Regression, the algorithm gives 9 outputs for each of the 9 positions in the tic-tac-toe board. We take the maximum value for each of the positions or each of the positions and we choose the maximum of the 9 outputs and make it 1 and the change the rest to 0 (analogous to One-hot encoding).
* In KNN and MLP, outputs greater than 0.5 are converted to 1 and consequently the rest to 0.