# Part 1:Classifiers and Regressors

**Submitted by:**

**Shivam Basia(UF ID:7838-4123)**

**Aakansh Togani(UF ID:6005-9201)**

**Pulin Soni(UF ID:7303-3793)**

**Course: Deep Learning for Computer Graphics**

## Problem Statement:

Built an AI Tic Tac Toe game along with evaluating multiple regressors and classifiers on the tic tac toe datasets.

# INDEX

## Program Execution:

To run and see regressors accuracy result and train model for tic tac toe game run the following command:

**python3 regressors.py**

To run classifiers run the following command:

**python3 classifiers.py**

To start playing the Tic Tac Toe game with computer AI run the following command:

**python3 tictactoe_game.py**

## Accuracy metrics:

Here,we use the confusion metrics for calculating accuracy for the classifiers.Since,in regressors also, after predicting we are converting it back into labels,we will use the confusion matrix for generating the accuracy metrics for regressors as well.

**Confusion Matrix:** The matrix shown (fig.1) is used to analyze how the model is able to predict things. The confusion matrix is also known as the error matrix. As the name suggests, by just analyzing a confusion matrix one will be able to determine the exactitude of the model. The matrix will give additional insights with respect to the predictive ability of the model. We have extracted a confusion matrix for all combinations of classifier and regressor models executed for the given data sets.The confusion matrix is a special kind of table that displays the frequency distribution of the parameters applied. Here, the confusion matrix consists of two dimensions namely: Actual and Predicted. And exact types of classes in both dimensions.

| Predicted class / Actual class | P | N |
|---|---|---|
| P | TP | FN |
| N | FP | TN |

**Figure 1**

The table obtained will contain values for predefined types. These types are: True Positives, True Negatives, False Positives and False Negatives.

**Classification Report:** This report gives us further insight on how our model is predicting. Whether the model is predicting more negatives or not predicting positives. The report gives us more performance metrics such as recall, precision, F-1 score and support.

Classification_report for initial training-test set

| | precision | recall | f1-score | support |
|---|---|---|---|---|
| 0 | 0.97 | 0.99 | 0.98 | 1086 |
| 1 | 0.95 | 0.85 | 0.90 | 225 |
| accuracy | | | 0.97 | 1311 |
| macro avg | 0.96 | 0.92 | 0.94 | 1311 |
| weighted avg | 0.97 | 0.97 | 0.97 | 1311 |

**Figure 2**

## Code Snippets and Explanations:

## Classifiers:

### Training on 80% dataset and subsequent cross validation

```
class Classifier():

    def func(self, clf, X, y):
        X_train, X_test, y_train, y_test = train_test_split(
            X, y, shuffle=True, test_size=0.20, random_state=42)
        clf.fit(X_train, y_train)
        y_predicted = clf.predict(X_test)
        print("Confusion_matrix for initial training-test set")
        print(confusion_matrix(y_test, y_predicted))
        print("Classification_report for initial training-test set")
        print(classification_report(y_test, y_predicted))
        print("Training score for initial training-test set")
        print(clf.score(X_train, y_train))
        print("Testing score for initial training-test set")
        print(clf.score(X_test, y_test))
        accuracies = cross_val_score(estimator=clf, X=X, y=y, cv=10)
        print("Cross-validation Accuracies:")
        print(accuracies)
        print("Cross-validation Accuracies Mean:")
        print(accuracies.mean())
        print("Cross-validation Accuracies Standard Deviation Mean:")
        print(accuracies.std())
        y_pred_cross_val = cross_val_predict(clf, X, y, cv=10)
        print("Confusion matrix of combined cross-validation data predicted results")
        print(confusion_matrix(y, y_pred_cross_val))
        print("Classification_report of combined cross-validation data predicted results")
        print(classification_report(y, y_pred_cross_val))
        cv_results = cross_validate(clf, X, y, cv=10, return_train_score=True)
        print("Cross-validation testing scores:")
        print(cv_results['test_score'])
        print("Cross-validation training scores")
        print(cv_results['train_score'])
```

**Figure 3**

**Function Classifier***:*

- This function is used for Linear SVM, K-neighbour and MLP classifier models.
- The function has the parameter **clf, X, y**
- **Clf:** This contains the object of the model.
- **X:** This parameter consists of the inputs of our classifier model from the CSV file provided.
- **y:** This acts as the labels for the classifier model in this supervised learning process.

- We have specified the parameter of the **train-test-split()** where we set **shuffle= True** which randomizes the data provided. Shuffling the data is done to make sure that your training/test sets are representative of the overall distribution of the data.
- **test_size=0.2** assigns the test size to be 20% of the data provided.
- **random_state=42** this ensures that your state of randomness does not keep changing.
- The shuffled data is then fitted into the classifier using the **clf.fit() function.**
- The model is then used to predict the output for an unknown train input set passed, which is stored in the X_test variable. The predicted output is stored for further processing.
- **confusion_matrix()** : This function is used to print the confusion matrix table (As explained earlier in the document)
- **classification_report()** : This function gives additional insight of the model. The function returns a report which consists of the precision, recall, F1 and Support scores for the executed model.
- **Score** is an additional quantifier reflecting on the accuracy of the model. We remove the score for the training and testing set individually.
- **cross_val_score()** : This function returns an array of accuracy for each fold executed internally with any explicit command. The function must be given the number of cross validation needed in the function parameter 'cv'.
  We calculate the mean accuracy for all the accuracies returned by the cross_val_score() function. We also calculate the standard deviation with respect to all the accuracies obtained via the cross_val_score().
- **cross_val_predict():** The data is split according to the cv parameter. Each sample belongs to exactly one test set, and its prediction is computed with an estimator fitted on the corresponding training set.
- **cross_validate():** Evaluate metrics by cross-validation and also record fit/score times.

## Models Explained with respect to the dataset :

- **Final Board Classification Dataset :**

```
if __name__ == '__main__':
    obj = Classifier()
    data = pd.read_csv('tictac_final.txt', sep=" ", header=None)
    col_X = [0, 1, 2, 3, 4, 5, 6, 7, 8]
    col_y = [9]
    X = data[col_X]
    y = data[col_y]
```

**Figure 4**

- Here the dataset used is the ‘tictac_final’.
- The given dataset consists of 10 columns.
- First 9 columns from index ‘0’ to ‘8’ are inputs for the desired model. The data of these columns are stored in variable ‘X’.
- The last column having index ‘9’ is the expected output or the labels for given sequential inputs. The data of this column is stored in variable ‘y’

**Linear SVM Classifier** : The algorithm creates a line or a hyperplane which separates the data into classes.

```
print("For final boards classification dataset:")
print("Linear SVM Classifier:")
clf_svm = svm.SVC(kernel='linear', degree=2, gamma='auto',
                  C=1.2, coef0=0.2, probability=True, random_state=42)

obj.func(clf_svm, X, y)
```

**Figure 5**

- **Kernel:** Specifies the kernel type to be used in the algorithm. It must be one of ‘linear’, ‘poly’, ‘rbf’, ‘sigmoid’, ‘precomputed’ or a callable.
- **Degree:** Degree of the polynomial kernel function (‘poly’). Ignored by all other kernels.
- **Gamma:** gamma=‘auto’, uses 1 / n_features.
- **C:** Regularization parameter. The strength of the regularization is inversely proportional to C.

```
For final boards classification dataset:
Linear SVM Classifier:
Confusion_matrix for initial training-test set
[[ 61   6]
 [  0 125]]
Classification_report for initial training-test set
              precision    recall  f1-score   support

          -1       1.00      0.91      0.95        67
           1       0.95      1.00      0.98       125

    accuracy                           0.97       192
   macro avg       0.98      0.96      0.96       192
weighted avg       0.97      0.97      0.97       192

Training score for initial training-test set
0.9869451697127938
Testing score for initial training-test set
0.96875
Cross-validation Accuracies:
[1.         1.         1.         1.         1.         1.
 1.         1.         1.         0.83157895]
Cross-validation Accuracies Mean:
0.983157894736842
Cross-validation Accuracies Standard Deviation Mean:
0.050526315789473676
```

**Figure 6**

- The classifier predicts **6 False negatives** values and **0 False positive** values
- The **training score** for the initial training-test set was **0.98694**
- The **testing score** for initial training-test set was **0.96875**
- The **cross-validation accuracies** of the 10-folds are displayed.
- The **cross-validation accuracy mean** is **0.98315**.
- **Cross-validation accuracy Standard Deviation Mean** is **0.05052**.

```
Confusion matrix of combined cross-validation data predicted results
[[316  16]
 [  0 626]]
Classification_report of combined cross-validation data predicted result
              precision    recall  f1-score   support

          -1       1.00      0.95      0.98       332
           1       0.98      1.00      0.99       626

    accuracy                           0.98       958
   macro avg       0.99      0.98      0.98       958
weighted avg       0.98      0.98      0.98       958

Cross-validation testing scores:
[1.         1.         1.         1.         1.         1.
 1.         1.         1.         0.83157895]
Cross-validation training scores
[0.98143852 0.98143852 0.98143852 0.98143852 0.98143852 0.98143852
 0.98143852 0.98143852 0.98146002 1.        ]
```

**Figure 7**

- The classifier predicts **16 False negative** values and **0 False positive** values
- The classification report of combined cross-validation predicted data is displayed.
- The **10 cross-validation testing scores** are displayed.
- **10 Cross-validation training scores** are displayed.

**KNN Classifier**: The classifier K-Nearest Neighbour clusters output data to a group of data to which the output is nearest too.

```
print("KNeighborsClassifier:")
clf_knn = KNeighborsClassifier(n_neighbors=5)
obj.func(clf_knn, X, y)
```

**Figure 8**

- **KNeighborsClassifier** : This function is imported from the sklearn class. It returns an object of a KNN model.
- **n_neigbors** : This parameter defines the number of data groups the model can have.
- **Confusion Matrix and Classification Report.**

```
KNeighborsClassifier:
Confusion_matrix for initial training-test set
[[ 67   0]
 [  0 125]]
Classification_report for initial training-test set
              precision    recall  f1-score   support

          -1       1.00      1.00      1.00        67
           1       1.00      1.00      1.00       125

    accuracy                           1.00       192
   macro avg       1.00      1.00      1.00       192
weighted avg       1.00      1.00      1.00       192

Training score for initial training-test set
1.0
Testing score for initial training-test set
1.0
Cross-validation Accuracies:
[0.97916667 1.         1.         1.         1.         1.
 0.90625    1.         1.         1.        ]
Cross-validation Accuracies Mean:
0.9885416666666667
Cross-validation Accuracies Standard Deviation Mean:
0.028125000000000004
```

**Figure 9**

- The above Confusion matrix is for the KNN classifier when trained using the ‘tictac_final’ dataset.
- The confusion matrix obtained predicts all the **True Positives and True Negatives correctly**, therefore the matrix consists of value ‘0’ at the position of False Positive and False Negative.

- We also display the Classification report for the initial random shuffled dataset.
- The score with respect to the initial shuffled data set for both training and testing have been displayed.
- The **Training Score** and **Testing Score** Obtained = **1.**
- We cross validate using 10-folds, and obtain accuracies for each fold, also display the mean accuracy with the standard deviation mean.
- The **cross-validation accuracy mean** is **0.9885**.
- **Cross-validation accuracy Standard Deviation Mean** is **0.0281**.

```
Confusion matrix of combined cross-validation data predicted results
[[323   9]
 [  2 624]]
Classification_report of combined cross-validation data predicted results
              precision    recall  f1-score   support

          -1       0.99      0.97      0.98       332
           1       0.99      1.00      0.99       626

    accuracy                           0.99       958
   macro avg       0.99      0.98      0.99       958
weighted avg       0.99      0.99      0.99       958

Cross-validation testing scores:
[0.97916667 1.         1.         1.         1.         1.
 0.90625    1.         1.         1.        ]
Cross-validation training scores
[1. 1. 1. 1. 1. 1. 1. 1. 1. 1.]
```

**Figure 10**

- The above figure contains the confusion matrix for the combined result of cross-validated data.
- Here the number of predictions increases due to the cross-validation.
- Here, we obtain **TP: 323 FP: 9 FN: 2 TN: 624**
- We also calculate the **Cross-Validation testing score** and The **cross-validation training score**.

**MLP Classifier**: MLPClassifier trains iteratively since at each time step the partial derivatives of the loss function with respect to the model parameters are computed to update the parameters. It can also have a regularization term added to the loss function that shrinks model parameters to prevent overfitting.

```
print("MLP Classifier:")
clf_mlp = MLPClassifier(random_state=42, max_iter=50,
                        solver='lbfgs', activation='tanh', early_stopping=True)
obj.func(clf_mlp, X_single, y_single)
```

**Figure 11**

- **Solver:** It is used for weight optimization 'lbfgs' is an optimizer in the family of quasi-Newton methods.
- **Activation:** Activation function for the hidden layer. We have applied the 'tanh', the hyperbolic tan function, which returns f(x) = tanh(x).
- **Early_stopping*:*** Whether to use early stopping to terminate training when the validation score is not improving.

```
MLP Classifier:
Confusion_matrix for initial training-test set
[[286   3   6   3  13   0   6   3   3]
 [  4 129   6   4  10   4   5   0   6]
 [ 10   7 149   4   6   0   4   2   5]
 [  3   2   2  97   6   0   3   0   4]
 [ 14   2   6   0 173   0   5   2   0]
 [  3   3   1   0   0  66   3   0   0]
 [  4   3   2   5   2   0  82   0   1]
 [  1   4   1   4   1   1   1  37   0]
 [  5   1   4   2   2   1   0   3  71]]
Classification_report for initial training-test set
              precision    recall  f1-score   support

           0       0.87      0.89      0.88       323
           1       0.84      0.77      0.80       168
           2       0.84      0.80      0.82       187
           3       0.82      0.83      0.82       117
           4       0.81      0.86      0.83       202
           5       0.92      0.87      0.89        76
           6       0.75      0.83      0.79        99
           7       0.79      0.74      0.76        50
           8       0.79      0.80      0.79        89

    accuracy                           0.83      1311
   macro avg       0.82      0.82      0.82      1311
weighted avg       0.83      0.83      0.83      1311
```

**Figure 12**

```
Training score for initial training-test set
0.883206106870229
Testing score for initial training-test set
0.8314263920671243
Cross-validation Accuracies:
[0.74542683 0.81068702 0.80916031 0.84732824 0.84122137 0.83358779
 0.82900763 0.86259542 0.86259542 0.87328244]
Cross-validation Accuracies Mean:
0.8314892478123254
Cross-validation Accuracies Standard Deviation Mean:
0.03517962976171464
```

**Figure 13**

- **Confusion matrix & Classification report** of the initial training-test set is displayed.
- The **Training score** for the initial training-test set is **0.88320.**
- The **Testing score** for the initial training-test set is **0.83142**.
- **Cross-validation reports** for the 10 folds are displayed.
- **Cross-validation accuracy mean** for the initial training-test set is **0.83148.**
- **Cross-validation accuracy STD mean** for the initial training-test is **0.03517.**

```
Confusion matrix of combined cross-validation data predicted results
[[1394   20   38   17   39    9   23    9   15]
 [  20  687   35   21   43   12   16    4    7]
 [  46   35  801   15   41    6   12    8    9]
 [  25   18    6  400   26    7    9    5   17]
 [  55   17   21   15  879    9   26    4   20]
 [  11   20    7    4    7  281   12    6    5]
 [  38   14   14   20    4    0  445    4    5]
 [   9   10    4    8    9    8    7  192    7]
 [  33   14   13    6   11   10    1    3  368]]
Classification_report of combined cross-validation data predicted results
              precision    recall  f1-score   support

           0       0.85      0.89      0.87      1564
           1       0.82      0.81      0.82       845
           2       0.85      0.82      0.84       973
           3       0.79      0.78      0.79       513
           4       0.83      0.84      0.84      1046
           5       0.82      0.80      0.81       353
           6       0.81      0.82      0.81       544
           7       0.82      0.76      0.79       254
           8       0.81      0.80      0.81       459

    accuracy                           0.83      6551
   macro avg       0.82      0.81      0.82      6551
weighted avg       0.83      0.83      0.83      6551

Cross-validation testing scores:
[0.74542683 0.81068702 0.80916031 0.84732824 0.84122137 0.83358779
 0.82900763 0.86259542 0.86259542 0.87328244]
Cross-validation training scores
[0.89397795 0.88297151 0.87398236 0.8775441  0.87856174 0.87737449
 0.87483039 0.86855495 0.87483039 0.87432157]
```

**Figure 14**

- Confusion matrix of combined cross-validation predicted data is displayed.
- Classification_report of combined cross-validation predicted data is displayed.
- The cross validation testing scores of the 10-folds are displayed.
- The cross validation training scores of the 10-folds are displayed.

**Intermediate Board Optimal Play (Single-Label):**

```
data_single = pd.read_csv('tictac_single.txt', sep=" ", header=None)

col_X = [0, 1, 2, 3, 4, 5, 6, 7, 8]
col_y = [9]
X_single = data_single[col_X]
y_single = data_single[col_y]
```

**Figure 15**

- Here the dataset used is the ‘tictac_single’.
- The given dataset consists of 10 columns.
- First 9 columns from index ‘0’ to ‘8’ are inputs for the desired model. The data of these columns are stored in variable ‘X_single’.
- The last column having index ‘9’ is the expected output or the labels for given sequential inputs. The data of this column is stored in variable ‘y_single’

**Linear SVM Classifier** : The algorithm creates a line or a hyperplane which separates the data into classes.

```
print("Linear SVM Classifier:")
clf_svm = svm.SVC(kernel='linear', degree=9, gamma='auto', C=9, coef0=0.11,
                  probability=True, random_state=42, class_weight='balanced')
obj.func(clf_svm, X_single, y_single)
```

**Figure 16**

- **Kernel:** Specifies the kernel type to be used in the algorithm. It must be one of 'linear', 'poly', 'rbf', 'sigmoid', 'precomputed' or a callable.
- **Degree:** Degree of the polynomial kernel function ('poly'). Ignored by all other kernels.
- **Gamma:** gamma='auto', uses 1 / n_features.
- **C:** Regularization parameter. The strength of the regularization is inversely proportional to C.

```
Confusion_matrix for initial training-test set
[[ 44 137  24  34   0  51   7  26   0]
 [  6  75   0  33   0  47   1   0   6]
 [ 17  78  28   8   0  46   0   5   5]
 [  4  23   7  25   0  40   0  18   0]
 [ 12  69  25  18   0  60   2  11   5]
 [  7  16   2   2   0  36   6   1   6]
 [  7  20  17  15   0  28   4   8   0]
 [ 11  15   6   2   0  10   3   2   1]
 [ 10  30   8   4   0  23   4   7   3]]
Classification_report for initial training-test set
              precision    recall  f1-score   support

           0       0.37      0.14      0.20       323
           1       0.16      0.45      0.24       168
           2       0.24      0.15      0.18       187
           3       0.18      0.21      0.19       117
           4       0.00      0.00      0.00       202
           5       0.11      0.47      0.17        76
           6       0.15      0.04      0.06        99
           7       0.03      0.04      0.03        50
           8       0.12      0.03      0.05        89

    accuracy                           0.17      1311
   macro avg       0.15      0.17      0.13      1311
weighted avg       0.19      0.17      0.14      1311

Training score for initial training-test set
0.1648854961832061
Testing score for initial training-test set
0.16552250190694126
Cross-validation Accuracies:
[0.13719512 0.14961832 0.1648855  0.17557252 0.15725191 0.18320611
 0.20458015 0.15267176 0.18167939 0.17251908]
Cross-validation Accuracies Mean:
0.1679179854775647
Cross-validation Accuracies Standard Deviation Mean:
0.01865786744900777
```

**Figure 17**

- The above figure consists of the confusion matrix for the Initial training-test set.
- The **confusion matrix** is a square matrix of size **9X9.**
- The size of the square matrix defines the number of different labels a Multi-Class-Single-Label classifier contains.
- We also display the **Classification Report** of the same model.
- The shown Classification Report contains parameters as mentioned earlier, for all the classes of output/labels.
- The figure also displays **cross-validated accuracy.**Here the number **of Folds =10**.
- Furthermore, we display the mean and standard deviation mean for all the calculated accuracies.

```
Confusion matrix of combined cross-validation data predicted results
[[176 728  52  85   4 302  16 182  19]
 [ 25 556   3   3   0 206  10  21  21]
 [ 92 408 100   8   4 275   1  64  21]
 [ 29 189  30  46   0 148   5  62   4]
 [ 93 434  39  69  15 294  13  85   4]
 [ 34  96  11   5   0 161  14  18  14]
 [ 42 178  63  18   9 144  14  69   7]
 [ 54  95  25   1   0  43   4  25   7]
 [ 45 186  19  14   0 119  19  50   7]]
Classification_report of combined cross-validation data predicted results
              precision    recall  f1-score   support

           0       0.30      0.11      0.16      1564
           1       0.19      0.66      0.30       845
           2       0.29      0.10      0.15       973
           3       0.18      0.09      0.12       513
           4       0.47      0.01      0.03      1046
           5       0.10      0.46      0.16       353
           6       0.15      0.03      0.04       544
           7       0.04      0.10      0.06       254
           8       0.07      0.02      0.02       459

    accuracy                           0.17      6551
   macro avg       0.20      0.17      0.12      6551
weighted avg       0.25      0.17      0.13      6551

Cross-validation testing scores:
[0.13719512 0.14961832 0.1648855  0.17557252 0.15725191 0.18320611
 0.20458015 0.15267176 0.18167939 0.17251908]
Cross-validation training scores
[0.15945717 0.17944369 0.18724559 0.17842605 0.14806649 0.16265265
 0.18962008 0.17130258 0.16841927 0.16977612]
```

**Figure 18**

- The above figure contains the confusion matrix for the combined result of cross-validated data.
- Here the number of predictions increases due to the cross-validation.
- We also display the **Classification Report** of the same model.
- The shown Classification Report contains parameters as mentioned earlier, for all the classes of output/labels.
- The figure also displays the **cross-validated accuracies.** Here the **number of Folds = 10**.
- We also calculate the **Cross-Validation testing score** and The **cross-validation training score**.

**KNN:** The classifier K-Nearest Neighbour clusters output data to a group of data to which the output is nearest too.

```
print("KNeighborsClassifier:")
clf_knn = KNeighborsClassifier(n_neighbors=5)
obj.func(clf_knn, X, y)
```

- **KNeighborsClassifier** : This function is imported from the sklearn class. It returns an object of a KNN model.
- **n_neigbors** : This parameter defines the number of data groups the model can have.

```
Confusion_matrix for initial training-test set
[[1127   75   53   10   70   15   32   12   12]
 [ 113  455   55   29   43   15    7    8   29]
 [ 182   85  493   15   47    7   17    3   24]
 [  98   65   64  151   35    6   32    9    7]
 [ 226  121  115   47  395   13   25    6   13]
 [  64   53   40   35   16   90   11    5    7]
 [ 111   67   66   36   15    4  176    2    8]
 [  36   45   13   15    9    5   15   69   11]
 [ 120   51   36   23   22    9    2    8  140]]
Classification_report for initial training-test set
              precision    recall  f1-score   support

           0       0.54      0.80      0.65      1406
           1       0.45      0.60      0.51       754
           2       0.53      0.56      0.55       873
           3       0.42      0.32      0.36       467
           4       0.61      0.41      0.49       961
           5       0.55      0.28      0.37       321
           6       0.56      0.36      0.44       485
           7       0.57      0.32      0.41       218
           8       0.56      0.34      0.42       411

    accuracy                           0.53      5896
   macro avg       0.53      0.44      0.47      5896
weighted avg       0.53      0.53      0.51      5896

Training score for initial training-test set
0.7862595419847328
Testing score for initial training-test set
0.5251017639077341
Cross-validation Accuracies:
[0.6875     0.72824427 0.71908397 0.77557252 0.80152672 0.78625954
 0.77557252 0.79541985 0.78015267 0.81984733]
Cross-validation Accuracies Mean:
0.7669179389312977
```

**Figure 19**

- The above Confusion matrix is for the KNN classifier when trained using the intermediate boards optimal play(single label) dataset.
- We also display the Classification report for the initial random shuffled dataset.
- The scores  with respect to the initial shuffled data set for both training and testing have been displayed.
- The **Training Score** is **0.78625** and **Testing Score** Obtained is **0.5251**
- We cross validate using 10-folds, and obtain accuracies for each fold, also display the mean accuracy with the standard deviation mean.

- The **cross-validation accuracy mean** is **0.766917**
- **Cross-validation accuracy Standard Deviation Mean** is **0.0394.**

```
Confusion matrix of combined cross-validation data predicted results
[[1366   33   42   18   40   10   28    8   19]
 [  73  628   38   22   36    7   15   12   14]
 [  89   52  742   20   32    8   14    2   14]
 [  45   34   21  347   25    8   11    9   13]
 [  80   48   43   23  823   10    9    3    7]
 [  32   24   12    9    9  236    8    6   17]
 [  62   25   23   24    8    4  388    7    3]
 [  24   18    5    9   11    5    4  166   12]
 [  50   23   19    9   13    6    5    6  328]]
Classification_report of combined cross-validation data predicted results
              precision    recall  f1-score   support

           0       0.75      0.87      0.81      1564
           1       0.71      0.74      0.73       845
           2       0.79      0.76      0.77       973
           3       0.72      0.68      0.70       513
           4       0.83      0.79      0.81      1046
           5       0.80      0.67      0.73       353
           6       0.80      0.71      0.76       544
           7       0.76      0.65      0.70       254
           8       0.77      0.71      0.74       459

    accuracy                           0.77      6551
   macro avg       0.77      0.73      0.75      6551
weighted avg       0.77      0.77      0.77      6551

Cross-validation testing scores:
[0.6875     0.72824427 0.71908397 0.77557252 0.80152672 0.78625954
 0.77557252 0.79541985 0.78015267 0.81984733]
Cross-validation training scores
[0.94893978 0.94708277 0.94657395 0.94759159 0.95267978 0.95267978
 0.94691316 0.94810041 0.94776119 0.94606513]
```

**Figure 20**

- The above Confusion matrix is for the KNN classifier when trained using the intermediate **board's optimal play(single label) dataset**.
- We also display the Classification report for the combined cross-validation predicted data.
- **Cross validation testing scores** are displayed.
- **Cross-validation training scores** are displayed.

**MLP Classifier** : MLPClassifier trains iteratively since at each time step the partial derivatives of the loss function with respect to the model parameters are computed to update the parameters. It can also have a regularization term added to the loss function that shrinks model parameters to prevent overfitting.

```
print("MLP Classifier:")
clf_mlp = MLPClassifier(random_state=42, max_iter=1000,
                        solver='lbfgs', activation='tanh', early_stopping=True)
obj.func(clf_mlp, X_single, y_single)
```

**Figure 21**

- **Solver:** It is used for weight optimization 'lbfgs' is an optimizer in the family of quasi-Newton methods.
- **Activation:** Activation function for the hidden layer. We have applied the 'tanh', the hyperbolic tan function, which returns f(x) = tanh(x).
- **Early_stopping*:*** Whether to use early stopping to terminate training when the validation score is not improving.

```
Confusion_matrix for initial training-test set
[[303   1   3   6   3   0   4   1   2]
 [  4 152   5   0   4   0   2   0   1]
 [  8   2 166   3   3   0   1   1   3]
 [  0   2   0 104   6   1   2   0   2]
 [  7   1   0   0 183   5   4   0   2]
 [  2   1   3   0   0  70   0   0   0]
 [  2   2   0   3   3   0  89   0   0]
 [  0   4   0   6   1   0   1  38   0]
 [  0   2   2   0   1   0   0   2  82]]
Classification_report for initial training-test set
              precision    recall  f1-score   support

           0       0.93      0.94      0.93       323
           1       0.91      0.90      0.91       168
           2       0.93      0.89      0.91       187
           3       0.85      0.89      0.87       117
           4       0.90      0.91      0.90       202
           5       0.92      0.92      0.92        76
           6       0.86      0.90      0.88        99
           7       0.90      0.76      0.83        50
           8       0.89      0.92      0.91        89

    accuracy                           0.91      1311
   macro avg       0.90      0.89      0.89      1311
weighted avg       0.91      0.91      0.91      1311

Training score for initial training-test set
1.0
Testing score for initial training-test set
0.9054157131960335
Cross-validation Accuracies:
[0.86585366 0.90839695 0.90839695 0.92366412 0.92824427 0.94503817
 0.93435115 0.94045802 0.95114504 0.94503817]
Cross-validation Accuracies Mean:
0.9250586482964065
Cross-validation Accuracies Standard Deviation Mean:
0.02423819606890225
```

**Figure 22**

- The above figure consists of the confusion matrix for the Initial training-test set.
- The **confusion matrix** is a square matrix of size **9X9.**
- The size of the square matrix defines the number of different labels a Multi-Class-Single-Label classifier contains.
- We also display the **Classification Report** of the same model.
- The shown Classification Report contains parameters as mentioned earlier, for all the classes of output/labels.
- The figure also displays the **cross-validated accuracies.** Here the **number of Folds = 10**.
- Furthermore, we display the mean and standard deviation mean for all the calculated accuracies.

```
Confusion matrix of combined cross-validation data predicted results
[[1481    8   18    5   21    7   10    9    5]
 [  10  769   22    7   13    4    7    6    7]
 [  18   14  896   10   10    0    9    3   13]
 [  11    6    5  462   11    3    2    6    7]
 [  17   10    6   10  986    9    3    1    4]
 [   9   10    2    6    3  316    2    4    1]
 [  13    5   11   14    2    1  495    1    2]
 [   2    1    1    5    1    3    2  237    2]
 [  15    4    5    3    6    5    2    1  418]]
Classification_report of combined cross-validation data predicted results
              precision    recall  f1-score   support

           0       0.94      0.95      0.94      1564
           1       0.93      0.91      0.92       845
           2       0.93      0.92      0.92       973
           3       0.89      0.90      0.89       513
           4       0.94      0.94      0.94      1046
           5       0.91      0.90      0.90       353
           6       0.93      0.91      0.92       544
           7       0.88      0.93      0.91       254
           8       0.91      0.91      0.91       459

    accuracy                           0.93      6551
   macro avg       0.92      0.92      0.92      6551
weighted avg       0.93      0.93      0.93      6551

Cross-validation testing scores:
[0.86585366 0.90839695 0.90839695 0.92366412 0.92824427 0.94503817
 0.93435115 0.94045802 0.95114504 0.94503817]
Cross-validation training scores
[1. 1. 1. 1. 1. 1. 1. 1. 1. 1.]
```

**Figure 23**

- The above figure contains the confusion matrix for the combined result of cross-validated data. The classifier K-Nearest Neighbour clusters output data to a group of data to which the output is nearest too.
- Here the number of predictions increases due to the cross-validation.
- We also display the **Classification Report** of the same model.
- The shown Classification Report contains parameters as mentioned earlier, for all the classes of output/labels.
- The figure also displays the **cross-validated accuracy.** Here the **number of Folds = 10**.
- We also calculate the **Cross-Validation testing score** and The **cross-validation training score**.

## Best Model for Classifiers:

As visible in the results above,all the 3 Classifiers trained on the Final dataset gave a very good accuracy of approximately 98-99% as well as none of them were overfitting and also MLP tends to give a slightly better result.

On the Intermediate Board optimal play single label, both MLP as well as KNN gave a very good accuracy of around 92-93% as well as both of them were not overfitted and MLP tends to give a slightly better result.However,on the other hand Linear SVM tends to perform very poor result of accuracy around 20%,the reason might be as the data is not linearly separable.

**Impact of training only 10% data on classifiers:**

**For final boards classification dataset:**

```
For final boards classification dataset:
Linear SVM Classifier:
Confusion_matrix for initial training-test set
[[289  15]
 [  0 559]]
Classification_report for initial training-test set
              precision    recall  f1-score   support

          -1       1.00      0.95      0.97       304
           1       0.97      1.00      0.99       559

    accuracy                           0.98       863
   macro avg       0.99      0.98      0.98       863
weighted avg       0.98      0.98      0.98       863
```

```
MLP Classifier:
Confusion_matrix for initial training-test set
[[290  14]
 [  0 559]]
Classification_report for initial training-test set
              precision    recall  f1-score   support

          -1       1.00      0.95      0.98       304
           1       0.98      1.00      0.99       559

    accuracy                           0.98       863
   macro avg       0.99      0.98      0.98       863
weighted avg       0.98      0.98      0.98       863
```

**Figure 24**

```
KNeighborsClassifier:
Confusion_matrix for initial training-test set
[[199 105]
 [ 23 536]]
Classification_report for initial training-test set
              precision    recall  f1-score   support

          -1       0.90      0.65      0.76       304
           1       0.84      0.96      0.89       559

    accuracy                           0.85       863
   macro avg       0.87      0.81      0.82       863
weighted avg       0.86      0.85      0.85       863
```

**Figure 25**

As visible from the above result, there was not much issue in the 2 classifiers(Linear SVM,MLP) when training data was reduced to 10% since all the 3 classifiers gave an accuracy of above 90 %.Though some effect was visible in KNN's accuracy as it got reduced to 87%.

```
For intermediate boards optimal play(single label) dataset:
KNeighborsClassifier:
Confusion_matrix for initial training-test set
[[1153   59   61    8   55    5   31   24   10]
 [  65  466   77   13   51    6   25   21   30]
 [ 123   73  543   23   44    4   27   10   26]
 [  79   49   38  171   41    5   57   14   13]
 [ 154   98   65   23  537   13   34   14   23]
 [  52   28   30   26   10  135   13    9   18]
 [  97   29   43   28   25    0  233   10   20]
 [  26   22   17    5   11    2    9  113   13]
 [  96   30   24    7   28    9    8   12  197]]
Classification_report for initial training-test set
              precision    recall  f1-score   support

           0       0.62      0.82      0.71      1406
           1       0.55      0.62      0.58       754
           2       0.60      0.62      0.61       873
           3       0.56      0.37      0.44       467
           4       0.67      0.56      0.61       961
           5       0.75      0.42      0.54       321
           6       0.53      0.48      0.51       485
           7       0.50      0.52      0.51       218
           8       0.56      0.48      0.52       411

    accuracy                           0.60      5896
   macro avg       0.60      0.54      0.56      5896
weighted avg       0.60      0.60      0.59      5896
```

**Figure 26**

```
Linear SVM Classifier:
Confusion_matrix for initial training-test set
[[313 246 139 265 152 212   8  71   0]
 [ 28 176  40 127  51 124  61  97  50]
 [ 57 171 183 158 103 136  16   4  45]
 [ 18  80  46 100  51  92  51  12  17]
 [ 23 160 126 228 239 148  14  18   5]
 [ 24  27  55  60  11  83  17  23  21]
 [ 45  67  73  82  31  60  78  18  31]
 [ 34  25  31  12   7  29  25  36  19]
 [ 46  54  15  18  46  76  18  45  93]]
Classification_report for initial training-test set
              precision    recall  f1-score   support

           0       0.53      0.22      0.31      1406
           1       0.17      0.23      0.20       754
           2       0.26      0.21      0.23       873
           3       0.10      0.21      0.13       467
           4       0.35      0.25      0.29       961
           5       0.09      0.26      0.13       321
           6       0.27      0.16      0.20       485
           7       0.11      0.17      0.13       218
           8       0.33      0.23      0.27       411

    accuracy                           0.22      5896
   macro avg       0.25      0.22      0.21      5896
weighted avg       0.31      0.22      0.24      5896
```

**Figure 27**

```
MLP Classifier:
Confusion_matrix for initial training-test set
[[1064   38   48   33   67   23   56   13   64]
 [  32  482   54   39   76   20   24    8   19]
 [  75   52  571   31   65   10   30   23   16]
 [  32   24   20  247   48   41   33   12   10]
 [  69   55   68   54  641   19   22    2   31]
 [  30   36   11   28   12  151   19   12   22]
 [  37   53   41   19   16   18  272   14   15]
 [  15   21   12   12    2   23    5  113   15]
 [  25   19   16   29   29   14   15   16  248]]
Classification_report for initial training-test set
              precision    recall  f1-score   support

           0       0.77      0.76      0.76      1406
           1       0.62      0.64      0.63       754
           2       0.68      0.65      0.67       873
           3       0.50      0.53      0.52       467
           4       0.67      0.67      0.67       961
           5       0.47      0.47      0.47       321
           6       0.57      0.56      0.57       485
           7       0.53      0.52      0.52       218
           8       0.56      0.60      0.58       411

    accuracy                           0.64      5896
   macro avg       0.60      0.60      0.60      5896
weighted avg       0.64      0.64      0.64      5896
```

**Figure 28**

As visible from the above result,the 2 classifiers'(MLP and KNN) accuracy went down when training data was reduced to 10%.The accuracy dropped significantly due to the fact that there was a lack of available data points for the algorithm to learn.The impact was much visible in MLP and KNN since there accuracy went down from 92-93% to 60% when training data was reduced.Linear SVM accuracy remains nearly the same as it previously only did not generated good results even when the training data was high.

## Regressors:

```
class Regressor():

    def funcreg(self,reg, X, y, filename):

        X_train, X_test, y_train, y_test = train_test_split( X, y, shuffle=True, test_size=0.20, random_state=42)
        reg.fit(X_train, y_train)
        y_predicted_proba = reg.predict(X_test)
        weig = 0.5
        #len(y[y == 1])/len(y[y == 0])
        y_predicted = np.where(y_predicted_proba >= weig, 1, 0)
        print("Confusion_matrix for initial training-test set")
        print(confusion_matrix(y_test, y_predicted))
        print("Classification_report for initial training-test set")
        print(classification_report(y_test, y_predicted))
        #print("Training score for initial training-test set")
        #print(reg.score(X_train, y_train))
        #print("Testing score for initial training-test set")
        #print(reg.score(X_test, y_test))
        accuracies = cross_val_score(estimator=reg, X=X, y=y, cv=10)
        print("Cross-validation-accuracies")
        print(accuracies)
        print("Cross-validation-accuracies Mean:")
        print(accuracies.mean())
        print("Cross-validation-accuracies Standard Devaition:")
        print(accuracies.std())
        pickle.dump(reg, open(filename, 'wb'))
```

**Figure 29**

- This function is used for KNN and MLP regression models. The object of each of these models is trained using respected imported classes and is passed in the given regression function for prediction and extracting of statistical parameters such as the confusion matrix, accuracy of the model object being passed through the main function.
- The above function has parameters reg, X, y, filename.
- **reg**: This parameter contains the object of the trained model.
- **X**: This consists of the set of data that act as inputs for the model. This data has been preprocessed from the given CSV file.
- **Y**: This consists of the set of data that act as a label for a respected set of inputs. This data has also been preprocessed from the given CSV file.
- Since the dataset for the regression model consists of a Multi-Label Dataset we have been given the liberty to make separate models for each output.
- filename: We notice that this parameter is not used in the classification function, that is because we need to store multiple trained models for regression. The filename parameter consists of a path that specifies the location where each individual model with respect to the ‘y’ label column has been written on the disk.
- Moving forward, we shuffle the obtained data using the function ‘train_test_split()’
- **train_test_split()** : This function returns X_train, X_test, y_train and y_test. It also shuffles the data with respect to the random state specified in the function parameter. This function also divides the given dataset into training and testing data; we can specify the distribution proportion using the ‘test_size’ parameter.
- The shuffled and preprocessed training data is fitted in the model object passed ‘reg’ using the ‘**fit’** function.
- The model is then used to predict the output for an unknown train input set which is stored in the X_test variable. The predicted output is stored for further processing.

- As we are required to treat the predicted output as a classifier output even though we use a regression model for prediction, the predicted output have been given value of 1 or 0 with respect to how they relate with the predefined ‘weigh’ variable(weigh=0.5)
- **confusion_matrix()** : This function is used to print the confusion matrix table (As explained earlier in the document)
- **classification_report()** : This function gives additional insight of the model. The function returns a report which consists of the precision, recall, F1 and Support scores for the executed model.
- Score is an additional quantifier reflecting on the accuracy of the model. We remove the score for the training and testing set individually.
- **cross_val_score()** : This function returns an array of accuracy for each fold executed internally with any explicit command. The function must be given the number of cross validation needed in the function parameter ‘cv’.
- We calculate the mean accuracy for all the accuracies returned by the cross_val_score() function. We also calculate the standard deviation with respect to all the accuracies obtained via the cross_val_score().
- **Pickle.dump** : This dump function is imported from the pickle class. The function writes the given model into the path specified ‘here: filename’.

**Intermediate Board Optimal Play (Multi-Label):**

```
if __name__ == '__main__':
    obj = Regressor()
    data_multiple = pd.read_csv('tictac_multi.txt', sep=" ", header=None)
    col_X = [0, 1, 2, 3, 4, 5, 6, 7, 8]
    col_y = [9, 10, 11, 12, 13, 14, 15, 16, 17]
    X_multiple = data_multiple[col_X]
    y_multiple=data_multiple[col_y]
```

**Figure 30**

- For the **‘tictac_multi’** dataset, all the following models are regression models.
- This dataset is a **multi-class-multi-label dataset.**
- In the given case we have **9 inputs** and **9 outputs.**
- **data_multiple** is a data frame that has a csv file with tuples and columns.
- **col_X :** This is an array, it contains the index position of each input column, it is later used to extract the input values from the dataframe data_multiple into a variable **‘X_multiple’**
- Col_y : This is an array, it contains the index position of each output column, it is later used to extract the output values from the dataframe data_multiple into a variable **‘y_multiple’**

**k-NN regressor:** The k-NN regression outputs value according to the average of its k-nearest neighbors.

```
print("KNN Regressor:")
for i in col_y:
    filename = 'Model_param_col_'+str(i-9)+'.pkl'
    print("For column no: "+str(i-9))
    reg_knn = KNeighborsRegressor(n_neighbors=9,weights='distance',algorithm='kd_tree',p=2,leaf_size=9,n_jobs=-1)
    obj.funcreg(reg_knn, X_multiple, data_multiple[i],filename)
```

**Figure 31**

- All of the regression models trained will have **multiple output** since the dataset is multiclass.
- The '**filename**' variable is **String type.** It contains a path where each of the models trained will be **written** on **disk space.**
- The loop in the above code runs until all the values in 'col_y' are executed.
- Since for each index position of the output the model is trained separately, we have to use the **pickle class** for **storing** all models in a **serialization format**. This will allow the model to be trained and be able to **predict output according to an index position**.
- **KneighborsRegressor():** This function returns an object of an inbuilt class; the object has the properties of a k-NN regressor.
- The object along with X_multiple, data_multiple[i] (output for a specific index position) and filename (path).

```
For column no: 0
Confusion_matrix for initial training-test set
[[971  17]
 [ 32 291]]
Classification_report for initial training-test set
              precision    recall  f1-score   support

           0       0.97      0.98      0.98       988
           1       0.94      0.90      0.92       323

    accuracy                           0.96      1311
   macro avg       0.96      0.94      0.95      1311
weighted avg       0.96      0.96      0.96      1311

Cross-validation-accuracies
[0.74449855 0.858954   0.83401277 0.8765574  0.87361868 0.92266674
 0.87459051 0.92029613 0.90131005 0.9193329 ]
Cross-validation-accuracies Mean:
0.872583772326226
Cross-validation-accuracies Standard Devaition:
0.05088777617738081
For column no: 1
Confusion_matrix for initial training-test set
[[1075   11]
 [  33  192]]
Classification_report for initial training-test set
              precision    recall  f1-score   support

           0       0.97      0.99      0.98      1086
           1       0.95      0.85      0.90       225

    accuracy                           0.97      1311
   macro avg       0.96      0.92      0.94      1311
weighted avg       0.97      0.97      0.97      1311

Cross-validation-accuracies
[0.6207188  0.77455456 0.78110271 0.83677188 0.84599373 0.82848989
 0.88456824 0.82554314 0.86120933 0.87314587]
```

**Figure 32**

- In the above figure, the **confusion matrix** along with the **classification report** with additional information is for only ‘Column 0’ and ‘Column 1’
- Similar outputs have been obtained for all the ‘9’ columns, each for a specific label/output/class.
- The figure contains the confusion matrix and classification report.
- We also display the accuracy once the dataset has been cross-validated.
- Additionally , the mean of all accuracy and the mean of standard deviation obtained via each fold (here:10) has been displayed.

**Linear regressor** : This is the foremost and the most basic type of regressor used for a model.

```
def linear_reg(self, X,y):
    np_x_train, np_x_test, np_y_train, np_y_test = train_test_split( X, y, shuffle=True, test_size=0.20, random_state=42)
    np_x_train=np.array(np_x_train)
    np_y_train=np.array(np_y_train)
    np_x_test=np.array(np_x_test)
    np_y_test=np.array(np_y_test)
    Y_pred = np.empty((np.shape(np_y_test)[0], np.shape(np_y_test)[1]))
    bias = 1
    for i in range(9):
        y = np_y_train[:, i]
        W = np.linalg.inv(np_x_train.T @ np_x_train) @ np_x_train.T @ y
        W = [weight + bias for weight in W]
        y_pred = np_x_test @ W
        Y_pred[:, i] = y_pred

    Y_pred = (Y_pred == Y_pred.max(axis=1)[:, None]).astype(int)

    total_acc = np.empty(9)
    for i in range(9):
        total_acc[i] = self.get_accuracy_score(np_y_test[:, i],
                                               Y_pred[:, i], normalized=False)

    acc = np.sum(total_acc) / (np.shape(np_y_test)[0] * 9)
    print("Accuracy LR: {0}".format(acc))

def get_accuracy_score(self,y_true, y_pred, normalized=True):
    pred_accu = accuracy_score(y_true, y_pred, normalize=normalized)
    return pred_accu
```

**Figure 33**

- The function called for this is different like the rest of the two regression, since here no library model has been called to return an object of the mentioned model.
- We created a model using normal equations.
- **train_test_split() :** This function shuffles and proportionally divides the dataset into training and testing.
- **W:** it is a variable which contains all the weight calculated for specific labels.
- **Y_pred :** the empty variable now contains the predicted values for each output column since the loop runs 9 times and each time takes a different index position column for y into account.
- **get_accuracy_score() :** this imported function is used to obtain a performance matrix to evaluate the model.

```
Linear Regressor:
Accuracy LR: 0.7230273752012882
```

**Figure 34**

- The above figure is the accuracy returned when using the linear regression user defined model via normal equation.

**MLP Regressor:** MLP Regressor trains iteratively since at each time step the partial derivatives of the loss function with respect to the model parameters are computed to update the parameters. It can also have a regularization term added to the loss function that shrinks model parameters to prevent overfitting.

```
print("MLP Regressor:")
for i in col_y:
    filename = 'Model_param_col_'+str(i-9)+'.pkl'
    print("For column no: "+str(i-9))
    reg_knn = MLPRegressor(random_state=42, max_iter=50, solver='lbfgs', activation='tanh')
    obj.funcreg(reg_knn, X_multiple, data_multiple[i],filename)
```

**Figure 35**

- All of the regression models trained will have **multiple outputs** since the dataset is multiclass.
- The '**filename**' variable is **String type.** It contains a path where each of the models trained will be **written** on **disk space.**
- The loop in the above code snippet runs until all the values in 'col_y' are executed.
- Since for each index position of the output the model is trained separately, we have to use the **pickle class** for **storing** all models in a **serialization format**. This will allow the model to be trained and be able to **predict output according to an index position**.
- **MLPRegressor():** This function returns an object of an inbuilt class; the object has the properties of a MLP Regressor.
- The object along with X_multiple, data_multiple[i] (output for a specific index position) and filename (path).

```
MLP Regressor:
For column no: 0
Confusion_matrix for initial training-test set
[[825 163]
 [ 10 313]]
Classification_report for initial training-test set
              precision    recall  f1-score   support

           0       0.99      0.84      0.91       988
           1       0.66      0.97      0.78       323

    accuracy                           0.87      1311
   macro avg       0.82      0.90      0.84      1311
weighted avg       0.91      0.87      0.88      1311

Training score for initial training-test set
0.6371645346470576
Testing score for initial training-test set
0.5861326638702307
Cross-validation-accuracies
[0.5592843  0.60452339 0.59429023 0.57366233 0.63032103 0.66176886
 0.55887115 0.64979357 0.65039632 0.63463189]
Cross-validation-accuracies Mean:
0.611754307784638
Cross-validation-accuracies Standard Devaition:
0.03696993772994624
```

**Figure 36**

- In the above figure, the **confusion matrix** along with the **classification report** with additional information is for only ‘Column 0’ and ‘Column 1’
- Similar outputs have been obtained for all the ‘9’ columns, each for a specific label/output/class.
- The figure contains the confusion matrix and classification report.
- We also display the accuracy once the dataset has been cross-validated.
- Additionally , the mean of all accuracy and the mean of standard deviation obtained via each fold (here:10) has been displayed.

## Best model for Regression:

**Regressors Final Accuracy:**

KNN Regressor: Accuracy 84.44%
Linear Regressor: Accuracy 72%
MLP Regressor:Accuracy 96%

As visible from the above result,MLP Regressor performs best out of the three Regressors.The reason can be given to the fact that a large dataset is available and thus,a very good depth can be accomplished.Also,dataset only contains -1,1 tanh activation function perfectly fits the model. KNN also performs very well but less than MLP primarily for the reason of being a lazy learning model with a local approximation.Linear Regression performs the worst of three,the primary reason is due to the fact that Linear Regression is highly sensitive to imbalance data.

## Optimal Model and observations for the tic tac toe game created:

All the 3 Models(Linear Regression,K-Nearest Neighbours,MultiLayer Perceptron) were used to train with the Intermediate Boards Optimal Play(Multi-Label) dataset and the simultaneous tic tac toe game was built with each of the 3 models been tried as a predictor for the Computer AI.

**Linear Regression:**

Linear Regression Model was unable to generate optimal outputs and lost both the game played with a human player.The reason may be as it predicts the next move as a vector with most optimal position as 1 and rest 0.So,if the most optimal position is already occupied,it predicts the next move which may tend to lose the computer ai the game.

## Game Snippets:

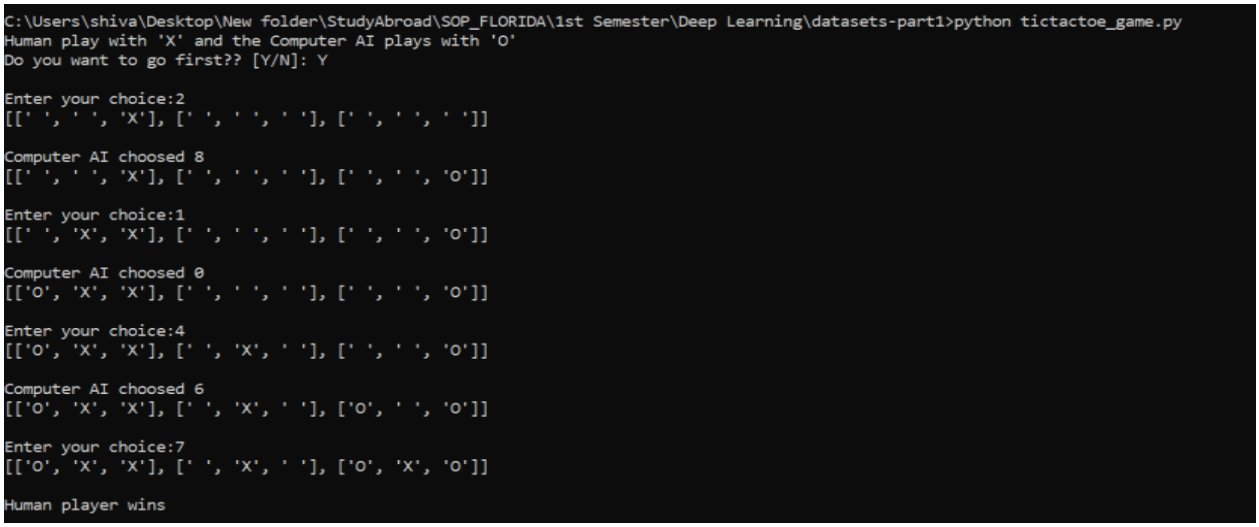


**Figure 37**

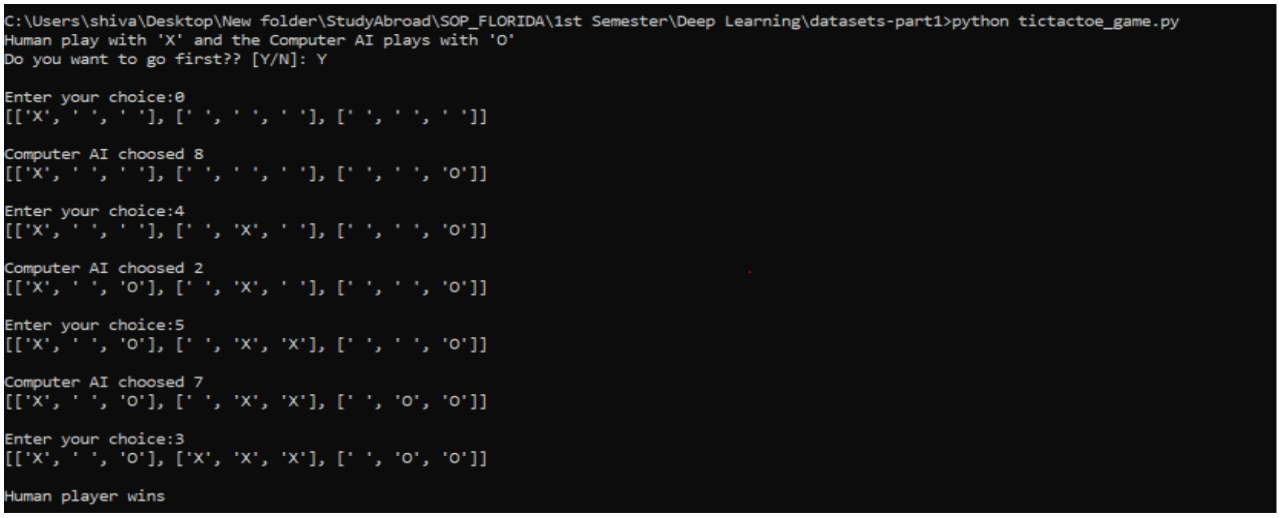


**Figure 38**

**K-Nearest Neighbours:**

K-Nearest Neighbours Model was able to generate optimal outputs and won 1 game and tied 1 with a human player.The reason may be as it predicts the next move as a multiple vector with most optimal position and assign 1 to them and rest 0.So,if the most optimal position is already occupied,it predicts the next move which is also optimal making the Computer AI to win and tie against the human player.KNN Regressor worked nicely as we were able to use the KD-Tree approach in the Regressor.K-D Tree's basic idea is that if point A is very distant from point B, and point B is very close to point C, then we know that points A and C are very distant, without having to explicitly calculate their distance.So,here the regressor performed better as it was able to calculate the distance of all 9 inputs explicitly and thus the regressor performed very well.

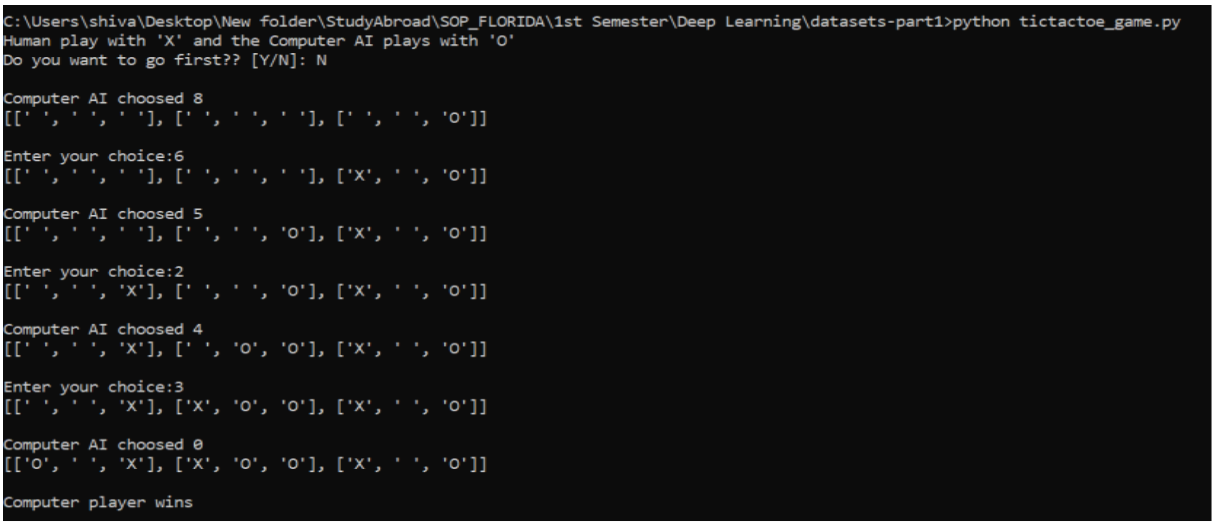


**Figure 39**

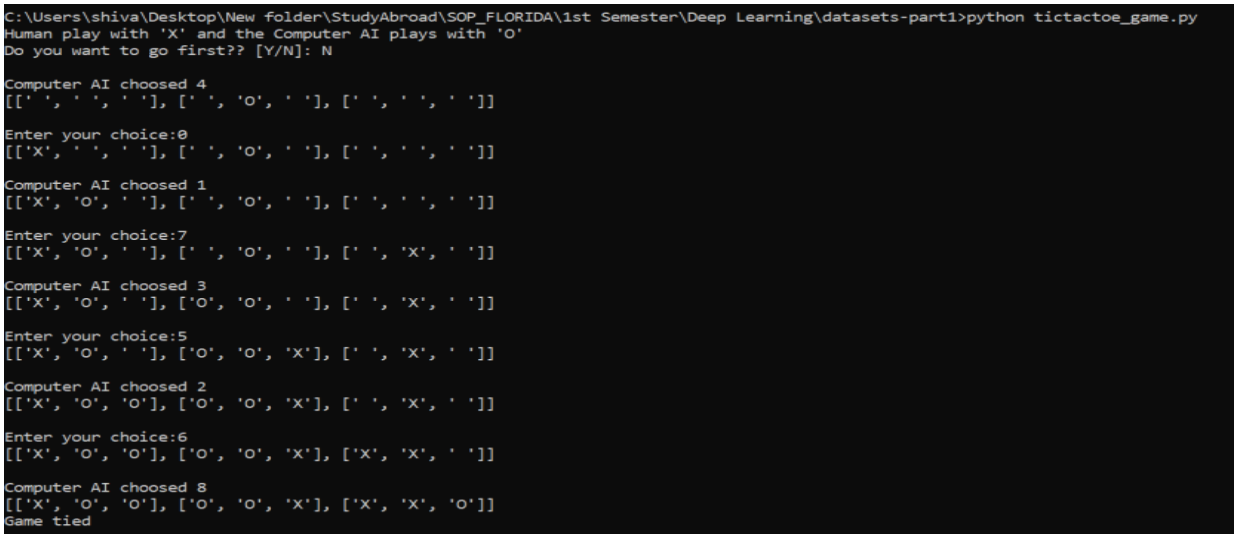


**Figure 40**

**MultiLayer Perceptron:**

MultiLayer Perceptron Model was able to generate optimal outputs and won 1 game and tied 1 with a human player.The reason may be as it predicts the next move as a multiple vector with most optimal position and assign 1 to them and rest 0.So,if the most optimal position is already occupied,it predicts the next move which is also optimal making the Computer AI to win and tie against the human player.

MLP regressor has the best accuracy among the three,the reason was being of able to train with a very good depth of available data points iteratively .The hidden units take the sum of this 9 inputs and assign weights to each one of them and then,apply the tanh activation function which compressed the domain from a set of -infinity to +infinity to a set of -1 and +1 and since the outputs are only either -1 or +1 here,tanh activation function worked nicely.

Since,we used a lot of iterations with 1000 and early stopping we were able to iteratively get to the depth of the dataset and as soon as loss increases,early stopping ends the training there.Also,we used the solver lbfgs(which stands for Limited-memory Broyden-Fletcher-Goldfarb-Shanno ) which basically approximates the second derivative updates with gradient evaluations.Also,it stores only last few updates,so it works well in this type of dataset where we have to predict the next step by evaluating the current board position as chances of getting overfitted gets reduced significantly.

```
C:\Users\shiva\Desktop\New folder\StudyAbroad\SOP_FLORIDA\1st Semester\Deep Learning\datasets-part1>python tictactoe_game.py
Human play with 'X' and the Computer AI plays with 'O'
Do you want to go first?? [Y/N]: Y

Enter your choice:8
[[' ', ' ', ' '], [' ', ' ', ' '], [' ', ' ', 'X']]

Computer AI choosed 4
[[' ', ' ', ' '], [' ', 'O', ' '], [' ', ' ', 'X']]

Enter your choice:0
[['X', ' ', ' '], [' ', 'O', ' '], [' ', ' ', 'X']]

Computer AI choosed 1
[['X', 'O', ' '], [' ', 'O', ' '], [' ', ' ', 'X']]

Enter your choice:7
[['X', 'O', ' '], [' ', 'O', ' '], [' ', 'X', 'X']]

Computer AI choosed 6
[['X', 'O', ' '], [' ', 'O', ' '], ['O', 'X', 'X']]

Enter your choice:2
[['X', 'O', 'X'], [' ', 'O', ' '], ['O', 'X', 'X']]

Computer AI choosed 5
[['X', 'O', 'X'], [' ', 'O', 'O'], ['O', 'X', 'X']]

Enter your choice:3
[['X', 'O', 'X'], ['X', 'O', 'O'], ['O', 'X', 'X']]
Game tied
```

**Figure 41**

```
C:\Users\shiva\Desktop\New folder\StudyAbroad\SOP_FLORIDA\1st Semester\Deep Learning\datasets-part1>python tictactoe_game.py
Human play with 'X' and the Computer AI plays with 'O'
Do you want to go first?? [Y/N]: Y

Enter your choice:2
[[' ', ' ', 'X'], [' ', ' ', ' '], [' ', ' ', ' ']]

Computer AI choosed 6
[[' ', ' ', 'X'], [' ', ' ', ' '], ['O', ' ', ' ']]

Enter your choice:0
[['X', ' ', 'X'], [' ', ' ', ' '], ['O', ' ', ' ']]

Computer AI choosed 1
[['X', 'O', 'X'], [' ', ' ', ' '], ['O', ' ', ' ']]

Enter your choice:4
[['X', 'O', 'X'], [' ', 'X', ' '], ['O', ' ', ' ']]

Computer AI choosed 8
[['X', 'O', 'X'], [' ', 'X', ' '], ['O', ' ', 'O']]

Enter your choice:3
[['X', 'O', 'X'], ['X', 'X', ' '], ['O', ' ', 'O']]

Computer AI choosed 7
[['X', 'O', 'X'], ['X', 'X', ' '], ['O', 'O', 'O']]

Computer player wins
```

**Figure 42**

Since MLP Regressor performed with the best accuracy, the final AI predictor was trained with the MLPRegressor.

## Conclusion:

We built a tic tac toe game in this project between a human player and a computer AI where the AI was trained with various classification and regression algorithms. We then did a comparative analysis to find the feasibility of each of those algorithms with the different datasets.

We conclude that the Multilayer Perceptron Regressor will build the best model for the given dataset. While playing the game when trained with k-NN and Linear regression model, we were easily able to beat the computer which was certainly not the case when the AI was build on the MLP Regressor model.