CAP 5404 PART1

classifier and regressor for connectFour

Yujie Wang(59913548)
Juyeon Park(57396009)
Preston Goren(78952836)

Introduction

This project is a deep learning project, our group analyzes the dataset that is provided to us via different ways. KNN, Multilayer perceptron, linearSVM, and linear regression. Our implementation is based on scikit-learn.

Github Link:

https://github.com/Wangyuji/DL_for_CG.git

Dataset:

The dataset we used is provided by our professor.
tictac_final.txt, tictac_multi.txt, tictac_single.txt all 3 datasets exist in the ../tictactoedatasets fold.

Program composition：

MLP_clf_reg.py contains classifier and regressor of Multilayer Perceptron
KNN_clf_reg.py contains classifier and regressor of KNN
SVM_Regression_clf_reg.py contains classifier of SVM and a regressor of Linear Regression

connectFour game is in the humanGame.ipynb.

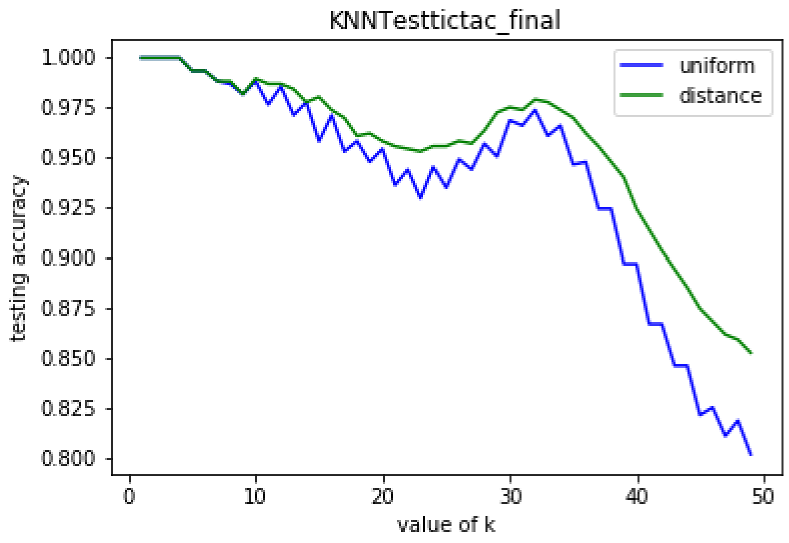
Run projects:

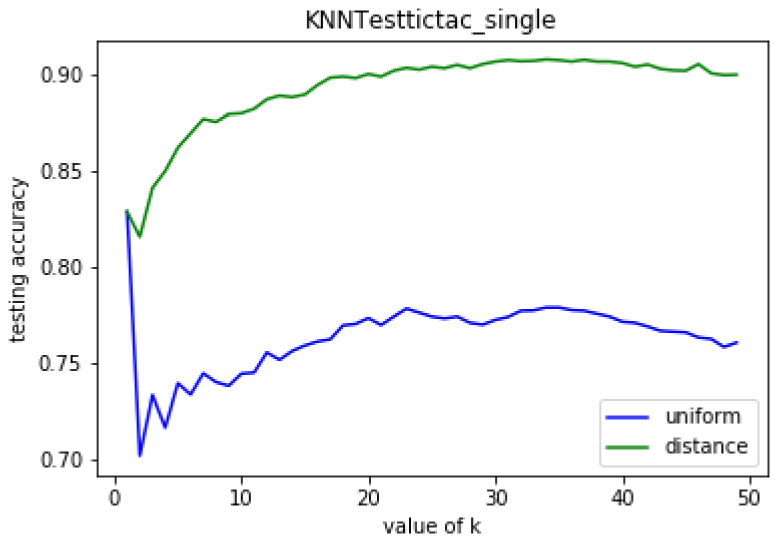
we use jupyterNotebook in Anaconda3 to compile the code.
Run **runAll.ipynb** will auto compile all the .py files and get the results of each models and run **humanGame.ipynb** will start connectFour game

Classifier and Regressor

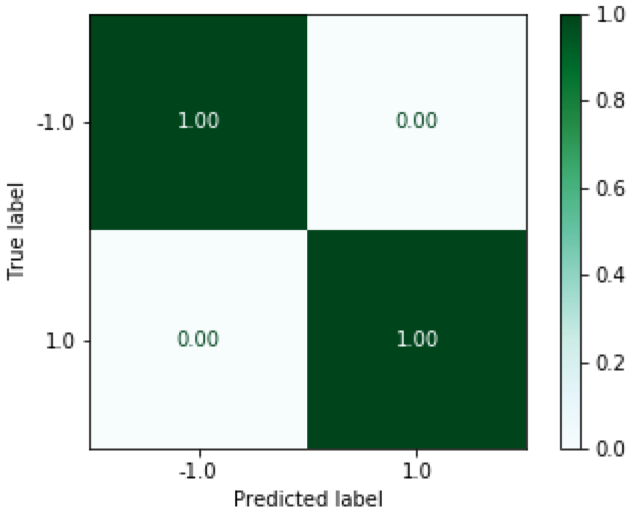
KNN Classifier

For the k-nearest neighbors classification, we plotted the mean testing accuracies of various values of k (1 through 50) and two weight functions (uniform and distance) using a grid search with 10-fold cross validation. This was done for both the tictac_final and tictac_single datasets.

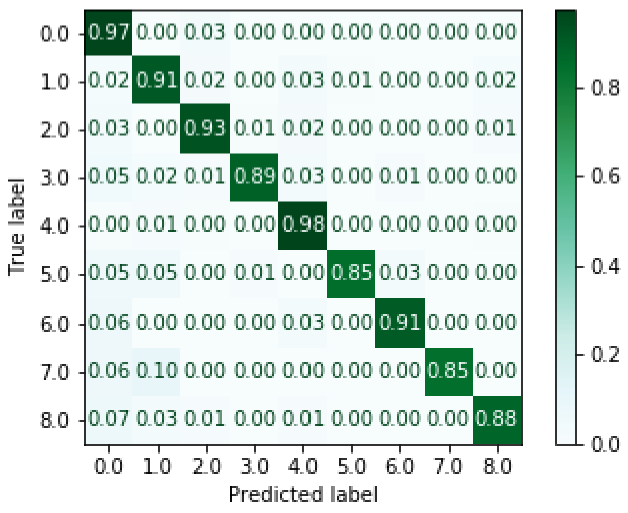




This grid search allowed us to determine the optimal hyperparameters for our KNN classifiers. The optimal parameters for tictac_final were n_neighbors = 1 and weights = 'uniform' and the optimal parameters for tictac_multi were n_neighbors= 34 and weights = 'distance'. The testing accuracy for the knn classifier on the tictac_final dataset was 100%, the testing accuracy for the knn classifier on the tictac_multi dataset was 92.8%. Confusion matrices for the knn classifier on the two datasets can be seen below.

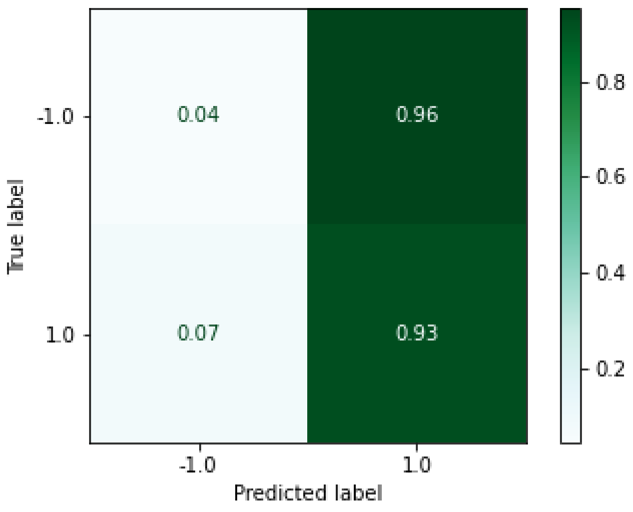


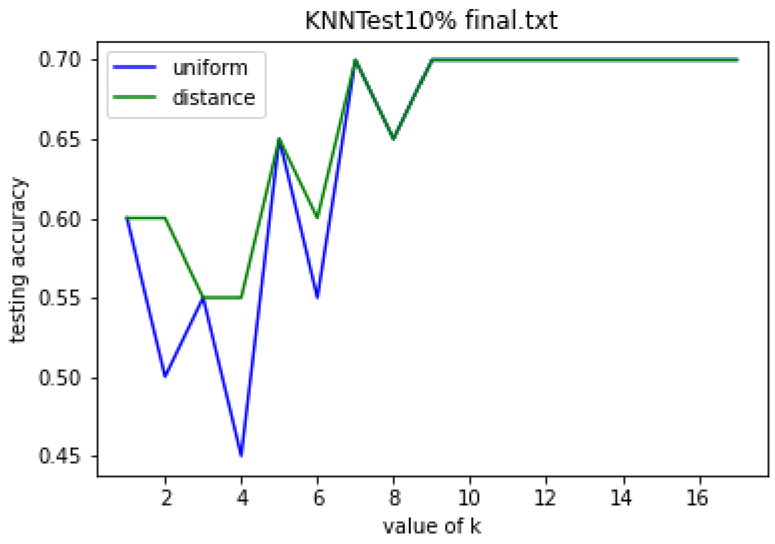
Confusion Matrix for knn classifier on tictac_final dataset



Confusion Matrix for knn classifier on tictac_single dataset

KNN Classifier Performance on 10% of Data
When trained on 10% of the tictac_final dataset, the KNN classifier performance dropped to 70.1% accuracy. The best parameters were determined to be n_neighbors = 7 and weights = ‘uniform’ via a grid search. The results of the grid search (various values of k for two different weighting systems plotted against test accuracy) can be seen below along with the normalized confusion matrix. The classifier performed significantly worse when trained on 10% of the data which makes sense because there is not as much to learn from.





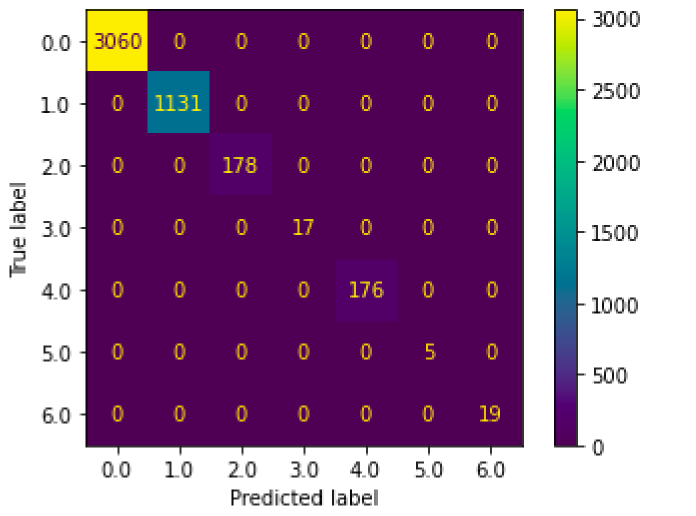
Confusion matrix and grid search results for KNN classifier trained on 10% of tictac_final data.

KNN Regressor

For the k-nearest neighbors regression, we used a grid search with 10-fold cross validation to compare various values of k (1 through 10) and two weight functions (uniform and distance) on the tictac_multi dataset. The grid search allowed us to determine the optimal hyperparameters for our KNN regressor. The optimal parameters for tictac_multi were n_neighbors = 9 and weights = ‘distance’. Accuracy was calculated by calculating the number of correctly predicted testing samples for each tic tac toe board position (9 total), this was then normalized for the number of samples in the testing set to end up with a final percentage. KNN regression on the tictac_multi dataset resulted in a testing accuracy of 78.09%.

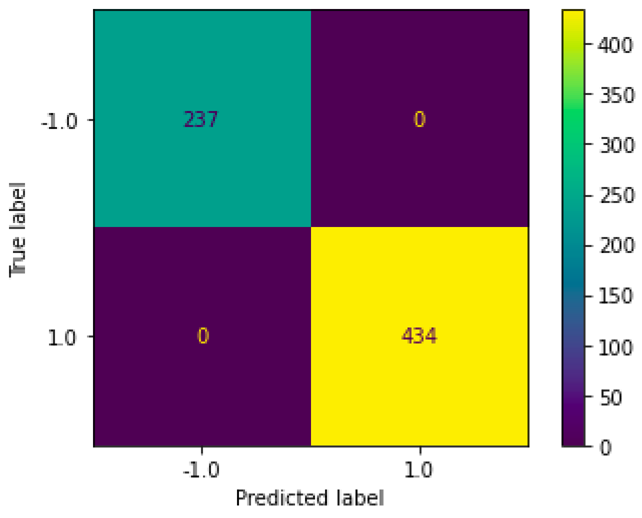
LinearSVM Classifier

For Linear Support Vector Machine (linearSVM), we plotted the accuracies with some values of C (regularization parameter, must be positive number). We used grid search with 10 folds and 10 random states. These are applied for 10% trained tictac_final, normal tictac_final and tictac_single datasets.



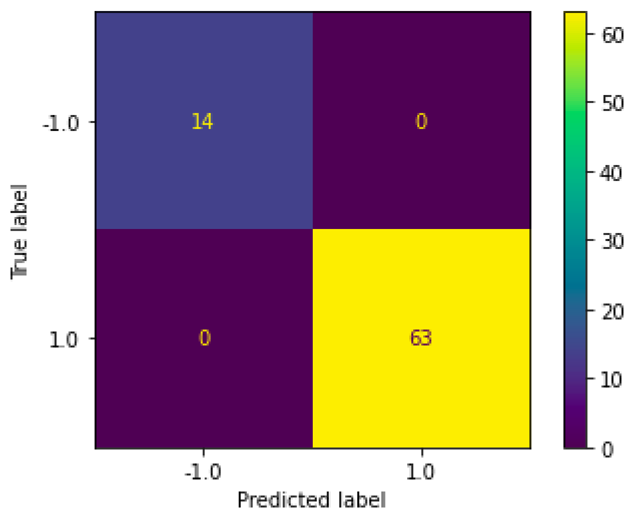
confusion matrix of the tictac_single dataset

We used five different values for the parameter C: 0.01,0.1,1,2,and 5 for both tictac_single and tictac_final. The above confusion matrix is the output of the tictac_single dataset. In this linear SVM classifier, the best parameter C we got was ‘5’, and the testing accuracy was 33.6%.



confusion matrix of the tictac_final dataset

This confusion matrix is the output of the tictac_final dataset. The best parameter C for this final dataset used linear SVM classifier was ‘1’, and the testing accuracy was approximately 98.6%.



confusion matrix of the tictac_final dataset with 10% trained as much data

The matrix above is a trained tictac_final dataset which used 10% of data. We got the ‘5’ for the best parameter C. And the test accuracy was 71.4%.

So we can now see the difference between the trained model only with 10% of data and with the regular amounts of the data. The biggest difference was the accuracy, which decreased from 98.6% to 71.4%.

Linear Regression Regressor

For linear regression regressor, we used 10 folds and squared loss function (MSE). We produced 9 vectors with for loop to get total accuracy score. We also trained and tested the MSE. We got 26.2% of accuracy when we trained, 26.5% after we tested. And the final accuracy score for linear regression regressor was 72.48%

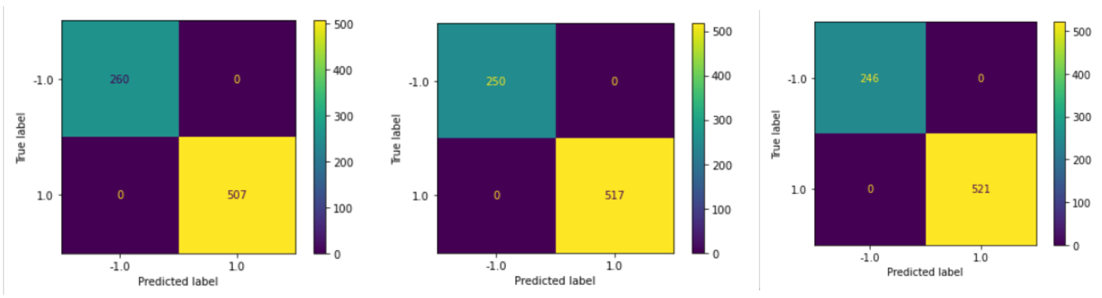
Multilayer Perceptron Classifier

In the code, we first set a set of parameters, param_setting. And set various parameters in it to bring into the generation function of the classifier and regressor. This includes the size of the hidden layer we want to use, the method of activation function (such as "relu" or "sigmod"), and the adjustment of the learning rate. What's more, we can set up many different sizes of hidden layers, and activation functions, and let the subsequent code show the way with the highest score.

In this classification model, I used n_splits = 10 and random_state =10 to generate my K-fold. And I tried to use different hidden layer sizes to train data, and collect the result each time. The following table and picture show the performance of MLP classifiers and all of them are based on the final.txt dataset.

hidden_layer_size	accuarcy
(150,100,50,10)	97.30%
(500,250,125,50)	98.10%
(200,50,15,3)	98.04%

The results of accuracy when the classifier has different hidden layer

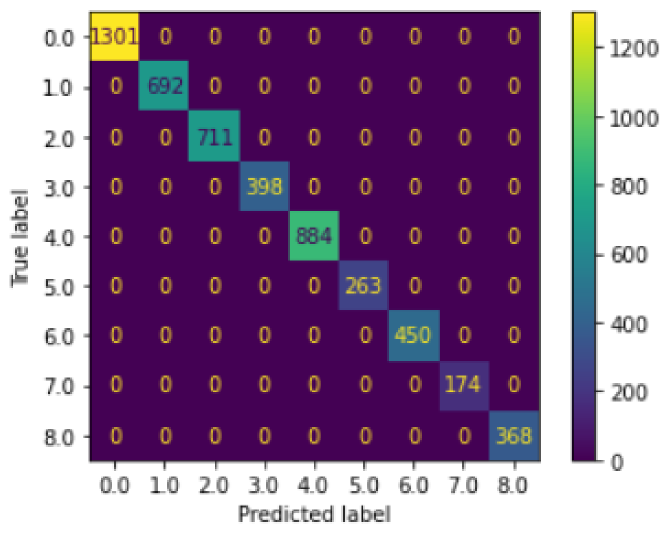


Confusion matrix with different hidden layers,
left to right [(150,100,50,10), (500,250,125,50), (200,50,15,3)]

In order to better test the processing of the mlp classifier on different datasets, I used 10% of the final.txt dataset and the single.txt dataset for separate tests. The test results are as follows:

hidden_layer_size	accuarcy
(150,100,50,10)	76.43%
(500,250,125,50)	71.56%
(200,50,15,3)	78.21%

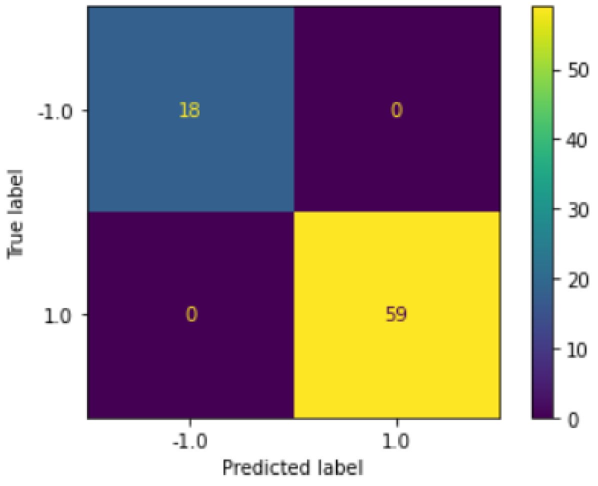
The results of accuracy when the classifier has different hidden layer, and all of three base on single.txt dataset



Confusion matrix bases on single.txt data set (hidden layer (150,100,50,10))

So, the accuracy is only 71.4%, when MLP classifier use 10% size of final.txt dataset

```
Test accuracy: 0.7142857142857143
Test accuracy with normalize flase:
 55
Confusion Matrix is:
[[0.43333333 0.36170213]
 [0.16666667 0.89361702]]
```



Confusion matrix bases on 10% of final.txt data set (hidden layer (150,100,50,10))

Multilayer Perceptron Regressor

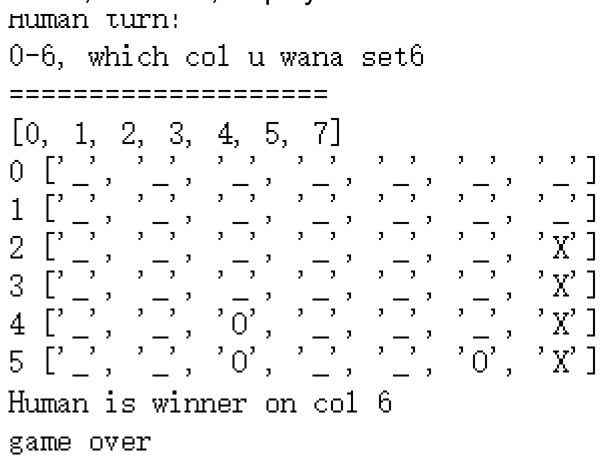
Like the MLP classifier, because the same MLP initialization function is used, the regressor can also have the function of adjusting the parameter settings.
For the test accuracy is 90.1%, and here is a table compare will other 2 regressor

regressor	accuarcy
MLP	90.16%
Linear Reg	72.48%
KNN	78.09%

ConnectFour Game play

The game code is in the humanGame.ipynb. Due to the principle of operation, it is difficult to predict the next movement position of the AI with other regressors. Which means AI uses the MLP regressor, because it has the highest performance in multi.txt dataset. So we try to use it for AI to predict and what's the next position to move.

The code will ask the user who will go first, AI or Human(if user chooses AI, computer go first, otherwise user go first). And then prepare a game board. Each round will record the positions entered by both parties, and display all the vacancies and existing pieces on the chessboard.
Similarly, the code will also check the chessboard every round to determine whether there is a victory or defeat, and if so, display the result and end the operation.



```
Human turn!
0-6, which col u wana set6
====================
[0, 1, 2, 3, 4, 5, 7]
0 ['_', '_', '_', '_', '_', '_', '_']
1 ['_', '_', '_', '_', '_', '_', '_']
2 ['_', '_', '_', '_', '_', '_', 'X']
3 ['_', '_', '_', '_', '_', '_', 'X']
4 ['_', '_', 'O', '_', '_', '_', 'X']
5 ['_', '_', 'O', '_', '_', 'O', 'X']
Human is winner on col 6
game over
```

Screen of connectFour game

What's more, due to the limited data set and our limited capabilities, the performance of AI is not very strong, and users can still easily beat the computer.

Conlusion

In this assignment, we distributed tasks reasonably, each of us wrote a classifier and regressor, and actively participated in group meetings, either through interviews or through Discord.
In general, it's possible that we tried too few times and that we didn't get the highest performance results, thus making the AI in the game "smarter". But each of us has learned the principles of various models, which means a lot to us.