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

Course Project Part I Simple Learning Models: Classifiers and Regressors
DUE: October 1st, 11:59 pm

Overview | Details | Resources | Getting Help | Submitting | Acknowledgements

Overview

Collaboration Rules: You may complete this project individually, or in groups of two or three. You may discuss concepts between groups, but **each group must submit their own original code**.

In this project, you will evaluate the performance of classifiers and regressors in simple two player game scenarios. Your single classifier program will generate statistical accuracy and confusion matrices for several classifiers and regressors (linear regression, linear SVM, k-nearest neighbors, and multilayer perceptron) using provided datasets. Each game play session will be synthetically generated. The two players in your games are both AI players. Your program will learn from these games to produce reasonable player moves for one of the players in the game.

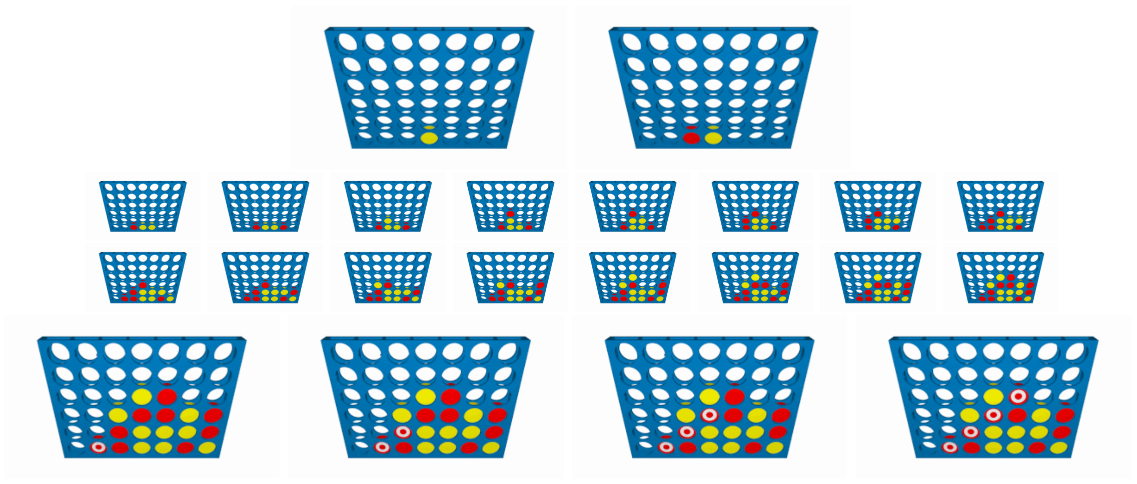

Figure 1: Connect Four. Original GIF Image Source: Wikipedia:Silver Spoon
Click here to see animation of the full game sequence.

Details

Set Up Your Code Base: See courseprojectpartI-docs.pdf for files referenced in this doc. Use HiPerGator or your personal computer. Everyone in the course has access to NVIDIA GPUs (GPU programming is required later) and APIs like PyTorch which are pre-installed on HiPerGator. See software list here. Remember that your HiPerGator account and any data on our storage space **will be removed without notice on December 17th**.You can find out more about the 70 million $ AI partnership between UF and NVIDIA and the new AI NVIDIA DGX A100 SuperPod here. This project will be developed using Python and the scikit-learn module. If you are not using HiPerGator, although scikit-learn is supported on Windows, macOS and Linux, we recommend that windows users install Virtual Box and work with Python and scikit-learn in a Linux environment. You will find information for installing Python here, and useful *getting started* information on scikit-learn here. Use the latest development version of scikit learn found here, which you must build from the source code. This will enable you to use the multilayer perceptron classes in scikit-learn.

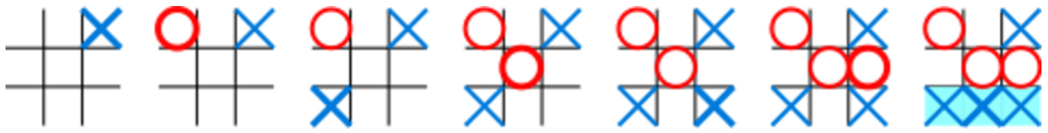

Figure 2: Tic Tac Toe. Image Source: Wikipedia:Traced by User:Stannered

Learn Game Rules: In this assignment you will evaluate the performance of a neural network on a simple game of *Connect Four* (Figure 1). Background reading and game rules can be found here. The Connect Four board you will use has 6 rows and 7 columns. There are $4,531,985,219,092$ positions for all game boards populated with 0 to 42 pieces. You will begin applying your learning models to a simpler case of a 3x3 *Tic Tac Toe* board for which the winning player must occupy three consecutive spots on the board (Figure 2). The game rules for *Tic Tac Toe* can be found here. Both games are categorized as zero-sum games and may be solved using the minimax algorithm described here. Minimax for Tic Tac Toe and Connect Four can be found here and here.

Examine Datasets: There are four datasets posted to canvas. In each case, there are two computer players; player X and player O.

Connect Four: This database contains all legal 8-ply positions in the game of Connect Four in which neither player has won yet, and in which the next move is not forced. There are 67557 instances. Attributes represent board positions on a 6×7 board. There are 42 attributes, each corresponding to one Connect Four square. Player X is the first player and player O is the second player. The outcome class is the game-theoretical value for the first player (2: win, 1: loss, 0: draw). Attribute Information: x=player x has taken, o=player o has taken, b=blank. Class Distributions: 44473 win(65.83%), 16635 loss(24.62%), 6449 draw(9.55%). The board states are shown in Figure 3.

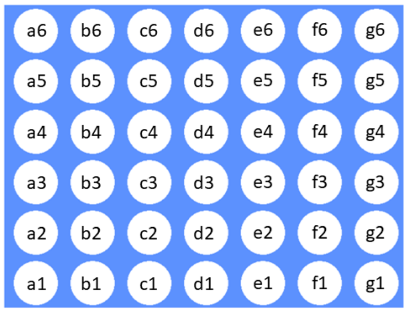

Figure 3: Connect Four States

In the following Tic Tac Toe datasets, player X is denoted as $+1$ and player O is denoted as -1. Empty squares are denoted as 0.

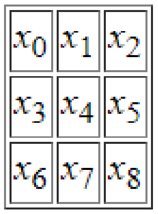

Figure 4: Tic Tac Toe States

Tic Tac Toe 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 $x_0, x_1 \cdots, x_8$, the states of each Tic Tac Toe square (Figure 4). The output feature y is the winning player. Each line of the input file lists the input features followed by the single output feature, as: $x_0, x_1 \cdots, x_8, y$. The dataset includes all 958 possible Tic Tac Toe game board configurations.

Tic Tac Toe 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 $x_0, x_1 \cdots, x_8$, 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).

Tic Tac Toe Intermediate boards optimal play (multi label): There are usually multiple optimal moves. This dataset can be viewed either as a regression or a classification problem with multiple output labels. Inputs are $x_0, x_1 \cdots, x_8$. Outputs are $y_0, y_1 \cdots, y_8$, where y_i is 1 if the given square is an optimal move for player O, otherwise 0. Each line lists x_i inputs followed by y_i outputs.

Classifiers: The players in the Tic Tac Toe games provided make optimal moves. Your goal is to learn from these games to produce reasonable moves for player O (-1). You can trivially load datasets into Python by loading the data matrix using the numpy loadtxt function, and then selecting out columns from it.

Write a program that outputs the statistical accuracy and confusion matrices that record the performance of a classifier on two classification datasets: *final boards classification dataset* and *intermediate boards optimal play (single label)*.

Report performance for three classifiers: linear SVM, k-nearest neighbors, and multilayer perceptron. Links to the relevant classes in scikit-learn are included in this text. You can manually choose the number of neighbors k, and the architecture of your multilayer perceptron, although be aware this introduces some overfitting risk. Your implementation must use k-fold cross validation (using e.g. 10 folds). Note that none of the datasets are randomly shuffled so you should randomly shuffle the order of the dataset before using cross-validation. See lecture notes on *MLBasics* for details on confusion matrices. Normalize the confusion matrices so that each row sums to one.

Please be sure to use the specified datasets and classifiers as you will only be evaluated on the binary classification and multi-class classification problem for these two datasets and these three classifiers.

Regressors: Write a program that evaluates a regressor on the *intermediate boards optimal play (multi label) dataset*. Evaluate three regressors: k-nearest neighbors, linear regression and multilayer perceptron.

For linear regression, implement the solution yourself via normal equations. Note that one linear regressor can be trained independently for each output. For simplicity, you can use a squared loss function. For linear regression with intermediate play datasets, you can consider a nine vector output. You can produce 9 linear models, where each model regresses a given single output against all the inputs. When choosing the optimal move, you can compare the 9 continuous outputs of the linear models directly, and choose the move that has the highest output. For k-nearest neighbors and multi-layer perceptron, you can encode outputs as a 9 vector with zeros for all positions except the optimal move(s).

Since this is really a multi-label classification problem that is being treated as a regression problem, please 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. Use k-fold cross-validation also when assessing the accuracy for this part.

Please be sure to test the specified regressors on the specified dataset for this multi-label problem as you will only be evaluated on this dataset and these regressors.

Human Game Play: Now use the knowledge you have gained working with simple learning models to develop a command line (or web-based) Connect Four Game where a human can play against a machine learning model. Suppose the computer is given a current board. Find the output of the model that is highest for the given board, which is not already taken by another player, and make the computer move correspond to the choice most preferred by the model. Use the provided Connect Four dataset for your game. You are free to use any neural network architecture of your choosing from scikit-learn (even ones not explored in this exercise). By now, we have covered neural networks in class.

Your report must address the items listed below. All explanations should discuss performance in terms of the network architecture and algorithm. Superficial answers will not count.

Evaluation on Tic Tac Toe Boards:

Record the accuracy and normalized confusion matrices for the 3 classifiers, and the accuracy for the 3 regressors from the tests above.

Explain which method worked best for classification and why?

Explain which method worked best for regression and why?

Investigate (and report) what happens to the accuracy of the classifiers if they are trained on $\frac{1}{10}$ as much data.

Explain why certain methods scale better to larger datasets than the others. Hint: the multilayer perceptron should work better than k-nearest neighbors regressors, but they both should have above **80**% accuracy, and should play a decent game of Tic Tac Toe in the next step.

Evaluation on Connect Four Data:

Explain the network architecture you chose for your Connect Four game and why. Provide a brief description of how your model works. Explain whether you can beat the computer at Connect Four, and how hard this is for your chosen model.

Explain your code and run it for us: Make a 5-7 minute video screen tour walking us through your code, explaining what you did in you own words. Run your code and show us the program generating results at the command line or web interface. We will suggest programs for doing this in canvas. You may however use any program you like. These are only suggestions.

In your video you must show us how to navigate your code:

- general structure
- classifier and regressor implementations
- normal equations
- loss functions
- human game play implementation

Optional extra credit: The above machine learning setup requires an expert human or computer player to observe and learn in a supervised manner from. You can explore the use of alternative techniques such as genetic algorithms.

Grading: You will be evaluated based on the following criteria (1) source code correctness and completeness 30%, (2) results from classifier and regressor evaluation (your written evaluation, and our output running your program) 30%, (3) the performance of your human gameplay program, your assessment of your ability to beat the computer and your assessment of your model 20% (4) video screen tour of your code 10% (5) written report (all other items not scored in 2 or 3) 10%.

Resources

Instruction documents referenced in this assignment will be posted to canvas.

All students in the class will have access to the HiPerGator.

If you are not in the CISE department, and would also like computer lab access, you can register for a CISE account at https://www.cise.ufl.edu/help/account. Linux systems are available for remote access top complete this assignment. **Remember to back-up your work regularly!!!** Use version control to store your work. DO NOT PUBLISIZE SOLUTIONS.

Getting Help

Send an email to me (ctoler@cise.ufl.edu) or the TA (khasnis.s@ufl.edu) for any issues or questions.

Office Hours are conducted in Zoom. Zoom links and available hours are posted on canvas.

Collaborating You may discuss this assignment with each other but each group should submit individual work. Remember to always credit outside sources you use in your code. University policies on academic integrity must be followed.

Submitting

Only one submission is required per-group. Non submitting group members should add the list of group members as a note in the canvas assignment. Upload one zip file to Canvas. If your zip file is too large for a canvas upload, you must submit your written report to canvas by the due date and upload your zip file to a file share (e.g. ONEDRIVE at UF) and email the link to the instructor by the due date. All submissions (Canvas or otherwise) must be completely uploaded by the due date and time. This assignment is subject to the late policy for course assignments.

Your submission should include everything needed to test, run and understand your code including:

- The complete source code including any third party libraries not available in the main HiPerGator distributions.
- A program that automatically prints the required classifier and regressor results.
- A program that automatically implements the *Connect Four* gameplay with a human.
- Any other input needed to run your code that is not mentioned in this list. For example some programs need images icons or other support data to run out-of-the-box.
- A written report that includes:
  - An explanation of your implementation.
  - Evaluation results requested in the assignment
  - Instructions on how to run your programs. Your code will be tested on all the provided datasets using scripts.
  - Anything you would like us to know (bugs, difficulties).
- 5-7 minute video screen tour walking us through your code and explaining your implementation. It should be located in a folder in your zip called videotour. You should explain your code and step through the key parts of your code. You also need to run the system so that we can see the results being generated. Program options will be posted to canvas but you may use any program you prefer.

Acknowledgements

Sources and Datasets:
UCI Machine Learning Repository
Connelly Barnes, UCI
David W. Aha