Practice One

Stephen Griffin: 100559805
Corey Gross: 100559804

Phase One

For the first phase, we used similar attributes to those in the third lab. They consisted of:

- Position of the snake's head (x and y)
- Position of the fruit (x and y)
- Length of the snake
- Delta x and y of snake to fruit
- Distance between the snake and the fruit (Euclidean distance)
- Direction the snake is facing

Other than the direction of the snake, all of these values are naturally numeric. In order to provide the model with better data, we labeled each cardinal direction with a number.

- UP = 0
- DOWN = 1
- LEFT = 2
- RIGHT = 3

Phase Two

After many attempts at determining which features to include and which not to include, we finally decided. Using a boolean to determine if a move was safe in a particular direction led to a massive overfit of the data. The model decided that since there was a safe move in a particular direction, it would travel the entire distance to the wall, and then crash without changing direction. Originally, we had used Manhattan Distance to the fruit, with the x and y coordinates of both the snake head and the fruit. We also included the length. With this data, we were struggling to find a classification that would fit the data above a percentage of about 50. Once we added the delta x and y along with the Euclidean Distance, the model increased efficiency. Then, we decided that our data was not strong enough to create a good model. We went back and replayed the snake game at a much lower difficulty setting. The training data set has two games, each with a score of about 8000 and 5000 respectively. The test data set has one game, with a score of about 5000. However, due to the way the game was played, there is about a 80-20 percent split between the two data files. Once the game data was significantly better, the model became better as well.

Classifier	Correct	MAE	MSE	Build Time (s)	Precision	Recall	F-Measure
J48	82.90%	0.1088	0.2787	0.08	0.830	0.829	0.829
Rand.Fst.	83.89%	0.1472	0.2691	1.21	0.841	0.839	0.839
MultiPerc.	85.45%	0.0894	0.2659	6.21	0.858	0.855	0.854
NaiveBay.	67.95%	0.1998	0.3399	0.01	0.725	0.680	0.679
IBk-10	62.44%	0.2056	0.3852	0	0.628	0.624	0.625

Overall, we can see that Multilayer Perceptron performed the best in almost every category. However, with only three games played in the entire data set (training and testing), it still took a build time of 6.21 seconds. If our goal is to minimize the amount of time for a larger data set, we might decide to choose J48 or Random Forests, as they perform similarly in most metrics. Random Forests performs slightly better than J48, but takes a decent amount of time longer as well.

Phase Three

The results obtained from the agent compared to the keyboard/bfs model is significantly worse. On average, the agent can correctly play the snake game up to about an average of 1.5 thousand points. It accurately tracks down fruits and plays pretty well, however, there is an issue within the snake game that is causing issues for the agent. When changing direction twice quickly, sometimes the snake game registers the two movements as one, going straight back into itself. This happens far more often in the agent than it does in the keyboard or bfs model. This is an error in the game rather than the agent, yet it prevails. While playing the game, the agent plays similarly to the bfs, but this is by design. When recollecting my data the second time around, I attempted to play in a more deterministic manner, as in the direction I would take to the fruit. Instead of rapidly changing direction to get a fruit that is one or two frames off, I would try to line up the move on the first turn. If not, I waited until I was in line with the fruit, and then changed direction once I arrived.

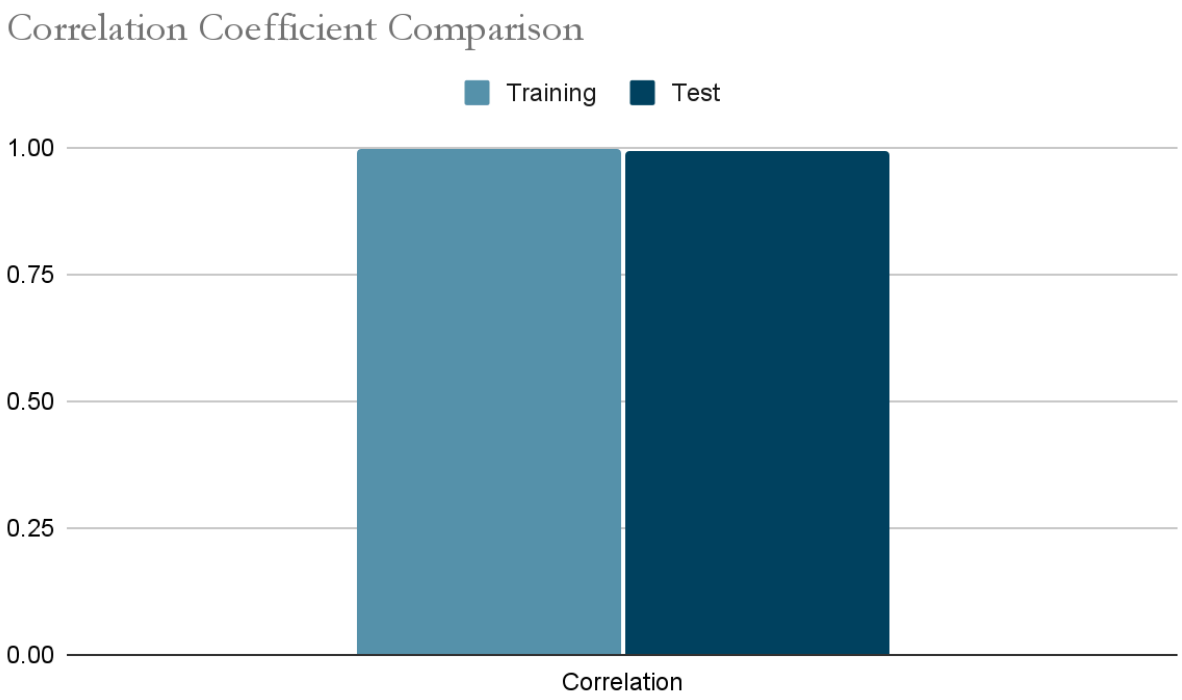
Phase Four

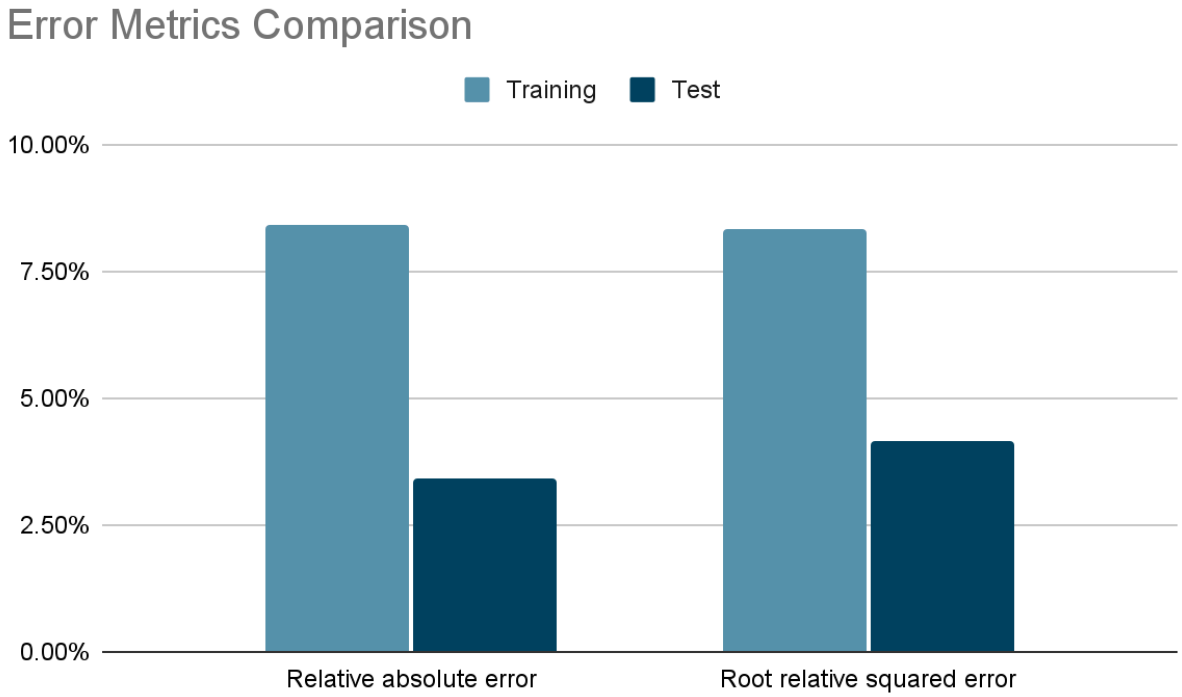
When recollecting the data for the fourth phase, I took a similar approach as I did in my second/third attempt in the third phase. I played the game in a more deterministic manner for the training data. The goal of phase four was to generate data from both the keyboard and the agent created from phase three. In my case, I gathered all of the training data from the keyboard, and

all of the test data from the agent. This gives more variability between the two sets of data. This time, the attributes we used is as follows:

- distance_to_fruit
- snake_length
- curr_score
- ticks_since_last_score
- next_score (which we are predicting)

M5 Regression





As we can see in the correlation coefficients and the error metrics, the test performs better for error metrics, and similarly in correlation. This means that the linear regression performed with M5 is great for the generalization of data, and is not overfitting.

Additive Regression with Decision Stump

	Training	**Test**
Correlation coefficient	0.9821	0.8212
Relative absolute error	17.4577 %	9.0527 %
Root relative squared error	19.0011 %	10.5355 %
Mean absolute error	254.6718	207.9649
Root mean squared error	325.8931	246.018
Instances	8644	1981

With Additive Regression, the linear regression performs slightly worse for correlation on the test set. However, the RAE and RRSE perform better than the training set. This means that the data is not generalized as well for the Additive Regression. It captures training patterns effectively but loses some predictive accuracy, meaning that it might be very slightly overfit for

this model. Interestingly, we see the test set having lower errors than the training set. This indicates that the training set might have more complex predictions and patterns. Luckily enough, the lower errors suggest that the data might not be as overfit as we might think, suggesting that the drop in correlation could be due to complexity differences between the two sets of data. Overall, we definitely prefer the M5 model for linear regression in the case of our data.

Questions

1. What is the difference between learning these models with instances coming from a human-controlled agent and an automatic one?

The difference between learning the models with instances coming from a human compared to those coming from an automatic agent is that humans are harder to predict. Data needs to be captured in a deterministic way so the algorithm can correctly predict/correlate.

2. If you wanted to transform the regression task into classification, what would you have to do? What do you think could be the practical application of predicting the score?

If we wanted to transform the regression task into classification, we would need to do a few things. First, we would define meaningful categories like score increases slightly/significantly or decreases slightly/significantly. Then, we would change the numeric attribute into categorical classes that represent when score slightly or drastically changes. Finally, we would have to train a classification model in Weka instead of a regression model. Some practical applications of predicting the score would be a version of the game where difficult is adaptable. For example, imagine a game that if a model predicts a significant increase in score, the game proactively makes the next fruit harder to reach. Another case would be for player performance monitoring. Predicting score can help identify a player's trajectory, which would be useful for personal recommendations or feedback for improvement. This would also help game developers use these models to ensure balanced game mechanics (fruit placement, scoring system, etc.).

3. What are the advantages of predicting the score over classifying the action? Justify your response.

Some advantages of predicting the score over classifying the action would be better information about the game state, strategic decision making, and continuous feedback or adaptation. Predicting the score gives direct information about the consequences or rewards of state/actions. Action prediction would only predict which immediate direction is best, without quantifying the consequences of its actions. Also, scenarios where feedback is necessary, score prediction is better. Action classification is categorical, and less nuanced as a result. It tells you only the "best" move, not how much better or worse alternatives may be. Score prediction is extremely valuable for incremental learning and adaptation.

4. Do you think some improvement in the ranking could be achieved by incorporating an attribute that would indicate whether the score at the current time has dropped?

Yes, some improvement in the ranking could be achieved by incorporating an attribute that would indicate whether the score at the current time has dropped. It would directly reflect if a risky move has been made, indicating that the snake made a suboptimal move or increased risk. This attribute would also capture the stability of gameplay, which is important for score prediction. Finally, it would help the model differentiate complex patterns, improving predictive accuracy.

Conclusions

Linear regression provided highly accurate and robust predictions with high correlation and low error. Additive regression performed well also, but slightly worse. Attribute selection proved to be extremely beneficial to the correctness of predictions, increasing the accuracy by nearly 30-40%. Proper data preprocessing was the large issue at the beginning of the practice, as well as data collection. When we played in a more deterministic manner, the agent predictions became far more accurate, and was actually able to play the game for a couple thousand ticks at a time (accurately). The obtained model would be most useful for player analytics and adaptive difficulty if the game developers decided to pursue so. It would also be useful for creating smarter, more challenging opponents or teammates in a cooperative version of the game. Machine learning can be applied in a range of scenarios outside of snake. For example, it can be used for financial predictions, healthcare analytics, or even predictive maintenance for machinery based on sensor data. The largest issue that we encountered during the agent creation for snake was data collection. Original sets of data were trained without much thought, and gameplay was not deterministic. Directions were chosen randomly, just attempting to get to the fruit without regard for how it could be perceived in classification. Once we reattempted gameplay and generated data that was more straightforward, agent creation improved significantly. One issue with the practice that we encountered was the snake game itself. Many of the times the snake dies is due to a bug in the game that allows the user to turn into itself on one move. If two subsequent moves are fast enough, the game registers only the second direction, and the snake dies while it looks like it is moving forwards. If the snake game was created without pygame, this might not be an issue. Just some criticism for future endeavors.