Assignment I

Machine Learning I
Data Science and Engineering

April 2024

Salma Alejandra Caisaguano Barzallo - 100496504
Raquel Jaén Delgado - 100496615

Universidad Carlos III de Madrid

Contents

1 Introduction

This project involves leveraging Machine Learning methodologies to develop predictive and classification models for the Snake game, utilizing Weka. Furthermore, it aims to construct an autonomous agent capable of controlling the snake based on the models generated.

2 Phase I: Instances collection

Feature Extraction Function

The feature extraction function in this code focuses on extracting relevant information from the game state.

```
def print_line_data(game):
    line_data = [
        DIFFICULTY,
        game.snake_pos[0],
        game.snake_pos[1],
        len(game.snake_body),
        game.food_pos[0],
        game.food_pos[1],
        game.food_eaten,
        food_eaten_count,
        game.old_direction,
        game.direction,
        game.region_direction[0],
        game.region_direction[1],
        game.obstacles,
        game.prev_score,
        game.score,
        game.end_reason,
        gameOver
    ]

    return line_data
```

The extracted features include:

- Difficulty level.
- Snake's head position (X, Y coordinates).
- Length of the snake's body.
- Position of the food (X, Y coordinates).

- Whether food has been eaten or not.
- Count of eaten foods.
- Direction of the snake (current and previous).
- Direction of the snake with respect to a central region (head and tail).
- Direction of the nearest obstacle.
- Current score and previous score.
- The cause of game over.
- Boolean value indicating whether the game is over or not.

Mechanism to Include Future Data

```
file_name = 'GameState.arff'
    if not os.path.isfile(file_name):
        with open(file_name, "w", newline='') as file:
            # Write header
            file.write('@relation GameState \n\n')
            # Write attributes
            attributes = [
                ('Difficulty', 'NUMERIC'),
                ('Head_X', "NUMERIC"),
                ('Head_Y', "NUMERIC"),
                ('Snake_Body_Length', "NUMERIC"),
                ('Food_Position_X', "NUMERIC"),
                ('Food_Position_Y', "NUMERIC"),
                ('Food_Eaten', '{True, False}'),
                ('Eaten_Food_Count', "NUMERIC"),
                ('Old_Direction', "{UP, DOWN, LEFT, RIGHT}"),
                ('Direction', "{UP, DOWN, LEFT, RIGHT}"),
                ('Head_Region_Direction', '{NORTH-EAST, NORTH-WEST, SOUTH-EAST,
                    SOUTH-WEST}'),
                ('Tail_Region_Direction', '{NORTH-EAST, NORTH-WEST, SOUTH-EAST,
                    SOUTH-WEST}'),
                ('Nearest_Obstacle', "{UP, DOWN, LEFT, RIGHT}"),
```

```
            ('Score', "NUMERIC"),
            ('Next_Score', 'NUMERIC'),
            ('Cause_Death', '{None, Body_Touched, Wall_Touched}'),
            ('OVER', '{True, False}}')
        ]
        for attribute in attributes:
            file.write('@attribute {} {}\n'.format(attribute[0], attribute[1]))
        file.write('\n@DATA\n')
else:
    with open(file_name, "a", newline='') as file:
        # Write data
        data = ','.join(map(str, print_line_data(game)))
        file.write(data + '\n')
```

The mechanism to include future data involves writing the current game state to a file in ARFF after each game iteration. It initializes a file named 'GameState.arff' if it does not exist and writes the ARFF header and attributes. Then, it appends the data of the current game state to the file. This mechanism allows for storing the game state data along with relevant attributes for each game instance. Therefore, future data can be included by adding more game instances to the same file.

3 Phase II: Classification

In this section, we focus on building a classification model to help our snake make decisions during gameplay, specifically deciding which direction to move in order to reach the food. We will start by cleaning up our data and selecting the most relevant attributes. Then, we will try out different classification algorithms to see which one works best for our snake. Finally, we will evaluate the performance of each model and compare them to find the most effective one for our game.

1. Preprocessing

Attributes concerning future events, like the score in the next tick, cannot be utilized as input for the classifier because they are not accessible when the snake must decide its next action. Therefore, we will remove the `next_score` attribute.

After removing the attribute `next_score`, we applied the `InfoGainAttributeEval` filter in

`Weka` to determine the importance of each attribute in predicting the snake's movement direction.

The `InfoGainAttributeEval` result provides a ranking of attributes based on their importance in predicting the target variable (in this case, the direction of the snake player's movement).

Rank	Attribute	Information Gain
1	Nearest_Obstacle	1.439601
2	Old_Direction	1.324535
3	Score	0.170569
4	Food_Position_Y	0.164937
5	Food_Position_X	0.145327
6	Head_Y	0.116241
7	Tail_Region_Direction	0.115306
8	Head_X	0.111579
9	Eaten_Food_Count	0.061451
10	Snake_Body_Length	0.061451
11	Head_Region_Direction	0.00216
12	Food_Eaten	0.000324
13	OVER	0.000248
14	Cause_Death	0.000212
15	Difficulty	0

Table 1: InfoGainAttributeEval Result

Upon examination of the results, it is evident that the `Difficulty` attribute has an Information Gain value of 0, suggesting it provides no relevant information for predicting the target variable. Therefore, we will remove it as it does not contribute to the predictive power of our classification model.

Additionally, considering the low Information Gain values of the last two attributes (`Cause_Death` and `OVER`), which are close to zero, there is indication that these attributes may not significantly contribute to predicting the snake's movement direction. Hence, it is reasonable to remove these attributes.

At the end we keep the following attributes:

Rank	Attribute	Information Gain
1	Nearest_Obstacle	1.439601
2	Old_Direction	1.324535
3	Score	0.170569
4	Food_Position_Y	0.164937
5	Food_Position_X	0.145327
6	Head_Y	0.116241
7	Tail_Region_Direction	0.115306
8	Head_X	0.111579
9	Eaten_Food_Count	0.061451
10	Snake_Body_Length	0.061451
11	Head_Region_Direction	0.00216
12	Food_Eaten	0.000324

Table 2: Final Attribute Selection

We still have some variables with low Information Gain values, but it is reasonable to explore the impact of removing these attributes on model performance.

2. Classification Algorithms

For this analysis, we will be trying the following classification algorithms: `J48`, `Random Tree`, `ZeroR` and `PART`

2.1. `J48`

We are starting our exploration of classification algorithms with J48. Known for its simplicity and effectiveness, `J48`, based on the C4.5 algorithm, is a popular choice for decision-making tasks. Our goal is to see how well `J48` can help our snake find and eat food in the game.

Correctly Classified Instances	5006	93.2911	%
Incorrectly classified Instances	360	6.7089	%
Kappa statistic	0.9105		
Mean absolute error	0.0526		

Table 3: Results `J48` 12 Attributes Model

The `J48` classification model with 12 attributes demonstrates strong performance in predicting the snake's movement direction in our game environment.

We will now remove the attribute `Food_Eaten`, and run again the algorithm

Correctly Classified Instances	4999	93.1606	%
Incorrectly classified Instances	367	6.8394	%
Kappa statistic	0.9088		
Mean absolute error	0.0537		

Table 4: Results `J48` 11 Attributes Model

Overall, while removing the `Food_Eaten` attribute resulted in a marginal decrease in accuracy and increase in error rate, the model with 11 attributes still demonstrates strong performance in classifying the snake's movement direction. Therefore, the removal of the `Food_Eaten` attribute may be justified if it simplifies the model without significantly compromising its predictive capability.

Now, what would happen if we remove now the attribute `Head_Region_Direction`

Correctly Classified Instances	5000	93.1793	%
Incorrectly classified Instances	366	6.8207	%
Kappa statistic	0.909		
Mean absolute error	0.0531		

Table 5: Results `J48` 10 Attributes Model

Comparing these models, we observe that the 12 Attributes Model has the highest accuracy (93.29%) and Kappa statistic (0.9105), indicating its superior performance in correctly classifying instances and agreement beyond chance. However, the differences in accuracy and performance metrics between the 12 Attributes Model and the subsequent models with fewer attributes are minimal.

Therefore, while the 12 Attributes Model slightly outperforms the others, the practical significance of this difference may be negligible. Additionally, the models with fewer attributes offer a more simplified representation, potentially leading to easier interpretation and implementation.

In conclusion, considering both performance and simplicity, the 10 Attributes Model (after removing `Food_Eaten` and `Head_Region_Direction`) may be considered the best option. It retains a high level of accuracy and predictive capability while also offering a slightly more streamlined model compared to the 12 Attributes Model.

2.2. `Random Tree`

The `Random Tree` algorithm is a powerful tool used in machine learning for making decisions, especially in classification tasks. It is a member of the decision tree family, but it has a unique twist that sets it apart. Unlike traditional decision trees that consider all features at each decision point, `Random Tree` randomly selects a subset of features to make its decisions. This randomness helps prevent overfitting and encourages diversity among the trees in the ensemble. By combining the predictions of many such trees, `Random Tree` can provide accurate and robust classifications.

In our experimentation with the `Random Tree` algorithm, we will conduct three runs using different variations of the data. Firstly, we'll employ the original dataset as is, without any modifications. Secondly, we'll discretize the numerical variables to observe the impact of this transformation on model performance. Finally, we will add a new variable, `Distance_To_Food`, to see the effect that it may have in the model.

1. Original Dataset

Correctly Classified Instances	4463	83.1718	%
Incorrectly classified Instances	903	16.8282	%
Kappa statistic	0.7754		
Mean absolute error	0.0846		

Table 6: Results `Random Tree` Original Attributes Model

2. Discretized Dataset

Correctly Classified Instances	4807	89.5826	%
Incorrectly classified Instances	559	10.4174	%
Kappa statistic	0.8611		
Mean absolute error	0.0858		

Table 7: Results `Random Tree` Discretized Attributes Model

3. Add New Attribute; `Distance_To_Food`

Correctly Classified Instances	4373	81.4946	%
Incorrectly classified Instances	993	18.5054	%
Kappa statistic	0.7531		
Mean absolute error	0.0926		

Table 8: Results `Random Tree` New Attribute Model

In conclusion, our experiments involved three approaches: using the original data, discretizing numerical values, and introducing a new attribute related to the distance to food. We observed that discretizing numerical values significantly improved the model's performance, resulting in higher accuracy. However, the addition of the new attribute did not yield substantial improvements. This underscores the importance of thoughtful data preprocessing, such as discretization, to enhance the effectiveness of the Random Tree algorithm in making informed decisions.

Among these, the model trained on the discretized dataset proved to be the most successful, exhibiting higher accuracy compared to the other approaches. This underscores the significance of meticulous data preprocessing, such as discretization.

2.3. `ZeroR`

Below, we are going to test `ZeroR`, which predicts the mean (for a numeric class) or the mode (for a nominal class) by identifying the class that has been repeated the most times and assigning it as the default predicted value.

Correctly Classified Instances	1358	25.3075	%
Incorrectly classified Instances	4000	74.6925	%
Kappa statistic	0		
Mean absolute error	0.375		

Table 9: Results `ZeroR` 12 Attributes Model

Correctly Classified Instances	1358	25.3075	%
Incorrectly classified Instances	4000	74.6925	%
Kappa statistic	0		
Mean absolute error	0.375		

Table 10: Results `ZeroR` 11 Attributes Model

The results from our experimentation with `ZeroR` reveal its straightforward approach but limited predictive power. Regardless of the number of attributes considered, `ZeroR` tends to achieve low classification accuracy, as evidenced by the consistently low correct classification rates ranging around 25.31%. This underscores the importance of comparing more sophisticated models like Random Tree with simpler baselines to gauge the effectiveness and practicality of different approaches in classification tasks.

2.4. `PART`

The PART algorithm, which stands for "Partial Decision Trees," is a hybrid approach that combines elements of decision trees and rule-based learning. Unlike traditional decision trees that aim to create a complete tree structure, PART generates rules to classify instances in a dataset. It begins by constructing a decision tree, similar to algorithms like C4.5, but instead of fully expanding the tree, it stops and generates rules at each node based on the majority class. This hybrid approach allows PART to handle both numerical and categorical attributes efficiently while producing comprehensible rules that are easy to interpret.

Correctly Classified Instances	4857	90.5143	%
Incorrectly classified Instances	509	9.4857	%
Kappa statistic	0.8735		
Mean absolute error	0.0588		

Table 11: Results PART Original Attributes Model

Correctly Classified Instances	4888	91.0921	%
Incorrectly classified Instances	478	8.9079	%
Kappa statistic	0.8812		
Mean absolute error	0.0667		

Table 12: Results PART Discretized Attributes Model

Correctly Classified Instances	4841	90.2162	%
Incorrectly classified Instances	525	9.7838	%
Kappa statistic	0.8695		
Mean absolute error	0.0601		

Table 13: Results PART Custom Parameters Model (C=0.3, M=2)

Throughout the study, different parameter configurations were explored to understand their impact on the performance of the PART algorithm. The first two sets of results were obtained using the default parameters of the PART algorithm. Subsequently, the performance was evaluated after adjusting the parameters to better suit the characteristics of the dataset. For example, the third set of results was obtained by setting the confidence threshold ('C') to 0.3 and the minimum number of instances per leaf ('M') to 2. These adjustments were made based on experimentation and observation of how the algorithm performed under different configurations.

3. Comparison

In summary, the J48 Decision Tree with the 10 Attributes Model, Random Tree with the discretized dataset, and the customized PART algorithm stood out as the most effective models

for the classification task. These models exhibited high accuracy and predictive capability while maintaining a balance between model complexity and interpretability. Additionally, the significance of preprocessing techniques, such as feature selection and discretization, was highlighted in enhancing the performance of classification algorithms.

4 Phase III: Building an automatic agent

While we could not successfully carry out this task, we believed that the smart agent created with `Weka` had the potential to make superior predictions. This optimism stemmed from the fact that `Weka` relies on machine learning techniques that learn from data inputs, gradually enhancing their abilities. Essentially, this implies that the intelligent agent could adapt and improve over time as it engages in more games, refining its strategies along the way.

5 Phase IV: Prediction

5.1 Keyboard Dataset

The selection of relevant attributes is crucial for building accurate predictive models. In this analysis, we utilize the `CorrelationAttributeEval` feature selection method provided by `Weka`. This method measures the correlation between each attribute and the target variable, `Next_Score`, attributes with higher correlation values are considered to be more relevant for predicting the target variable, whereas attributes with lower or negative correlation values may have less predictive power.

The following table summarizes the results obtained from the CorrelationAttributeEval analysis:

Based on the analysis, attributes such as `Score`, `Eaten_Food_Count`, and `Snake_Body_Length` show strong positive correlations with the `Next_Score`. These attributes are likely to be the most relevant for predicting the score in the next tick. On the other hand, attributes like `Head_X` and `Food_Position_Y` have negative correlations with the `Next_Score`, indicating they may have little predictive power.

However, it is important to note that while correlation is a useful measure for feature selection, it may not always capture the complexity of relationships between variables and the target variable. Attributes with very low correlation coefficients or negative correlations may be candidates for fur-

Rank	Attribute	Correlation	Average Rank
1	Score	1.0	1.0
2	Eaten_Food_Count	0.986	2.3
3	Snake_Body_Length	0.986	2.7
4	Head_Y	0.025	4.5
5	Nearest_Obstacle	0.024	5.0
6	Direction	0.022	6.3
7	Head_Region_Direction	0.022	7.1
8	Old_Direction	0.022	7.1
9	Tail_Region_Direction	0.02	9.0
10	OVER	0.01	10.1
11	Cause_Death	0.01	10.9
12	Food_Eaten	0.005	12.0
13	Difficulty	0.0	13.0
14	Food_Position_X	-0.006	14.0
15	Food_Position_Y	-0.014	15.0
16	Head_X	-0.041	16.0

Table 14: Attribute Correlation with Next_Score

ther investigation or potential removal. In this analysis, attributes such as `Food_Position_X`, `Food_Position_Y`, and `Head_X` have negative correlations with `Next_Score`, indicating they may have little predictive power for the target variable. Additionally, attributes like `Difficulty`, `Food_Eaten`, `OVER`, and `Cause_Death` also show low correlations `Next_Score`.

After identifying the most relevant attributes for predicting the `Next_Score` through correlation analysis, the next step is to build regression models using various algorithms. This section explores the performance of different regression techniques in predicting the score in the next tick. The goal is to evaluate how well each algorithm generalizes from the training data to unseen data and to determine which algorithm yields the best predictive performance. The following regression algorithms will be considered in this analysis:Linear Regression Random Tree Support Vector Regression (SVR)

`Linear Regression`

We will analyse the application of linear regression in building predictive models for the `Next_Score`.

Based on the output from the Linear Regression algorithm:

The model produced a prediction equation for the `Next_Score` variable, taking into account the coefficients of the relevant attributes. Notably, the attributes with non-zero coefficients in the model are `Score` and `Eaten_Food_Count`.

$$\texttt{Next\_Score} = 1 \times \texttt{Score} + 0.0073 \times \texttt{Eaten\_Food\_Count} + 100.022$$

The model's performance on the test set was exceptional, achieving a perfect correlation coefficient of 1. Additionally, both the mean absolute error and root mean squared error were 0, indicating that the model's predictions perfectly matched the actual values in the test set. This indicates that the linear regression model was able to accurately capture the relationship between the selected attributes and the `Next_Score` variable.

A perfect fit in a regression model, where the correlation coefficient is 1 and the mean absolute error and root mean squared error are both 0, is unusual and could indicate potential issues with the model. This suggests that the model is likely overfitting the training data, meaning it is capturing noise rather than the underlying relationship between the variables.

Additionally, to address the unusual occurrence of a perfect fit in the linear regression model, we will explore the effect of variable deletion on model performance.

We will delete the attributes `Head_X` and `Food_Position_Y` and `Difficulty`.

The model still produced a prediction equation for the `Next`$_{\texttt{S}}$`core` variable, considering the coefficients of the relevant attributes. Notably, the attributes with non-zero coefficients in the model are `Score` and `Eaten_Food_Count`.

$$\texttt{Next\_Score} = 1 \times \texttt{Score} + 0.0028 \times \texttt{Eaten\_Food\_Count} + 100.0084$$

Similar to the previous model, the performance metrics on the test set remained exceptional, with a perfect correlation coefficient of 1 and mean absolute error and root mean squared error both equal to 0.

While deleting attributes such as `Head_X`, `Food_Position_Y`, and `Difficulty` did not significantly affect the model's performance metrics, it is essential to interpret these results cautiously.

Random Tree

To begin our analysis, we will train a decision tree model using the dataset and evaluate its performance on the test set.

Correlation Coefficient	MAE	RMSE	Relative Abs. Error (%)
0.9795	74.2648	151.5995	11.5545

Table 15: Summary of Random Tree Algorithm Results

The Random Tree algorithm, applied to predict the Next_Score variable, exhibits a notable correlation coefficient of 0.9795, indicating a strong positive correlation between predicted and actual values. However, the model's performance metrics suggest room for improvement.

We will now run the algorithm with the dataset where the attributes Head_X and Food_Position_Y and Difficulty were deleted.

Correlation Coefficient	MAE	RMSE	Relative Abs. Error (%)
0.9935	57.0736	89.3726	8.8798

Table 16: Summary of Algorithm Results after Deleting Attributes

In summary, deleting the attributes Head_X, Food_Position_Y, and Difficulty resulted in an overall improvement in the RandomTree algorithm's performance metrics. The correlation coefficient increased, indicating a stronger relationship between predicted and actual values, while the MAE, RMSE, and relative absolute error decreased, suggesting a reduction in prediction errors.

Support Vector Regression (SVR)

We will analyze the application of Support Vector Regression (SVR) in building predictive models for the Next_Score. SVR works by finding the hyperplane that best fits the training data while minimizing prediction errors. This technique is particularly effective in capturing complex relationships between variables and the target variable.

Overall, these results suggest that the SVR model has effectively captured the underlying patterns in the data and can make highly accurate predictions for the Next_Score variable.

Correlation Coefficient	MAE	RMSE	Relative Abs. Error (%)
1.0	1.551	1.9834	0.2413

Table 17: Summary of `SVR` Algorithm Results

However, as with any model, it's essential to validate its performance on unseen data and consider potential sources of bias or overfitting.

We will now run the algorithm with the dataset where the attributes `Head_X` and `Food_Position_Y` and `Difficulty` were deleted.

Correlation Coefficient	MAE	RMSE	Relative Abs. Error (%)
1.0	1.245	1.5281	0.1937

Table 18: Summary of `SVR` Algorithm Results after Deleting Attributes

These results indicate that even after deleting the attributes `Head_X` and `Food_Position_Y` and `Difficulty`, the `SVR` algorithm still achieved a perfect correlation coefficient of 1. The mean absolute error (MAE) and root mean squared error (RMSE) decreased slightly compared to the results without attribute deletion, indicating a slight improvement in prediction accuracy. Overall, the `SVR` algorithm performed exceptionally well even after removing these attributes from the dataset.

In the first part of this phase, we focused on building predictive models for the `Next_Score` variable, essential for our automatic agent. Attribute selection highlighted `Score`, `Eaten_Food_Count`, and `Snake_Body_Length` as significant predictors.

Regression algorithms were applied, including Linear Regression, Random Tree, and Support Vector Regression (`SVR`). Linear Regression exhibited a perfect correlation coefficient, suggestive of potential overfitting. Random Tree showed promise, with notable improvements after removing less relevant attributes. `SVR` demonstrated exceptional performance, maintaining accuracy even after attribute deletion.

5.2 Intelligent Agent Dataset

13. Attribute selection

Rank	Attribute	Correlation	Average Rank
1	Score	0.99988	1.0
2	Eaten_Food_Count	0.94991	2.3
3	Snake_Body_Length	0.94991	2.7
4	Difficulty	0.27904	4.5
5	Old_Direction	0.2011	5.0
6	Nearest_Obstacle	0.048	6.3
7	Head_X	0.04461	7.1
8	Direction	0.04136	7.1
9	OVER	0.02269	9.0
10	Cause_Death	0.02269	10.1
11	Head_Region_Direction	0.02244	10.9
12	Tail_Region_Direction	0.02104	12.0
13	Food_Position_X	0.02038	13.0
14	Food_Position_Y	0.01225	14.0
15	Food_Eaten	0.00814	15.0
16	Head_Y	-0.06739	16.0

Table 19: Attribute Correlation with Next_Score

As it is observed in the table `Score`, `Eaten_Food_Count`, and `Snake_Body_Length` have a very high correlation with `Next_Score` (close to 1), indicating that these attributes are strongly related to the next score in the game. Specifically, `Score` has the highest correlation (0.99988), which makes sense as the score generally increases as the snake eats more food and its body lengthens.

`Difficulty`, `Old_Direction`, and `Nearest_Obstacle` have a positive but much lower correlation with `Next_Score`. This suggests that these attributes may also influence the next score, but their impact is less direct or consistent.

`Head_X`, `Direction`, `OVER`, `Cause_Death`, `Head_Region_Direction`, `Tail_Region_Direction`, `Food_Position_X`, `Food_Position_Y`, and `Food_Eaten` have even lower correlations, indicating that these attributes have little to no direct impact on the next score.

Surprisingly, `Head_Y` has a negative correlation with `Next_Score`, though it's small (-0.06739). This could suggest that moving the snake's head in the Y direction (up or down) might have a slight negative impact on the next score, though more analysis would be needed to confirm this.

2. Regression Algorithms

Once the key attributes for forecasting the `Next_Score` have been identified through correlation analysis, the next stage involves constructing regression models. For this part we are going to consider the algorithms used with the previous dataset.

`Linear Regression`

The model produced a prediction equation for the `Next_Score` variable, taking into account the coefficients of the relevant attributes.

$$\begin{aligned}\texttt{Next\_Score} = {} & 0.0593 \times \texttt{Difficulty} + 0.7987 \times \texttt{Snake\_Body\_Length} \\ & + 0.7382 \times \texttt{Eaten\_Food\_Count} + 3.8676 \times \texttt{Old\_Direction} = \texttt{DOWN}, \texttt{UP}, \texttt{RIGHT} \\ & - 4.5557 \times \texttt{Old\_Direction} = \texttt{RIGHT} - 4.2051 \times \texttt{Direction} = \texttt{DOWN}, \texttt{UP} \\ & - 1.7499 \times \texttt{Nearest\_Obstacle} = \texttt{UP}, \texttt{DOWN} + 0.9758 \times \texttt{Score} \\ & - 67.7206 \times \texttt{Cause\_Death} = \texttt{None}, \texttt{Body\_Touched} \\ & + 70.8914 \times \texttt{Cause\_Death} = \texttt{Body\_Touched} - 71.3948 \times \texttt{OVER} = \texttt{True} + 64.9255\end{aligned}$$

The model's performance on the test set was pretty similar to its homologous done with the keyboard dataset. It obtained a perfect correlation coefficient of 1 as well as zero-mean absolute error and root squared error, which shows that the perfect alignment between the model's predictions and the actual values in the test set.

This elevated degree of precision might indicate that the model is overfitting the training data, which could lead to problems when forecasting future data.

In order to tackle this issue, we will investigate the impact of removing variables on the performance of the model.

This is the produced prediction equation for `Next_Score` after deleting the attributes `Food_Position_Y` and `Food_Eaten` and `Head_Y`.

$$\begin{aligned}\texttt{Next\_Score} = {} & 0.0208 \times \texttt{Snake\_Body\_Length} + 101 \times \texttt{Food\_Eaten} = \texttt{True} \\ & - 0.0208 \times \texttt{Eaten\_Food\_Count} + \texttt{Score} \\ & - 70.9527 \times \texttt{Cause\_Death} = \texttt{None}, \texttt{Body\_Touched} \\ & + 70.9527 \times \texttt{Cause\_Death} = \texttt{Body\_Touched} \\ & - 70.9527 \times \texttt{OVER} = \texttt{True} + 69.8903\end{aligned}$$

There is a slight change in its performance. The value for its correlation coefficient is 0.9999,

the mean absolute error is 7.747 and the root mean square error is 15.4697. There is still a very strong relationship between the predictions and the actual values.

`Random Tree`

This are the results obtained after training a decision tree using the dataset.

Correlation Coefficient	MAE	RMSE	Relative Abs. Error (%)
0.994	112.6685	184.9971	11.1533

Table 20: Summary of `Random Tree` Algorithm Results

The application of the Random Tree algorithm for predicting the `Next_Score` variable demonstrates a significant correlation coefficient of 0.994, suggesting that the model's predictions are very closely aligned with the actual values . Nonetheless, lower values for MAE and RMSE would be more desirable.

We will now run the algorithm after deleting the least relevant attributes.

Correlation Coefficient	MAE	RMSE	Relative Abs. Error (%)
0.9917	113.7596	182.6771	11.2614

Table 21: Summary of Algorithm Results after Deleting Attributes

The results are very alike. There is a slight improvement in some metrics and a worsening in others. Depending on the context we might consider using one or the other.

`Support Vector Regression` (`SVR`)

After running the Support Vector Regression (`SVR`) these are the results produced .

Correlation Coefficient	MAE	RMSE	Relative Abs. Error (%)
1.0	3.9843	4.7857	0.3944

Table 22: Summary of `SVR` Algorithm Results

In general, these results indicate that the `SVR` model has successfully identified the inherent trends in the data, enabling it to predict the `Next_Score` variable with high precision. Nonetheless, like all models, it is crucial to test its effectiveness on new, unseen data and be mindful of overfitting

We will now run the algorithm with the dataset where the attributes `Head_Y` and `Food_Position_Y` and `Food_Eaten` were deleted.

Correlation Coefficient	MAE	RMSE	Relative Abs. Error (%)
0.9999	6.1843	15.128	0.6122

Table 23: Summary of `SVR` Algorithm Results after Deleting Attributes

These results indicate that even after deleting the non-significant attributes, the `SVR` algorithm still achieved an almost perfect correlation coefficient of 0.999. The mean absolute error (MAE) and root mean squared error (RMSE) have experienced a moderate increment.

As we have observed in the preprocessing part of this process, not all the attributes contribute equally to the prediction of the target and there are some which are almost non-significant and deleting them has little to no effect on the evaluation metrics.

All the algorithms used (Linear Regression, Random Tree, and Support Vector Regression (`SVR`)) provided extraordinarily high correlation coefficient values . Linear Regression and `SVR` demonstrated exceptional performance with almost perfect correlation coefficient (suggesting a potential overfitting) while mantaining small values for MAE, RMSE and Relative Alsolute Error. On the other hand, Random Tree yielded values of MAE and RMSE which could be improved.

6 Questions

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

Learning models with instances from a human-controlled agent and an automatic one can be different. People might act in more varied and unpredictable ways, which affects the data. Also, humans might have biases that machines do not, while machines might give more consistent data.

6.2. 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?

To transform the regression task into classification, we would discretize the continuous output into categories. Predicting the score in gaming can enhance player feedback, adjust game difficulty, personalize gaming experiences, and facilitate competitive gaming.

6.3. What are the advantages of predicting the score over classifying the action? Justify your answer.

Predicting the score gives detailed feedback on how well players are doing and helps improve game strategies. It also lets game designers make better decisions based on player data. Classifying actions might make things too simple and not give enough information about player behavior or game performance.

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

Adding an attribute to show if the score went down might help improve rankings. This would let models notice changes over time and adjust predictions.

7 Conclusions

To sum up, our analysis helped us understand how different machine learning methods work for tasks like sorting things into groups (classification) and predicting future values (regression). We found that some methods, like decision trees and a special algorithm called PART, worked well for classification. And for predicting scores in our game, we saw that Support Vector Regression (SVR) did really well.

Apart from the technical aspect, our findings hold significant implications across various domains. For instance, in gaming, our developed models can enhance the gameplay experience by enabling automatic agents to make informed decisions using predictive algorithms. Furthermore, our exploration underscores the adaptability of machine learning across diverse sectors, including healthcare and finance, where predictive modeling can play a pivotal role in driving informed decision-making processes and optimizing operational efficiency.

In relation to building an automatic agent, we have encountered many problems. Since we did not have a Linux team, we installed the virtual box, which was very time consuming. After following the instructions and modifying the Python scripts, opening them in order to build the agent was not possible. Even though we were not able to complete this task succesfully, our suposition was that the intelligent agent built using Weka would yield better predictions as it uses machine learning algorithms which can learn from input data and improve their performance over time. This means that the intelligent agent could adapt and improve as it plays more games.

This project had its fair share of challenges. We had to make sure our models did not just remember the data (overfitting) and figure out which factors were most important. Overcoming these obstacles meant finding the right balance between capturing patterns in the data and being able to predict new information. We also had to work out which features really mattered in making predictions. It took a lot of testing and experimenting to find the best variables that made our models work well. But by facing these challenges head-on, we improved our methods and created better predictive models.

This project has been both challenging and rewarding. We have learned a lot about how machine learning can help solve real-world problems. It has shown us the potential of these techniques and how they can make a real difference in various situations. This project has given us a deeper appreciation for the power of machine learning and the exciting possibilities it holds for the future.