Lab 3 – Building a Ms. Pacman pro-player decision tree with ID3

Group 2: Hallberg Sofia, Kareld Oscar

INTRODUCTION

Ms Pacman is a maze arcade game where the player takes the role of Ms Pacman and earns points by eating pills in the maze. The maze is also inhabited by the four ghosts, Blinky, Inky, Pinky and Sue, who are trying to eat Ms PacMan. There are also a number of power pills in the maze and when Ms Pacman eats a power pill the ghosts get eatable instead. The aim of a game is for Ms Pacman to collect as many points as possible by eating pills without being eaten by a ghost.

The purpose of the assignment is to implement a functional AI controller able to learn from data recorded from playing the game. The controller should be a decision tree trained with the ID3 algorithm.

The ID3 algorithm generates a decision tree from a set with training data. When creating a node in the decision tree the ID3 algorithm iterates through every unused attribute in the path and calculates the information gains for the attributes. The attribute with the largest information gain is selected to represent the value of the node and the data set is partitioned by the selected attribute. The algorithm recurse with the partitioned data and the remaining attributes as parameters to create childnodes until one of the following scenarios occur:

- Every element in the subset belongs to the same class
- There are no more attributes to be selected
- The subset is empty

The final node is turned into a leaf-node classified with the class label of the final subset of the branch.

When running the game the decision tree is used to determine Ms Pacman next move taking the game state as input.

DESIGN AND IMPLEMENTATION

The algorithm is implemented in three modules

- Tree building
- Attribute selection
- Subset creation

Tree building

The tree building recursively builds the tree node by node, finally returning the root node to be used as entry point for the AI controller.

```
Node buildTree (dataSet DS, attributeList AL)
    create a node N
    if all tuples in the DS has the same class label C
        classify N with C
        return N
    if AL is empty
        classify N with the most common class label C in DS
        return N
    else
        select attribute A using the attributeSelector (DS, AL)
        label N with A
        remove A from attributeList
        for all possible values V of A
            make ALC as a copy of AL
            create subset S using subsetCreator (DS, A, V)
            if S is empty
                create child-node CN and add to N
                classify CN with the most common class label C in DS
            else
                create child-node recursively with buildTree(S, ALC)
        return N
```

Attribute selection

The attribute selection takes a dataset and a list with attributes as input and returns the attribute from the list, with the highest information gain.

```
Attribute attributeSelector (dataSet DS, attributeList AL)
    count number of tuples in dataset
    count number of tuples with the different classes
    calculate average information in dataset
    for all attributes A in AL
        calculate number of tuples with the different values for A
        calculate information gain
    return the attribute with the highest information gain
```

Subset creation

The subset creation takes a dataset, an attribute and a value as input parameters and returns a subset containing the tuples from the dataset where the given attribute has the given value.

```
Dataset subsetCreator (dataSet DS, attribute A, value V)
    create empty subset SUB
    for all tuples in DS
        if value is V for attribute A
        add tuple to SUB
    return SUB
```

Data gathering

The data for training and testing the algorithm was gathered when Ms PacMan was played a number of times. The players tried to play as well as possible to provide the algorithm with information telling how Ms PacMan should react in different situations.

The recorded data was divided into training data (80%) and test data (20%).

Input parameters

Parameters used when building the algorithm were restricted to:

- Position of PacMan
- Position of ghosts
- If ghosts are edible or not
- Direction of ghosts

The reason for not using further parameters is to limit the expansion of the decision tree and limit the manual effort when building the algorithm.

Other possible parameters neglected due to expected lack of information for the decisiont:

- Remaining lives for PacMan
- Current score
- Total game time
- Current level time
- Number of total pills in level
- Number of total power pills in level

The main reason for not including these parameters is that the decision made by the AI should depend upon these parameters. A given situation should result in the same action taken by the AI regardless if the number of lives left is one or two or if the game has progressed for one minute or three. For the same reason, the number of pills and power pills are not used as input parameters, the information doesn't have value as long as there is no information about the position of the pills.

There were a number of parameters that we thought had merit in the decision making of the AI, but in the end decided not to implement it into our solution. For example, it would be reasonable to take the current level into account since the position of PacMan brings much more information when it's known in what maze the position is. Since the level was not used as a parameter the data gathering is made in the same maze (the first).

The parameters are discrete from the beginning or are discretized before they are used in the tree building. The continuous parameters (position of PacMan and the distances between PacMan and the ghosts) were discretized using functionalities included in the framework.

Printing the first path of the tree shows that the attribute with highest gain is BLINKY_EDIBLE followed by SUE_EDIBLE, PINKY_EDIBLE and BLINKY_DIR if the boolean attributes are true. It does make sense that ghost edible is an important attribute since distances to the ghosts and their positions are not important when they are edible.

Training accuracy

The final version of the algorithm has accuracy 85% for the training data and 82% for the test data. The small difference between the training accuracy and the test accuracy indicates that the algorithm is not overfitting. Also the moderate accuracy of 80% indicates that the algorithm does not overfit.

RESULTS

The resulting AI is not an optimal Ms Pacman agent, but with a larger amount of training data the foundation is expected to generate a mode decent AI. The amount of data was restricted to keep the running time of the program reasonable and it would be interesting to see how accurate the implementation could be given a more substantial amount of data (i.e. hundreds of thousands of tuples from several hours of quality gameplay). If the algorithm for calculating gain, and the resulting decision tree, works correctly, this could result in an agent that is very good at avoiding the ghosts (when not edible) but not at all interested in collecting pills and power pills. In that regard, the AI agent can be seen as some kind of opposite to the NearestPillPacMan().

The initial solution did not take the direction of the ghosts into account, resulting in an accuracy around 60%. Once ghost directions were used as attributes, accuracy increased to over 80%. The improvement of accuracy was expected since the position of a ghost is not enough for the AI to assess what direction itself should head for.

Except for a bigger amount of training data resulting in a longer start up time for the application, there are no performance issues during the actual gameplay. This is probably a testament to the strength of traversing through a decision tree. Even if the tree as a whole is large, the time to find the next move is short since the tree is shallow.

The decision tree contains more than 10 000 nodes, but the depth of the tree never exceeds 13, which is why it takes time to build the tree but to follow a path from root to leaf is a quick operation.

Possible improvements are to make the algorithms more generalizable to facilitate an easy extension of attributes. As the algorithms are implemented, adding a new attribute is time consuming. With a more scalable application it could be easier to try different combinations of attributes.

Another potential improvement is to decrease the time complexity of the training process. Small improvements to speed up the process have been done, but there are still procedures iterated more than necessary.

Other potential use for decision tree algorithms is an AI GPS system choosing an optimal route for a driver. A GPS that always chooses the shortest distance route, not taking into account the amount of traffic, recent accidents, recurring events and a number of other parameters will result in a lot of frustrated drivers in this day and age.

One of our bigger takeaways from this assignment is that this was quite hard - even in a controlled setting like the provided framework. We can only imagine the difficulty of creating decision trees based on thousands of parameters where the data could be incomplete or in other ways hard to discretize.

REFERENCES

https://mau.instructure.com/courses/8846/files/1436794?module_item_id=348857
2022-01-15