Practice 1: Classification and Prediction

Alejandro Taboada Esteban - 100472778
Martín Ranea Marina - 100474046

Phase 1: Instances collection

First we changed the printLineData() function with the aim of getting some attributes that can help us do the prediction. We selected to store the following information:

The Pac-Man Position
The direction Pac-Man is going to
The number of living ghosts
The position of each ghost
The distance to each ghost
The number of food remaining
The distance to the closest food
The score
The action performed

In the file game.py it is necessary to create the header of the columns for a .arff file. And in the same file, we added the next action (Action) and the next score (Score). After that, each training instance had information about three types: current state, performed action and information about the next turn.

At that moment we needed to collect the data for each file. We had three files for the data collected through the keyboard game and another three through the intelligent Pac-Man from tutorial 1. For each of them, there is one of training instances generated by manually playing with BustersKeyBoardAgent on 5 maps, combining games where the ghosts are static and move randomly, one of test instances generated using the same agent and maps, and one of test instances generated by using the same agent on 5 different maps.

Phase 2: Classification

In this phase, we apply the same actions to the files that collect instances for the intelligent Pac-Man we programmed and for the files that collect instances for the Pac-Man that is not automatic (keyboard), and like this we are going to be able to compare the performance of both cases.

For the data preprocessing, we tried 2 different filters to choose the attributes, and after comparing them we used the second one.
Firstly, we will explain how we have done the preprocessing and evaluation of the

intelligent Pac-man, and then the same for the keyboard mode.

1. INTELLIGENT PACMAN

1.1. Data preprocessing

Once we have the 3 files, we apply two different attribute evaluators that show the importance of each attribute taking into account that we want to predict the attribute *Action*. We start applying *gainRatioAttributeEval* and the search method *Ranker* for generating a ranking of the attributes. Later, we will use the attribute evaluator *CfsSubsetEval.*

gainRatioAttributeEval
We have obtained the results of the following table:

Ranked attributes:

1.011	17	NumDots
1.009	6	G1PosY
989	5	G1PosX
977	13	DistG1
964	12	G4PosY
956	2	Pacy
932	11	G4PosX
869	3	PacDir
845	19	Score
797	10	G3PosY
787	7	G2PosX
661	9	G3PosX
65	16	DistG4
603	1	Pacx
585	8	G2PosY
404	4	NumAgents
345	18	DistDot
0	15	DistG3
0	14	DistG2

As we can see in the table, the last two attributes do not affect when predicting *Action,* so we removed them. We also removed *DistanceToGhost1* and *DistanceToGhost2* because they have a relevance smaller than 0.1.

We applied again the attribute evaluator that we used before and we obtain the following table:
Ranked attributes:

1.009	5	G1PosY
989	4	G1PosX
977	12	DistG1
964	11	G4PosY
956	2	Pacy
932	10	G4PosX
869	3	PacDir
845	16	Score
797	9	G3PosY
787	6	G2PosX
661	8	G3PosX
65	15	DistG4
603	1	Pacx
585	7	G2PosY
0	14	DistG3

We can see in the table that we can eliminate *DistanceToGhost3*, and we find new attributes that influence when predicting the class. In order to have the attribute *Action* at the last one, we apply the filter *Reorder*. So, we save this file. We did the same changes (the ones on the training dataset) in the test files.

CfsSubsetEval
Then, we applied the attribute evaluator *CfsSubsetEval* and the search method *BestFirst* to the file that we had before applying the previous attribute evaluator. And we obtained the following output:

Selected attributes: 1,3,5,6,12,17,19 : 7
Pacx
PacDir
G1PosX
G1PosY
G4PosY
NumDots
Score

So we removed from this file all the attributes that did not appear in the table.

Now, we created the new attributes. In order to do that, we applied the same filters as before and we do the same transformations of the attributes, to be able to eliminate one of the directions (south, noth, west and east), or try to group them

With the attribute evaluator CfsSubsetEval and the search method BestFirst, we obtain:

Selected attributes: 1,3,5,6,12,17,19 : 7
Pacx
PacDir
G1PosX
G1PosY
G4PosY
NumDots
Score

1.2. Evaluation

We performed the following algorithms using cross-validation in the dataset of training tutorial 1 (with filter1 and filter2).

Algorithm	**training_tutorial1_ filter1**	**training_tutorial1_ filter2**
J48	85.78%	86.74%
IBK k = 1	86.02%	85.74%
IBK k = 3	85.78%	86.26%
IBK k = 5	82.40%	82.16%
PBK k = 10	86.57%	87.98%
ZeroR	42.65%	42.65%
Naive Bayes	81.20%	76.86%
RandomForest	85.41%	85.74%
M5Rules	86.75%	86.31%

The models that we are going to use for phase 4 in python are M5Rules and IBK for k = 10, because they have the highest accuracies for the training set, even if J48 had good values too.

2. KEYBOARD PACMAN

Then, we did the same for the keyboard Pacman.

2.1. Data preprocessing

We did the same as before, we applied the attributes evaluators *gainRatioAttributeEval* and *CfsSubsetEval.*

gainRatioAttributeEval

We have obtained the following results:

Ranked attributes:

3.011	3	PacDir
2.427	19	Score
1.381	17	NumDots
1.133	12	G4PosY

1.109	2	Pacy
1.096	13	DistG1
1.065	5	G1PosX
1.064	6	G1PosY
1.017	16	DistG4
991	1	Pacx
868	11	G4PosX
751	8	G2PosY
731	4	NumAgents
709	15	DistG3
582	7	G2PosX
494	9	G3PosX
448	10	G3PosY
431	14	DistG2
0	18	DistDot

We removed all the attributes with an influence smaller than 0.1.

We again created the attributes *north-south* and *east-west*, using the same filters and doing the same transformations as before. We applied the attribute evaluator again and we obtained the set for filter1.

CfsSubsetEval

Then, we applied the attribute evaluator *CfsSubsetEval* and the search method *BestFirst* to the training_keyboard_classification file(the one that we had before applying the previous attribute evaluator).
And we obtained one more table.

So we removed from the file all the attributes that did not appear in the table. Now, we create new attributes as before. As we did not have the attribute South, we only created the attribute *east-west.* In order to do that, we applied the same filters as before and we did the same transformations of the attributes.

We applied again the attribute evaluator *CfsSubsetEval* and the search method

BestFirst and we obtained the final set for the filter2

2.2. Evaluation

We tried these algorithms and got the following accuracy using the training sets of the Pac-Man keyboard and the cross-validation method.
Even if now the best model would be RandomForest, we will do the same ones as above to simplify the job

Phase 3: Regression

At first, we applied the attribute evaluator *CfsSubsetEval* with the search method *GreedyStepwise* to decide which attributes to remove and then we started with the algorithms.
We decided to do the same as in the previous phase and compare first the models generated using the cross-validation method. Then, we selected the best ones and selected the best ones to apply them into the two testing sets.

1. INTELLIGENT PAC-MAN

1.1. Data preprocessing
We obtained the following results:

Selected attributes: 1,3,5,6,12,17,21 : 7
Pacx
PacDir
G1PosX
G1PosY
G4PosY
NumDots
FutureScore

So we removed the attributes that do not appear in the table and we saved the file.

1.2. Evaluation
Then, we execute some different models and we obtain the following results.

	training_tutorial1_filter1
IBK k=1	0.91
IBK k=3	0.90
IBK k=5	0.91
LinearRegression	0.99
REPTree	0.97
RandomForest	0.97
M5P	0.98

So linear regression is the algorithm that gets the best scores.

2. KEYBOARD PAC-MAN
2.1. Data preprocessing

We did the same process as before, obtaining the set with the filter applied.

2.2. Evaluation

Then, we execute some different models and we obtain the following results.

TABLA

	training_keyboard_regression_filter
IBK k=1	0.49
IBK k=3	0.65
IBK k=5	0.68
LinearRegression	0.72
REPTree	0.68
RandomForest	0.57
M5P	0.64

The results are very similar, so we could choose either.

Phase 4: Building an automatic agent

Once we have tried the best models that we obtained in phase2, we realized that our Pac-Man does not work very well. The models that we have tried are IBK for k = 10 and M5Rules. We applied them. While we performed the training of the tutorial1 and keyboard versions, we obtained some graphs of the correlation and the error found, which we found interesting.

Finally, the models were created and the IBK for k = 10 performed a bit better, even if it does not work as good as we expected, which we found disappointing

Questions

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

The primary distinction lies in the fact that playing the game with a static ghost using the automatic agent will result in the collection of identical instances every time we play on the same map. However, when using the keyboard agent, it is less likely that we will take the same steps each time. If we use the automatic agent with static ghosts, we may end up with a model that overfits because the same instances are repeated multiple times. To avoid this issue, we used random and static ghosts, which allowed us to achieve better accuracy by incorporating instances from the automatic agent.

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?

Our opinion is that the most suitable approach is to treat the score values as nominal types of attributes with intervals in between. This method would prove beneficial in determining whether the Pac-Man agent has consumed food or a ghost.

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

Selecting the attribute to predict depends on the specific objective at hand, suggesting that there are no advantages of predicting the score over classifying the action.

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?

In our perspective, it would not be beneficial as we are already aware that the score diminishes over time and increases whenever Pac-Man consumes food or a ghost.

Conclusions

- Technical conclusions about the task that has been carried out.

Once we applied machine learning to Pac-Man, we recognized that we require additional examples and a longer duration to experiment with various approaches and observe the outcomes. To achieve superior outcomes, we executed the identical set of five maps multiple times, which may result in overfitting in the datasets.

- More general insights such as: what can the obtained model be useful for, if during the practice you have come up with other domains in which machine learning can be applied, etc.

In the event that the model can accurately forecast actions using fresh examples, it would be valuable in effectively playing this game.

- Description of the problems encountered when doing this practice.

Initially, we encountered an issue with the data collection in the previous tutorial as we only gathered instances with limited attributes. Subsequently, we realized that more attributes were required and made changes to the printLineData() function and arff file headers creation in game.py. In addition, we had difficulties with the Java bridge not working on our laptops, leading us to install the virtual machine of Ubuntu to proceed with the project.

When testing the models, Pac-Man's performance was subpar, and we recognized the need to gather more instances as we only had 500 for the training sets and 200 for the testing sets initially. We also attempted to create new attributes during the preprocessing stage, but found that they were not sufficiently relevant and opted to remove them and try alternative options.

- Personal comments. Opinion about the practice. Difficulties encountered, criticisms, etc.

We are somewhat disappointed that we invested a significant amount of time in data preparation, only to find that our model's performance was worse than expected when applied to gameplay. It appears that all our efforts were in vain, and we ended

up making final predictions without extensive preprocessing, simply relying on attributes that we assumed would be the most fitting. Despite this setback, we did enjoy the experience of programming around a famous game that we have all played, and also learned a lot deeper about how modeling works, and that it is not always easy to find a model that fits your data.