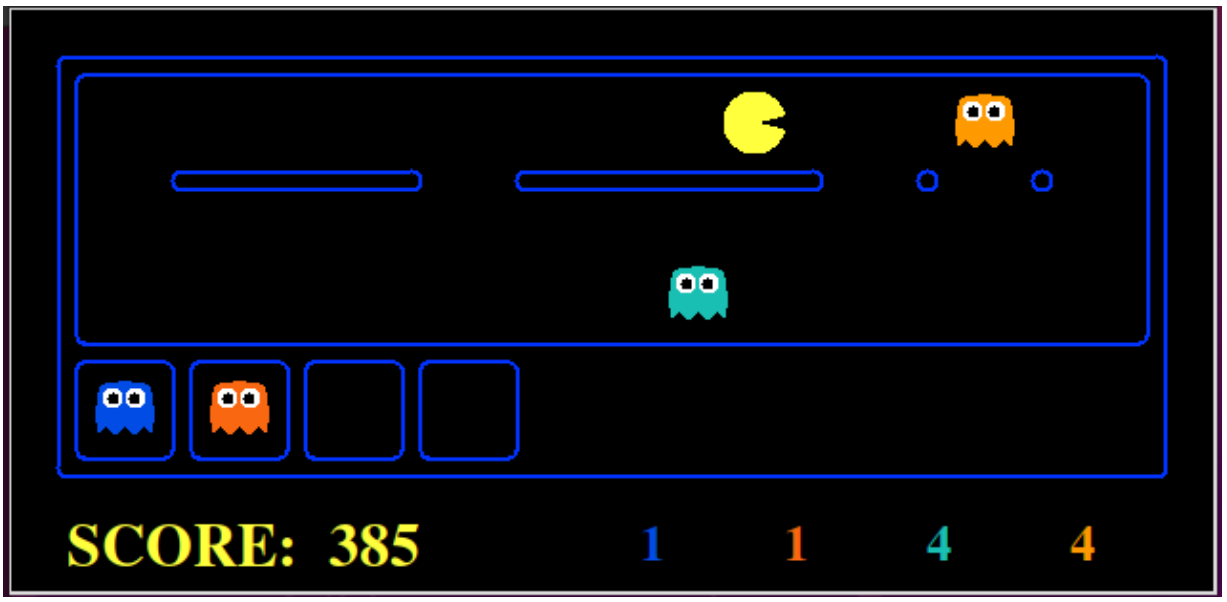
Departamento de Informática
Universidad Carlos III de Madrid

Bachelor in Data Science and Engineering
Machine Learning I
Course 2020-2021

Assignment 1: Classification and Prediction

Lucía León Marco & Elena Gómez Espinosa
100451897-100451953



2 Phase 1: Instances collection

The first step to preparing the data is to modify the *printLineData()* function with the aim of getting some attributes that can help us do the prediction. Hence, inside the function, we created a variable called *state* in which we are going to save the information of the Pac-Man in the current state.
We selected to store the following information named as:

- Px, Py: the Pac-Man position (each coordinate in one column),
- PDir: the performed action,
- GhostsNum: the number of ghosts (the number of agents minus one that is the Pac-Man),
- DistanceToGhostX: the ghosts distances to each Pac-Man (if the ghost is no more living, its distance will be set as -1) being X the number of the corresponding ghost,
- PrevScore: that is the actual score,
- North, South, East, West: four columns for each legal action, where if the direction is legal we will put a 1 and 0 otherwise,
- FoodNum: the number of food and,
- FoodDistance: the distance to the nearest food (set as -1 if it no longer exists).

Hence, we had the *printLineData()* function modified.
Then, 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 our "intelligent Pac-Man".
Two of those three files are done with the same five maps and used one for training and the other for testing, and the last file is a testing dataset but performed with the other five maps. We also played with static and random ghosts

The first five maps that we used are: classic, contestClassic, smallHunt, minimaxClassic and trappedClassic. And the other five are: oneHunt, openClassic, originalClassic, testClassic and NEWmaze.

3 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).

Once we have finished phase one, we have six files that provide us with all the considered attributes of the intelligent Pac-Man. Now, we test how selecting a different set of attributes affects each dataset. First of all, we divided the files that we will use for classification and for regression.

The files that we will use for regression are exactly the ones that we have collected. We called them **training_tutorial1_regression**, **test_samemaps_tutorial1_regression,**

test_othermaps_tutorial1_regression and **training_keyboard_regression**, **test_samemaps_keyboard_regression, test_othermaps_keyboard_regression** . The ones that we will use for classification are the same, but removing the variable *Score*.We called them **training_tutorial1_classification**, **test_samemaps_tutorial1_classification, test_othermaps_tutorial1_classification** and **training_keyboard_classification**, **test_samemaps_keyboard_classification, test_othermaps_keyboard_classification .**

Then, we removed the instances that contain *Stop* from the attributes *Action* and *PDir* (previous direction), because we did not want the next action of our Pac-man to be *Stop.* We applied the filter *Removewithvalues* (unsupervised) to columns *Action* and *PDir* and we have selected that it removes the index 5 (the index of *Stop*).
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 for classification, 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:

0.3933	14	East
0.3677	11	North
0.3239	3	PDir
0.2719	13	West
0.2579	12	South
0.1928	10	PrevScore
0.178	2	Py
0.145	1	Px
0.1406	15	FoodNum
0.0739	9	DistanceToGhost4

0.056	5	LivingGhost
0.0541	7	DistanceToGhost2
0	16	FoodDistance
0	4	GhostsNum
0	6	DistanceToGhost1
0	8	DistanceToGhost3

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

We continue creating new attributes. Firstly, we converted the legal actions (*North*, *South*, *West*, *East*) into numeric attributes, using the filter *NominalToBinary* (for unsupervised attributes). And we create a new attribute called *totalLegalActions* which is the sum of the attributes *North*, *South*, *West,* and *East*. For that, we used the filter *AddExpression*.

After that, we applied again the attribute evaluator that we applied before, and we realized that the new attribute does not influence when predicting *Action,* so we removed it.

Now, we created two new attributes, *north-south*, which is the sum of *north* and *south*, and *east-west,* which is the sum of *east* and *west*. Then we converted the attributes *North*, *South*, *West*, *East* to nominal using the filter *NumericToNominal* and did the same with the new attributes *north-south* and *east-west*.

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

0.393	8	East
0.368	5	North
0.324	3	PDir
0.288	11	north_south
0.273	12	west_east
0.272	7	West
0.258	6	South
0.193	4	PrevScore
0.178	2	Py

0.145	1	Px
0.141	9	FoodNum

We can see in the table the 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 with the name: **training_tutorial1_classification_filter**
We did the same changes (the ones on the training dataset) in the test files and we called them: **test_samemaps_tutorial1_classification_filter** and **test_othermaps_tutorial1_classification_filter**.

CfsSubsetEval

Then, we applied the attribute evaluator *CfsSubsetEval* and the search method *BestFirst* to the file **training_tutorial1_classification** (the one that we had before applying the previous attribute evaluator). And we obtained the following output:

Selected attributes: 2,3,10,11,12,14,15:7
Py
PDir
PrevScore
North
South
East
FoodNum

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

Now, we created the same new attributes as before. In that case, we did not have the attribute West so we only created the attribute *north-south.* In order to do that, we applied the same filters as before and we do the same transformations of the attributes.

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

Selected attributes: 2,3,10,11,12,14,15:7
Py
PDir
PrevScore

North
East
FoodNum
North-South

As the table shows, South did not appear in the ranking now, so we removed it.
We saved this file and we named it **training_tutorial1_classification_filter1.**
Doing the same changes that we did for the training in the test files, we named the files: **test_samemaps_tutorial1_classification_filter1** and **test_othermaps_tutorial1_classification_filter1**.

1.2. Evaluation

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

Algorithm	**training_tutorial1_ classification_filter**	**training_tutorial1_ classification_filter1**
J48	85.7831%	86.747%
IBK k = 1	86.0241%	86.747%
IBK k = 3	85.7831%	86.2651%
IBK k = 5	82.4096%	82.1687%
PART	84.5783%	86.988%
ZeroR	42.6506%	42.6506%
Naive Bayes	81.20%	76.8675%
RandomForest	87.71%	86.747%
Id3	79.759%	78.3133%

Now, we test the models with the highest accuracy

	test_samemaps_tutorial1_ classification_filter	**test_othermaps_tutorial1_ classification_filter**
RandomForest (training_tutorial1_ classification_filter)	99.0361%	67.0455%

	test_samemaps_tutorial1_ classification_filter1	test_othermaps_tutorial1 _classification_filter1
J48 (training_tutorial1_ classification_filter1)	92.3372%	59.4697%
IBK k = 1 (training_tutorial1_ classification_filter1)	95.4023%	63.6364%
RandomForest (training_tutorial1_ classification_filter1)	95.0192%	68.9394%

These are the models that we will try using python in phase 4.

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

0.411	11	North
0.3638	3	PDir
0.3128	14	East
0.2824	13	West
0.257	12	South
0.2268	1	Px
0.1682	2	Py
0.1527	10	PrevScore
0.1248	5	LivingGhost

0.1024	15	FoodNum
0.0781	16	FoodDistance
0.0555	9	DistanceToGhost4
0	6	DistanceToGhost1
0	7	DistanceToGhost2
0	4	GhostsNum
0	8	DistanceToGhost3

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 obtain:

Ranked attributes

0.411	6	North
0.364	3	PDir
0.313	9	East
0.287	11	north-south
0.282	8	west
0.257	7	South
0.249	12	east-west
0.227	1	Px
0.168	2	Py
0.153	5	PrevScore
0.125	4	LivingGhost
0.102	10	FoodNum

Now, there is not any attribute with an influence smaller than 0.1, so we saved this file and we called it **training_keyboard_classification_filter.** We did the same changes in the test sets and we called them **test_samemaps_keyboard_classification_filter** and **test_othermaps_keyboard_classification_filter.**

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 the following output:

Selected attributes:3,10,11,13,14:5
PDir
PrevScore
North
West
East

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:

Selected attributes:1,2,3,6
PDir
PrevScore
North
east-west

Now, we remove the attributes that do not appear in the table. We saved the file and called it **training_keyboard_classification_filter1.** We did the same changes in the test sets and we called it **test_samemaps_keyboard_classification_filter1** and **test_othermaps_keyboard_classification_filter1.**

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.

Algorithm	**training_keyboard_ classification_filter**	**training_keyboard_ classification_filter1**

J48	82.1346%	81.4385%
IBK k = 1	80.6425%	79.8144%
IBK k = 3	79.5824%	80.7425%
IBK k = 5	79.8144%	79.3503%
PART	79.1183%	80.0464%
ZeroR	31.0905%	31.0905%
Naive Bayes	77.7262%	76.1021%
RandomForest	84.45%	80.2784%
Id3	71.9258%	75.174%

Then, we selected the models with higher accuracy and tried them with the test sets, samemaps and othermaps.

	test_samemaps_keyboard_ classification_filter	**test_othermaps_keyboard _classification_filter**
J48 (training_keyboard_ classification_filter)	88.9313%	80.7122%
RandomForest (training_keyboard_ classification_filter)	91.2214%	78.0415%

	test_samemaps_keyboard_ classification_filter1	**test_othermaps_keyboard _classification_filter1**
J48 (training_keyboard_ classification_filter1)	81.6794%	62.908%

These are the models that we will try using python in phase 4.

4 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:1,2,3,6,7,8,10,15,16,17 : 10
Px
Py
PDir
DistanceToGhost1
DistanceToGhost2
DistanceToGhost3
PrevScore
FoodNum
FoodDistance
Action

So we removed the attributes that do not appear in the table: We saved the file and we called it **training_tutorial1_regression_filter**. We did the same in the test sets and we called them **test_samemaps_tutorial1_regression_filter** and **test_othermaps_tutorial1_regression_filter**.

1.2. Evaluation

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

	training_tutorial1_regression_filter
IBK k=1	0.91
IBK k=3	0.9053
IBK k=5	0.9092
LinearRegression	0.9901
REPTree	0.9871

RandomForest	0.9895
M5P	0.9902

We test the ones that better predict the scores. And we obtained the following results:

	test_samemaps_tutorial1_ regression_filter	**test_othermaps_tutorial1 _regression_filter**
LinearRegression (training_tutorial_ regression_filter)	0.991	0.9937
M5P (training_tutorial_ regression_filter)	0.991	0.9937

So linear regression and M5 are the algorithms that get the best scores.

2. KEYBOARD PAC-MAN

2.1. Data preprocessing

We obtained the following results:

Selected attributes:1,2,5,9,10,15,16: 7
Px
Py
LivingGhost
DistanceToGhost4
PrevScore
FoodNum
FoodDistance

So we removed the attributes that do not appear in the table: We saved the file and we called it **training_keyboard_regression_filter**. We did the same in the test sets and we called them **test_samemaps_keyboard_regression_filter** and **test_othermaps_keyboard_regression_filter**.

2.2. Evaluation

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

	training_keyboard_regression_filter
IBK k=1	0.4988
IBK k=3	0.667
IBK k=5	0.679
LinearRegression	0.705
REPTree	0.6929
RandomForest	0.5878
M5P	0.6599

We test the ones that better predict the score. And we obtained the following results:

	test_samemaps_keyboard_ regression_filter	**test_othermaps_keyboard_ regression_filter**
LinearRegression	0.9897	0.995
REPTree	0.9843	0.9906

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

5 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 **j48.model** and **ibk1.model** with the arff **training_tutorial1_classification_filter1** .Although we obtained high accuracies, predictions are not good enough.
So, now we will try different solutions:

- Collecting more instances (between 2000-3000 for keyboard and tutorial1).
- Selecting the attributes logically instead of using attribute evaluators. The selected attributes are: Px, Py, DistanceToGhost1, DistanceToGhost2, DistanceToGhost3, DistanceToGhost4, North, south, west, east and action(class).

We named the new arff files **NEWinstances_keyboard**, **NEWinstances_tutorial1**. The files we had used for the models are named as j48_keyboard.model with the corresponding name for each one.

We collected between 2000 and 3000 instances for the training file of tutorial1 using the following maps: classic, contestClassic, smallHunt, minimaxClassic and trappedClassic. Then we tried the models that fit better in the pre-processing with cross-validation although the data is different.

- J48: it does not work very well with the maps used for the training set but with other maps it does. (**j48_2.model** (model) with **NEWinstances_tutorial1** (arff))
- RandomForest: it works better than J48 (with the maps used for the training set) but it predicts the action in other maps more or less the same. (**randomforest_tutorial1.model** with **NEWinstances_tutorial1**)
- Ibk with k=1: is the one that works better in the maps used for training but the prediction of the actions for other maps are not correct (classic, contestClassic, smallHunt, trappedClassic). (**ibk_tutorial1.model** with **NEWinstances_tutorial1**)

Using the same maps and algorithms we tried the keyboard Pac-Man and get the following results:

- J48: it works more or less well (for predicting NEWmaze). **(j48_keyboard.mode**l with **NEWinstances_keyboard**)
- RandomForest: it works more or less well.**(randomforest_keyboard.model** with **NEWinstances_keyboard**)
- Ibk k=1: this is the one with best results (for predicting oneHunt, NEWmaze, smallHunt). (**ibk_keyboard.model** with **NEWinstances_keyboard**)

The best ones are ibk for the instances collected from the intelligent Pac-Man and from the ones collected from the keyboard (**ibk_tutorial1.model** with **NEWinstances_tutorial1** and **ibk_keyboard.model** with **NEWinstances_keyboard**).

After applying machine learning Pac-Man works worse than the one we programmed in tutorial 1. That is because our Pac-Man worked in almost every map and for computing the predictions we will need more instances and time to try in different ways how it works.

6 Questions

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

   The main difference is that if we run the game with the automatic agent with static ghosts, we will collect the same instances each time we play in the same map. But using the keyboard agent is less probable that you make the same steps each time. So, using the automatic one with static ghosts you could get a model with overfitting because you have the same instances repeated more times. In our case, for that reason we used random and static ghosts. We have obtained better accuracy using the instances coming 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?**

   We think that the best option would be to make intervals between the score values as nominal types of attributes. It will be useful for knowing if the Pac-Man agent has eaten a ghost or food.

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

   There are no advantages of predicting the score over classifying the action because one selects the attribute to predict based on the objective it has.

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?**

   From our point of view it won't be useful because we already know that the score decreases with the time and increases when the Pac-Man eats a ghost or food.

7 Conclusions

- **Technical conclusions about the task that has been carried out.**
  After applying machine learning to Pac-Man we realized that we will need more instances and time to try in different ways how it works, the results we get… For getting better results we run many times the same five maps and it may cause 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.**
  If the prediction of the actions were good with the new instances, the model will be useful to play this game in a successful way.

- **Description of the problems encountered when doing this practice.**
  The first problem we found was that in the previous tutorial we collected instances with a few attributes and then we realized that it was necessary to collect more. So we have to modify the *printLineData()* function and the creation of the headers of the arff file in the *game.py*.
  Then, the Java bridge does not work in our laptops and we needed to install the virtual machine of ubuntu to continue doing the assignment.
  When trying the models the Pac-Man works very badly so we have to collect more instances because at first we only had 500 for the training sets and 200 for the testing ones.
  Also, in the preprocessing we tried to create new attributes but they did not have such relevance to continue using it so we deleted them and tried new ones.

- **Personal comments. Opinion about the practice. Difficulties encountered, criticisms, etc.**
  It has been a bit disappointing that we had spent a lot of time preparing the data and when we have played using the models that we have created we look at the results as a disaster. So, all the work had been useless and our final predictions had been done without a deep preprocessing, just using the attributes that we think will fit better. By the way, it has been an enjoyable experience because it was programming a game.