ASSIGNMENT 1: CLASSIFICATION AND PREDICTION

Made by: **Marina Gómez Rey** (100472836)
María Ángeles Magro Garrote (100472867)

INDEX

Introduction

The aim of this project is to generate an autonomous Pac-Man that, based on some explanatory variables and generating and preprocessing some data, can play on its own in the most efficient way, also, if possible, maximizing the score of the game.

In order to do that, we had to decide which variables could be the most useful, try different combinations of them, as well as different algorithms (models) and select the best ones after generating a sufficient amount of data. After that, the selected model is tried and tested with the automatic agent.

As the practice is asked to be repeated at least two times to check the performance of each final model, we decided to proceed in the following way: the first one consists on focusing on eating the ghosts as quickly and efficiently as possible, and in this dataset, apart from the variables we had already taken in other tutorials, we had to add the future score and the current one.

After that model, our second iteration was focused on maximizing the score in general, adding the pac-dots to our analysis, so we had to include relevant variables related to these elements.

Model 1

As previously stated, the main objective of this model was to generate an automatic Pac-Man that can play in the most efficient way focusing on eating the ghosts as fast as possible.

● First approach

Phase 1: Instances collection

- Variable selection

As a first approach, we used the variables we had obtained in tutorial 3:

- Tick
- Pacman position in X
- Pacman position in Y
- Legal action North (True or False)
- Legal action South (True or False)
- Legal action East (True or False)
- Legal action West (True or False)
- Ghost position 1 X (coordinates of the first ghost in X)

- Ghost position 1 Y (coordinates of the first ghost in Y)
- Ghost position 2 X (coordinates of the second ghost in X)
- Ghost position 2 Y (coordinates of the second ghost in Y)
- Ghost position 3 X (coordinates of the third ghost in X)
- Ghost position 3 Y (coordinates of the third ghost in Y)
- Ghost position 4 X (coordinates of the fourth ghost in X)
- Ghost position 4 Y (coordinates of the fourth ghost in Y)
- Distance to ghost 1 (using the Distancer.getdistance function, that takes into account the walls)
- Distance to ghost 2
- Distance to ghost 3
- Distance to ghost 4
- Pacman direction (North, South, East, West, Stop)

Apart from those previous variables, as asked in the statement, we added the current score and the future one.

- Addition of the future score: regarding the future score, we had to overcome some issues to include it. Our idea was to generate a line of the present state without this last attribute, and then include the future score in the next tick and add the rest of the data in a new line.

  In order to do that, we created a condition for the line because, depending if it's the first tick or not, the format changes slightly.
  In the first tick of a game, the line is generated with all the attributes except the future score. Afterwards, the rest of the lines include:
  - First, the current score added to the previous line (this is the future score)
  - Then a '\n', to continue to the next line
  - Finally the next line data is generated again without the future score.

  However, we had a problem with the fact that the last line of a game was generated without the future score, which is a problem as weka cannot recognize an empty space. In order to overcome that issue, on each game (except the first one, when the document is empty) the previous last line is erased, overcoming the issue of having a not finished last line. This is done by opening the file, erasing the last line (position -1) with the pop function, and appending the file again so that the changes are saved. However, in the last game of all of them, the last line must be erased manually, which is a more approachable solution.

- Arff creation

Once we had selected the variables, we had to generate the data in a format that weka could recognize. This is done in the printLineData function.

Firstly, we generated a valid header that included all the variables with @attribute before and the correct types for each variable (all numeric except the legal actions and the Pac-Man direction).

Then, as stated before in the future score, weka cannot recognise empty values, and the distances to the ghosts become a None when they are eaten, for that reason, when this happens, we add an if statement where these values are changed into a "?", that can be recognised by weka.

The printLineData, which generates this information as a string, is then called in the chooseAction function so that on each tick the new line is appended to the document (we first check if the document is already existing or not so that we do not keep opening the file continuously, what can produce lag).

Phase 2: Classification

Data preprocessing, feature creation and selection was done in this step.

In weka, we decided to remove the Stop action as it is not useful in practice (this action only happens in the beginning, while the agent is not moving because the player has not pressed any arrow yet, and we do not want our agent to learn to remain stopped since the beginning). However, instead of deleting it in Python, we used the RemoveWithValues filter in weka to erase all lines that had a 'Stop' as taken action.

Finally, we had to reorder the variables so that the action Pac-Man has just taken is the last one, as it is the one we want to predict.

Before creating any variables, we decided to do subsets of our actual features to test the importance of them and see which variables were more relevant to the dataset so that we could reduce the dimension and, consequently, the complexity. Five datasets were created, in all of them the future score is removed as it must not be used right now:

- Dataset 1: all the variables are taken into account
- Dataset 2: the ghost coordinates are erased to see if their positions are relevant
- Dataset 3: the tick is erased to see if the moment in which the actions are happening is really relevant
- Dataset 4: the Pac-Man positions are erased
- Dataset 5: both the ghost and Pacman positions are erased

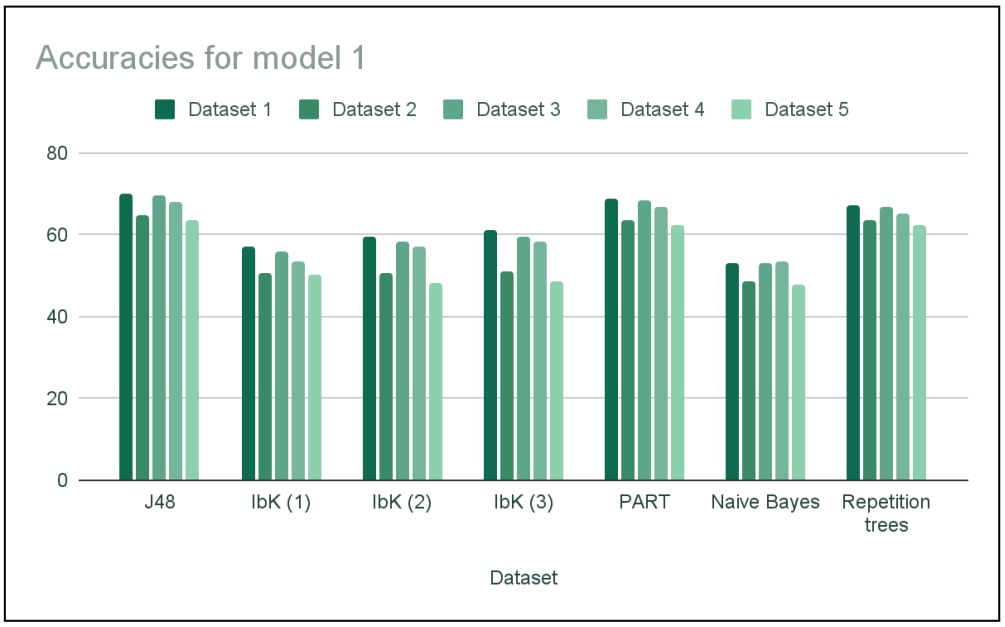
Also, we tested using different classification models to find the one that gives the best results and compare the tendencies with different results from different algorithms (all using cross-validation with 10 folds in this first approach). This was done with the experimenter in weka, as it repeats each model several times so that the results are more precise. Apart from

the ones we saw in the previous tutorials, we included repetition trees as classification algorithms as they provide better results than simple trees, as it can perform several tests rather than just one so that more complex boundaries can be taken.

The obtained results were:

Dataset	J48	IbK (1)	IbK (2)	IbK (3)	PART	Naive Bayes	Repetition trees
Dataset 1	**69,87**	**57,26**	**59,58**	**61,05**	**68,98**	52,87	**67.23**
Dataset 2	64,91	50,45	50,71	50,98	63,43	48,5	63.40
Dataset 3	69,6	55,96	58,24	59,62	68,24	53,23	66.98
Dataset 4	67,9	53,4	57,1	58,35	66,74	**53,39**	65.29
Dataset 5	63,61	50,1	48,3	48,72	62,26	48,02	62.23

Table which gathers accuracy among different datasets and models



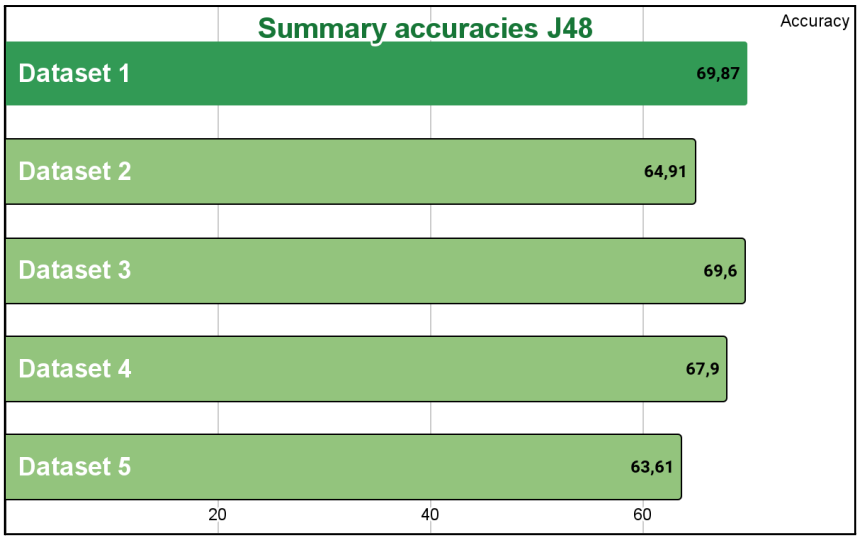
Bar graph with the accuracies obtained for model 1

The main conclusions obtained were two: dataset 1 provided us a better explanation of the data based on the results of the accuracy, as they are the highest in all models except the last one, where the difference is not very pronounced. Also, J48 gave a better accuracy than the rest of models. ZeroR was not even tested due to its algorithm, where all instances are classified as the majority class, which is not useful, and as we have 4 different classes, always gives an accuracy around 25%.

Regardless of that, other conclusions can be drawn based on the previous results. On one hand, the second best results are given by dataset 3, which suggest that the tick is not that

relevant as erasing it does not have such an intense impact on the final accuracies. On the other hand, the dataset which provided the worst results is number 5, which erases the positions of both Pac-Man and the ghosts, so these variables seem to have relevance.

The accuracies obtained with J48 can be summarized in the following graph:



Graph which gathers accuracy among different datasets in J48

After having done this, we decided to create some new features based on the already existing ones that could provide extra information:

- **Number of legal actions:** we added the number legal actions Pac-Man could take in that specific tick adding together the North, South, East and West variables (as their values are 1 if they can be taken or 0 if not).
- **High score:** we decided that a game has a high score if the third quartile of the score variable is passed (max*¾, the maximum value is 827). This would be 620,25 which is approximately 620.
- **Total distance to ghosts**: the distances to the ghosts are added together to have an overall vision of the remaining distance to cover.

*As the first dataset was the one that provided the best results in the previous step, we decided to keep working with this one and try these new variables on it.

The models are tested in the same way as before although only J48 will be displayed as it was proven that this model gives the best results:

Dataset	J48 accuracy
With # legal actions	69.88 %
With high score	70 %

With total distance to ghosts	69.84 %

Table with accuracies among different datasets

Seeing the accuracy these new variables provided, we decided to keep the high score as it increased the results slightly.

Our main doubt now was: should we balance? Should we normalize? J48 is known to be sensitive to unbalanced datasets and normalizing may help if the range of values among features varies widely. Despite this, balancing the target attribute direction did not provide us a better accuracy in J48 (68.07%) and normalizing numerical variables neither (69.59%), so we kept the variables in the same way.

Furthermore, the accuracy of the model was tested with the test sets obtained (note that the test sets have the same preprocessing applied as the training one):

Dataset	Accuracy
Test set same map	58 %
Test set different map	61.79 %

Table with accuracies among different datasets

Phase 3: Building an automatic agent

Once the model (J48) and the training set (dataset 1 with all variables plus the high score one) of approximately 5000 instances were finished, they were imported to our code together with weka in order to use them to predict the next movement of the Pacman.

In order to do that, we had to create a new method in which the instance per tick is updated so the weka function could predict the next movement based on these new attributes, comparing them to our tree and training set.
This task was solved with the getState(self, gameState) function. It is very similar to printLineData, the same conditions (such as converting None to "?" so that weka can recognise them) were included and after, the x line is generated as a list with the variables in the same order as they appear in the training set. An important thing here is to take care of the types of the variables, as they have to be numbers if they are numeric, but they have to be converted into string if they are nominal. Also, the movement Pac-Man has just taken is not included as this is the variable to predict.

Once that new method was created, the chooseAction one was updated so that we call getState to get the x, and the movement comes from calling the predict method from weka with that x, the model (J48) and the training set. Finally, the movement is returned.

However, when the Automatic Agent was programmed we realized that something was wrong, as Pac-Man tried to move to illegal actions and it did not hunt the ghosts at all. In order to overcome these issues, we decided to look in the created model for the decision rules the agent was following to try to find the origin of this strange behavior.
After taking a deep look at the J48 we found out that:

- The tree was giving too much importance to the tick and the score, which are not significant at all to decide which direction to take, as they vary a lot depending on the maps and are only information for the player, not for the agent.

- Something similar happened with the coordinates of both Pac-Man and the ghosts, they vary a lot from different maps and the distances seem more useful.

Having taken all of this into consideration, we decided that we had to start again taking other more useful variables into consideration, so that Pac-Man could really learn where to go from them, and that can be used regardless of the map we are at. This leads to our second approach.

● Second approach

The problems encountered in the first approach were taken into account to create this model.

Phase 1: Instances collection

As the prediction of the movements previously were far from being correct, some drastic changes were made. The coordinates did not provide useful information as a model could not interpret them as well as the tick and score.
To overcome these problems, we thought about the algorithm we used to create the first Automatic Agent in Tutorial 1, which worked perfectly. That algorithm consisted on computing the distances from Pac-Man to the ghosts, taking that Pac-Man moved to one of the directions. Then, the minimum distance from that previous four (one for each direction) is taken and if it comes from a legal action, it is the one we take.

For that reason, we wanted to include the distances to all the four ghosts, having that Pac-Man has gone through one of the four directions, so in total, sixteen new variables are added. Also, the legal actions are kept as we think it is very important that Pac-Man learns that if a legal action is False, he should not take it.

So, the final variables were:

- Legal action North
- Legal action South

- Legal action East
- Legal action West
- ***North ghost 1 (the distance to ghost 1 if we moved to the north)***
- ***South ghost 1 (the distance to ghost 1 if we moved to the south)***
- ***East ghost 1***
- ***West ghost 1***
- ***North ghost 2***
- ***South ghost 2***
- ***East ghost 2***
- ***West ghost 2***
- ***North ghost 3***
- ***South ghost 3***
- ***East ghost 3***
- ***West ghost 3***
- ***North ghost 4***
- ***South ghost 4***
- ***East ghost 4***
- ***West ghost 4***
- Pacman direction
- Future score

Note that if the ghost has been eaten or the Pacman cannot move to that direction, a "?" was appended, as it is a symbol weka can recognise (if not it would be kept as an empty value).

Phase 2: Classification

The same preprocessing as in approach 1 was made:

- Stop direction instances were erased with the filter RenameWithValues .
- We reordered the variables so that the direction was in the last position.
- The future score was removed as it is not used right now.
- Finally, the classes were balanced because the model selected (J48 as it gives the best results) is sensible to unbalanced classes and they were considerably unbalanced.
- The feature selection has been already done in the previous steps (with the inclusion of new variables to keep the distances to the ghosts).

The J48 was made with an accuracy of 68.33% when applying 10 folds cross-validation. Again, looking at the tree leaves we saw the selection of some illegal movements such us:

- Legal action South = False: South (12.87/2.51)

Despite this, we decided to test this model with the test sets of both the same and different maps:

Dataset	Accuracy
Test set same map	54.9414 %
Test set different map	66.493 %

Table with accuracies among different datasets

Phase 3: Building an automatic agent

We modified the getState function to have the new variables we are using in this approach so the list has the correct attributes, and we call once again the chooseAction function with the actual training set and model.

Although the movement seemed to have improved slightly compared to its antecessor, once again Pac-Man was trying again to move to illegal actions.

As a consequence, the game stopped each time an action was done which was not in the legal set of them. To overcome that problem, we included some lines in the chooseAction function where if the selected move is not included in the legal actions list, it takes another random one. However, this solution is not optimal as many times it enters into an infinite loop when there are only one or two legal actions. If the ghosts move randomly, it may go out of that infinite loop when a ghost approaches it (it takes too much time to finish the game), but when the ghosts are still, Pac-Man keeps in that loop.

Keeping this in mind, we decided that this approach could be enhanced if only the distances to the closest ghost are taken into account as we did in Tutorial 1. It is a simplified approach of this second one and could make our model better.

● Third approach

As explained before, a simplified version of the second approach was taken here.

Phase 1: Instances collection

Taking into account that the model did not work fully yet, we decided to simplify even more the variable selection. Now, instead of appending 16 distances (4 from each ghost), only the 4 distances for the closest ghosts were appended. Again, the legal actions were kept as they seem relevant for the agent to learn to not go to an illegal one. Our variables were:

- Legal action North
- Legal action South
- Legal action East
- Legal action West

- **North closest ghost (distance to the closest ghost if we move to the North)**
- **South closest ghost**
- **East closest ghost**
- **West closest ghost**
- Pacman direction
- Future score

Again, if the legal action of that direction could not be taken, a "?" was appended. These changes were, as before, included in the printLineData, which is the function that creates the arff file weka can learn.

Phase 2: Classification

About the pre-processing of this third approach, very similar actions were taken:

- The Stop action was erased in weka.
- The variables were reordered so that the action taken was the last one.
- The future score variable was erased.
- As the model we use is J48 and it is sensible to unbalanced classes, we decided to balance them, as the imbalance in the classes was pronounced.

The generated J48 with 10 fold cross-validation gave an accuracy of 66,58%.

However, again, some leaves in the tree (cross-validation is the one used) were found erratic such as:

- Legal action North = False and North <= 11: North (12.21)

In which it is suggesting to move to an illegal action.

Phase 3: Building an automatic agent

As before, we had to create again the method getState to generate line x as a list of the attributes the training set has, which was again done very similarly to the printLineData created in this model.
However, when trying it, again, some illegal moves were done by the Pac-Man (as expected as the tree included illegal moves), so we found no better results with this model compared to the previous one.

After this, we tried to solve this problem by doing more preprocessing to our dataset. When playing, it is normal that in some ticks the Pac-Man is facing a wall and in those ticks, the generated line produces noise, as the action is illegal but the taken movement rather than being Stop, is the direction the Pac-Man is facing. For that reason, if the Legal action North was False and the direction was North, we would erase that instance. The same with South,

East and West, so the model would not learn from those instances. Despite this, the movement choices did not improve.

Furthermore, we also tried to use the automatic agent (the ones created in Tutorial 1, that provided the most optimal route for Pac-Man to eat the ghosts) to generate the training set instead of the human controlled one in order to have the most accurate dataset and avoid the issues a person can produce playing, such as the generation of that noise when facing a wall because of lack of enough reaction speed. The performance seemed to improve slightly but again illegal movements were found and finishing the game took too many ticks so it was not still a sufficiently good solution.

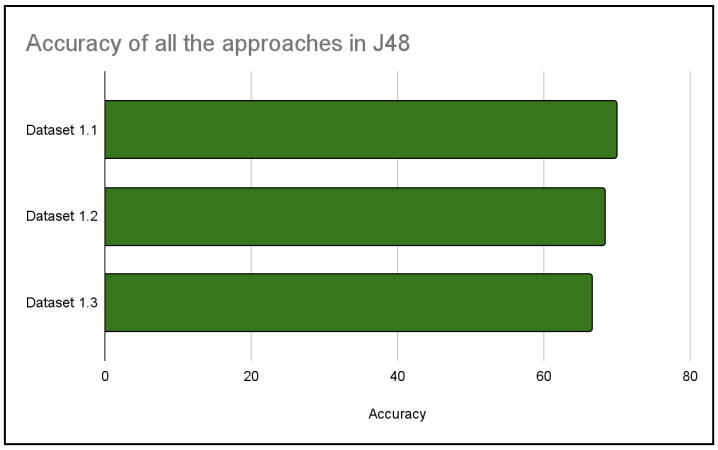
Conclusions from Model 1

Three approaches were tested:

- First approach: all variables plus the new high score one were included, in which we discovered that, contrary to our initial beliefs, variables such as the tick or the score are not really useful to create an automatic agent as they vary a lot from map to map and do not give useful information, but noise.
- Second approach: a completely different approach was taken and the distances to the ghosts taking the movements Pac-Man could take in the tick were taken into account. This seemed to improve the performance but it was not good enough still as illegal moves were taken and it took too much time to finish a game.
- Third approach: a simplified version of the previous one was taken, taking only the distances to the closest ghost as it would eliminate some complexity, but the results did not become much better.

Legal actions were inserted in all the models as these variables are crucial in practice to let the agent learn to take only legal actions.

The accuracies obtained from each model can be summarized in the following graph:



Accuracies of all the approaches in J48

Regarding the algorithm, we saw that the best one, taking that it gives the best accuracy in all datasets, is J48, so this is the one we have used through all the different approaches. Also we selected it because the results can be understood easily as it gives a tree we can follow to see what the agent is doing. We have tried it with cross-validation and two different test sets (same map and different maps), but the selected one to use in the automatic agent is the cross validation.

As seen, cross validation testing accuracy was almost the same for the three approaches. Furthermore, for all models, we had to add the fact that when receiving an illegal action, we had to choose an alternative one randomly although this sometimes leads to infinite loops.

The **second approach** was the most fluid when it came to movement so it was decided to be kept as the definitive model from this first objective, which is to make the most efficient autonomous Pac-Man.

As the second approach was the selected one, we decided to try playing some games to see its efficiency.

- Default map with random ghosts (the map we created in the tutorial) - 216 ticks to finish.
- Six hunt with random ghosts - 765 ticks.
- Small Hunt with random ghosts - 228 ticks.

After seeing the behavior among these different maps, it has been proved that it mainly has problems when there are very few legal actions available, that is when it keeps in infinite loops until some ghost approaches it.

Phase 4: prediction (second approach)

In this phase, instead of predicting the next move, we have to predict the future score, which is a numeric attribute so instead of classification algorithms we use regression ones, based on the actual one and the movement Pac-Man is going to take.

As previously stated, the dataset we have chosen from model one is the second one, which has as variables the distances to the ghosts with the possible movements of Pac-Man.
As preprocessing steps we had already done the feature selection, so the only things we need to do so is placing the future score as the last column and erasing once again the instances with Stop as taken action.

In order to decide a model, we had to check different regression models. We used random forest, which is a model that creates trees and computes the mean to see how well they perform according to some parameters; linear regression, which approximates a linear

function between the predictors (variables) and the variable to predict; and M5, which generates a set of if-then rules and can deal with every type of variable.

In regression, accuracy is not used for testing the quality of the model. Instead, other measures such as the MSE (Mean Squared Error) or the correlation coefficient are provided. The correlation coefficient is a number among -1 and 1 which measures the linear relation among the independent variables and the target one. A number close to 1 will tell us that there is a positive linear relationship while a number close to -1 a negative one. Furthermore, a number close to 0 will tell us that there is no linear relationship at all.

When performing the tests in weka, it gives as results the correlation coefficient as well as different ways of measuring the error, but we are going to check the previously explained two, as they are very representative.

Their correlation coefficients and MSE were tested with 10 fold cross validation and both test sets (same and different maps) in the different models:

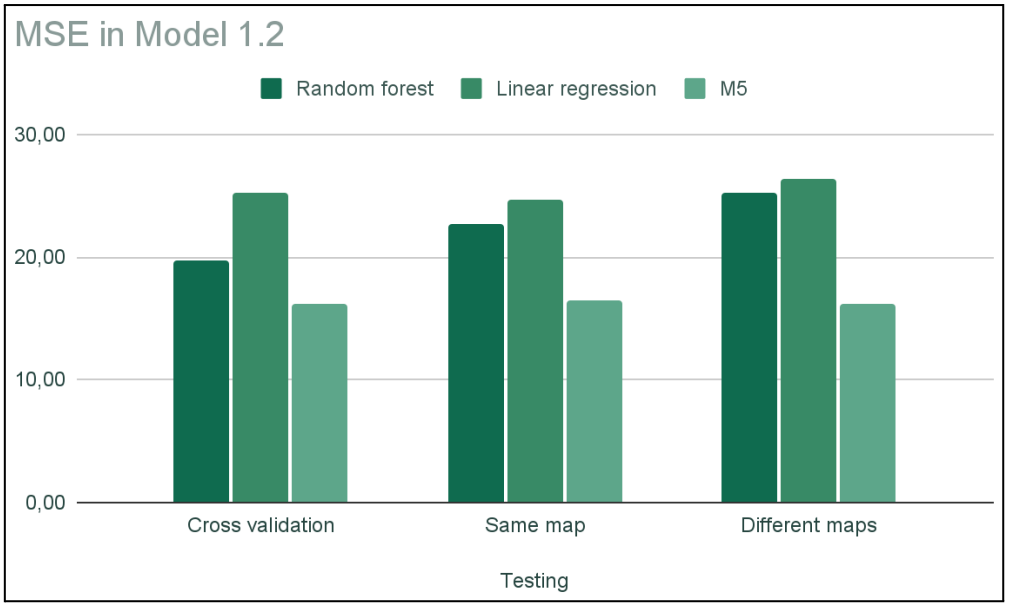
Testing	Random forest	Linear regression	M5
Cross validation	0.9843	0.9752	0.9838
Same map	0.9807	0.9728	0.9825
Different maps	0.9809	0.9755	**0.9847**

Correlation coefficient of model 1.2 among different regression techniques and testings

Testing	Random forest	Linear regression	M5
Cross validation	19.7701	25.2604	**16.1453**
Same map	22.7234	24.6279	16.4232
Different maps	25.2914	26.298	16.1988

MSE of model 1.2 among different regression techniques and testings

Based on these results, we can see that the correlation coefficients in all models and tests are very close to 1, which suggests a strong linear relationship between the predictors and the variable to predict, in this case, the future score.



MSE of the tested models in model 1.2

When taking a look into the Mean Squared Error with the help of the above graph, it can be clearly seen that M5 is the model that gives the lowest value, which suggests that this is the best model among the three tested ones. Also, this is the one that gives the highest correlation coefficient. On the other hand, the linear regression is the one with the worst results, having the highest MSE among the three.

Model 2

Our main objective with the second model was to maximize the score of the game, which gave us the following idea: what if we added the dots to our gameplay? Dots are an important part of the Pacman game because the score can be maximized when taking them into account. So, the idea was to treat them as the ghost agents. This is also a good idea because we know there are four ghosts per game, but we do not know how many dots there are, so appending only the distance to the closest dot (if there is none just add "?") solves this problem.

Phase 1: Instances collection

The variables used in this model where:

- Legal action North
- Legal action South
- Legal action East
- Legal action West
- North closest ghost (distance to the closest ghost if we move to the North)
- South closest ghost
- East closest ghost

- West closest ghost
- **North closest dot (distance to the closest dot if we move to the North)**
- **South closest dot**
- **East closest dot**
- **West closest dot**
- Pacman direction

A main problem was encountered for finding the coordinates of the pac dots, as there is no concrete attribute for this.

In order to find that information, which is saved in a matrix (list with the coordinates of the positions), we created a grid (with the getFood function) where if there is a pac dot in that position, there is a T and if not, an F. Then, if you try to get a line it takes it in the wrong way, taking a column in reverse order. To fix that, the lines of the grid are copied in a matrix, but transposed. Then, the rows of the matrix are checked, and if in a specific position there is a True, that position is appended to the pacdots list.

Phase 2: Classification

The preprocessing applied in this case is very similar to the ones computed before:

- The future score is erased, as well as the current one.
- The Stop action is erased too
- The classes are balanced in order to use J48

The accuracy of the decision tree with cross validation was 56,5991%.
Furthermore, when testing it with the test sets:

Dataset	Accuracy
Test set same map	47.7569 %
Test set different map	51.9762 %

As observed, the accuracy with respect to model 1 has decreased. Despite this, this model is expected to help us better to predict the score as the pac-dots, which were not taken into account before (and are a source of extra points), now are.

Phase 3: Building an automatic agent

In the same way as model 1, the getState function was modified to contain the new generated variables and the predict method from weka is called.

However, the results did not improve much and again illegal actions were performed. That is why, once again, the random movement had to be added so that Pac-Man did not perform an illegal action.

Let's see the performance in some maps:

- Predetermined map with random ghosts: 318 ticks.
- Small hunt with random ghosts: 204 ticks.
- Big hunt with random ghosts: 308 ticks.

This model seems to be more efficient, it plays better, takes less ticks and seems more intelligent.

Phase 4: Prediction

The same way as it was done with model 1, now we have to focus on predicting the future score, for which we have to use regression models instead of classification ones. The models used are Random forest, linear regression and M5.

Again, we are going to study the correlation coefficient and the Mean Squared Error.

The given results are:

Testing	Random forest	Linear regression	M5
Cross validation	0.9882	0.9853	0.9873
Same map	0.9891	0.9873	0.9894
Different maps	0.9878	0.9873	**0.9899**

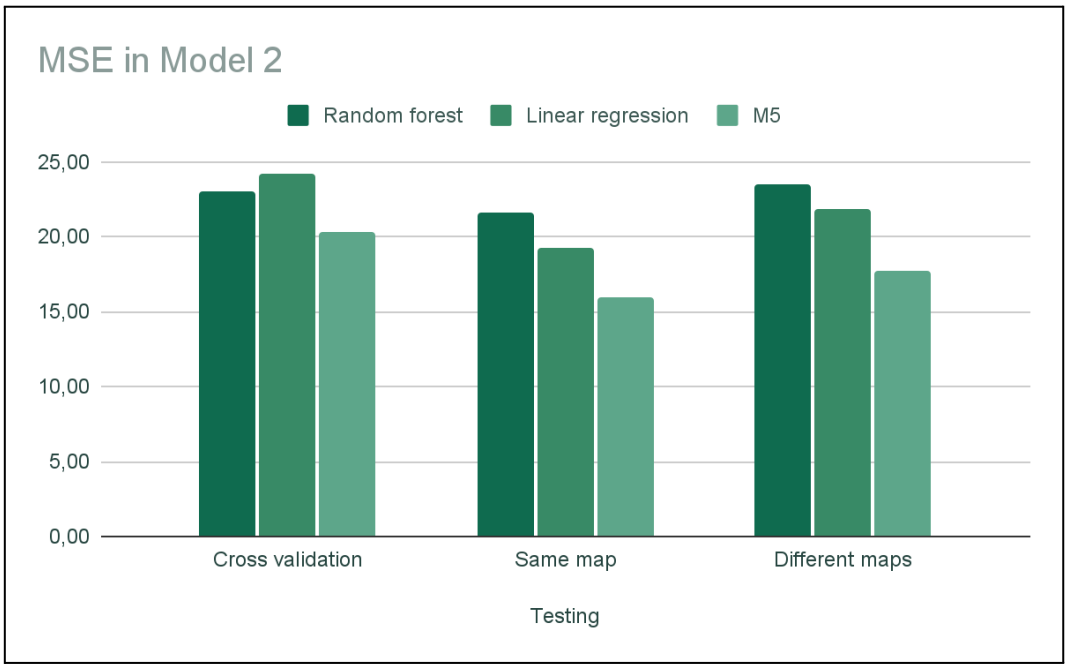
Correlation coefficient of model 2 among different regression techniques and testings

Testing	Random forest	Linear regression	M5
Cross validation	23.0025	24.2528	20.2822
Same map	21.6298	19.3171	**15.9728**
Different maps	23.5595	21.8433	17,7016

MSE of model 2 among different regression techniques and testings

Analyzing the results, it can be seen that again the correlation coefficients are very close to 1, which suggests a linear relationship once more. Also, M5 provides the highest results among all different tested models.

Let's compute the graph for the MSE to see it better.



Bar graph that represents the MSE for model 2

The same that happened in model 1 happens here, M5 provides the best results as the Mean Squared Error is the lowest, so it is the model that best predicts the future score.

Also, when comparing the results from model 1.2 and model 2, it can be seen that this second one has a lower MSE value and a higher correlation coefficient so, as expected this model predicts better the future score. This proves the hypothesis we already had that adding the pac-dots would enhance the probability to better predict the score.

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 between both approaches is the source of the data. When coming from a human controlled agent, the data is created by a person manually. In this case, it would be by creating the data using the keyboard agent of pacman. This method is more demanding as the person should try to generate the best quality data (playing Pacman in the best way) and when it comes to having a large dataset, the time needed to create it can be demanding. Also, it can include human errors, such as the ones we encountered when facing a wall that can make the agent learn wrongly how to play, suggesting illegal actions.

When an automatic agent is used, the data is generated automatically, which would be equivalent here to using the algorithm we have previously created in Tutorial 1 to move the

Pacman in the most efficient way. As a consequence, it can be more useful when large amounts of data are required and it consists of repetitive tasks. It is less consuming for a person and should provide less human errors, which can enhance the performance.

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?

Classification and regression are two types of supervised learning which mainly differs on the type of output received when predicting. In classification the output variable is categorical or numerical discrete and in regression the variable is numerical continuous.

So basically, the first thing we would need to focus on is to define discrete classes or categories for a numerical variable. This would be equivalent to setting different ranges of values per class in the output variable.

This method has been used for example, when the creation of High_score was done. If the score was higher than 620, a True was added, while if it was less, a False. Again, there are other implementations which could be useful such as dividing the score in High, Medium or Low.

This can be helpful to simplify our model and make it more interpretable. Furthermore, due to having classes, testing the imbalance of the output variable can be done. Also, there are many algorithms that only can predict categorical variables, so they would be available.

Predicting the score can be useful because we know that when Pac-Man eats a ghost or a pac-dot the score is increased, so that can help to make a more efficient game as if we can predict that taking a specific move the score will be increased, we know that that movement is the optimal one. Also, as we want to maximize the score, this also can be helpful.

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

The main difference encountered was the huge increase in accuracy when predicting the score and testing the quality of the models.

Also, another difference found for predicting were the models used. For the action, a classification task was being done, so the appropriate models were used, such as J48 and IBK, and the same happened with predicting the score, which was a regression task, in which only a few models fitted such as random forests, M5 and linear regression.

Furthermore, when it comes to predicting a continuous numerical variable, a more flexible prediction can be done than when the target variable is a categorical feature. For instance,

misclassifying a category may provoke a higher error than predicting a numerical variable a few units from the real value. Due to this fact, more accurate predictions may be obtained.

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 a new attribute about the decrease of the score may be misleading for the model and produce noise.

Firstly, the score in Pac Man only depends on the player and may increase if ghosts or dots are eaten. The score may fluctuate over time as there are several ticks when the Pac Man is moving and it does not eat anything. In this case, losing a score does not tell us anything about the ability of the player.

Again, the score at a particular moment does not tell us anything about the player's skills and as a consequence about the possible score the player may have.

Concluding, this attribute may not contribute to the quality of our model while many others such as the speed or obstacles could help more. A temporary drop may not be relevant for predicting the ranking of the Pac Man.

This conclusion also comes from the problems we encountered from the generation of our model 1, as our first approach included the score and could lead to irregularities in some leaves.

Conclusions

This practice was very complete in the sense that all the steps of a machine learning task had to be computed.

Firstly, the generation of instances based on some variables we thought could be useful to solve the task we were given, in this case, generate an automatic Pac-Man that could learn by itself had to be done. In order to predict that, it was compulsory to include the movement Pac-Man has taken, as it is the variable to predict, and the future score for phase 4. This task was done modifying the printLineData function in Python so that it generated an .arff file weka could learn, for what some aspects such as the empty values had to be taken into account because they had to be changed into “?”. Also, this led to the erasing of the last line of each game because the future score remained blank at that moment. The header was also important because the types of the variables had to be defined there.

We have really seen the importance of this first step as in our first approach, that included many variables we thought to be useful, some were proven to be useless and generated noise such as the tick, the score or the coordinates. For this reason, we had to change our vision and make a different approach with other variables that seemed to be more useful based on the algorithm we created in Tutorial 1. These were based on the future distances to the ghosts with the different movements Pac-Man could take.

After the collection of the variables, data preprocessing had to be done, what includes erasing noise (instances or attributes (trying different subsets of the variables)) where we had to erase the lines with the Stop action, as they did not provide useful information; creating new features that could include useful information for the agent to learn, normalizing or balancing classes, which we did with the direction as the classes were unbalanced.

Once the best selected set of variables was found, different models had to be tested using both 10 fold cross-validation and test sets (that come both from the same map and different maps). Firstly, in order to predict the next move, some classification algorithms were used such as J48, IbK or PART. After the evaluation, the model that proved to be the most useful was J48, as it provided the best accuracy and the interpretation was easy (it generates a tree) so that we could understand the results and what our model was doing.

Once the training set and the model were selected, the next step was to generate the automatic agent based on them. To do that task, a new function was included in Python (getState) in which the line with the selected attributes was included taking care of the types of the variables for each tick. With this method, that generates the actual line (x), the data set and the model, the predict function in weka gives as a result the next movement to take.

However, trying this we realized that our agent wanted to move to illegal actions, so we added a randomizer in which if the movement is not legal, it takes another one. Nevertheless, it is not an optimal solution as sometimes it gets stuck in an infinite loop.

Finally, the last step is to predict the future score. Again we preprocessed the data but now, the algorithms had to be for regression, as the variable is numerical. After trying several ones, the one that gave the lowest error and highest correlation coefficient was M5, so this is the best one.

Regarding the generated models, we wanted to create two, each one of them with its own objective. The first one had as a goal to generate an agent that could play the most efficiently as possible, eating the ghosts fast; while the second one's goal was to maximize the score by also eating the pac-dots.

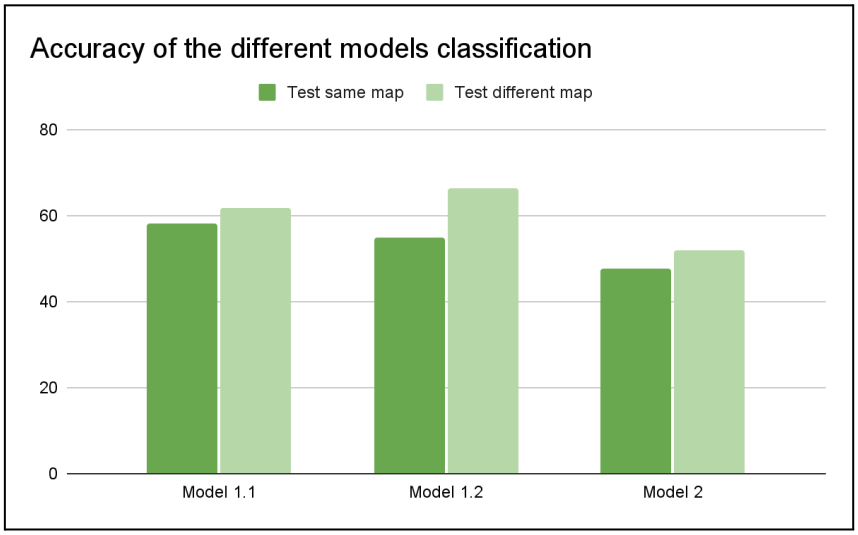
For the first model, as explained before, the first approach which had all variables from Tutorial 3 was proven with the J48 to have given too much importance to non explanatory variables, so we changed the approach to the second one with the distances to the ghosts, and erasing the previous variables except the legal actions. Although the results were improving,

we took a third approach where only the distances to the closest ghost were included, but it did not improve so we took as a final result the second approach of this first model.

For the second model, the pac-dots were included treating them the same way as the ghosts, so as variables we had the distances to the closest ghost, and pac-dot, and the legal actions.

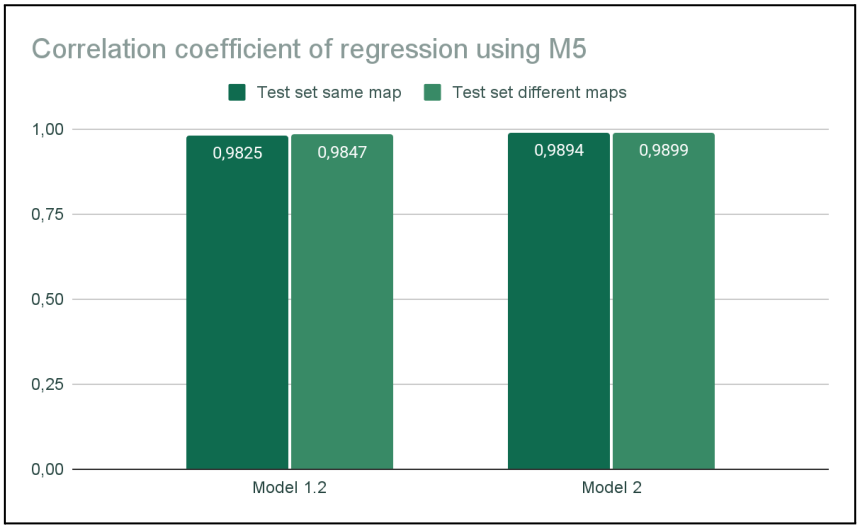
The performance of this last model seemed to be the best as the movements of the Pac-Man are more fluent and it takes less ticks to finish. However, the accuracy is better for the first approach. Also, the regression models give very similar results that seem better for the second approach, as a source of points such as the pac-dots are taken into account.

The final accuracies can be seen summarized in the following graph:

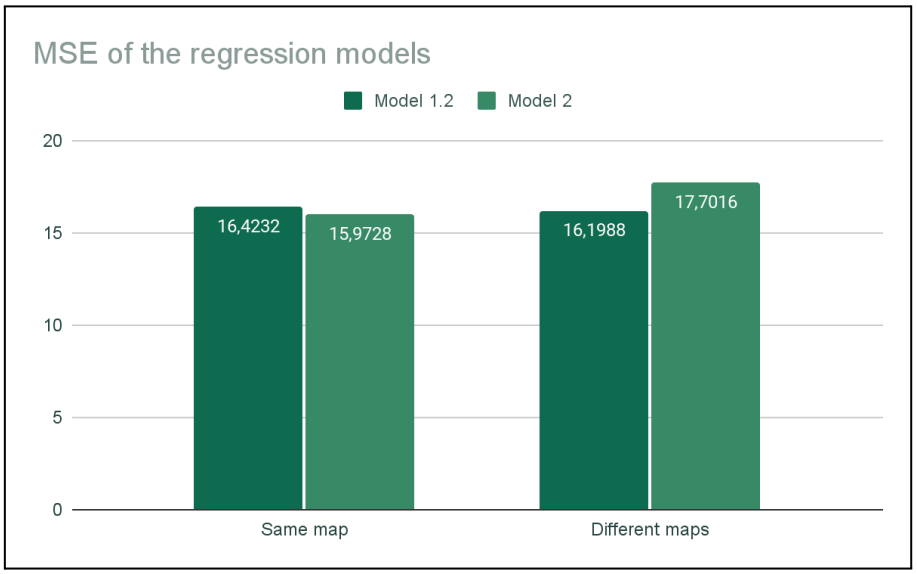


Accuracy of the different models obtained

The correlation coefficients and MSE are summarized in the following graphs:



Correlation coefficients for the regression models



MSE of the regression models