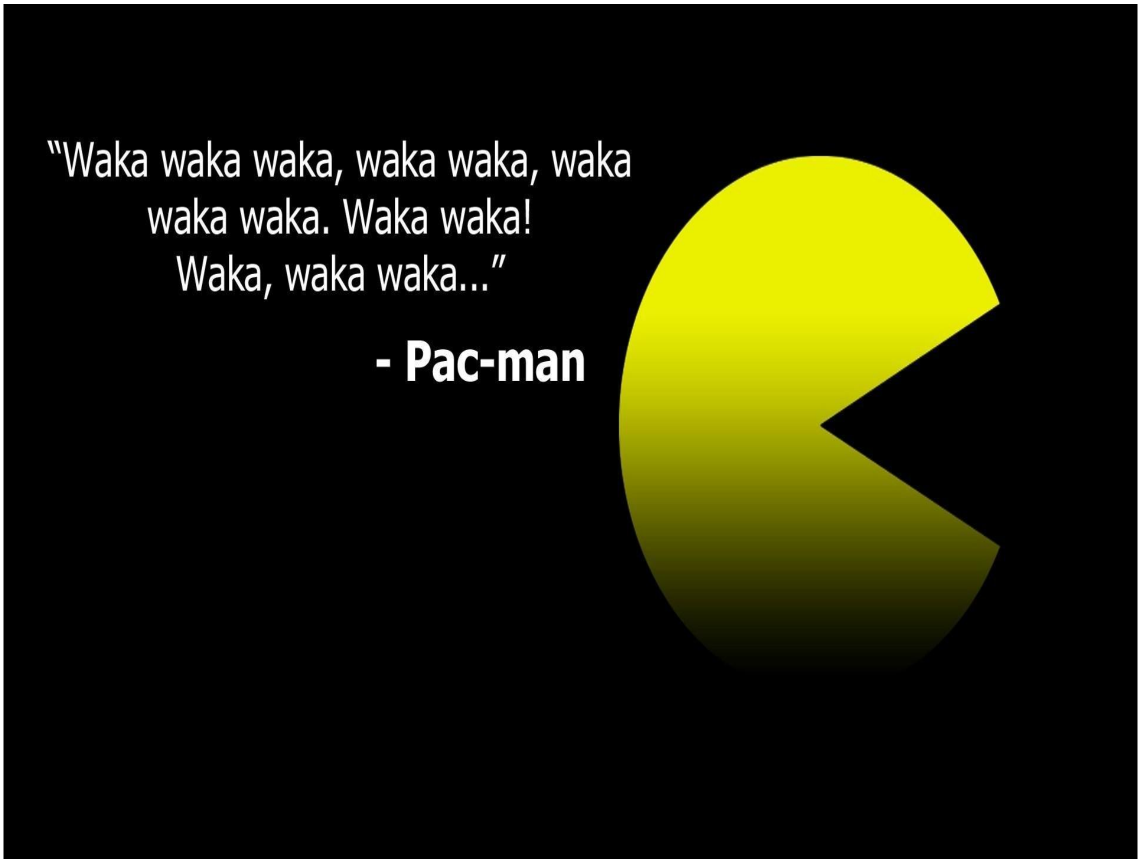
# MACHINE LEARNING TUTORIAL I

# CARLOS III UNIVERSITY

DATA SCIENCE AND ENGINEERING BACHELOR DEGREE



ALEJANDRO PÉREZ, 100429952
CHRISTOPHER MANZANO, 100429927

APRIL 9TH, 2021

# Introduction

We aimed to create a machine learning based agent that could solve the problem of hunting the ghosts at the game pacman. In order to implement our agent we applied the crisp dm methodology which was developed along the different phases of our practice.
We simplified the hunting problem as a classifying problem in which we want to predict the correct movement in order to hunt the ghosts.

For this classification task we used as training set the outputs given by the KeyboardAgent and the BasicAgentAA from tutorial 1. We remark that for this task we created a new BasicAgentAA which could solve all the maps (in contrast with the original one which tended to get stacked in some situations).

We arrived at the conclusion that we needed more instances and a better train set in order to make the classifier work. Besides, we discovered that the new agent of tutorial 1 we programmed was not the best option.We concluded it was better to use our original programed agent. We will explain these results in advance in phase 4.

We remark that in our standard all the different BasicAgentAA we implemented are only one agent. Which has different behaviors. All these behaviors are methods inside the BasicAgentAA. Thus, our original first tutorial programmed pacman is implemented in the behavior1 method. In addition, our new first tutorial programmed pacman is implemented in the behavior2 method. Our machine learning agents were implemented as another behavior method in the BasicAgentAA. Finally, the machine learning agent we reached is implemented on the behavior3 method.

## Phase 1: Instance Collection

### PrintLineData() function

We implemented the *printLineData()* method in the class BustersAgent so that the other classes could inherit it. In order to get the future attributes, we added the attribute *lastGameState* to the *BustersAgent* class. This attribute saves the current *gameState*. Our new *printLineData()* returns a text file with *.arff* extension containing the information from *lastGameState* (*pacX*, *pacY*, *score*,..) plus the score of the actual *gameState* (*nextScore*).

Thus, we are printing data from the past together with the score of the present. This way we lose the first and last instances at every execution, but we get to print the future tick information in return.

### Attribute selection

In the first part of this project, we tweaked our *printLineData()* function as we just described to create text files with the *.arff* extension for *Weka* to understand. We then selected the variables shown in the table below. We found these to be the most relevant to work with in *Weka*:

| **Attribute** | **Type** | **Description** |
|---|---|---|
| *pacX* | Numeric | Pac-Man's position on the *x* axis |
| *pacY* | Numeric | Pac-Man's position on the *y* axis |
| *LegalNorth* | {1,0} | Whether Pac-Man can go *North* (1) or not (0) |
| *LegalSouth* | {1,0} | Whether Pac-Man can go *South* (1) or not (0) |
| *LegalEast* | {1,0} | Whether Pac-Man can go *East* (1) or not (0) |
| *LegalWest* | {1,0} | Whether Pac-Man can go *West* (1) or not (0) |
| *Gosth1x* | Numeric | Ghost 1's position on the *x* axis |
| *Gosth1y* | Numeric | Ghost 1's position on the *y* axis |
| *Gosth2x* | Numeric | Ghost 2's position on the *x* axis |
| *Gosth2y* | Numeric | Ghost 2's position on the *y* axis |
| *Gosth3x* | Numeric | Ghost 3's position on the *x* axis |
| *Gosth3y* | Numeric | Ghost 3's position on the *y* axis |
| *Gosth4x* | Numeric | Ghost 4's position on the *x* axis |
| *Gosth4y* | Numeric | Ghost 4's position on the *y* axis |
| *DistGosth1* | Numeric | Pac-Man's distance to Ghost 1 |
| *DistGosth2* | Numeric | Pac-Man's distance to Ghost 2 |

| | | |
|---|---|---|
| *DistGosth3* | Numeric | Pac-Man’s distance to Ghost 3 |
| *DistGosth4* | Numeric | Pac-Man’s distance to Ghost 4 |
| *GhostDir1* | {North,South,East,West,Stop} | Ghost 1’s movement direction |
| *GhostDir2* | {North,South,East,West,Stop} | Ghost 2’s movement direction |
| *GhostDir3* | {North,South,East,West,Stop} | Ghost 3’s movement direction |
| *GhostDir4* | {North,South,East,West,Stop} | Ghost 4’s movement direction |
| *NearestFoodDist* | Numeric | Pac-Man’s distance to the nearest Pac-Dot |
| *NumOfFood* | Numeric | Number of Pac-Dots remaining on the map |
| *Score* | Numeric | Current Score |
| *NextScore* | Numeric | Score in the next tick (predicted) |
| *Direction* | {North,South,East,West,Stop} | Pac-Man’s movement direction |

## Data collection schema

| **Map** | **Training** | **Test (maps from training)** | **Test (other maps)** |
|---|---|---|---|
| *oneHunt* | 1 000 | 400 | - |
| *classic* | 1 000 | 400 | - |
| *bigHunt* | 1 000 | 400 | - |
| *trickyClassic* | 1 000 | 400 | - |
| *newmap* | 1 000 | 400 | - |
| *trappedClassic* | - | - | 400 |
| *20Hunt* | - | - | 400 |
| *capsuleClassic* | - | - | 400 |
| *smallHunt* | - | - | 400 |
| *openClassic* | - | - | 400 |

Note: The number of entries for every data set is not 100% exact.

Note: The training data size of 1 000 comes from 500 lines from the keyboard agent plus another 500 from the agent in tutorial 1.

## Phase 2: Classification

### Data Analysis

We created different versions of the datasets in order to compare which one gave the best result. We arrived at four initial versions: original dataset, alpha version, beta version, and gamma version.

*alpha*

- We removed the attributes related to the score and the food since we consider they are not useful in our task.
- We created new attributes: *meanghostx*, *meanghosty,* and *meandist*, which are the averages of the positions of the ghost in x, the positions of the ghosts in y, and their distance to Pac-Man. Since we consider that the mean could be useful to make Pac-Man go for the ghosts which were more close to each other.

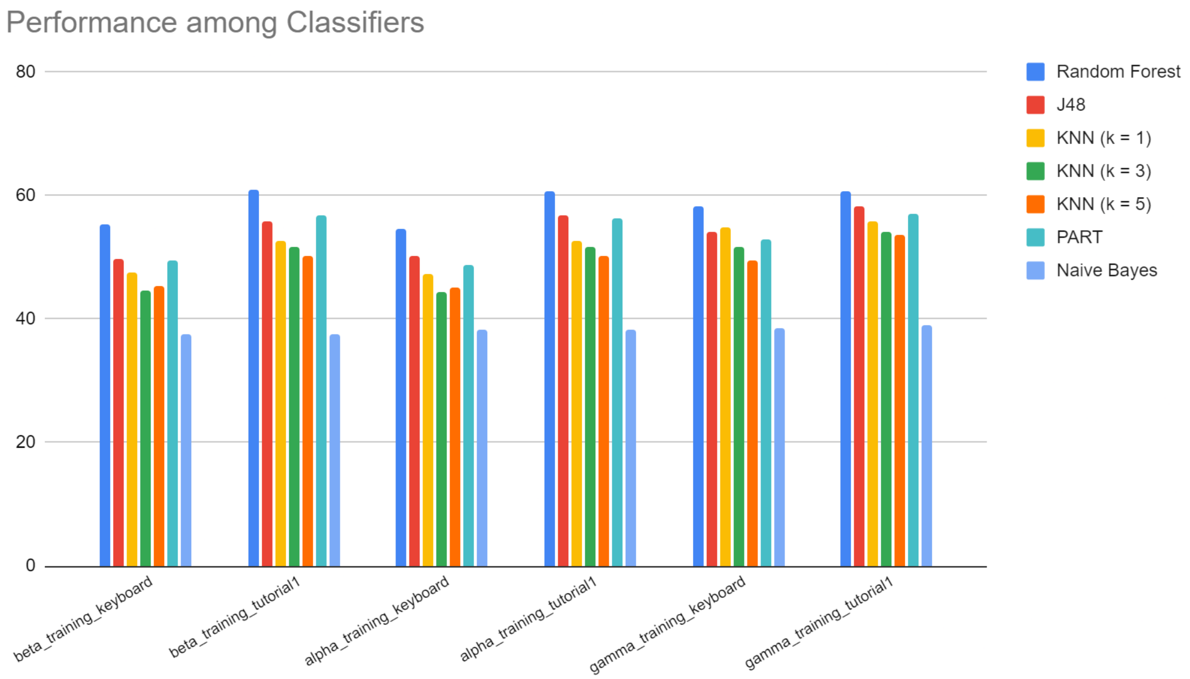
*beta*

- We created the attributes meanghostx and meanghosty to express the place where the ghosts are less spread.
- We created the attribute meandist for expressing the mean distance to all the ghosts.

*gamma*

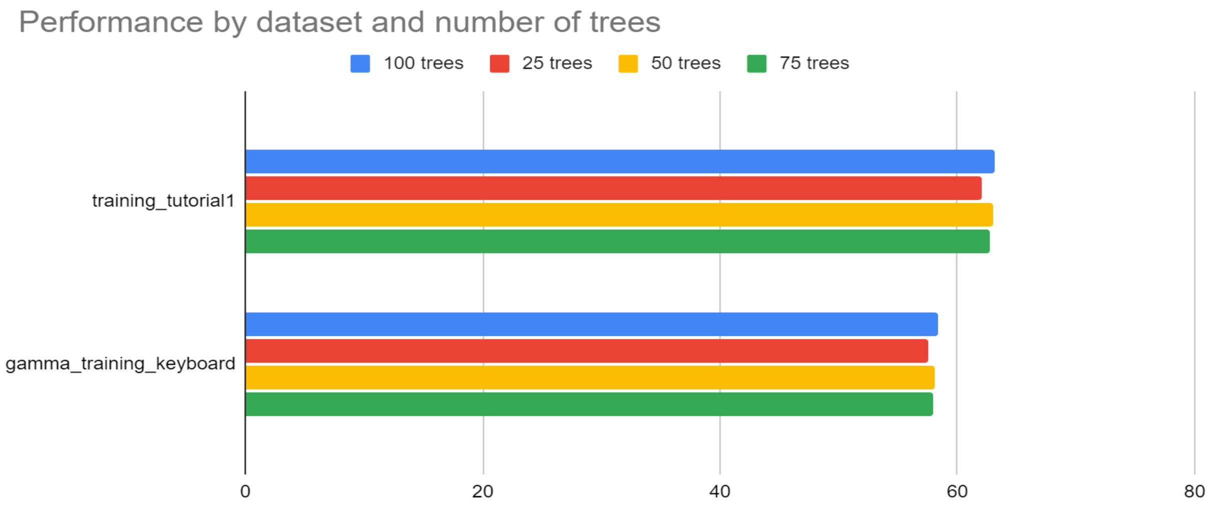
- We removed the ghosts' directions.
- We removed the created attributes.
- We removed the ghosts' distances.

We used ZeroR as our main algorithm to compare against J48, random forest, PART, Näive Bayes, and IBK with 1, 3, and 5 nearest neighbors. For this comparison, we used a cross-validation t-test with 10 folds over all our training datasets' versions. Finally, we got the following results:



These results give us the intuition that a random forest is the best option for our classification task. Now we aim at fixing the dataset we are going to use. And we can see that our best option is the original training_tutorial1. Also, we are going to keep the gamma version of training_keyboard (since its performance is not significantly different from the original version) in order to compare both of them in practice given the fact that a good accuracy does not imply a good performance at solving the problem.

Finally, we tried a different number of trees in a random forest in order to maximize the accuracy:



We are going to keep 50 as the number of trees since there is not a significant difference with 100. As a conclusion, we decided that our optimal configuration is using the beta and gamma versions from *training_tutorial1* and *training_keyboard* respectively in a random forest with 50 trees.

Finally, our results (accuracy) following this setup with the test sets are:

| **Dataset** | **Same Maps** | **Other Maps** |
|---|---|---|
| *beta-tutorial1* | 42.8125 % | 43.7564 % |
| *gamma-keyboard* | 49.8958 % | 34.6425 % |

## Phase 3: Prediction

For this phase, we modified the versions of the dataset in order to make a regression:

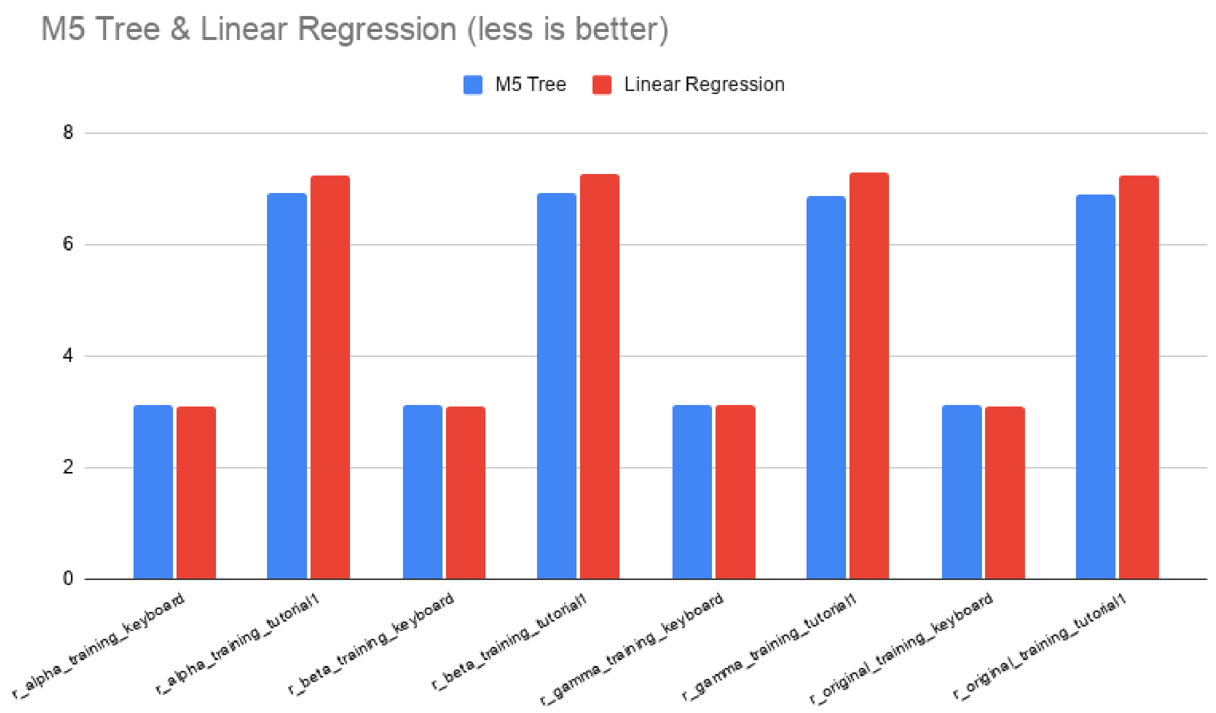
*r_original*
- Interchanged Pac-Man direction and next score

*r_alpha*
- We removed the ghost directions

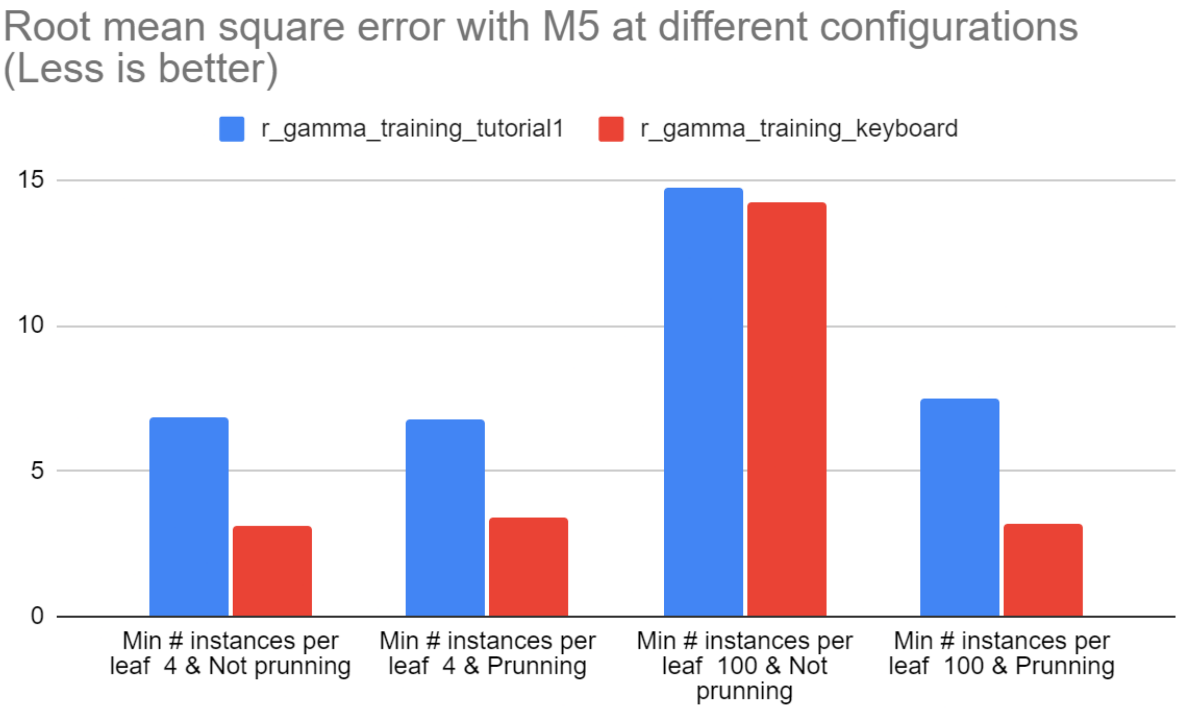
*r_beta*
- We removed the Pac-Man legal actions

*r_gamma*
- We removed the data related to the position of all the agents

We are going to compare the performance of the regression with M5 and linear regression with a cross-validation t-test taking as a comparison parameter the root mean square error.



Now we try to determine the perfect configuration of the M5 algorithm. For this task, we aim to compare the minimum number of instances required for getting a leaf and whether prune or not.



We conclude that we have to keep M5 without pruning and whit a minimum number of instances per leaf around 4. In this case, we decided to choose 4.

Finally, with this configuration, we obtain the following results (root relative squared error) with the test sets:

| **Dataset** | **Same Maps** | **Other Maps** |
|---|---|---|
| *r_gamma_tutorial1* | 6.8127 % | 18.1024 % |
| *r_gamma_keyboard* | 4.886 % | 10.1109 % |

## Phase 4: Building an automatic agent

Our chosen classifiers from phase 2 were two randomForest with different training sets and different numbers of trees.

After finally connecting weka and using our two models we discovered that our classifiers are potatoes. They did not accomplish the hunting task at all. Their set of decision rules make them easily reach a state where the output is an illegal movement.

This issue is unfixable with the ghosts at static since all these patterns are deterministic. We tried to output a random movement and a movement given by the behavior1. But this ends up repeating over and over. With ghosts at random this issue is a bit less important. However it also appears.

We remark that the agent trained with the programmed pacman has learnt patterns of behavior. There are moments in which the agent tries to overcome an obstacle which is not present in the map. For instance, in the map oneHunt, the agent follows the strategy of newmap in order to find the entrance to the base of the ghosts. We consider this curious.

At the end our hypothesis is that the keyboard agent and the programmed agent are too complex to be inferred by the random forest algorithm. At least with the current attributes we are using for training.

On the one hand, the keyboardAgent set of decisions depends on us. We usually try to maximize efficiency by choosing the right path and the right order to eat each ghost based on the set of inputs we receive (which is quite larger eg. the whole map perspective). On the other hand, the new programmed python also needs information about the map. In addition it does not have a hunting criteria. It hunts in the order we want it to hunt.

As a consequence we think that a lot of the decisions the agents take along the map do not reflect a pattern but a stochastic behavior. In other words, the data we had was not enough. Neither in quality or quantity.

## Phases 2 and 4: Remastered

Having concluded phases 3 and 4 we felt somewhat underwhelmed. Thus, inspired by some models we were shown in class, we decided to redo this last part for the sake of trying to comprehend how the model is trained and also to try to get Pac-Man to solve a few more maps. We considered the following attributes:

| **Attribute** | **Type** | **Description** |
|---|---|---|
| *pacX* | Numeric | Pac-Man's position on the *x* axis |
| *pacY* | Numeric | Pac-Man's position on the *y* axis |
| *LegalNorth* | {1,0} | Whether Pac-Man can go *North* (1) or not (0) |
| *LegalSouth* | {1,0} | Whether Pac-Man can go *South* (1) or not (0) |
| *LegalEast* | {1,0} | Whether Pac-Man can go *East* (1) or not (0) |
| *LegalWest* | {1,0} | Whether Pac-Man can go *West* (1) or not (0) |
| *Gosthx* | Numeric | Position on the *x* axis of the closest ghost |
| *Gosthy* | Numeric | Position on the *y* axis of the closest ghost |
| *DirToGhost1* | {North,South,East,West,Stop} | Best move to reach Ghost 1 according to behavior 1 |
| *DirToGhost2* | {North,South,East,West,Stop} | Best move to reach Ghost 2 according to behavior 1 |
| *DirToGhost3* | {North,South,East,West,Stop} | Best move to reach Ghost 3 according to behavior 1 |
| *DirToGhost4* | {North,South,East,West,Stop} | Best move to reach Ghost 4 according to behavior 1 |
| *Move* | {North,South,East,West,Stop} | Pac-Man's movement direction (value to predict) |

Our idea was to create several behaviors using weka models, as well as keep other models from previous tutorials (eg behavior 1), plus one more behavior in charge of selecting the best behavior to follow according to the map we're playing. We split our data collection into three types of files, depending on the size of the map (large, medium, easy). For each type of map we recorded training sets, as well as test sets on both the same maps and other maps. We selected about 4.500 instances for every training set, as the number of instances we used last time wasn't nearly enough to produce accurate results. We chose three maps for every training set and also those same 3 for a test set as well as another test set with three more maps.

After the data collection, we opened the various arff files in weka to measure their performance with J48 as the chosen classifier. We also tried random forest but the results were quite similar, so we decided to stick to J48. Here's the table of performance we got for every file:

| **Filename (training)** | **Test options** | **Accuracy** |
|---|---|---|

| | | |
|---|---|---|
| *training_largemaps.arff* | Cross-validation (10 folds) | 99.6256 % |
| *training_largemaps.arff* | *test_largemaps.arff* | 97.9021 % |
| *training_largemaps.arff* | *test_otherlargemaps.arff* | 91.1133 % |
| *training_mediummaps.arff* | Cross-validation (10 folds) | 98.4065 % |
| *training_mediummaps.arff* | *test_mediummaps.arff* | 98.0703 % |
| *training_mediummaps.arff* | *testing_other_medium_maps.arff* | 95.8017 % |
| *training_easy_maps.arff* | Cross-validation (10 folds) | 98.3444 % |
| *training_easy_maps.arff* | *testing_same_easy_maps.arff* | 97.5847 % |
| *training_easy_maps.arff* | *testing_other_easy_maps.arff* | 99.5023 % |

As we can see, all the results are quite astonishing! Far better than our previous 60% accuracy. We saved one model for every type of map (large, small, easy) and implemented the various behaviors into our basicAgentAA method. We have a total of 6 possible behaviors: two of them are from previous tutorials, three are the predictions made by weka depending on the type of map, and one more that was in charge of selecting the best behavior to choose depending on various parameters, but we didn't have time to implement that part. Neither we thought it was needed given our astonishing results.

We tried the performance of the newly-created models from weka and pacman's behavior was very nice. We were quite proud of these models, as they all did very well and were quite efficient in solving most maps. Of course since our models were trained using mainly the rules from behavior 1, our poor Pac-Man still got stuck in the same places as our old behavior 1.

Nonetheless, having trained this model with weka was quite challenging yet fun. We noticed how Pac-Man's behavior was even better than behavior 1 in some cases, and rarely behaved in an unexpected way. We are really looking forward to diving deeper in better ways to make pacman solve every map efficientñy in future projects.

## Questions

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

We think that the instances given by a human are more complex. The decisions take into account data and logic that may not be possible to infer from the attributes that we selected for training our algorithms. Thus we think that for our algorithms a lot of the attributes for training were behaving like random variables. This makes the task of training have a lot of noise.

### 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 could use the result given by the regression model as a new attribute in our classification task in order to make it classification. Predicting the score could be useful for having into account the idea of not only hunting the ghosts but also doing it in the best possible way.

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

The prediction of the score does not depend on a lot of the attributes that we used as a training set as we can see in the phase 3.
Also this task required less computational cost.
Finally a regression is a little more easy to predict given that by definition a regression line takes into account an error.

### 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?

We think that not since our rules for pacman from tutorial 1 do not take into account the score, neither we did while creating the training set.
Furthermore, since our goal is to catch the ghosts, the score is expected to increase only when we catch them and to decrease otherwise (we are not taking into account the food). Thus, if we implement it, we'll have 4 increases and all the other decreases per game. Which do not add any further information.

# Conclusions

**Technical insights**
We have aimed to build an intelligent model which could solve all the maps. In order to do this we approached this task as a classification problem where we had to classify the correct direction pacman should take in order to hunt the ghosts. We discover that this approach is not that accurate since a good classification rate does not mean that pacman will hunt the ghosts. We discovered that after testing our pacman and seeing that it behaves worse than expected. In addition, we discovered that we should have included more useful training attributes. Our pacman from tutorial1 takes into account the walls and we too when played with the keyboard agent. Thus we suspect that since it does not have any information related to that… Some examples might appear as random decisions at the moment of classifying.

**General insights**
Sadly we are not really proud of our results. We think we could have done better if we had taken into account more useful attributes in this classification task. Since we consider it was a mistake of us that the classifier does not perform well, we consider that approaching a lot of different things as classification tasks is a good strategy for different machine learning applications such as betting or not betting in a betting house, giving or not the job to an applicant… among others.

**Problems encountered**
We encounter a series of problems such as a bad model using our original tutorial 1 agent. So we decided to build another agent for creating the training instances.Conversely, this agent resulted in even worse results at trying to solve the problem. We think it may happen because this agent uses information related to the walls and takes as parameter the order in which we want to hunt the ghosts.

In addition we had a lot of technical difficulties with the weka wrapper, the java bridge, the dataset format… etc.

**Personal comments**
We think it was a really interesting practice. We finally put in practice a lot of concepts related to machine learning such as feature selection, feature generation, cross-validation, regression trees, decision trees, rules, etc.

We also discovered that machine learning is not just applying an algorithm but a complex process that requires mastering several different skills such as statistics, data mining, problem solving and business understanding.

Our only criticism is that we would have liked to have a little more guidance in this practice. It is the first time we employ the machine learning problem solving approach in a project. We were not sure how to compare the algorithms. What was expected for us to write in this document. What does make sense to do and what does not. We understand we had to discover some of these things by ourselves. But We still think we could have saved time if the paper was a bit more specific.

In general it was a really educational experience in which we learned a lot of this amazing world of machine learning. Thanks for giving us the opportunity of doing this. And for reading this report.