Grado en ciencia e ingeniería de datos

Machine Learning I - Academic year: 2021/22

CLASSIFICATION AND PREDICTION

LUCÍA CORDERO SÁNCHEZ

ALONSO MADROÑAL DE MESA

NIAs:

100451778 100454449

Date: 1st April, 2022

1 Introduction

Pac-Man optimization path project consists of many different phases in which we pre-process and clean our data so that we can build trustworthy models to help Pac-Man success in its task: reaching all ghosts in the shortest possible way when ghosts move or are fixed. In this report, we will try to provide an objective and concise insight on the interpretation of these models.

2 Phase 1

Explanation of the feature extraction function that has been programmed, as well as the mechanism to include future data in the same instance.
First, we decided which of the variables were more important, thinking about which of them may be more relevant to have in a prediction model. Moreover, the purpose of our function is to collect all the variables from the game into a .arff file. This step of modifying the header is made to prepare the data to be preprocessed by Weka.

Explaining in depth the details of the technical part, we have to highlight the method we have used so that, considering the function is filling the k-th row of attributes, the variable ”nextScore” is registered as an attribute from the previous instance (k-1).
An exception we have also taken into account is that when we are in the first tick of the game, there is no previous instance so the function does not register that current score.

Finally, in the last iteration or tick we find the same type of exception: no next score is reflected because there is no more instances, so we remove the whole instance to avoid mistakes.

3 Phase 2: Classification

Explanation of the experimentation as explained in the statement. It must include the justification of the selected algorithms, of the selected attributes and of any treatment on the data that has been carried out. It will conclude with an analysis of the results produced by the chosen algorithms and justification for the choice of the final model.

Pre-processing: treatment of the data

First of all, we have implemented some variables in our Python code to make it easier for Pac-Man to optimize the ticks in each journey: *getCloserGhostDist*, and *getNorth*, *getSouth*, *getEast*, *getWest*.

getCloserGhostDist returns the ghost that is closer to our PacMan, and *get[Direction]* is a binary variable, that indicates whether we can move in the next position to North, South, East or West or not (eg: if PacMan is not allowed to move to the North, *getNorth = False* in that tick). These variables will give us an insight on which direction is more prohibited by walls, for example.

We decided not to implement **getCloserGhostPosition** as it may be redundant in our code, and introduce more noise to the models than the usefulness it has for the learning of PacMan.

We have implemented the following changes into the data to make it more robust to outliers, understandable and as free of noise as possible:

- **Nominal to binary.** We have used the unsupervised filter ”Nominal to binary” for all the variables that were boolean, as well as for the ones we have created: *NorthPos*, *SouthPos*, *EastPos*, *WestPos*, *LivingGhost1*, *LivingGhost2*, *LivingGhost3*, *LivingGhost4* and *currentAction* (dividing it in groups of all the possible values it has). Moreover, we removed the ”STOP” by not storing this data from the list of legal actions.
- **Removing the variables.** To predict as accurately as possible, we have

decided to remove *foodNum* and *foodDist*, as none of them are very related with which will be the next movement of Pac-man, nor useful enough to predict it.

- In a more general insight, we will **avoid using "future attributes"** in the classification task because we don't really know how they will behave: attributes for the future are not available at the moment.
- **Remove the existent outliers** by filtering them to have more homogeneous data.
- **Rename distances.** First, in Python we set the value of distance for a dead ghost to 1000 (so that we cannot have it confused with a real distance, because it is an impossible value). Taking into account that we would have a very biased model if we did not change the distance from 1000 to other value, so we set them with Weka to -1. In this task, we covered also the removal of NAs with the same method.
- **To standardize.** We have normalized the numerical type of attributes because they will be less likely to affect our models if their distances from each other are reduced.
  Since not all of them are originally in the same "scale", in an algorithm that measures the weight or importance of each variable, non-normalized numerical data would mean a lot of bias (as it gives more importance to the largest numbers, that do not necessarily correspond to the more important variables).
  This situation is avoided by transforming the data this way, which will allow us to compare them fairly.

Related to a task of **Exploratory Data Analysis**, we have studied the importance of our weighted variables and ranked them: overall, the most important variables are the ones related to the free positions the Pac-Man has to move around in each tick (EastPos, WestPos, NorthPos, SouthPos; the variables we have created).

Other relevant attributes are Pac-Man positions in X and Y. This insight has been obtained by applying Attribute Selection filter with the evaluator

”**GainRatioAttributeEval**” and search mode ”Ranker”.
GainRatioAttibuteEval evaluates the worth of an attribute by measuring the gain ratio with respect to the class, and *Ranker* mode simply ranks this ratio in decreasing order. The last ones are interpreted as attributes that return more noise than relevant information, so we are not interested in them.

Ranking	Attribute
1	GhostPosition4X
2	GhostPosition4X
3	Y
4	X
5	EastPos
6	GhostDIst4
7	Score
8	EastPos
9	WestPos
10	GhostPosition2X

Mainly because we have 18 variables, we only ranked the 10 most relevant. From it, we can conclude that ghost positions, as well as the current position of Pac-Man and whether they can move to the DirectionPos or not is very relevant. Due to the lack of importance of the variables foodNum and foodDist, we decided to remove them (as being told previously). In this case, these variables added more noise than relevant information to our models.

Finally, using unsupervised techniques, we applied Clustering and Principal Components Analysis, but the conclusions were not satisfactory because data are not as intercorrelated to have components that explain considerably good individually the information, but we need many of them to achieve a good explanation.

Analysis of the results

Once we have treated our data, we have started our experiments to see which of them give us the best accuracy. We tried ZeroR, J48, Naïve Bayes, Simple Logistic Regression, Logistic, KNN and k-d trees with the following results:

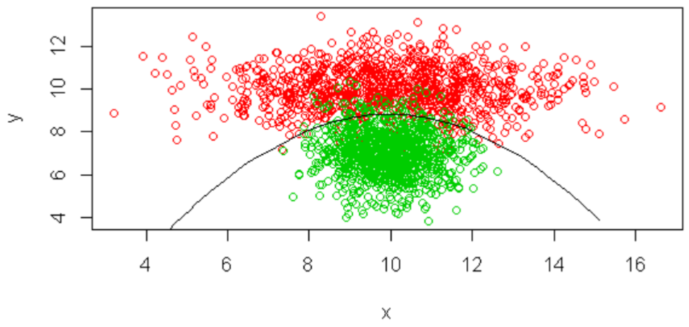
Algorithm	Accuracy
ZeroR	42,27%
J48	77,03%
RandomForest	79,96%
Naïve Bayes	62,55%
Simple Logistic Regression	76,82%
Logistic	75,08%
IBk, k = 1	78,40%
IBk, k = 5	72,23%
IBk, k = 9	71,98%
IBk, k = 13	70,49%
IBk, k = 17	69,44%
Kd trees, k = 1	78,40%

The best algorithms for our model are **RandomForest** and **J48**, as well as Simple Logistic Regression in the third place. This way, we prove the hypothesis we had at first: our data works better with models whose performance improves with linearity. Figuring out that our data may be linear, it is interesting to consider why our results turned out like this with the best and the worst performances:

Random Forest is a **very robust algorithm** in terms of dealing with extreme values. It is an ensemble of decision trees, this is to say that many trees, constructed in a certain "random" way form a Random Forest. Each tree is created from a different sample of rows and at each node, a different sample of features is selected for splitting. A Random Forest's **nonlinear nature** can be an advantage over linear algorithms, making it a great option.

In the second place, the decision boundaries of J48 can be made, in a way, "stepwise linear". It uses the "divide and conquer" method to generate the initial decision tree from a training dataset. It is based on the use of the gain ratio criterion; probably, as it follows a similar nature of Random Forest trees it works well too (it is also in the family of decision trees).

The worst performances are observed over Naïve Bayes: this classification algorithm is a simple probabilistic classifier with a strong assumption of independence.



Although the assumption of attribute independence is generally a poor assumption and is often violated for true data sets like this one, in which the variables are very correlated. Moreover, it is very effective for performing over huge loads of data, which is not our case either. This is how Naïve Bayes perform in a simulation of a $\mathcal{N}$ distribution: two Gaussian clouds are simulated and fitted a decision boundary (which is clearly very inaccurate).

Final decision

Our final choice was deciding between J48 algorithm and Random Forest. Overall, in the training set Random Forest performed slightly better, but we opted for J48 to develop our automatic agent because the difference with the accuracy on the test set was lower (this tables refer to RF and J48, respectively):

Data	Accuracy
Train	79,96 %
Test	72,47 %

Data	Accuracy
Train	77,03 %
Test	75,69 %

4 Phase 3: Regression

This phase aims to explore Weka's prediction algorithms. The experimentation procedure and its documentation in the memory will be equivalent to that of phase 2.
Once the information has been obtained from a series of games played by the different selected agents, a prediction model must be built.

For regression, we have followed a very similar procedure to Phase 2: apply ”Nominal to binary”; removing foodNum and foodDist (the rankings of information gain for the variables was almost invariant, so we will skip the representation); remove the outliers; rename distances as done in the previous phase, to standardize our data and finally to check again through the ”Ranker” that our choices would be satisfactory for the model, only keeping the first ones for creating it.

The main reason why we opted for removing more variables is simple: if we have a small dataset with a lot of variables, we may find ourselves analizing very complex data due to the rise in dimensionality and complexity (this is called typically the ”Curse of dimensionality”, and we want to stay away from it).
A fundamental different with phase 2 is that in phase 3 we keep nextAction as an important variable for the regression task, so we don't avoid anymore the ”future data”.

Related to the models we tried, we have to make some notes: we will always compare processed with unprocessed data, as well as data from the tutorial 1 built agent compared to the keyboard agent (played manually).

Parameters of measure
At first, we did not know how to interpret regression: there is no ”accuracy” formally, no parameter that estimates in percentage or in relative units how much you ”success” or ”fail”. In this task, we will work with errors.
Errors are ”large” or not depending on the data with which we work with.

Table 1: Comparison in terms of the Square Root Relative Error in Keyboard-Agent

	Linear Regression	Logistic	M5P	RandomForest	J48
Processed data	**24,4**	24,05	20,15	20,59	20,45
Raw data	14,24	**17,36**	12,63	12,31	12,64

Table 2: Comparison in terms of the Square Root Relative Error with Tutorial1 Agent

	Linear Regression	Logistic	M5P	RandomForest	J48
Processed data	11,70	**16,68**	10,00	9,27	9,60
Raw data	19,01	**20,44**	15,02	16,50	16,75

For example, an error of our prediction with respect to a measure may be the same in two different regression tasks, and insignificant in one of them but very relevant in the other one.
In this sense, the best measure (with a similar function to the percentage of accuracy) is r-squared for linear regression (that measures how much our prediction line adjusts to the real data). To get a broad explanation of our models, we will use Square Root Relative Error and the correlation coefficient.

Note: For all experiments, we have used $\alpha = 0.05$ in a two-tail test.

First, in Table 1 we will compare the processed and raw data in terms of the Square Root Relative Error, which is defined as the square root of the sum of squared errors of a predictive model normalized by the sum of squared errors of a simple model. In other words, the square root of the Relative Squared Error (RSE).
Compared to Table 2, in which we pick the data playing by hand, it is clear that **the error is way smaller for the programmed agent from Tutorial 1 than from the one we covered ourselves**. Moreover, we can conclude that trees and linear regression adapt very well to the data, as they have the lower errors. Marked in blue and bold, the models with worse prediction: usually, logistic.

Table 3: Comparison for the Correlation Coefficient in KeyboardAgent

	Linear Regression	Logistic	M5P	RandomForest	J48
Processed data	0,97	0,97	0,98	0,98	0,98
Raw data	0,99	0,98	0,99	0,99	0,99

Table 4: Comparison for the Correlation Coefficient in Tutorial Agent

	Linear Regression	Logistic	M5P	RandomForest	J48
Processed data	0,99	0,99	1,00	1,00	1,00
Raw data	0,98	0,98	0,99	0,98	0,98

To interpret Table 3 and Table 4, we will follow a similar reasoning: processed data loses a little bit of correlation in general with respect to raw data (whose nature is correlated), and models are almost equal in general, so the only conclusion we would extract from the correlation coefficient is a measure of linear correlation (which is the case, but may not be in any other).Finally, another conclusion would be that playing with the keyboardAgent (ourselves), correlation would be lower (we will fail in optimizing the way as humans sometimes, so the nature of the relations won't be as good reflected as in a computer agent).

5 Phase 4: Automatic Agent

Explanation of which model has been selected and why to build the automatic agent, and a brief description of how it works.

For creating our Automatic Agent we first tried to used the data with the adjustments made in the pre-processing phase. Unfortunately, it was very difficult to translate to python all those measures, so we opted for used our "raw data" from the beginning and play in a backward selection way: from a model with all the variables, we used the Ranking (provided before in phase 2) to determine which variables could be discarded, until we had sufficient variables to have a decent model for our Automatic Agent, but not too many to have our complexity increased exponentially.

The goal was to **keep an agent that was good enough to predict the class with the chosen classification method (J48) without falling into very time-consuming operations** with too many attributes.

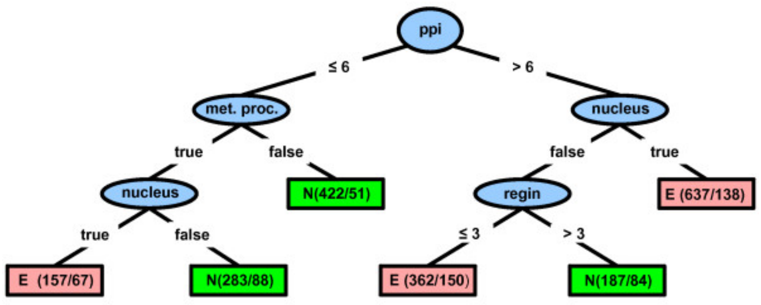
The following measures were given by our final model classification method J48:

Parameter	Quantity
Correctly classified instances	1157
Incorrectly classified instances	125
Kappa statistic	0,8754
Mean Absolute Error	0,057
Root Mean Error	0,1698
Relative Absolute Error	18,42 %
Root relative squared Error	42,92 %

When the Cohen-Kappa index is over 0.7, it is considered in J48 that we have a good model. On the contrary, RandomForest had only 3 incorrectly classified instances against 1280 correct ones for our first dataset, so if we had used this one we would have found overfitting issues.

<u>J48</u>

J48 is a machine learning decision tree classification algorithm based on Iterative Dichotomiser 3. It is very helpful in examine the data categorically and continuously.



Regarding the results that are shown to us in this procedure, there are several indicators of the goodness of fit of the model that may be interesting:

1. Percentage of well-classified instances globally
2. Cohen's Kappa Index: Goodness-of-fit index of the complete model. It is understood that the model is appropriate if this value exceeds the value 0.7.
3. Mean absolute error: Statistic that indicates the estimation error made. Lower values indicate that the model is more appropriate.
4. Percentage of the area under the ROC curve: The ROC curve indicates, on an abscissa axis and ordinates, the relationship between the sensitivity (true positives between the total of positives) and specificity (true negatives among the total negatives) of the model established classification. A curve is established for each category, in relation to the rest of the categories, indicating the capacity of the model to detect cases belonging to that category of the criterion variable.
5. Confusion matrix: Contingency table that relates the classification given by the model with the real value that the subject reaches in the criterion variable.
6. Accuracy: Percentage of well-classified instances among all those selected as positive

Data sets

The basis for the creation of the final data set was first the raw data with approximately 500 observations and all variables, in which we applied feature selection by removing the least relevant ones.

After this, we concluded that 500 instances were very few to get any conclusion, so we decided to combine six maps and both the keyboard and automatic agents already built to get a bigger database.

Unfortunately, our database had around 3000 instances and accuracy measurements when we applied the same adjustments to the database in Weka Explorer were similar.

In this sense, we have to take into account that on a regular basis an algorithm will always play ”better” than humans (so our data had a little bit of variance due to this), and that some thousands of instances are not determinant to built a trustworthy model.

Building the agent
The main process for implementing the agent was to launch the Java Virtual Machine, to create our list ”x” for the weka.predict function and to check that our model was correctly done, as well as fixing that the data were extracted from the biggest database done (with slightly better results).

6 Questions

1. What is the difference between learning these models with instances coming from a human-controlled agent and an automatic one?
It is clear that as humans our abilities to play a game in the most efficient way will never be as good as the ones a computer has: they will probably find the shortest path to follow, while for a human that path for Pac-Man is unlikely. The main point is that, if we consider that Pac-Man will learn from the behaviour patterns of our data, we should consider having only the best way possible (which is the one from computation and not by hand), so that Pac-Man does not learn from inefficient movements.

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?
Some regression models are already classification models, like logistic regression. Some ideas are to split the data at a certain cutpoint to get a classification or

to use discretised percentiles to define categories instead of numerical values. In general, regression trees convert to classification trees if the dependent variable changes, but it will probably translate into a loss of information.

3. What are the advantages of predicting the score over classifying the action? Justify your answer.
From the point of view of the learning, it is more useful for Pac-Man to have predicted the score over the actions, since it is easier to identify patterns in the movements that increase or decrease the score, rather than in a list of four actions. This change would result in a better-informed Pac-Man every time it plays, since eventually it would have enough information to decide based on the current position to which direction the score will be more minimized.

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?
Sure! This was the main point stated in the third question: by having a numerical value rather than a categorical one, it would be easier in terms of computation to achieve an optimal path (more accuracy because we give to the numbers a ”weighted” importance, not as when we discretize categorical variables that they all weight equally).

7 Conclusions

We would divide our conclusion in three different aspects: the technical conclusions, insights on struggling and personal comments. Finally, we will add a glossary of all the .arff files and models attached to our practice.

From a technical point of view, we find that extracting clear conclusions for a sample of this size (small) is hard, but assignment 1 was very interesting to gain insight on how to interpret each parameter of the models and how to express in a simple and conclusive way.
A curious thing is that, although at first we thought we would struggle so much

with Weka to create the models, it was not the case thanks to the resources we have found on the net and the previous introduction to Weka taught in the subject.

If we had to choose a very time-consuming problem with which we struggled from the very beginning until the end was the Virtual Machine and the whole setup to use the console and to connect Python to Weka. Having so many files and some errors from lines that we hadn't written was not easy to solve, as we had a double challenge: interpret the code and solve the problem.
Moreover, we found out that Javabridge may have very confusing exceptions that are helpless to warn you of any failure (or better said, of where to find it).

Luckily for us, the best part of this assignment was that we all experienced some common failures, so we could help each other out to solve them. As a personal comment, we find that phases 2 and 3 were more disconnected from phase 4 than expected: we could not use the data we had processed (which was shown to be better) but the raw data because it is technically a big issue to deal with (to translate to Python all those conditions of pre-processing applied to the data in Weka), and having to get again new data sets with considerably more instances that did not guarantee a better performance was unsatisfactory.

Finally, we have discovered that models are very different from each other in terms of the applications, so this project has helped us to understand some complex theoretical knowledge given in the master class, and complete it with extra resources. More concretely, we have learned about the scope of random forest, the importance of linearity for using or not different approaches and the different ways to measure "errors" depending on the goal of the task.

Glossary
1. All files that do not end up with "-" or "t" are the original (raw data).
2. All files that end up with "t" belong to some data processing from Phase 2.
3. All files that end up with "t2" belong to some data processing from Phase 3, as well as model experiments.
4. All files that end up with "t3" or "t4" are failed experiments from Phase 3 that we finally did not use, but that are relevant to see how we got to the final solution proposed.