# Matching a machine learning model with the best data representation in high state space problem : Rubick's cube

Amine Bechrifa Sana Jallouli

## Abstract

*In this project, we compare à selection of machine learning classifiers to find the most accurate one for solving 3x3 Rubik's Cube. For each classifier, we also find the data representation that leads to the best accuracy. The classifiers compared are logistic regression, decision tree classifier, random forest ,support vector machine SVC, adaboost classifier and gradient boosting classifier as well as a 324-1024-1024-1024-1024-50-24 Leaky ReLU network with a Softmax output. Each one fed with "hot representation" then with integer representation. To overcome the high dimensionality, we used the models to predict the number of moves separating a state and the final one. The results obtained depict à very small difference between the two data representations, however a significant gap between the classifiers accuracy.*

## 1. Introduction

The 3x3 Rubik's cube is an instance of problems involving sequence of learning under a discrete representation of multiple potential actions. The cube having multiple configurations representing the state spaces, makes it difficult to learn the different potential moves leading to the solution.Indeed, it has 43,252,003,274,489,856,000 possible states.Solving such problems can be assimilated to other investigations in problem decomposition thus leading to solving problems such as Genetic Programming and protein unfolding.Our goal is solve à 3 by 3 Rubik's cube using different classifiers to find the most accurate solver.We also aim to find the date representation leading to the best accuracy.The data is represented by à line of integers defining à cube, either in hot representation or int representation. Thus, each colored face of one of the small cubes constituting the Rubik's cube, is always placed in the same index of the line with à different corresponding color.For instance, à red face of à small cube is [0,0,0,1,0,0] in hot representation and "3" in int representation. We directly predict the best move leading to the shortest path to the solved state. These moves are represented by integers ranging from 0 to 23.The large state space and a requirement to learn invariances relative to the specific colours present,make it difficult to directly predict the move to make and led to an inaccuracy worse than predicting the move randomly. Aiming to achieve more accurate predictions, we instead used the number of moves leading to the solved states as the target data, which is also represented by an integer. To better represent the cube we relied on the Rubik's Cube website where the exact configuration of the Rubik's cube is explained. We also used the pyCuber documentation for better manipulation of the cubes's representation.

## 2. Methodology & Experimental Results

**1. Data Processing**

a.Flattening the data

To be able to use the data in the model training, we needed to flatten it. The 3d cube represented by the Pycuber library as à 3d matrix is reshaped into 1 line.The content of these matrices is the representation of the colors as strings. We transformed the strings in both int representation and "hot" representation [figure 1]. The int representation is defined by à ligne of 54 integers ranging from 0 to 5 , each corresponding to à color. The "hot" representation is defined by à line of 324 binary elements, where each 6 defines à color [figure 2 ]. Generating the data We initialize 3x3 solved cubes using the pyCuber library. We then generate,for each cube, 10 random moves , which are represented by à string in the Pycuber library. For each of these moves, we apply the move on the cube, flatten the result to get à line of integers and add it to an array while plugging the number of moves applied in that step in à second array. We then get 2 arrays of 10 elements, where the first one represents the cubes shuffled a different number of times (from 0 to 9 times). The second array, which constitutes our target ,contains the corresponding number of moves each cube has been subjected to.

These two arrays represent the input to our models. To generate the full data set, this process is repeated N times to get à data set of 10N elements where N is the number of cubes that have been shuffled from 0 to 9 times. We set N to 10,000 meaning that 100,000 cubes and we also generated 20,000 for the testing.

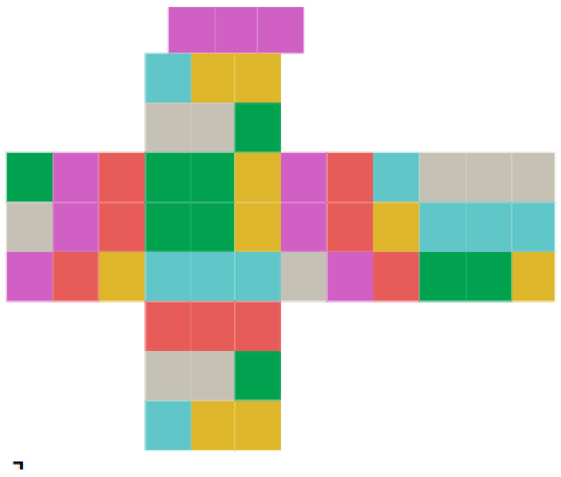
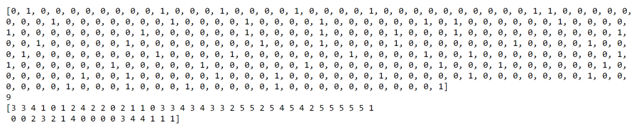


Figure 1.



```
[0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0,
0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 1, 0, 1, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0,
1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1,
0, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0,
0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1,
1, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 1, 0,
0, 0, 0, 0, 0, 1, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0,
0, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1]
9
[3 3 4 1 0 1 2 4 2 2 0 2 1 1 0 3 3 4 3 4 3 3 2 5 5 2 5 4 5 4 2 5 5 5 5 5 1
 0 0 2 3 2 1 4 0 0 0 0 3 4 4 1 1 1]
```

Figure 2.

b.Benefits of this approach

The first advantage of our generated data is its dimensionality that makes the training faster.By generating the same number of cubes for each number of shuffles, we are making sure that the model will not be biased toward any specific prediction. Moreover,when the paths intersect, they make the data better by providing for the same cube a prediction closer to zero which is the solved state. After training, à cube that has been generated after 7 shuffles can be predicted as one that has been subjected to one that is subjected to 4 shuffles.

**2.The Training, Testing of the models**

The models are trained to find the numbers of moves separating the shuffled cube and the solved state. For that, the training was done using different models with different parameters and fed with the two data representations , namely the "hot" representation and int representation"

The models are trained using 100,000 cubes and their corresponding target.Each input of the model is à line of integers along with the target that is also an integer representing the number of moves needed to reach the solved state.

We use the 20,000 samples of cubes generated and their corresponding targets as testing data. The different models used are listed along with their parameters and different data representation. The results obtained for each model are also given. From these results we depict that the model parameters leading to the best accuracy are the same for training and testing and this for all the models.

a.Tree Classifier :

Using the tree classifier, we focused on the max depth parameter and we trained à model for each of the following values 1,5,10,30,50,100. The best training and testing accuracy is obtained starting from max depth= 30 and remaining unchanged above 30 [figure 3,4 apendix].

| | hot_representation | Int_representation |
|---|---|---|
| Training accuracy | 0.8122 | 0.43855 |
| Testing accuracy | 0.1146 | 0.4446 |

b. Random Forest Classifier: [figure 5, 6 apendix]

Using the Random Forest classifier, we focused on the max depth parameter and we trained à model for each of the following values 1,5,10,20,50,100.. The best training and testing accuracy is obtained starting from max depth= 20 and remain unchanged above 20 [figure 4 apendix].

| | hot_representation | Int_representation |
|---|---|---|
| Training accuracy | 0.8122 | 0.81146 |
| Testing accuracy | 0.45805 | 0.4558 |

c. Logistic Regression : [figure 7,8 apendix]

Using the Logistic Regression, we focused on the C parameter and we trained à model for each of the following values 10-5,10-4,0.001,0.01,0.1,1.The best training and testing accuracy is obtained at C=0.1 [figure 5 apendix]

| | hot_representation | Int_representation |
|---|---|---|
| Training accuracy | 0.42458 | 0.37648 |
| Testing accuracy | 0.4113 | 0.3694 |

d. Support Vector Machine (SVM) :

Using the Support Vector Machine (SVM), we focused on the C parameter and we trained à model for each of the following values0.01,0.1,1,10,100,1000. The best training and testing accuracy is obtained at C=10

| | hot_representation | Int_representation |
|---|---|---|
| Training accuracy | 0.449273 | 0.4539 |
| Testing accuracy | 0.377 | 0.36 |

e.AdaBoost Classifier : Using the classifier AdaBoost Classifier, we focused on the learning rate parameter and we trained à model for each of the following values0.0001,0.001,0.01,0.1,1,10 . The best training and testing accuracy is obtained at learning rate=0.001

| | hot_representation | Int_representation |
|---|---|---|
| Training accuracy | 0.32954 | 0.3128 |
| Testing accuracy | 0.330 | 0.312 |

e.GradientBoost Classifier :

Using the classifier AdaBoost Classifier, we focused on the learning rate parameter and we trained à model for each of the following values0.0001,0.001,0.01,0.1,1,10. The best training and testing accuracy is obtained at learning rate=0.001

| | hot_representation | Int_representation |
|---|---|---|
| Training accuracy | 0.5209 | 0.444 |
| Testing accuracy | 0.506 | 0.447 |

e.Neural Network A : [figure 12 apendix]

For the hot representation the Neural Network is a 324-1024-1024-1024-1024-50-24 Leaky ReLU network and Softmax output For the integer representation the Neural Network is a 324-1024-1024-1024-1024-50-24 Leaky ReLU and Softmax output For this Neural Network, we used Sparse Categorical Cross Entropy with Adam optimizer with the metrics as the accuracy. Using this Neural Network, we focus on the learning rate parameter and we trained à model for each of the following values0.0001,0.001,0.01,0.1,1,10. . The best training accuracy is obtained at learning rate=0.001

| | hot_representation | Int_representation |
|---|---|---|
| accuracy | 0.536 | 0.461 |

f.Neural Network B : [figure11 apendix]

For the hot representation the Neural Network is a 324-1024-1024-1024-50-24 Softmax network with Softmax output. For the integer representation the Neural Network is a 324-1024-1024-1024-50-24 Softmax with Softmax output. For this Neural Network, we used Sparse Categorical Cross Entropy with Adam optimizer with the metrics as the accuracy. Using this Neural Network, we focus on the learning rate parameter and we trained à model for each of the following values0.0001,0.001,0.01,0.1,1,10. The best training accuracy is obtained at learning rate=0.001

| | hot_representation | Int_representation |
|---|---|---|
| accuracy | 0.528 | 0.459 |

**3.Prediction Results**

From the plotting of the accuracy versus the parameters[figure , ], we can see that the shape of the curve, for both hot and int representations, is the same.We depict that the data representation impacts the accuracy of the models but doesn't have an impact on its variation. Comparing the results, the "hot" representation in each model , leads to results that are either as accurate as the int representation or better. For the Tree classifier, Random Forest classifier, Gradient Boosting classifier, and AdaBoost classifier there is no significant difference between the results of both representations. For all the other methods namely Support Vector Machine, Logistic Regression and the Neural Networks, there is an improvement in accuracy when using à hot representation. The best accuracy was achieved by the Neural Network with 4 layers and Leaky Relu activation function , softmax output fed with "hot " representation of data.

**4.Solving the Rubik's Cube**

To solve the Rubix Cube, we take the shuffled cube and apply it on each of the 24 possible moves while storing the results in an array. We then predict, for each of these resulting cubes, the number of moves that separate their state and the solved one. We take the cube for which we predicted the least number of moves and repeat the same process until the number of moves predicted is 0. For some of our first trials, we found that for some of the shuffled cubes, our algorithm got a stack going back and forth in the same move and its opposite.To solve this issue, after doing the first prediction, we removed the action that is opposite from the last move we made from the initial possibilities of moves. Another problem that arises is that, when finding the minimum number of moves equal between multiple actions, it chooses the last one one that it found. This issue is overcome by implementing an algorithm that takes all these elements, for which we predicted the same number of moves, and apply on each of them the whole process to find the best choice. For complexity limitation, this algorithm is only allowed to repeat itself 5 times for one run in case it keeps predicting the same number of moves for all them again.

Our best predictor is the Neural Network À, trained with the "hot" representation, with an accuracy of 0.536.We now use this model to solve the Rubix cube. Each time we predict the number of moves remaining to reach the solved state, we have à probability of 0.536 of finding the correct one. If the number of times we shuffle the cube is x, then the probability of the cube being solved is

$$(0.532)^{x-1}$$

. indeed, we discard the state where the cube is shuffled only one time as in the first step of the algorithm we apply on the cube all the possible moves, which means that we are sure that one of them leads to the solution.

## 3. Conclusions

From our experiment, and using our data set , we are able to conclude that the best classifier is The 324-1024-1024-1024-1024-50-24 Leaky ReLU network with a Softmax output with learning rate 0.001 and fed with hot representation. Furthermore, we can conclude that for our problem , "hot" representation leads to an accuracy that is at least as good as the integer representation but is better for most of the models. While having the possibility to generate as much data as we want for our Rubik's Cube, we were not able to conclude the scalability of our results because we lacked time and/or hardware resources. In addition, even using our best predictor, the probability of solving the 3x3 Rubik's Cube falls under 0.08.

## 4. Appendix

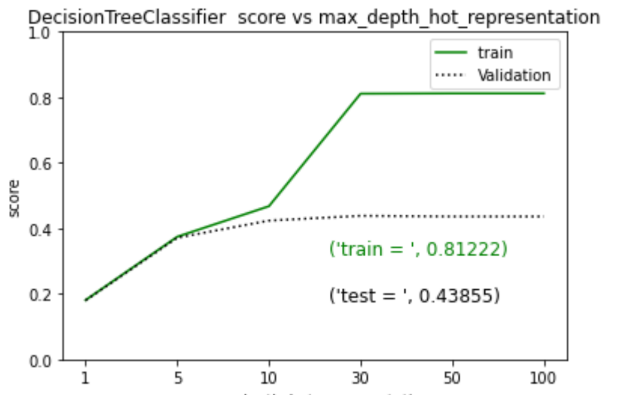


Figure 3.

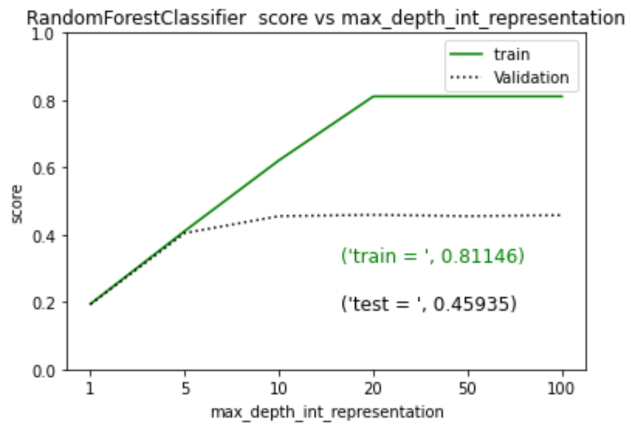


Figure 6.

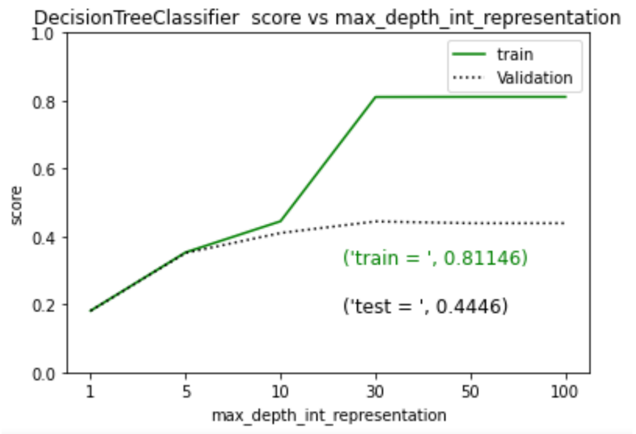


Figure 4.

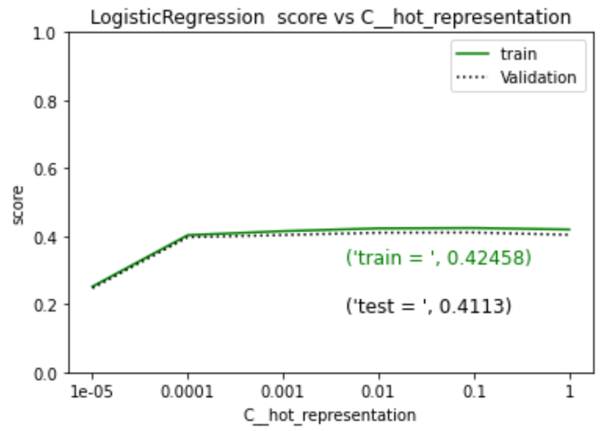


Figure 7.

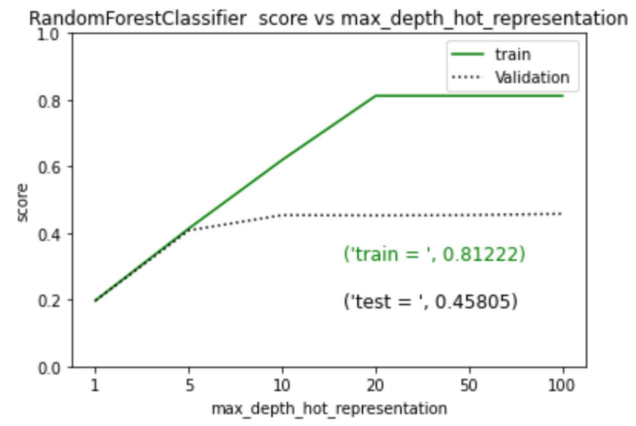


Figure 5.

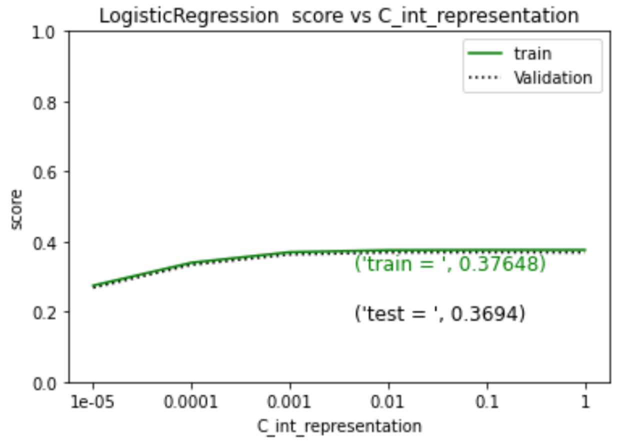


Figure 8.

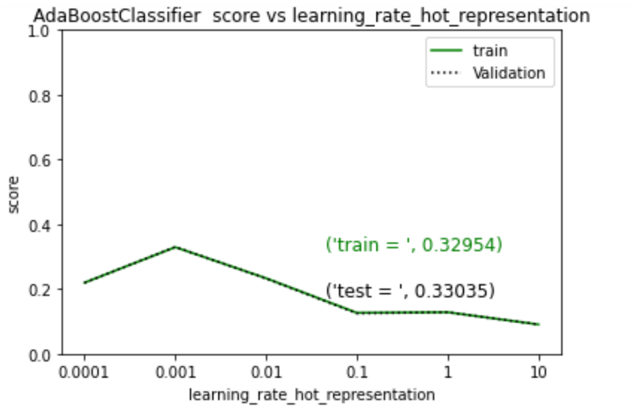


Figure 9.

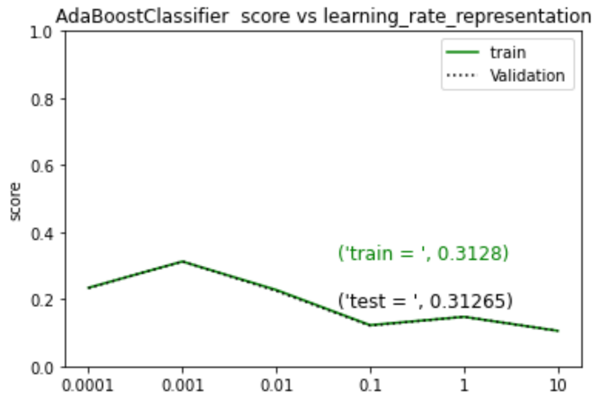


Figure 10.

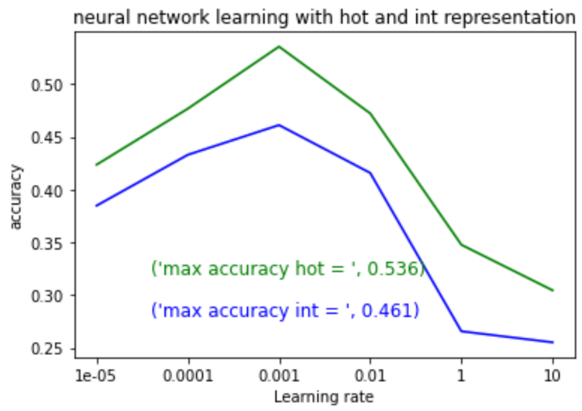


Figure 12.

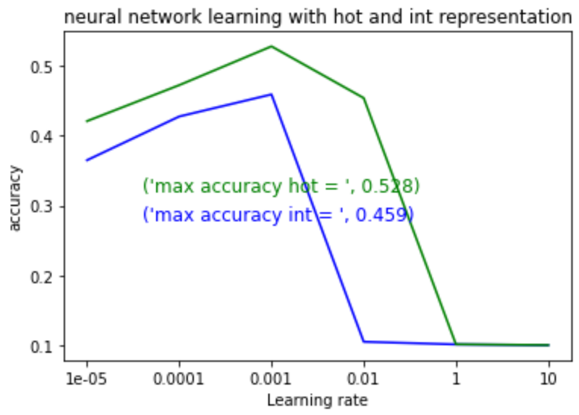


Figure 11.