Jack Charbonneau
Jannik Haas
Jian Liu
Mario Arduz
Joan Wong

Assignment 3 Write Up - Group 1

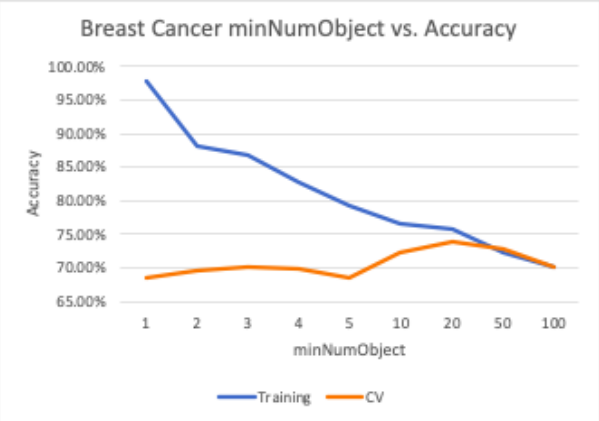
Part 1. Overfitting

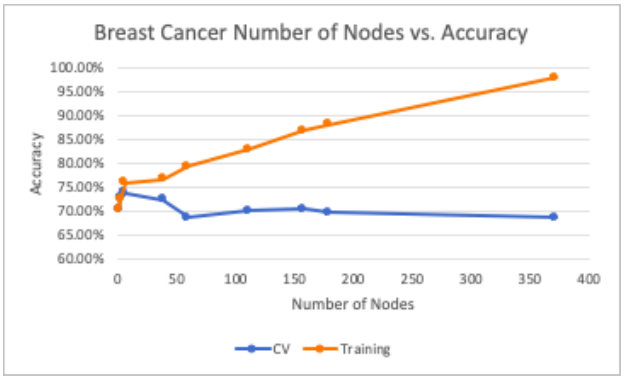
1. ***Warmup***

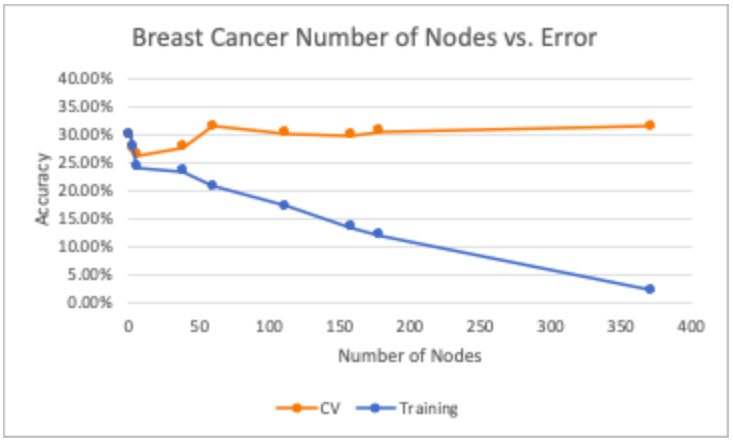
a. The accuracy on the training data is 75.87% and the 10-fold cross validation accuracy is 75.52%. The J48 model does not seem to suffer from overfitting on this task as the accuracy was almost the same for training and cross validation.

b. For the unpruned model, the training accuracy was 88.11% and the cross-validated accuracy was 69.58%. We can see that the model is overfitting a lot as there is a very large difference between the training and cross validation accuracy. The unpruned tree is most likely too flexible and, therefore. overfits to the training data causing an increase in training accuracy, but a large decrease in cross validation accuracy. We can see the CV accuracy for the unpruned tree is also significantly lower than the pruned tree.

2. Experiments with minNumObj







I have included graphs of both the minNumObject and the number of nodes of the trees corresponding to *the minNumObjects tested vs the accuracy* as well as *the number of nodes vs the error*. The number of nodes vs error follows the more typical bias variance tradeoff graph that we saw in lecture..

a. The model overfits to the training data for minNumObj up to 20 (number of nodes greater than 6). As can be seen from the last graph showing the *number of nodes vs. error*, the training error and cross validation (CV) error are very close up until the number of nodes is 6, which is the model created by minNumObj of 20. After this, however, the two lines diverge and the training error continues to decrease, while the CV error increases, hence these models overfit to the training data. When minNumObject is 1, the tree is able to make a single node for almost every single observation which will create a very flexible and complex tree and a training accuracy of 97.9%, however this will also lead to overfitting as the CV accuracy was only 68.53%.

b. The model underfits the data for a very high minNumObj. You can see this from the minNumObjects of 50 and 100 (corresponding to number of nodes of 3 and 1 respectively) where the training and CV errors are higher than our best model (number of nodes of 6), therefore underfitting and not modeling the true underlying trends of the data.

c. Increasing the minNumObj over 100 will not change the accuracy for this specific model. With a minNumObj of 100, the model is already equivalent to a zero rules model with only a single node. The training and CV error are the same as the model simply guessing the class with the higher number of observations.

3. Predicting overfitting

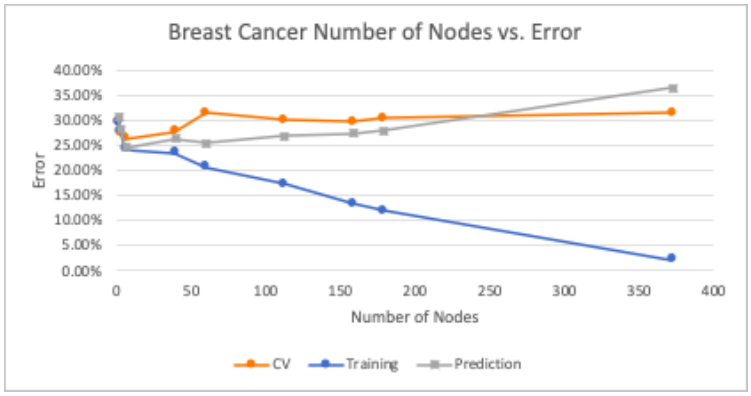
a. The formula for predicting the CV accuracy we have come up with is as follows: CV Accuracy = Training Accuracy - (Tree Size - k) * λ * k/n where k = number of predictive features, n = number of observations, and λ is the tuning parameter set to 3 in this case. For predicting the CV Error, we simply subtract the CV Accuracy formula from 1. The first part of the complexity penalty is tree size minus the number of variables in the data. We included this term due to the fact that a tree with a large number of

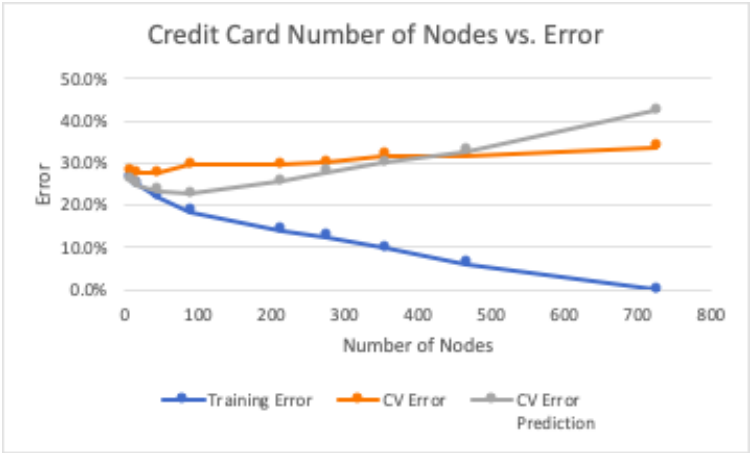
nodes is more likely to overfit the data, however, complex datasets with a large number of variables need a more complex tree. Therefore, this term takes the difference between the two to penalize large trees for low number of variable data more than ones with a large number of variables. λ is the tuning parameter, which in the case of the breast cancer data, set to 3 since it reduced the Mean Absolute Error (MAE) of the prediction. Lastly, we added *k/n* as the last part of the complexity penalty. If the data has a large number of variables and small number of observations, the complexity penalty increases, due to the fact that it is likely we will not get a good understanding of the trends behind the data from the small dataset. However, a large dataset with a small number of variables will get penalized less, since we have more observations which will be representative of the true nature of the data. We achieved an MAE of 2.37% for predicting the CV accuracy/error for the breast cancer dataset.

b. We used the same formula for the credit dataset and adjusted the tuning parameter accordingly. The tuning parameter that minimized the MAE was 2.9. However, due to the fact that the difference between λ = 3 and λ = 2.9 was insignificant, we simply continued to use 3 for λ. This prediction gave us a MAE of 3.70% for the credit dataset.

c. Our final formula for predicting the CV accuracy is as follows:

$$\text{CV Accuracy} = \text{Training Accuracy} - (\text{Tree Size} - k) * 3 * k / n$$



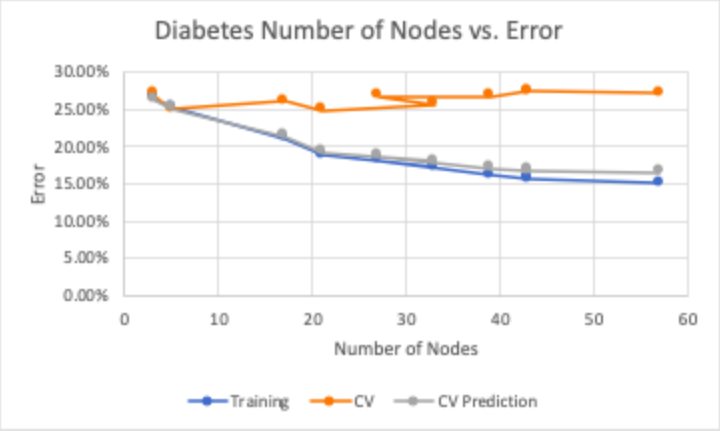


As you can see from the graphs above, the formula was unable to predict some jumps in the CV error for both datasets and underestimated the error for lower complexity (lower tree size) models, and overestimated the error for very complex trees.

	Breast Cancer Data		k =	9			
	Training	CV	n =	286			
Pruned	75.87%	75.52%	lambda =	3			
unPruned	88.11%	69.58%					
unPruned Testing with minNumObjects							
minNumOb	Training	CV	# Nodes	Training Error	CV Error	CV Error Prediction	Absolute Value of CV error-CV error prediction
1	97.90%	68.53%	372	2.10%	31.47%	36.37%	4.90%
2	88.11%	69.58%	179	11.89%	30.42%	27.94%	2.48%
3	86.71%	70.28%	158	13.29%	29.72%	27.35%	2.37%
4	82.87%	69.93%	112	17.13%	30.07%	26.86%	3.21%
5	79.37%	68.53%	60	20.63%	31.47%	25.44%	6.02%
10	76.57%	72.38%	39	23.43%	27.62%	26.26%	1.36%
20	75.87%	73.78%	6	24.13%	26.22%	24.41%	1.81%
50	72.38%	72.73%	3	27.62%	27.27%	28.19%	0.92%
100	70.28%	70.28%	1	29.72%	29.72%	30.48%	0.76%
						MAE:	2.37%

Credit Card Data			k =	20			
	Training	CV	n =	1000			
Pruned	85.50%	70.50%	lambda =	3			
unPruned	94%	68.30%					
unPruned Testing with minNumObjects							
minNumOb	Training	CV	# Nodes	Training Error	CV Error	CV Error Prediction	Absolute Value of CV error-CV error prediction
1	100.00%	66.40%	725	0.0%	33.60%	42.3%	8.70%
2	94.00%	68.30%	466	6.0%	31.70%	32.8%	1.06%
3	90.10%	68.30%	355	9.9%	31.70%	30.0%	1.70%
4	87.40%	70.00%	275	12.6%	30.00%	27.9%	2.10%
5	86.10%	70.40%	214	13.9%	29.60%	25.5%	4.06%
10	81.50%	70.40%	89	18.5%	29.60%	22.6%	6.96%
20	78.10%	72.50%	45	21.9%	27.50%	23.4%	4.10%
50	74.90%	72.50%	16	25.1%	27.50%	24.9%	2.64%
100	73.50%	72.20%	9	26.5%	27.80%	25.8%	1.96%
						MAE:	3.70%

Diabetes Data				k=	8			
	Training	CV		n=	768			
pruned	84.11%	73.83%		lambda =	3			
unpruned	84.38	72.66%						
unPruned Testing with minNumObjects								
minNumObj	Training	CV	# Nodes	CV Prediction	Training Erro	CV Error	CV Error Prediction	Absolute Value of CV error- CV error prediction
1	85.02%	72.91%	57	83.49%	14.98%	27.09%	16.51%	10.58%
2	84.38%	72.66%	43	83.29%	15.62%	27.34%	16.71%	10.63%
3	84.24%	72.66%	43	83.15%	15.76%	27.34%	16.85%	10.49%
4	83.85%	73.31%	39	82.88%	16.15%	26.69%	17.12%	9.57%
5	81.90%	73.31%	27	81.31%	18.10%	26.69%	18.69%	8.00%
10	82.81%	74.35%	33	82.03%	17.19%	25.65%	17.97%	7.68%
20	81.25%	75.13%	21	80.84%	18.75%	24.87%	19.16%	5.71%
50	78.91%	73.96%	17	78.63%	21.09%	26.04%	21.37%	4.67%
100	74.74%	75.00%	5	74.83%	25.26%	25.00%	25.17%	0.17%
200	74%	73%	3	73.73%	26.43%	26.95%	26.27%	0.68%
								6.82%



As you can see from the graph above, this is our prediction formula applied to the diabetes dataset. From this graph, we can see that our formula performs poorly on predicting the CV error for the diabetes dataset with a MAE of 6.82%. The J48 models for the diabetes dataset overall had much fewer nodes than for the breast cancer and credit card data, therefore making the complexity penalty very low which can be seen in the predictions. The variables for the diabetes dataset were strictly numeric, while the breast cancer and credit card data variables were mostly categorical. We suspect the J48 model created less splits for the numeric data which created smaller trees overall, making our prediction formula inaccurate. This leads to an interesting thought, however, to consider creating separate prediction formulas for data with mostly categorical variables, mostly quantitative variables, and data with equal parts of both. Interestingly, for the diabetes data, the initial pruned model with the default settings overfit the data about the same amount as the unpruned model with default settings, whereas for both the credit card and breast cancer data, the initial unpruned models overfit significantly more than the pruned models.

4. Bias/Variance

a. In machine learning, as you create more complex and flexible models, the models make less assumptions about the nature of the data and are able to find very non-linear trends (low bias), however a small change in the training data you feed to the model will create a very different model, and there is a

high chance of overfitting (high variance). Low flexibility models do the exact opposite with having a high bias and low variance. The tradeoff is finding the balance between the bias and variance to accurately model the underlying trends of the data without overfitting to the training data.

b. The approach of changing the minNumObj variable directly impacts the flexibility of the model. A low number will allow the J48 tree to fit almost perfectly to the training data, creating a very flexible and complex tree, however, if you attempt to train or use this tree on a new set of data, it will likely perform poorly because it overfits to the training data and did not actually model the underlying trends of the data. As we increase the minNumObj variable, we decrease the model complexity and, therefore, increase the bias of the model. Since we now need more objects in each node to split the tree, the tree will not overfit to one single observation that could have just been an outlier. This hopefully creates a model that models the underlying trends of the data, while also having low variance meaning that it can be used on unseen or test data and still perform well. Once the minNumObj variable gets too high, however, the bias of the model will be too high and, therefore, underfit the data without capturing the underlying trends of the data. With the breast cancer data, we saw that a minNumObj of 100 actually just created a zero-rule model since it did not have enough observations within the nodes to create even one split, so it just guessed the class with a higher number of observations.

c. The goal of a random forest is to increase the variance within each individual tree so that each tree is able to take more risks in the hopes that not all the trees make the same mistakes and, therefore, the overall forest has higher prediction accuracy. In a random forest, we are training a large number of decision trees with different subsets of the data as well as only using a subset of features at each split. In general, decision trees are a fairly flexible/complex model with high variance, meaning that when we train the same tree with different datasets, there is a high chance that we will end up with two very different models. This is why in a random forest we will end up with a large number of very different trees created by different datasets that alone have a low predictive power, but when taking the majority vote of all the trees, the model tends to perform fairly well. In logistic regression, the bias is fairly high and variance is low. This means the model makes strong linear assumptions about the nature of the data as well as the fact that training the same model on two different datasets will likely not make that much of a difference in the two models created. Therefore, the random regression technique will most likely create a large number of similar models. There can definitely be a benefit drawn from using this method as the variance induced will most likely make some difference in the models, however, the induced variance we create by doing random sampling on the training data will be mostly outweighed by the high bias and low variance of the logistic regression model. Therefore, while there is potential for random regression to be beneficial and useful, it would not bring as much of a benefit as a random forest.

Part 2. Gridworlds

1. What maps you used for testing. Explain how the maps guided your development.

To begin with, we set γ, the discount factor, to 1, *p*, the correct move, to 0.8, and *r*, the reward, to -0.04. We set ε, the random move, and α, the step size, to any arbitrary value (a rough estimate) to start off but were tested and tuned by the map configurations below. When ε = 0.2 and α = 0.06, the agent worked well on the two simple maps, Map 1 and Map 2. Then, we test these two parameters on more complex maps to see if we can further optimize the values. For Map 3, we found that the agent was stuck in the local plateau because α was too small. (See Question 2 for details). And for Map 4, we found that the agent was not able to see some states of the map, so we figured that there was a need for a higher exploration rate. (See Question 2 for details). Map 3 and Map 4 have higher rewards in directions which are not the closer ones to the end state, encouraging for exploration. After many iterations of testing and tuning, the parameters are finalized to ε = 0.4 and alpha = 0.1, which worked well on all four maps shown below.

```
[[1., 0., 0., 0., 0., 0.],
 [-1., 0., 0., 0., 0., 0.],
 [0., 0., 0., 0., 0., 0.],
 [0., 0., 0., 0., 0., 0.]])
```

Map 1

```
[[1., 0., 0., 0., 0., 0.],
 [-1., 0., 0., 0., 0., 0.],
 [0., 0., 0., 0., 0., 2.],
 [0., 0., 0., 0., 0., 0.]])
```

Map 2

```
([[1., 0., 0., 0., 0., 0., 0., 5.],
  [-1., 0., 0., 0., 0., 0., 0., 0.],
  [0., 0., 0., 0., 0., 0., 0., 0.],
  [0., 0., 0., 0., 0., 0., 0., 0.],
  [0., 0., 0., 0., 0., 0., 0., 0.]])
```

Map 3

```
([[1., 0., 0., 0., 0., 0., 0., 0., 0., 5.],
  [-1., 0., 0., 0., 0., 0., 0., 0., 0., 0.],
  [0., 0., 0., 0., 0., 0., 0., 0., 0., 0.],
  [0., 0., 0., 0., 0., 2., 0., 0., 0., 0.],
  [0., 0., 0., 0., 0., 0., 0., 0., 0., 0.],
  [0., 0., 0., 0., 0., 0., 0., 0., 0., 0.]])
```

Map 4

2. How you came up with an exploration policy that does a good job on a variety of maps. How did you set the stepsize parameter?

To maximize future reward, it is not wise to always take the action with the highest Q(s, a) value. Taking other actions to get useful information is necessary. Therefore, we choose to use epsilon-greedy policy to balance exploration and exploitation, requiring an exploration rate. At each iteration, we generate a random number, and if this number is smaller than the exploration rate, then the agent takes a random move. Otherwise, it takes the action with the highest Q(s, a) value.

We test different maps to figure out an optimal value for the exploration rate of our agent. We initially set ε, our random move, to be 0.2, and it worked well on Map 1. However, when we tested the agent on a bigger and more complex map (Map 4), it failed to find the optimal path to the highest value. As shown in Figure 1, the learned policy took the agent to the terminal state, (3, 5), which had a reward of 2, and the agent failed to discover states on the right side of the map, indicating the lack of exploration.

Policy:									
(0, 0)up	(0, 1)down	(0, 2)left	(0, 3)down	(0, 4)down	(0, 5)down	(0, 6)left	(0, 7)right	(0, 8)right	(0, 9)up
(1, 0)up	(1, 1)down	(1, 2)down	(1, 3)down	(1, 4)down	(1, 5)down	(1, 6)down	(1, 7)left	(1, 8)up	(1, 9)up
(2, 0)right	(2, 1)right	(2, 2)right	(2, 3)right	(2, 4)right	(2, 5)down	(2, 6)left	(2, 7)left	(2, 8)left	(2, 9)up
(3, 0)right	(3, 1)right	(3, 2)right	(3, 3)right	(3, 4)right	(3, 5)up	(3, 6)left	(3, 7)left	(3, 8)down	(3, 9)right
(4, 0)up	(4, 1)up	(4, 2)up	(4, 3)up	(4, 4)up	(4, 5)up	(4, 6)left	(4, 7)up	(4, 8)up	(4, 9)right
(5, 0)up	(5, 1)up	(5, 2)up	(5, 3)up	(5, 4)up	(5, 5)up	(5, 6)up	(5, 7)left	(5, 8)left	(5, 9)up
Highest Q value:									
(0, 0)0.0	(0, 1)1.71	(0, 2)1.57	(0, 3)1.77	(0, 4)1.8	(0, 5)1.62	(0, 6)0.37	(0, 7)0.79	(0, 8)3.04	(0, 9)0.0
(1, 0)0.0	(1, 1)1.76	(1, 2)1.8	(1, 3)1.84	(1, 4)1.88	(1, 5)1.92	(1, 6)1.88	(1, 7)1.11	(1, 8)0.45	(1, 9)1.34
(2, 0)1.76	(2, 1)1.8	(2, 2)1.84	(2, 3)1.88	(2, 4)1.92	(2, 5)1.96	(2, 6)1.92	(2, 7)1.72	(2, 8)0.36	(2, 9)0.09
(3, 0)1.8	(3, 1)1.84	(3, 2)1.88	(3, 3)1.92	(3, 4)1.96	(3, 5)0.0	(3, 6)1.96	(3, 7)1.58	(3, 8)0.0	(3, 9)0.0
(4, 0)1.76	(4, 1)1.8	(4, 2)1.84	(4, 3)1.88	(4, 4)1.92	(4, 5)1.96	(4, 6)1.92	(4, 7)0.25	(4, 8)-0.0	(4, 9)0.0
(5, 0)1.72	(5, 1)1.76	(5, 2)1.8	(5, 3)1.84	(5, 4)1.88	(5, 5)1.9	(5, 6)1.77	(5, 7)0.51	(5, 8)0.02	(5, 9)0.0

Figure 1

Thus, we need to set a higher exploration rate. We then tested on ε = 0.4, and as shown in Figure 2, the agent was then able to discover all the states and successfully found a path to the terminal state, (0, 9) which had the highest reward.

Policy:									
(0, 0)up	(0, 1)down	(0, 2)down	(0, 3)right	(0, 4)right	(0, 5)right	(0, 6)right	(0, 7)right	(0, 8)right	(0, 9)up
(1, 0)up	(1, 1)right	(1, 2)right	(1, 3)right	(1, 4)up	(1, 5)up	(1, 6)up	(1, 7)up	(1, 8)right	(1, 9)up
(2, 0)right	(2, 1)right	(2, 2)right	(2, 3)up	(2, 4)up	(2, 5)up	(2, 6)up	(2, 7)up	(2, 8)right	(2, 9)up
(3, 0)right	(3, 1)up	(3, 2)up	(3, 3)up	(3, 4)up	(3, 5)up	(3, 6)up	(3, 7)left	(3, 8)left	(3, 9)up
(4, 0)up	(4, 1)up	(4, 2)up	(4, 3)up	(4, 4)up	(4, 5)left	(4, 6)left	(4, 7)left	(4, 8)left	(4, 9)up
(5, 0)up	(5, 1)up	(5, 2)up	(5, 3)up	(5, 4)up	(5, 5)up	(5, 6)left	(5, 7)left	(5, 8)left	(5, 9)left
Highest Q value:									
(0, 0)0.0	(0, 1)4.58	(0, 2)4.64	(0, 3)4.76	(0, 4)4.8	(0, 5)4.84	(0, 6)4.88	(0, 7)4.92	(0, 8)4.96	(0, 9)0.0
(1, 0)0.0	(1, 1)4.64	(1, 2)4.68	(1, 3)4.72	(1, 4)4.76	(1, 5)4.8	(1, 6)4.84	(1, 7)4.88	(1, 8)4.92	(1, 9)4.96
(2, 0)4.56	(2, 1)4.6	(2, 2)4.64	(2, 3)4.68	(2, 4)4.72	(2, 5)4.76	(2, 6)4.8	(2, 7)4.84	(2, 8)4.88	(2, 9)4.92
(3, 0)4.52	(3, 1)4.56	(3, 2)4.6	(3, 3)4.64	(3, 4)4.68	(3, 5)0.0	(3, 6)4.76	(3, 7)4.72	(3, 8)4.6	(3, 9)4.59
(4, 0)4.48	(4, 1)4.52	(4, 2)4.56	(4, 3)4.6	(4, 4)4.64	(4, 5)4.6	(4, 6)4.51	(4, 7)4.0	(4, 8)1.62	(4, 9)0.42
(5, 0)4.44	(5, 1)4.48	(5, 2)4.52	(5, 3)4.56	(5, 4)4.59	(5, 5)4.51	(5, 6)3.96	(5, 7)2.68	(5, 8)1.06	(5, 9)0.14

Figure 2

As for the step size parameter, α, if this value is set too high, the agent may be unstable or jump over local plateau. For example, when $\alpha = 0.8$, the average reward graph (Figure 3) shows that the agent ignored the local plateau, which is the terminal state, (0, 0). If α is too small, however, it may take too long to converge or get stuck in the local plateau. For instance, when $\alpha = 0.8$, the average reward graph (Figure 4) shows that the agent was stuck in the local plateau, which is the terminal state, (0, 0).

Therefore, we tested on $\alpha = 0.1$. As shown in Figure 5, the performance converged; the agent successfully found a path to the terminal state, (0, 7), which had the highest reward. After testing on other maps, the optimal value for α was found to be 0.1.

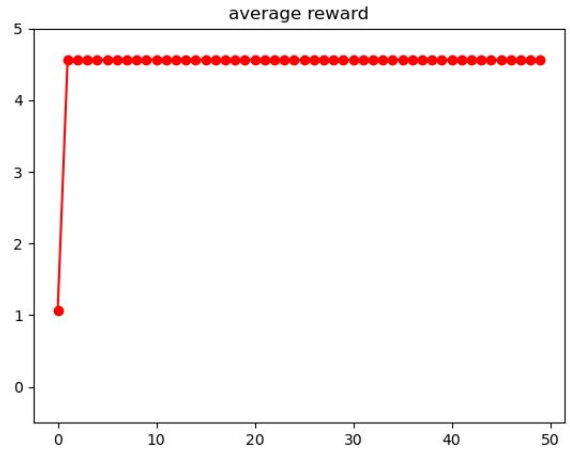


Figure 3. $\alpha = 0.8$ on Map 3

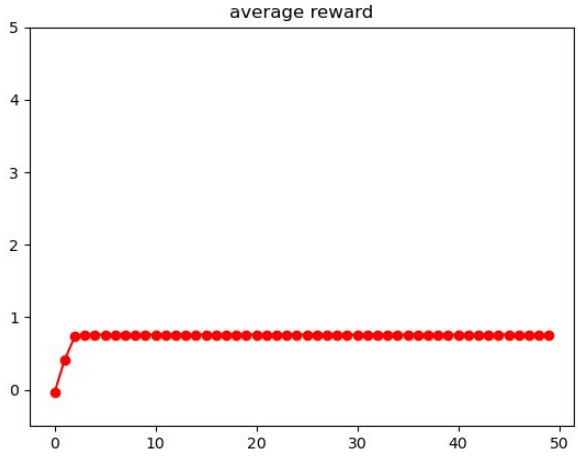


Figure 4. $\alpha = 0.06$ on Map 3

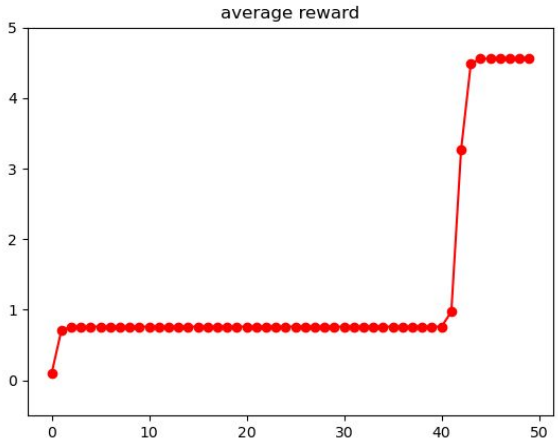
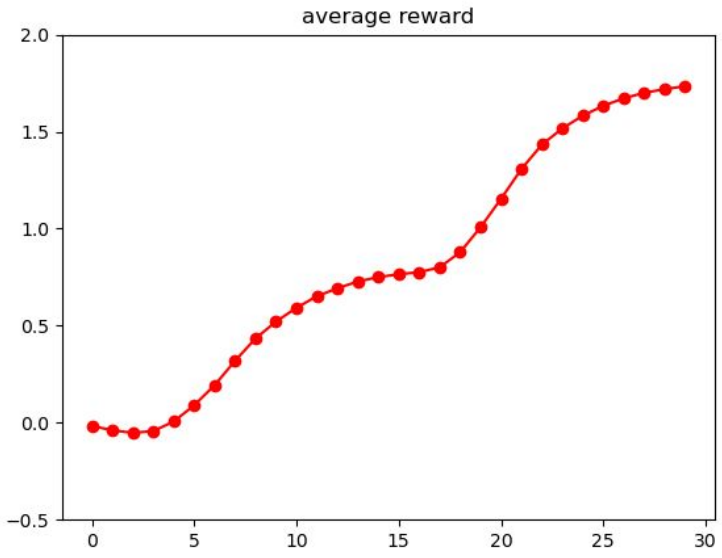


Figure 5. $\alpha = 0.1$ on Map3

3. Plot a graph of performance on one of your maps.

```
[[1., 0., 0., 0., 0., 0.],
 [-1., 0., 0., 0., 0., 0.],
 [0., 0., 0., 0., 0., 2.],
 [0., 0., 0., 0., 0., 0.]])
```

We ran our agent, setting ε (the random move) = 0.4, γ (discount factor) = 1, *p*(correct move) = 0.8, and *r*(reward) = -0.04, on Map 2 for 3,000 trails. Since this is a small, simple map, we set the step size to be very small (α = 0.01) to show details of the graph. The performance graph is presented below with the number of 100 training runs as its x-axis and the average reward received as its y-axis:

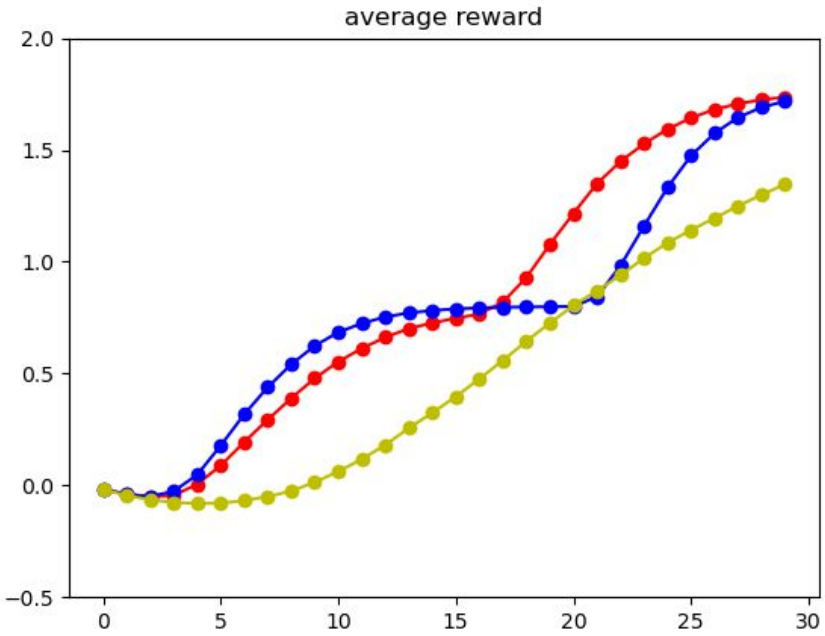


As shown in the performance graph, the curve increases rapidly in the beginning, and then it gradually slows down, reaching a plateau, which is the reward for reaching the terminal state, (0, 0). Then the curve grows quickly again and reaches another plateau, which is the reward of reaching the terminal state, (2, 5), and this terminal state has a bigger reward, 2.

4. ***Plot performance (on the same) graph of two simple policies and contrast their performance with your agent:***

a. ***Random: an agent that makes a random action at each timestep.***

b. ***ε-greedy with ε=0.1***



We applied two other policies on Map 2 with the same parameters, except ε, the performance graph is as shown above. The x-axis is the number of 100 training runs, and the y-axis is the average reward received. The red curve represents our ε-greedy policy with ε = 0.4, the blue curve represents the ε-greedy policy with ε = 0.1, and the yellow curve represents the random policy.

As shown in the performance graph, the blue curve grows faster than the red curve. Because ε = 0.1 policy acts more greedily than our ε = 0.4 policy does, ε = 0.1 policy has a higher possibility to take action with the highest Q(s, a) value and reaches the first terminal. But the red curve takes a shorter time than the blue curve does to reach the second terminal, which has a bigger reward. As opposed to ε = 0.1 policy, our ε = 0.4 policy explores more. Exploration helped our agent gather more useful information about the environment to maximize its future reward. As for the yellow curve, it grows much slower than both red and blue curves and does not reach any plateau, indicating that it failed to find a path to any terminals in 3,000 trails by always acting randomly and had a very low efficiency.

Part 3. Escaping from gridworld: the robotic truck driver

Explain your state representation, and why you chose it.

A state is represented as a moment in time where the agent (truck) has to make a decision: to either deliver packages or wait for another time tick. The time steps that happen when the agent is delivering packages are not considered as states because they do not offer the possibility to choose between continuing the delivery or going back to the warehouse. Instead, they are considered part of the transitions that are applied to states when the agent makes a decision.

Each state is described by approximations of the capacity of the truck and the maximum distance that has to be covered to deliver all packages. The capacity corresponds to the number of packages that can be loaded before making the delivery, and the maximum distance is the location of the package that is located at the furthest position with respect to the warehouse.

These two elements are used to represent the state because it is expected that they are going to influence the action of the agent. It is anticipated that a state with higher capacity, or with a low number of packages that are loaded to the truck, could be more likely to take the action of waiting at the warehouse. Conversely, a state where the capacity of the truck is low, and there is not a possibility to keep loading packages, the agent should be more inclined to take the action to deliver. In the case of the maximum distance to be covered, and assuming that the truck is not at full capacity, the agent should be more prone to deliver the orders when they are closer, and wait when they are positioned far from the warehouse.

The state representation consists in a tuple of two bounded values, which are approximations of the capacity and maximum distance to be delivered. To begin with, the absolute values of the capacity and distance are converted to relative values. The value of the actual capacity is divided by the maximum capacity of the truck, which is given as an input. Similarly, the value of the maximum distance to be covered is divided by the length of the road, which is also an input of the problem. This transformation is applied to have more generality in the treatment of the states.

Next, each of the relative values is bounded to the floor division with 0.2[1], which outputs 6 categories, where 1 is reserved to represent maximum capacity (when there are not any packages in the warehouse) and when the truck has to deliver a package at the furthest location. Therefore, one can note that one of these representations is reserved to the state where there are no packages at the warehouse: then the relative capacity equals 1 and the relative distance equals 0. This transformation reduces the number of state representations, with the aim that the agent would learn the action that offers the highest reward more easily.

To have a complementary explanation of the state representation, the new table is presented with some examples of possible states with a maximum capacity of 25 and a length of the road that equals 30.

[1] Since the float representation of 0.2 is inexact, 0.2 - 10**-14 is used instead.

Capacity	Distance	Relative Capacity	Relative Distance	New Capacity	New Distance
25	0	1.00	0.00	1	0
11	5	0.44	0.17	0.4	0
23	13	0.92	0.43	0.8	0.4
1	27	0.04	0.90	0	0.8
3	10	0.12	0.33	0	0.2
22	19	0.88	0.63	0.8	0.6
19	12	0.76	0.40	0.6	0.4
24	28	0.96	0.93	0.8	0.8
12	19	0.48	0.63	0.4	0.6
16	21	0.64	0.70	0.6	0.6
17	13	0.68	0.43	0.6	0.4

The output of the learned policy is in terms with the last two columns of the previous table. For that matter, the optimal policy will be valid for a set of states that are organized in ranges of 0.2 in their relative capacity and distance.

Also note that this task is not a well-defined, finite task. For example, the gridworld stops when the agent reaches the goal state. This simulation could continue indefinitely. In such a scenario, future summed rewards would go to infinity. How will you account for that? (there's a hyperparameter we talked about that is helpful).

Despite the design of the problem does not explicitly define a terminal state, and rewards could be obtained indefinitely, the gamma hyperparameter allows us to discount the value of future rewards based on how far they are obtained from the current time. It is expected that rewards (limited by a maximum value) in a sufficiently distant time, are going to have a present value close to zero. Consequently, the sum of future rewards, discounted by the gamma factor, would equal a defined value lower than infinity.

Furthermore, the update of Q values considers this element by calculating the rewards at each time tick, and adjusts them based on the gamma factor and the temporal distance from the moment when the action was taken. For instance, if an action on delivery was taken at *time = t*, the total reward at *time = T* would be discounted by a factor equal to *gamma**(T-t)*. All rewards are discounted until the moment when the agent is in a position to elect an action. Then the procedure is repeated, defining a new initial time *t*, and setting another *T* in the consecutive time steps.

The update of the Q values takes into consideration all the adjusted rewards that are between the current and future state, and also an adjusted maximum Q value for the next state. Again, this adjustment is proportional to the temporal difference of the times of the different states.

The motivation behind this adjustment, besides excluding the chance of having infinite rewards, is to have a more congruent comparison between states. Any action of delivery has the option to arrive at different states, with different time ticks that depend on the maximum distance of the package that has to be delivered. If the location of the farthest package of the delivery is closer, it will take less time to get to the next state, and the contrary would happen with a package that is far more distanced from the warehouse. Therefore, it should follow that the update of the Q value from the best future action would take into account the time difference of the possible state alternatives.

Explain the policy your simulation learns for a road length of 25, a truck capacity of 30, and a penalty of starting the truck on a deliver action of -250. How much reward does your agent receive on average after it is trained? After your agent is finished training, run it for enough iterations to get a large enough sample for average reward.

To explain the optimal policy under these conditions, 500 iterations were runned, where each of them allowed the agent to do 1,000 random actions. In consequence, a total of 1,000 updates were distributed to the different Q values for the existing states.

The next table displays the frequency of the best action at the third column. Since there are only two possible actions, they were encoded as 0 when the agent should stay at the warehouse and 1 when the agent should deliver. For that matter, the percentage that is shown below corresponds to the number of times, among the 500 iterations, that the Q values determined that the best action is to deliver a package. Similarly, when the percentage is lower, the agent opted for staying at the warehouse. Finally, the last column refers to the number of times that this state was present in the 500 iterations.

Capacity	Distance	Action	N
1	0	0%	500
0.8	1	5%	500
0.8	0.8	19%	500
0.8	0.6	30%	500
0.8	0.4	58%	500
0.8	0.2	98%	500
0.8	0	100%	500
0.6	1	62%	85
0.6	0.8	73%	171
0.6	0.6	94%	35
0.6	0.4	100%	3

Before interpreting the results of the optimal policy, the last column of the table shows that all states are not equally likely. Under the circumstances of the experiment, it is unlikely that the capacity of the experiment is going to be lower than 0.6. This argument could serve as a motivation to expand the representations of the state. Nevertheless, this would reduce the generality of this representation, and at

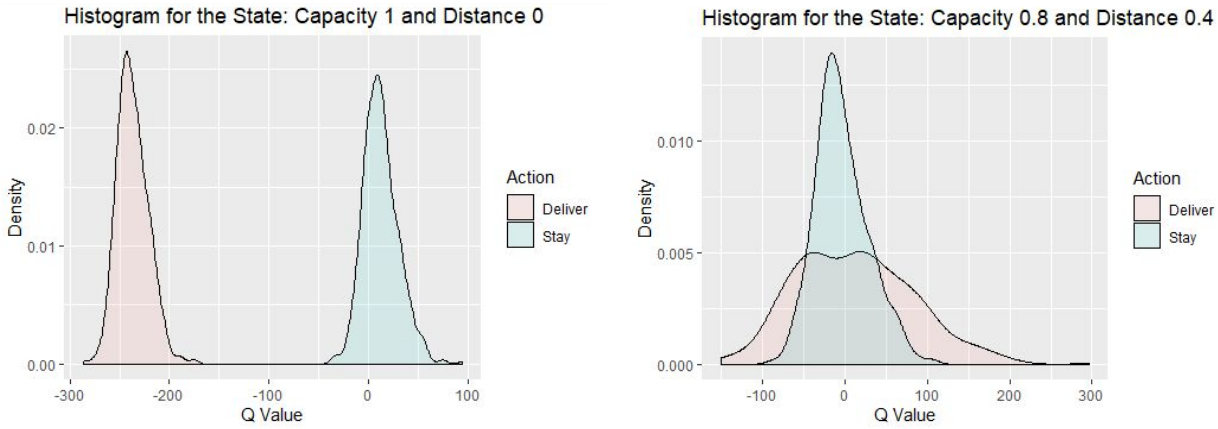
this point, it is uncertain how it would impact other possible combinations of the maximum capacity of the truck and the length of the road.

The main result of this table is that the agent adopts actions with the anticipated logic: with a decreasing capacity to load packages, the agent should be prone to deliver them. Similarly, given a level of capacity of the truck, the agent should be less likely to deliver the packages when the distance to the farthest location is high. Despite each state being represented by rounded versions of the relative values, this logic is maintained for the policy, which supports this state representation.

An example of the optimal policy is that the agent learned to stay at the warehouse when there are no packages to deliver, and almost all of the time, the agent learned to stay when it had a delivery located at 25 units from the warehouse with less than 20% of the packages loaded into the truck,.

It is important to acknowledge that when the action column is close to 50%, the agent is taking decisions that are comparable to one that is random. This is because the evaluation of the Q values offers similar results for both actions, and the election of the best action fits the randomness of each iteration. This behaviour is not necessarily wrong, but it implies that the expected value for both actions is similar.

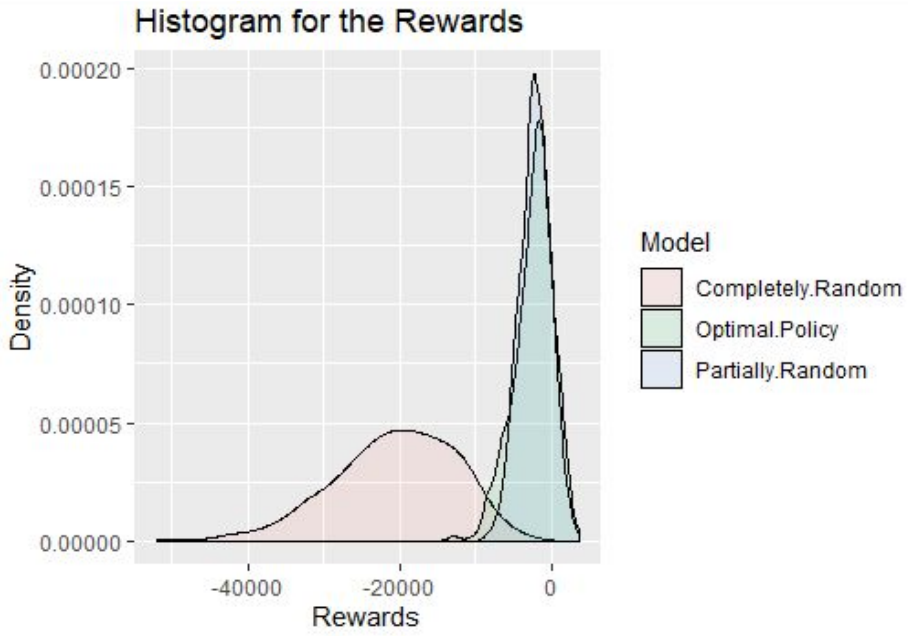
The following two graphs show this argument. At the left, the truck is at the warehouse and there are no packages to deliver. The best decision in this state is clear to identify because there are two distributions of rewards that are well separated, and the one that is positioned at the right offers more value, which happens to be the stay action. In the second case, the state has an actual relative capacity between [0.8,1) and a maximum distance within the range [0.4, 0.6). Both possible actions have a similar mean of the Q values distributions, but the decision to stay at the warehouse offers a *less risky* outcome, in contraposition with the decision to deliver that potentially offers higher and lower rewards than staying.



It is important to remark that these results were obtained from 500 random experiments and that the evaluation of the state under an unique run would offer only one of the points that correspond to the distribution shown in the previous graphs. Only if the distributions of the values are well separated, the results of the optimal policy are going to be consistent through different runs. Even more notable, it could

be possible that the agent indeed knows the best policy, but due to randomness of the actual environment, that optimal policy could lead to lower rewards.

The analysis of rewards after learning the optimal policy consists in summing the value of rewards that an agent obtains in the period of 1,000 time ticks. This experiment was runned 500 times and three different kinds of policies were compared: a completely random policy, a partially random policy and the optimal policy which was learned after 1,000 random actions with their correspondent updates of the Q values. The results are shown next.



	Optimal Policy	Completely Random	Partially Random
Min	-13,282	-52,008	-8,715
Median	-2,073	-20,245	-2,173
Mean	-2,418	-20,689	-2,222
Max	3,604	-2,972	3,219

The completely random policy takes a random action each time it has the opportunity to decide the transition from one state to the other. The partially random policy was explicitly coded to only take a random action when the warehouse had a positive number of packages waiting to be delivered. In other words, whenever the truck was empty, it was forced not to deliver the packages.

As the histogram of rewards and the descriptive statistics show, the optimal policy is notably better than the completely random policy. It is expected that a trained agent will obtain higher rewards than a random agent after it has been trained. This is true for the values shown at the table next to the histogram.

Nevertheless, the optimal policy is not different from the partially random policy because both of these actions output a distribution of rewards that are similar to one another.

There could be several reasons to explain this similarity. One could be that the main gain of the optimal policy and the partially random policy is not to deliver the truck when there are no packages in the warehouse. It is possible that some decisions of delivery when there are a positive number of packages at the warehouse are trivial (as it was shown for the histogram of the Q values for a state with 0.8 of capacity and 0.4 of maximum distance). Another reason could be that the design of the penalties of not delivering the package in time are not big enough to encounter a difference. In the current design of the game, the agent receives a reward of the package as long as it delivers it, and the truck can fit a large number of packages such that in the random actions used to train the agent, the relative capacity did not fall below 60%. Similarly, it is also possible that the state representation of the game is not the ideal one, and it is not capturing sufficient information for the agent to understand the environment and obtain a significantly higher sum of rewards when the optimal policy is applied.

Even though these are some of the possible explanations in the absence of differences between the optimal policy and a partially random policy, the agent was able to learn the fundamental issue of the problem. The optimal policy corresponds to a logic between the states, where a lower capacity is more related to a decision to deliver the packages, and a higher maximum distance of delivery is related to the decision of waiting at the warehouse. Finally, and perhaps more importantly, the agent learned not to deliver packages when the warehouse was empty without having previous information about rewards and penalties.