CS 534

Assignment #3: Machine learning

This assignment is composed of three parts. Parts 2 and 3 were designed to be done in sequence. Part 2 will help prepare you for Part 3. Part 1 is conceptually more complex, but does not require coding. It is also independent of Parts 2 and 3. Part 1 is about as difficult as Part 2 (depending on your coding skill). Part 3 is more complicated. (the above is guidance for teams who divide the assignment into parts amongst members; Part 1 was designed for folks who were weaker with coding).

This assignment is probably (hopefully!) longer than the prior assignments.

# Part 1: overfitting

## Warmup

For this question it is recommended, but not required, that you use the machine learning package Weka. Open the file *breast-cancer.arff* in the data subdirectory of Weka. The sample files come with the Weka download and should be in the same directory as the Weka application.

You will work with the J48 model. This model is a decision tree classifier similar to the one we discussed in class. Experiment with the interface to learn how to train a J48 model. What is the accuracy (% correct) on the training data? With a 10-fold cross validation? Does the J48 model seem to suffer from overfitting on this task?

Look at the options for J48. A critical one is *unPruned*; set that variable to true. How does performance change on the training set? On cross validation? Is J48 overfitting now?

## Experiments

For these experiments, you will leave unPruned set to true and will experiment with the *minNumObj* parameter. This parameter controls the minimum number of nodes required to create a leaf in the tree. For example, if minNumObj=50, then the tree creation process will not perform any splits that result in a node having fewer than 50 elements.

Collect data with minNumObj = {1, 2, 3, 4, 5, 10, 20, 50, 100}. You can run these experiments via the GUI. If you are more comfortable with the command-line interface that should work as well. Record and graph performance on the training data and on a 10-fold cross validation. Is there overfitting occuring? Where? Is there *underfitting* occuring? Where? Explain what is going on in relation to the video we discussed for class. Will increasing minNumObj beyond 100 probably improve performance? Why or why not? (it’s not enough to say you did the experiment of minNumObj = 200 and…, instead you should argue from principles what will happen).

While running these experiments, you should also record the number of nodes in the tree (for the next task).

## Predicting the amount of overfitting

It would be nice to be given a J48 model and be able to predict how well it would do on unseen test data, without actually having to do the test process. Your job is to build a formula that will take as input anything about the training process (e.g., number of elements in the tree, number of rows in the dataset, number of columns, model fit on the training data), and predict how well the model would do on a 10-fold CV.

To give your results some generalizability, open the file *credit-g.arff*. Construct a similar table as you did before of minNumObj vs. performance on the training data and on 10-fold CV.

Build a single formula that predicts the 10-fold CV performance of both datasets. Explain your formula, and create a table showing training set performance, actual 10-fold CV performance, and predicted 10-fold CV performance.

You might want to look at BIC for inspiration. Think of your task as:

CV % correct = training data % correct - complexity penalty

(If you’re curious, open *diabetes.arff* and see how well your formula does. Don’t be too depressed; constructing accurate models of cross validated accuracy is **hard.** That’s one reason I prefer reporting CV results rather than using something like BIC or adjusted R2).

## Bias/variance tradeoff

Learn about the bias/variance tradeoff in machine learning and describe it in approximately 100 words.

How does the approach of varying minNumObj relate to bias/variance? Hint: consider a model fit with minNumObj=1 vs one fit with minNumObj=100.

We learned about RandomForests in class. Consider a similar approach of using random sampling of the data to construct a set of models. However, instead of using a decision tree as the classifier it uses logistic regression. How much benefit would we expect to see from this *RandomRegression* technique vs. the gain from RandomForests? Why? Explain your answer in terms of bias and variance.

# Part 2. Gridworlds

For this task you will implement a gridworld system similar to the one we discussed in class. Specifically, you will implement Q-learning with ƛ=0 (this last part just means do what we did in class :-). You will learn a function Q(s,a), that computes the expected future reward for taking action *a* from state *s*. Yes, for this task you know the transition model and so could learn a utility function directly, but you should pretend that you don’t know it as that is good practice for Part 2 (as well as for most challenging real-world RL problems).

## Inputs

Your program should input:

* A filename specifying the rectangular gridworld you will be working with. You may assume that any non-zero entry is a terminal state. See the sample CSV file in the assignment #3 page on Canvas. The agent always starts in the bottom-left corner.
* Move cost. How expensive each move is (e.g., -0.04). The agent can move in the same 4 directions as in the in-class examples.
* The probability a move will go in the desired direction. The remaining probability is split among moving 90 degrees left or right of the desired direction. For example, passing in a 0.6 means if the agent moves right, it has a 60% chance of going right, a 20% chance of going up, and a 20% chance of going down.

Your program should train until performance converges or 20 seconds have elapsed. We will **not** test your code with large gridworlds, and computational efficiency should not be a major concern. If your code is taking several minutes please put in some sanity checks for the sake of our TA.

## Outputs

Your program should output:

* The learned policy. Again, make it easy for the TA to see what is going on. You don’t have to be as graphical as the examples we did in class. For the sample board provided, something as simple as: RIGHT, UP, UP, UP, LEFT is fine.
* The expected reward. I.e., after it trains, it should display the highest Q(s,a) value where s is the start state. For the sample board, you should have a value just under 0.8 for the best action from the start state.

## Writeup

In your writeup, explain:

* What maps you used for testing. The provided maze is just for documentation. You should have at least two other maps to test and tune your algorithm. Explain how the maps guided your development. Please do not use more than 5 maps in your writeup.
* 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?
* Plot a graph of performance on one of your maps. The x-axis should be number of training runs, and the y-axis should be average reward received. See the sample graph ([Sample graph](https://docs.google.com/spreadsheets/d/1ozZJ2vKxVOXHK0ZnLBcUwO4D76hyv_s9xON9J1nYx1Y/edit?usp=sharing)) for more information.
* Plot performance (on the same) graph of two simple policies and contrast their performance with your agent:
  + Random: an agent that makes a random action at each timestep. Comparing performance with random is a good baseline. If your agent cannot beat random performance it is not doing very well.
  + ε-greedy with ε=0.1.

# Part 3: Escaping from gridworld: the robotic truck driver\*

You will now focus on creating RL agents that have to act in a more realistic environment. For this part, you will have to create a robotic truck driver. Fortunately, the only decisions you will have to make are whether to wait for more cargo, or whether to head out and deliver the cargo already on the truck.

## The environment

The environment consists of a truck, a warehouse, customer houses, and time. The world is simple, with a warehouse at position 0. Each tick, there is a probability of 0.15 that a package will be created at the warehouse. If a package is created, the probability of a package being created at the next timestep increases by 0.02, to a maximum of 0.25. If no package is created, the probability decreases by 0.02, to a minimum of 0.05. For example, if at the first timestep a package is created, at the next timestep there will be a 0.17 probability of a package being created. Packages need to know when they were created (for computing reward) and the house number it should be delivered to. You can assume all house numbers are equiprobable. Note that although the transition model is explicit, trying to use it to make decisions would be difficult, and you are better off just using Q-learning.

The warehouse is able to store an infinite number of packages.

If your truck is at the warehouse, packages can be loaded onto the truck (this action takes no time) up to the maximum capacity of your truck. An important note is that the warehouse operates as a FIFO queue; that is, packages should be loaded onto the truck in the same order as they arrive. If a package is generated and your truck is at the warehouse, you can assume the package goes directly to the truck (if there is space). Loading the truck isn’t part of the problem you need to solve.

Customer houses are spaced equally along the road. For example, if the road is of length 20, then there is a customer house at each integer (1, 2, 3, … 18, 19, 20). The warehouse is at position 0.

## Actions

If your truck is at the warehouse, it can take one of two actions:

* Wait. The truck does not do anything and allows one unit of time to pass. This option can make sense if the truck is unloaded or mostly empty. Delivering a package to house 20 means the truck is away from the warehouse for 40 units of time during which packages can pile up.
* Deliver. The truck departs the warehouse with whatever packages it has. It drives out to the furthest house, dropping each package off at the appropriate house along the way. The truck moves 1 house per timestep. For example, if the truck has packages destined for houses 5, 11, and 13, the truck would spend 5 timesteps driving to house 5. It would then deliver that package and have packages for 11 and 13 remaining. It would spend 6 timesteps driving to house 11 and deliver that package. Then it would spend 2 timesteps to get to house 13 for its final delivery. It will then spend 13 timesteps driving back to the warehouse. If the agent chooses deliver and the truck is empty, it simply receives the penalty (described below) for starting the truck, does not leave the warehouse. The agent can select a different action at the next timestep.

Important: once the truck starts a delivery action it **must** complete it. So in the above example, if a truck was at house #1 and a new package arrived at the warehouse, the truck cannot immediately turn back to the warehouse. Instead it will be back in the warehouse in 25 timesteps. Thus, performing a deliver action must be a careful decision.

## Rewards

The agent has three sources of rewards:

1. Reward for delivering a package. Delivering a package is worth 30 times the road length, independent of the actual house number of the package. So for the above example, if the road length were 20, then the agent would receive a reward of (20\*30) = 600 per package, or 1800 in total.
2. A penalty for time required to deliver a package. At every timestep, for every package (both in the truck and in the warehouse), the agent receives a penalty of (current time - time package created). For example, if the package in the above example destined for house #5 sat in the warehouse for 3 timesteps before delivery began, the penalty would be 1+2+3+4+5+6+78 = 36. The first 3 terms are from the package sitting in the warehouse, the final 5 terms are when the package was on the truck to be delivered. The penalty is not given as a lump sum at the end, but is instead given to the agent at each timestep (this approach makes your life easier).
3. A penalty for starting the truck for the delivery action. For example, if this reward is -250, the agent loses 250 points for undertaking a delivery action. One sanity check is the agent should learn not to select the deliver action when the truck is empty. The agent should learn this behavior, and you should not code your simulation to forbid it.

## Inputs

Your program will accept the following command-line inputs:

* The truck’s capacity. For example, a 10 means the truck can handle 10 packages.
* The length of the road. For example, a 15 means the road is 15 units long with the furthest customer at house 15.
* The penalty for starting the truck on a run. For example, a value of -50 indicates an immediate reward of -50 for selecting the *deliver* option.
* The number of training iterations to run for. If given a -1 it should keep running continuously. (We won’t test running forever, it’s to motivate you to think about how to structure rewards.)

## Outputs

Your program should output the learned policy. Exactly how you do that is up to you. If you used a tabular representation you should use that to display the policy. If there are insufficient samples it is acceptable to output a “?” for a particular value. If you use a function approximator you should think about some key values to use to instantiate your function. This section is purposefully vague, as determining a useful representation is part of your task.

## Representation

One challenge with this problem is creating a state representation for your learner. For a gridworld, the state is (usually!) simply the (x,y) coordinates of the agent’s current position. How will you represent the state for this problem? Remember, the purpose of the state representation is to enable your learner to make better decisions by being able to predict what the future reward for various actions will be. What information is useful to know?

As a starting point, performing a deliver action when the truck is empty is probably a bad idea. What information can you provide your agent that will enable it to learn that? What other things might influence its decision making?

There are various approaches. I used a tabular representation for states similar to that used for the gridworld -- although obviously with different dimensions than the x- and y-coordinates. You could also use a function approximator. If you go this route, feel free to use any code you can find for a function approximator (just cite it in your writeup).

## Writeup

Explain your state representation, and why you chose it.

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).

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.

\* As a confession, I did this problem about 24 years ago as a grad student. I worked on it for several days and was worried as my presentation was the next day and I had nothing. The night before my presentation I deleted all of my code and started over. That worked much better and I got a result quickly. Think carefully about how RL programs are structured (rewatch the Q learning class video) and figure out where the various components of your program go. Then start writing code.