EECS545 Machine Learning
Homework #4
110 points

Due on 12pm (noon), MONDAY 3/28.

Reminder, while you are encouraged to think about problems in small groups, all written solutions must be independently generated. Please type or hand-write solutions legibly. While these questions require thought, please be as concise and clear in your answer as possible. Please address all questions to `eecs545qa@umich.edu` with a reference to the specific question in the subject line (E.g. RE: Homework # 3, Q2(c)). For any solutions that require programming, please submit commented code along with any figures that you are asked to plot.

Note this assignment is due on **Monday**, March 28 at 12:00 PM (noon). To ease your workload, we have made this homework the last homework of the semester. As such, it is slightly longer than past homeworks (worth **150** points), and you will have nearly a month to complete it. We do, however, strongly recommend starting early.

Note: Because of the Monday due date, all course participant will have access to the CSE building each late day after the deadline, so late days will be simply counted by whether or not the homework has been submitted by noon or not. Many of you have already used up late days, so please be aware of the 20% penalty for each day late. The solution for this homework will be released immediately following 96 hours after the deadline, so as always, you will receive no credit for any homework turned later than 96 hours after the deadline.

1 [30 points] Properties of Graphical Models

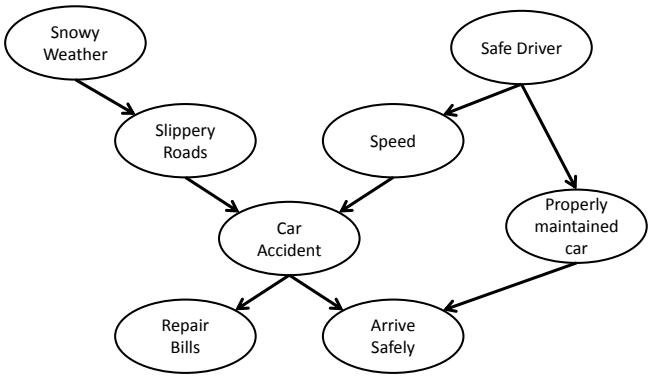
In this question, you will explore the construction of, independence properties of, and inference in Bayes nets.

(a) [**7 points**] Do Bishop PRML Exercise 8.3 (page 419).

(b) [**7 points**] Do Bishop PRML Exercise 8.4 (page 419).

(c) [**8 points**] Do Bishop PRML Exercise 8.11 (page 420).

(d) **[8 points]** Consider the following Bayes net:



(i) Is arriving safely independent of snowy weather?

(ii) Is a properly maintained car independent of a repair bill?

(iii) Is a properly maintained car conditionally independent of snow weather given a repair and a safe driver?

(iv) Is a repair bill conditionally independent of arriving safely, given a car accident?

(v) Is it possible to make properly maintained car independent of slippery roads given a repair bill by reversing one edge in the network? If so, which edge?

2 [22 points] Inference in Graphical Models

(a) **[8 points]** Do Bishop PRML Exercise 8.19 (page 421).

(b) **[6 points]** Do Bishop PRML Exercise 8.22 (page 421).

(c) **[8 points]** Do Bishop PRML Exercise 8.24 (page 421).

3 [26 points] Expectation Maximization

(a) **[6 points]** Do Bishop PRML Exercise 9.5 (page 456).

(b) **[6 points]** Do Bishop PRML Exercise 9.6 (page 456).

(c) **[6 points]** Do Bishop PRML Exercise 9.11 (page 457).

(d) **[8 points]** Do Bishop PRML Exercise 9.13 (page 457).

4 [22 points] K-means for image compression

In this problem, we will apply the K-means algorithm to lossy image compression, by reducing the number of colors used in an image. CTools contains a 512x512 image of a mandrill represented in 24-bit color. This means that, for each of the 262144 pixels in the image, there are three 8-bit numbers (each ranging from 0 to 255) that represent the red, green, and blue intensity values for that pixel. The straightforward representation of this image therefore takes about $262144 \times 3 = 786432$ bytes (a byte being 8 bits). To compress the image, we will use K-means to reduce the image to k = 16 colors. More specifically, each pixel in the image is considered a point in the three-dimensional (r; g; b)-space. To compress the image, we will cluster these points in color-space into 16 clusters, and replace each pixel with the closest cluster centroid. Follow the instructions below. Be warned that some of these operations can take a while (several minutes even on a fast computer)!

(a) Copy mandrill-large.tiff from Ctools. Start up MATLAB, and type `A = double(imread('mandrill-large.tiff'));` to read in the image. Now, `A` is a "three dimensional matrix," and `A(:,:,1)`, `A(:,:,2)` and `A(:,:,3)` are 512×512 arrays that respectively contain the red, green, and blue values for each pixel. Enter `imshow(uint8(round(A)));` to display the image.

(b) Since the large image has 262144 pixels and would take a while to cluster, we will instead run vector quantization on a smaller image. Repeat (a) with `mandrill-small.tiff`. Treating each pixel's (r, g, b) values as an element of $\mathbb{R}^3$, run K-means with 16 clusters on the pixel data from this smaller image, iterating (preferably) to convergence, but in no case for less than 30 iterations. For initialization, set each cluster centroid to the (r, g, b)-values of a randomly chosen pixel in the image.

(c) Take the matrix `A` from `mandrill-large.tiff`, and replace each pixel's (r, g, b) values with the value of the closest cluster centroid. Display the new image, and compare it visually to the original image. Hand in all your code and a printout of your compressed image.

(d) If we represent the image with these reduced (16) colors, by (approximately) what factor have we compressed the image?

5 [10 points] PCA

In class, we showed that PCA finds the "variance maximizing" directions onto which to project the data. In this problem, we find another interpretation of PCA. Suppose we are given a set of points $\{x^{(1)}, \ldots x^{(N)}\}$. Let us assume that we have as usual preprocessed the data to have zero-mean and unit variance in each coordinate. For a given unit-length vector $\mathbf{u}$, let $f_{\mathbf{u}}(x)$ be the projection

of point x onto the direction given by $\mathbf{u}$. In other words, if $V = \{\alpha\mathbf{u} : \alpha \in \mathbb{R}\}$, then

$$f_{\mathbf{u}}(x) = \arg\min_{v \in V} ||x - v||^2.$$

Show that the unit-length vector u that minimizes the mean squared error between projected points and original points corresponds to the first principal component for the data. I.e., show that

$$\arg\min_{\mathbf{u}:\mathbf{u}^T\mathbf{u}=1} \sum_{i=1}^{N} ||x^{(i)} - f_{\mathbf{u}}(x^{(i)})||_2^2$$

gives the first principal component.

Remark. If we are asked to find a k-dimensional subspace onto which to project the data so as to minimize the sum of squares distance between the original data and their projections, then we should choose the k-dimensional subspace spanned by the first principal components of the data. This problem shows that this result holds for the case of $k = 1$.

6 [16 points] Markov Decision Process

Consider an MDP with finite state and action spaces, and discount factor $\gamma < 1$. Let B be the Bellman update operator with $V \in \mathbb{R}^{|S|}$ a vector of values for each state. I.e., if $V^{'} = B(V) \in \mathbb{R}^{|S|}$, then

$$V^{'}(s) = R(s) + \gamma \max_{a \in A} \sum_{s' \in S} P_{sa}(s')V(s').$$

(a) **[12 points]** Prove that, for any two finite-valued vectors V_1, V_2, it holds true that

$$||B(V_1) - B(V_2)||_\infty \leq \gamma ||V_1 - V_2||_\infty$$

where

$$||V||_\infty = \max_{s \in S} |V(s)|.$$

(This shows that the Bellman update operator is a "γ-contraction in the max-norm.")

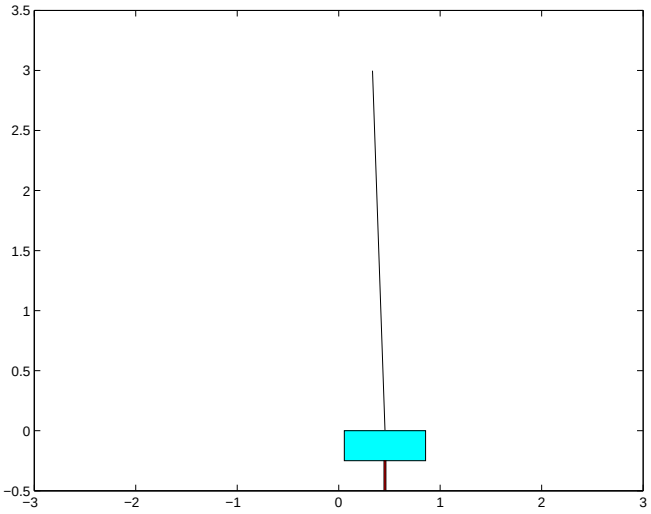
(b) **[4 points]** We say that V is a fixed point of B if $B(V) = V$. Using the fact that the Bellman update operator is a γ-contraction in the max-norm, prove that B has at most one fixed point—i.e., that there is at most one solution to the Bellman equations. You may assume that B has at least one fixed point.

7 [28 points] Reinforcement Learning: The inverted pendulum

In this problem, you will apply reinforcement learning to automatically design a policy for a difficult control task, without ever using any explicit knowledge of the dynamics of the underlying system.

The problem we will consider is the inverted pendulum or the pole-balancing problem.

Consider the figure shown. A thin pole is connected via a free hinge to a cart, which can move laterally on a smooth table surface. The controller is said to have failed if either the angle of the pole deviates by more than a certain amount from the vertical position (i.e., if the pole falls over), or if the cart's position goes out of bounds (i.e., if it falls off the end of the table). Our objective is to develop a controller to balance the pole with these constraints, by appropriately having the cart accelerate left and right.



We have written a simple Matlab simulator for this problem. The simulation proceeds in discrete time cycles (steps). The state of the cart and pole at any time is completely characterized by 4 parameters: the cart position x, the cart velocity $\dot{x}$, the angle of the pole θ measured as its deviation from the vertical position, and the angular velocity of the pole $\dot{\theta}$. Since it'd be simpler to consider reinforcement learning in a discrete state space, we have approximated the state space by a discretization that maps a state vector $(x, \dot{x}, \theta, \dot{\theta})$ into a number from 1 to `NUM_STATES`. Your learning algorithm will need to deal only with this discretized representation of the states.

At every time step, the controller must choose one of two actions — push (accelerate) the cart right, or push the cart left. (To keep the problem simple, there is no *do-nothing* action.) These are represented as actions 1 and 2 respectively in the code. When the action choice is made, the simulator updates

the state parameters according to the underlying dynamics, and provides a new discretized state.

We will assume that the reward $R(s)$ is a function of the current state only. When the pole angle goes beyond a certain limit or when the cart goes too far out, a negative reward is given, and the system is reinitialized randomly. At all other times, the reward is zero. Your program must learn to balance the pole using only the state transitions and rewards observed.

The files for this problem are available on CTools. Most of the the code has already been written for you, and you need to make changes only to `control.m` in the places specified. This file can be run in Matlab to show a display and to plot a learning curve at the end. Read the comments at the top of the file for more details on the working of the simulation.

(a) To solve the inverted pendulum problem, you will estimate a model (i.e., transition probabilities and rewards) for the underlying MDP, solve Bellman's equations for this estimated MDP to obtain a value function, and act greedily with respect to this value function.

Briefly, you will maintain a current model of the MDP and a current estimate of the value function. Initially, each state has estimated reward zero, and the estimated transition probabilities are uniform (equally likely to end up in any other state). During the simulation, you must choose actions at each time step according to some current policy. As the program goes along taking actions, it will gather observations on transitions and rewards, which it can use to get a better estimate of the MDP model. Since it is inefficient to update the whole estimated MDP after every observation, we will store the state transitions and reward observations each time, and update the model and value function/policy only periodically. Thus, you must maintain counts of the total number of times the transition from state s_i to state s_j using action a has been observed (similarly for the rewards). Note that the rewards at any state are deterministic, but the state transitions are not because of the discretization of the state space (several different but close configurations may map onto the same discretized state).

Each time a failure occurs (such as if the pole falls over), you should re-estimate the transition probabilities and rewards as the average of the observed values (if any). Your program must then use value iteration to solve Bellman's equations on the estimated MDP, to get the value function and new optimal policy for the new model. For value iteration, use a convergence criterion that checks if the maximum absolute change in the value function on an iteration exceeds some specified tolerance.

Finally, assume that the whole learning procedure has converged once several consecutive attempts (defined by the parameter `NO_LEARNING_THRESHOLD`) to solve Bellman's equation all converge in the first iteration. Intuitively, this indicates that the estimated model has stopped changing significantly.

The code outline for this problem is already in `control.m`, and you need to write code fragments only at the places specified in the file. There are

several details (convergence criteria etc.) that are also explained inside the code. Use a discount factor of $\gamma = 0 : 995$.

Implement the reinforcement learning algorithm as specified, and run it. How many trials (how many times did the pole fall over or the cart fall off) did it take before the algorithm converged?

(b) Plot a learning curve showing the number of time-steps for which the pole was balanced on each trial. You just need to execute `plot_learning_curve.m` after `control.m` to get this plot.

Credits

Some questions adopted/adapted from `http://www.stanford.edu/class/cs229/ps/ps3.pdf`, `http://www.stanford.edu/class/cs229/ps/ps4.pdf`, `http://www-anw.cs.umass.edu/rlr/domains.html`, and from Bishop PRML.