CS 229, Spring 2016
Problem Set #4: Unsupervised learning & RL

Due Thursday, June 2 at 11:00 pm on Gradescope.

Notes: (1) These questions require thought, but do not require long answers. Please be as concise as possible. (2) If you have a question about this homework, we encourage you to post your question on our Piazza forum, at `https://piazza.com/stanford/spring2016/cs229`. (3) If you missed the first lecture or are unfamiliar with the collaboration or honor code policy, please read the policy on Handout #1 (available from the course website) before starting work. (4) For problems that require programming, please include in your submission a printout of your code (with comments) and any figures that you are asked to plot.

If you are scanning your document by cellphone, please check the Piazza forum for recommended cellphone scanning apps and best practices.

1. **[11 points] EM for MAP estimation**

   The EM algorithm that we talked about in class was for solving a maximum likelihood estimation problem in which we wished to maximize

   $$\prod_{i=1}^{m} p(x^{(i)};\theta) = \prod_{i=1}^{m} \sum_{z^{(i)}} p(x^{(i)}, z^{(i)};\theta),$$

   where the $z^{(i)}$'s were latent random variables. Suppose we are working in a Bayesian framework, and wanted to find the MAP estimate of the parameters θ by maximizing

   $$\left(\prod_{i=1}^{m} p(x^{(i)}|\theta)\right) p(\theta) = \left(\prod_{i=1}^{m} \sum_{z^{(i)}} p(x^{(i)}, z^{(i)}|\theta)\right) p(\theta).$$

   Here, $p(\theta)$ is our prior on the parameters. Generalize the EM algorithm to work for MAP estimation. You may assume that $\log p(x, z|\theta)$ and $\log p(\theta)$ are both concave in θ, so that the M-step is tractable if it requires only maximizing a linear combination of these quantities. (This roughly corresponds to assuming that MAP estimation is tractable when x, z is fully observed, just like in the frequentist case where we considered examples in which maximum likelihood estimation was easy if x, z was fully observed.)

   Make sure your M-step is tractable, and also prove that $\prod_{i=1}^{m} p(x^{(i)}|\theta)p(\theta)$ (viewed as a function of θ) monotonically increases with each iteration of your algorithm.

2. **[22 points] EM application**

   Consider the following problem. There are P papers submitted to a machine learning conference. Each of R reviewers reads each paper, and gives it a score indicating how good he/she thought that paper was. We let $x^{(pr)}$ denote the score that reviewer r gave to paper p. A high score means the reviewer liked the paper, and represents a recommendation from that reviewer that it be accepted for the conference. A low score means the reviewer did not like the paper.

We imagine that each paper has some "intrinsic," true value that we denote by μ_p, where a large value means it's a good paper. Each reviewer is trying to estimate, based on reading the paper, what μ_p is; the score reported $x^{(pr)}$ is then reviewer r's guess of μ_p.

However, some reviewers are just generally inclined to think all papers are good and tend to give all papers high scores; other reviewers may be particularly nasty and tend to give low scores to everything. (Similarly, different reviewers may have different amounts of variance in the way they review papers, making some reviewers more consistent/reliable than others.) We let ν_r denote the "bias" of reviewer r. A reviewer with bias ν_r is one whose scores generally tend to be ν_r higher than they should be.

All sorts of different random factors influence the reviewing process, and hence we will use a model that incorporates several sources of noise. Specifically, we assume that reviewers' scores are generated by a random process given as follows:

$$\begin{aligned} y^{(pr)} &\sim \mathcal{N}(\mu_p, \sigma_p^2), \\ z^{(pr)} &\sim \mathcal{N}(\nu_r, \tau_r^2), \\ x^{(pr)}|y^{(pr)}, z^{(pr)} &\sim \mathcal{N}(y^{(pr)} + z^{(pr)}, \sigma^2). \end{aligned}$$

The variables $y^{(pr)}$ and $z^{(pr)}$ are independent; the variables (x, y, z) for different paper-reviewer pairs are also jointly independent. Also, we only ever observe the $x^{(pr)}$'s; thus, the $y^{(pr)}$'s and $z^{(pr)}$'s are all latent random variables.

We would like to estimate the parameters $\mu_p, \sigma_p^2, \nu_r, \tau_r^2$. If we obtain good estimates of the papers' "intrinsic values" μ_p, these can then be used to make acceptance/rejection decisions for the conference.

We will estimate the parameters by maximizing the marginal likelihood of the data $\{x^{(pr)}; p = 1, \ldots, P, r = 1, \ldots, R\}$. This problem has latent variables $y^{(pr)}$ and $z^{(pr)}$, and the maximum likelihood problem cannot be solved in closed form. So, we will use EM. Your task is to derive the EM update equations. Your final E and M step updates should consist only of addition/subtraction/multiplication/division/log/exp/sqrt of scalars; and addition/subtraction/multiplication/inverse/determinant of matrices. For simplicity, you need to treat only $\{\mu_p, \sigma_p^2; p = 1 \ldots P\}$ and $\{\nu_r, \tau_r^2; r = 1 \ldots R\}$ as parameters. I.e. treat σ^2 (the conditional variance of $x^{(pr)}$ given $y^{(pr)}$ and $z^{(pr)}$) as a fixed, known constant.

(a) In this part, we will derive the E-step:

(i) The joint distribution $p(y^{(pr)}, z^{(pr)}, x^{(pr)})$ has the form of a multivariate Gaussian density. Find its associated mean vector and covariance matrix in terms of the parameters $\mu_p, \sigma_p^2, \nu_r, \tau_r^2$, and σ^2.

[Hint: Recognize that $x^{(pr)}$ can be written as $x^{(pr)} = y^{(pr)} + z^{(pr)} + \epsilon^{(pr)}$, where $\epsilon^{(pr)} \sim \mathcal{N}(0, \sigma^2)$ is independent Gaussian noise.]

(ii) Derive an expression for $Q_{pr}(y^{(pr)}, z^{(pr)}) = p(y^{(pr)}, z^{(pr)}|x^{(pr)})$ (E-step), using the rules for conditioning on subsets of jointly Gaussian random variables (see the notes on Factor Analysis).

(b) Derive the M-step updates to the parameters $\{\mu_p, \nu_r, \sigma_p^2, \tau_r^2\}$. [Hint: It may help to express the lower bound on the likelihood in terms of an expectation with respect to $(y^{(pr)}, z^{(pr)})$ drawn from a distribution with density $Q_{pr}(y^{(pr)}, z^{(pr)})$.]

Remark. In a recent machine learning conference, John Platt (whose SMO algorithm you've seen) implemented a method quite similar to this one to estimate the papers' true scores μ_p. (There, the problem was a bit more complicated because not all reviewers reviewed every paper, but the essential ideas are the same.) Because the model tried to estimate and correct for reviewers' biases ν_r, its estimates of μ_p were significantly more useful for making accept/reject decisions than the reviewers' raw scores for a paper.

3. **[14 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 $\{x^{(1)}, \ldots, x^{(m)}\}$. 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 u, let $f_u(x)$ be the projection of point x onto the direction given by u. I.e., if $\mathcal{V} = \{\alpha u : \alpha \in \mathbb{R}\}$, then

   $$f_u(x) = \arg\min_{v \in \mathcal{V}} ||x - v||^2.$$

   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

   $$\arg\min_{u:u^Tu=1} \sum_{i=1}^{m} \|x^{(i)} - f_u(x^{(i)})\|_2^2 \ .$$

   gives the first principal component.

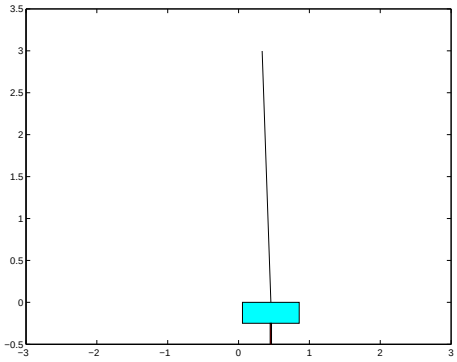
   Remark. If we are asked to find a k-dimensional subspace onto which to project the data so as to minimize the sum of squares distance between the original data and their projections, then we should choose the k-dimensional subspace spanned by the first k principal components of the data. This problem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

4. **[12 points] Independent components analysis**

   For this question you will implement the Bell and Sejnowski ICA algorithm, as covered in class. The files you'll need for this problem are in `/afs/ir/class/cs229/ps/ps4/q4`. The file `mix.dat` contains a matrix with 5 columns, with each column corresponding to one of the mixed signals x_i. The file `bellsej.m` contains starter code for your implementation.

   Implement and run ICA, and report what was the W matrix you found. Please make your code clean and very concise, and use symbol conventions as in class. To make sure your code is correct, you should listen to the resulting unmixed sources. (Some overlap in the sources may be present, but the different sources should be pretty clearly separated.)

   Note: In our implementation, we **annealed** the learning rate α (slowly decreased it over time) to speed up learning. We briefly describe in `bellsej.m` what we did, but you should feel free to play with things to make it work best for you. In addition to using the variable learning rate to speed up convergence, one thing that we also tried was choosing a random permutation of the training data, and running stochastic gradient ascent visiting the training data in that order (each of the specified learning rates was then used for one full pass through the data); this is something that you could try, too.



5. **[16 points] Markov decision processes**

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

6. **[25 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.[1]

   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

[1] The dynamics are adapted from `http://www-anw.cs.umass.edu/rlr/domains.html`

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 in `/afs/ir/class/cs229/ps/ps4/q6`. 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.[2]

(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.

[2] Note that the routine for drawing the cart does not work in Octave.

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.