CS 189 Introduction to Machine Learning
Fall 2017

HW1

This homework is due **Friday, September 1 at 12pm noon.**

1 Getting Started

You may typeset your homework in latex or submit neatly handwritten and scanned solutions. Please make sure to start each question on a new page, as grading (with Gradescope) is much easier that way! Deliverables:

1. Submit a PDF of your writeup to assignment on Gradescope, “HW[n] Write-Up”
2. Submit all code needed to reproduce your results, “HW[n] Code”.
3. Submit your test set evaluation results, “HW[n] Test Set”.

After you’ve submitted your homework, be sure to watch out for the self-grade form.

(a) Before you start your homework, write down your team. Who else did you work with on this homework? List names and email addresses. In case of course events, just describe the group. How did you work on this homework? Any comments about the homework?

(b) Please copy the following statement and sign next to it:

I certify that all solutions are entirely in my words and that I have not looked at another student’s solutions. I have credited all external sources in this write up.

2 "Sample Complexity" of coupon collecting

Roald Dahl's mathematical version of his book has the following problem. Willy Wonka has hidden 50 different types of cards in his chocolate wrappers, and some of the people to find all 50 types will be allowed to enter his factory and participate in a questionable experiment. Charlie goes to a local store to buy his chocolates, and can afford at most one chocolate a day. The store stocks 20 chocolates of each card type, and every time a chocolate is bought, it is replaced with another chocolate containing an identical card. Charlie picks a chocolate uniformly at random.

(a) What is the expected number of days it takes Charlie to qualify for Willy Wonka's challenge if he picks a random chocolate every time he goes to the shop?

Solution: Let X_i be the number of days required to pick up the ith distinct card. (where the first card is the 0th card) Each X_i is a geometric distribution with parameter $p = \frac{50-i}{50}$. Then, $E[X_i] = \frac{50}{50-i}$.

$$E[T] = \sum_{i=0}^{49} E[X_i] = 50 \sum_{i=0}^{49} \frac{1}{50-i} = 50 \sum_{i=1}^{50} \frac{1}{i} \sim 225$$

(b) The game has now changed. Instead of having to collect all d card types, at the end of the game Willy Wonka will pick a card uniformly at random and if someone has that card in their collection, they win a smaller prize. **If you want your probability of winning to be at least $1-\delta_w$, how many cards should you have in your collection?**

Solution: We provide solutions for both possible interpretations, although we expected the first.

Solution 1 (Card types)

Say Charlie has c distinct card types. Willy Wonka then picks one of Charlie's card types with probability $\frac{c}{d}$. We need this probability to exceed $1-\delta_w$, so we have

$$\frac{c}{d} > 1-\delta_w \implies c > d(1-\delta_w)$$

Solution 2 (Cards)

We look at the complement instead: examine the probability Charlie *doesn't* win a prize. If Charlie doesn't win, Willy Wonka picked a card that Charlie doesn't have. For each of Charlie's picks, Charlie must pick one of the other cards, which occurs with probability $\frac{d-1}{d}$. Repeating this n times: $(\frac{d-1}{d})^n$. Remember, this is the complement of the event we're interested in, so:

$$1 - (\frac{d-1}{d})^n$$

We need this probability to be greater than $1-\delta_w$, so

$$1-(\frac{d-1}{d})^n > 1-\delta_w$$
$$(\frac{d-1}{d})^n < \delta_w$$
$$n > \frac{\ln(\delta_w)}{\ln(\frac{d-1}{d})}$$

When d is large, we can use the approximation $\ln(1-x) \sim -x$ which holds for small enough x to conclude that
$$n > d\ln(1/\delta_w).$$

As you can see, one can think of this as Willy Wonka having picked the card before the game was even started, since this does not influence the resulting probability.

(c) Charlie just decides to buy chocolates for n days. At the end of this, he has some random number of distinct cards. **Given that there are d distinct cards and assuming that Charlie buys αd chocolates, what is the probability that Charlie wins a prize?**

Solution: See the second solution of the previous part, where $n = \alpha d$:

$$1-(\frac{d-1}{d})^{\alpha d}$$

(d) What happens as d gets large?

Solution: We can approximate the above using the fact that $\lim_{d\to\infty}(1+\frac{x}{d})^d = e^x$, giving us:

$$1-(\frac{d-1}{d})^{\alpha d} = 1-(1-\frac{1}{d})^{\alpha d} = 1-(1-\frac{\alpha}{\alpha d})^{\alpha d} = 1-(e^{-\alpha})$$

(e) Now, think about something that might seem completely different. We want to learn a completely unstructured function f on a finite domain of size d. We get training data from random samples, each of which picks a domain point U uniformly at random and then returns us $(U, f(U))$ — i.e. the domain point along with the function f's evaluation on that domain point. **How big of a training set should we collect so that with probability $1-\delta$ we will be able to successfully estimate the function when presented with a uniformly drawn domain point.**

Solution: The analogy is as follows: estimating the function correctly when presented with a point is equivalent to having a card when Willy Wonka picks a prize-winner. So, we have our probability of succeeding, given n draws already, $1-(\frac{d-1}{d})^n$. We need this probability to exceed $1-\delta$, so we have

$$1-(\frac{d-1}{d})^n > 1-\delta \implies (\frac{d-1}{d})^n < \delta \implies n \geq \frac{\ln\delta}{\ln(\frac{d-1}{d})}.$$

As before, when d is large, we can use the approximation $\ln(1-x) \sim -x$ which holds for small enough x to conclude that

$$n > d \ln(1/\delta).$$

3 The accuracy of learning decision boundaries

This problem exercises your basic probability (e.g. from 70) in the context of understanding why lots of training data helps improve the accuracy of learning things.

For each $\theta \in (1/4, 3/4)$, define $f_\theta : [0,1] \to \{0,1\}$, such that

$$f_\theta(x) = \begin{cases} 1 \text{ if } x > \theta \\ 0 \text{ otherwise.} \end{cases}$$

The function is plotted in Figure **??**.

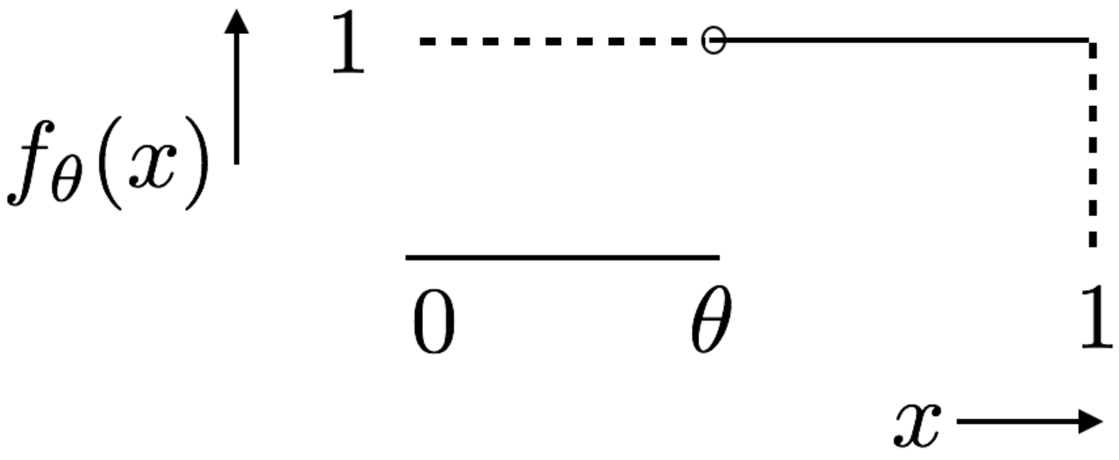

Figure 1: Plot of function $f_\theta(x)$ against x.

We draw samples $X_1, X_2, \dots, X_n$ uniformly at random and i.i.d. from the interval $[0,1]$. Our goal is to learn an estimate for θ from n random samples $(X_1, f_\theta(X_1)), (X_2, f_\theta(X_2)), \dots, (X_n, f_\theta(X_n))$.

Let $N_0 := \#\{i : f_\theta(X_i) = 0\}$ denote the number of samples we obtain that have function evaluation 0.

Let $T_{min} = \max(\{\frac{1}{4}\} \cup \{X_i | f_\theta(X_i) = 0\})$. We know that the true θ must be larger than T_{min}.

Let $T_{max} = \min(\{\frac{3}{4}\} \cup \{X_i | f_\theta(X_i) = 1\})$. We know that the true θ must be smaller than T_{max}.

The gap between T_{min} and T_{max} represents the uncertainty we will have about the true θ given the training data that we have received.

(a) What is the probability that $T_{max} - \theta > \varepsilon$ as a function of ε.

Solution: The probability of $T_{max} > \theta + \varepsilon$ is the probability that none of the samples X_i lie in $(\theta, \theta + \varepsilon]$. We sample X_i uniformly from $[0,1]$, so the probability of success for X_i is ε. For n samples, we have $(1-\varepsilon)^n$. Note that since T_{max} is by construction $<= \frac{3}{4}$, we are doomed if $\theta + \varepsilon > \frac{3}{4}$, so:

$$\Pr(T_{max} - \theta > \varepsilon) = \begin{cases} (1-\varepsilon)^n & \theta + \varepsilon < \frac{3}{4} \\ 0 & \text{o.w.} \end{cases}$$

(b) What is the probability that $\theta - T_{min} > \varepsilon$ as a function of ε.

Solution: Similar analysis here, except our lower bound is $\frac{1}{4}$:

$$\Pr(\theta - T_{min} > \varepsilon) = \begin{cases} (1-\varepsilon)^n & \theta - \varepsilon > \frac{1}{4} \\ 0 & \text{o.w.} \end{cases}$$

(c) Suppose that you would like to have an estimate for θ within an accuracy of 2ε with probability at least $1-\delta$. Please bound or estimate how big of an n do you need?

Solution: One way to obtain θ within a window of size 2ε is to have both T_{max} and T_{min} be within ε of it. Define random variables $L = \theta - T_{min}, U = T_{max} - \theta$, and note that we doomed if $L + U > 2\varepsilon$. For this to be true, we must have either $L > \varepsilon$ or $U > \varepsilon$, and so we have the inclusion of events: $\{L + U > 2\varepsilon\} \subseteq \{L > \varepsilon\} \cup \{U > \varepsilon\}$. Consequently,

$$\begin{aligned} \Pr(L + U > 2\varepsilon) &\leq \Pr(\{L > \varepsilon\} \cup \{U > \varepsilon\}) \\ &\leq 2(1-\varepsilon)^n \qquad \text{union bound.} \end{aligned}$$

We must ensure that this probability is upper bounded by δ, which ensures that we succeed with probability at least $1-\delta$. Solving for n, we have

$$\begin{aligned} 2(1-\varepsilon)^n &< \delta \\ n &> \frac{\ln(\frac{2}{\delta})}{\ln(1/(1-\varepsilon))}. \end{aligned}$$

Again, using the approximation $\ln(1-x) \sim -x$, we have $n > \frac{1}{\varepsilon}\ln(2/\delta)$ for ε small.

(d) Let us say that instead of getting random samples $(X_i, f(X_i))$, we were allowed to choose where to sample the function, but you had to choose all the places you were going to sample in advance. Propose a method to estimate θ. How many samples suffice to achieve an estimate that is ε-close (within an interval of size 2ε) as above? (**Hint:** You need not use a randomized strategy.)

Solution: Pick n points uniformly spaced on the interval $(\frac{1}{4}, \frac{3}{4})$. Then, the ith sample $X_i = \frac{1}{4} + \frac{i}{2n}$. Since we have n points, we create intervals of length $\frac{1}{2n}$. If our intervals are smaller than 2ε, we can guarantee that we estimate θ within an interval of 2ε. Solving for n, we have $\frac{1}{2n} < 2\varepsilon$ and so $n > \frac{1}{4\varepsilon}$ samples are sufficient.

(e) Suppose that you could pick where to sample the function adaptively — choosing where to sample the function in response to what the answers were previously. Propose a method to estimate θ. How many samples suffice to achieve an estimate that is ε-close (within an interval of size 2ε) as above?

Solution: Use binary search: start with three pointers, $s = 1/4, e = 3/4$ with m as the midpoint. If $f(m) = 0$, set $s = m$ and recompute the midpoint (i.e., search over the second half of the range). Otherwise, $f(m) = 1$ and set $e = m$ (i.e., search over the first half of the range). For each point sampled, we reduce the size of the range by half, so after n points, the interval we consider is $\frac{1}{2^{n+1}}$. We want this to be less than 2ε, and so

$$\frac{1}{2^{n+1}} < 2\varepsilon \implies n > \log_2(1/\varepsilon) - 2.$$

(f) Compare the scaling of n with ε and δ in the three sampling approaches above: random, deterministic, and adaptive.

Solution:

(a) For random, n is logarithmic in $1/\delta$. For ε, we use the approximation $\ln(1/(1-\varepsilon)) \sim \varepsilon$ to conclude that n is inversely related to ε.

(b) For deterministic, n is inversely related to ε. Note that this is the same scaling as choosing random evaluation points.

(c) For adaptive, n is logarithmic in $\frac{1}{\varepsilon}$.

4 Eigenvalue and Eigenvector review

A square matrix $A \in \mathbb{R}^{d\times d}$ has a (right) eigenvalue $\lambda \in \mathbb{C}$ and (right) eigenvector $\vec{x} \in \mathbb{C}^d \setminus 0$ if $P\vec{x} = \lambda\vec{x}$. Left eigenvalues and eigenvectors are defined analogously — $\vec{x}^T P = \lambda\vec{x}^T$. Since the definition is scale invariant (if $\vec{x}$ is an eigenvector, then $t\vec{x}$ is an eigenvector for any $t \neq 0$), we adopt the convention that each eigenvector has norm 1.

(a) Compute the right and left eigenvalues and eigenvectors of the following matrices.

i) $A = \begin{bmatrix} 3 & 2 \\ 1 & 3 \end{bmatrix}$

Solution:

Left eigenvectors are given by row vectors. We provide their column versions in the following solutions.

Right: $\lambda_1 = 3+\sqrt{2}, \lambda_2 = 3-\sqrt{2}, v_1 = \begin{bmatrix} \sqrt{2/3} \\ \sqrt{1/3} \end{bmatrix}, v_2 = \begin{bmatrix} -\sqrt{2/3} \\ \sqrt{1/3} \end{bmatrix}$

Left: $\lambda_1 = 3+\sqrt{2}, \lambda_2 = 3-\sqrt{2}, v_1 = \begin{bmatrix} \sqrt{1/3} \\ \sqrt{2/3} \end{bmatrix}, v_2 = \begin{bmatrix} -\sqrt{1/3} \\ \sqrt{2/3} \end{bmatrix}$

ii) $B = \begin{bmatrix} 5 & 2 \\ 2 & 5 \end{bmatrix}$

Solution: Since this matrix is symmetric, the right and left eigenvectors are the same too.

Right: $\lambda_1 = 7, \lambda_2 = 3, v_1 = \begin{bmatrix} 1/\sqrt{2} \\ 1/\sqrt{2} \end{bmatrix}, v_2 = \begin{bmatrix} -1/\sqrt{2} \\ 1/\sqrt{2} \end{bmatrix}$

Left: $\lambda_1 = 7, \lambda_2 = 3, v_1 = \begin{bmatrix} 1/\sqrt{2} \\ 1/\sqrt{2} \end{bmatrix}, v_2 = \begin{bmatrix} -1/\sqrt{2} \\ 1/\sqrt{2} \end{bmatrix}$

iii) A^2

Solution: We have $A^2 = AA = \begin{bmatrix} 11 & 12 \\ 6 & 11 \end{bmatrix}$.

Right: $\lambda_1 = 11+6\sqrt{2}, \lambda_2 = 11-6\sqrt{2}, v_1 = \begin{bmatrix} \sqrt{2/3} \\ \sqrt{1/3} \end{bmatrix}, v_2 = \begin{bmatrix} -\sqrt{2/3} \\ \sqrt{1/3} \end{bmatrix}$

Left: $\lambda_1 = 11+6\sqrt{2}, \lambda_2 = 11-6\sqrt{2}, v_1 = \begin{bmatrix} \sqrt{1/3} \\ \sqrt{2/3} \end{bmatrix}, v_2 = \begin{bmatrix} -\sqrt{1/3} \\ \sqrt{2/3} \end{bmatrix}$

iv) B^2

Solution: B is a symmetric matrix, so its eigenvalues are given by the square of the eigenvalues of B, with the eigenvectors staying the same. Hence, we have:

Right: $\lambda_1 = 49, \lambda_2 = 9, v_1 = \begin{bmatrix} 1/\sqrt{2} \\ 1/\sqrt{2} \end{bmatrix}, v_2 = \begin{bmatrix} -1/\sqrt{2} \\ 1/\sqrt{2} \end{bmatrix}$

Left: $\lambda_1 = 49, \lambda_2 = 9, v_1 = \begin{bmatrix} 1/\sqrt{2} \\ 1/\sqrt{2} \end{bmatrix}, v_2 = \begin{bmatrix} -1/\sqrt{2} \\ 1/\sqrt{2} \end{bmatrix}$

iv) AB

Solution: It is fine to leave the eigenvectors for the following two problems unnormalized.

Right: $\lambda_1 = 18+\sqrt{177}, \lambda_2 = 18-\sqrt{177}, v_1 = \begin{bmatrix} \frac{1}{11}(1+\sqrt{177} \\ 1 \end{bmatrix}, v_2 = \begin{bmatrix} \frac{1}{11}(1-\sqrt{177}) \\ 1 \end{bmatrix}$

Left: $\lambda_1 = 18+\sqrt{177}, \lambda_2 = 18-\sqrt{177}, v_1 = \begin{bmatrix} \frac{1}{16}(1+\sqrt{177} \\ 1 \end{bmatrix}, v_2 = \begin{bmatrix} \frac{1}{16}(1-\sqrt{177}) \\ 1 \end{bmatrix}$

v) BA

Solution:

Right: $\lambda_1 = 18+\sqrt{177}, \lambda_2 = 18-\sqrt{177}, v_1 = \begin{bmatrix} \frac{1}{11}(-1+\sqrt{177} \\ 1 \end{bmatrix}, v_2 = \begin{bmatrix} \frac{1}{11}(-1-\sqrt{177}) \\ 1 \end{bmatrix}$

Left: $\lambda_1 = 18+\sqrt{177}, \lambda_2 = 18-\sqrt{177}, v_1 = \begin{bmatrix} \frac{1}{16}(-1+\sqrt{177} \\ 1 \end{bmatrix}, v_2 = \begin{bmatrix} \frac{1}{16}(-1-\sqrt{177}) \\ 1 \end{bmatrix}$

(b) Compute the singular value decompositions of the matrices above. In addition, please compute the SVD of: $C = \begin{bmatrix} 5 & 2 \\ 2 & 5 \\ 3 & 2 \\ 1 & 3 \end{bmatrix}$

Solution: We begin with the reduced SVD of C:

$$C = \begin{bmatrix} 0.59 & -0.66 \\ 0.60 & 0.59 \\ 0.42 & -0.24 \\ 0.34 & 0.40 \end{bmatrix} \begin{bmatrix} 8.34 & 0 \\ 0 & 3.39 \\ 0 & 0 \\ 0 & 0 \end{bmatrix} \begin{bmatrix} 0.69 & -0.73 \\ 0.73 & 0.69 \end{bmatrix}^\top$$

i. $A = \begin{bmatrix} 0.76 & -0.65 \\ 0.65 & 0.76 \end{bmatrix} \begin{bmatrix} 4.54 & 0 \\ 0 & 1.54 \end{bmatrix} \begin{bmatrix} 0.65 & -0.76 \\ 0.76 & 0.65 \end{bmatrix}^\top$

ii. $B = \begin{bmatrix} \frac{1}{\sqrt{2}} & -\frac{1}{\sqrt{2}} \\ \frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} \end{bmatrix} \begin{bmatrix} 7 & 0 \\ 0 & 3 \end{bmatrix} \begin{bmatrix} \frac{1}{\sqrt{2}} & -\frac{1}{\sqrt{2}} \\ \frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} \end{bmatrix}^\top$

iii. $A^2 = \begin{bmatrix} 0.79 & -0.61 \\ 0.61 & 0.79 \end{bmatrix} \begin{bmatrix} 20.4 & 0 \\ 0 & 2.4 \end{bmatrix} \begin{bmatrix} 0.61 & -0.79 \\ 0.79 & 0.61 \end{bmatrix}^\top$

iv. $B^2 = \begin{bmatrix} \frac{1}{\sqrt{2}} & -\frac{1}{\sqrt{2}} \\ \frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} \end{bmatrix} \begin{bmatrix} 49 & 0 \\ 0 & 9 \end{bmatrix} \begin{bmatrix} \frac{1}{\sqrt{2}} & -\frac{1}{\sqrt{2}} \\ \frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} \end{bmatrix}^\top$

v. $AB = \begin{bmatrix} 0.78 & -0.63 \\ 0.63 & 0.78 \end{bmatrix} \begin{bmatrix} 31.71 & 0 \\ 0 & 4.64 \end{bmatrix} \begin{bmatrix} 0.68 & -0.73 \\ 0.73 & 0.68 \end{bmatrix}^\top$

vi. $BA = \begin{bmatrix} 0.73 & -0.68 \\ 0.68 & 0.73 \end{bmatrix} \begin{bmatrix} 31.71 & 0 \\ 0 & 4.64 \end{bmatrix} \begin{bmatrix} 0.63 & -0.78 \\ 0.78 & 0.63 \end{bmatrix}^\top$

(c) Show from the definition of a right eigenvalue that the quantity λ is an eigenvalue with associated eigenvector $\vec{x}$ iff for all $1 \le i \le n$, we have

$$(\lambda - A_{ii})x_i = \sum_{j \neq i} A_{ij} x_j.$$

Solution: Distribute, and we have $\lambda x_i - A_{ii} x_i = \sum_{j \neq i} A_{ij} x_j$. So,

$$\lambda x_i = \sum_j A_{ij} x_j = A_i x_i$$

If this is true for all i, then $\lambda \vec{x} = A\vec{x}$ and λ is therefore an eigenvalue.

(d) Now for an arbitrary eigenvalue λ of A and its associated eigenvector $\vec{x}$, choose index i such that $|x_i| \ge |x_j|$ for all $j \neq i$. For such an index i, show that

$$|\lambda - A_{ii}| \le \sum_{j \neq i} |A_{ij}|.$$

You have just proved Gershgorin's circle theorem, which states that all the eigenvalues of a $d \times d$ matrix lie within the union of d disks in the complex plane, where disk i has center A_{ii}, and radius $\sum_{j \neq i} |A_{ij}|$.

Solution: From the previous part, we see that

$$|\lambda - A_{ii}| = \frac{1}{|x_i|} \left| \sum_{j \neq i} A_{ij} x_j \right| \leq \sum_{j \neq i} |A_{ij}| \frac{|x_j|}{|x_i|}. \tag{1}$$

Choosing the index i proves the theorem.

5 Fun with least squares

Ordinary least squares for a scalar helps us learn linear predictors for a scalar b given observations $\vec{a}$. This is done by taking our training data points and constructing a tall vector $\vec{b}$ by stacking up our training data's b_i and a corresponding tall matrix A by stacking up our training data's $\vec{a}_i^T$, and then finding the weights $\vec{x}$ that minimize $\|A\vec{x} - \vec{b}\|_2^2$. The resulting weights $\vec{x}$ can be used to predict b's by $\vec{x}^T \vec{a}$.

Let us think about the components of $\vec{a}$ as being a sequence of measurements in time. The first measurement, the second measurement, the third measurement, and so on.

(a) Suppose that you wanted to construct not one linear predictor for b but a sequence of them. One that just used the first measurement. One that used the first two measurements. All the way up to one that uses all the measurements. How would you do this in the most straightforward manner?

Solution: Before providing a solution to the problem itself, let us expound a little on what is being asked. Think of the components of $\vec{a}$ as signifying the speeds of a robot over time, and b as where the robot ends up. We obtain training samples of these measurements as $\{\vec{a}_i, b_i\}$ for $i \in \{1, 2, \dots, n\}$, by collecting n independent trajectories of the robot. We then stack these training samples up as rows of a matrix A and vector $\vec{b}$, respectively. Now, predicting the scalar b from $\vec{a}$ can be done in the traditional way, via the training samples. Note that we have turned the problem of predicting a scalar from a vector into a regression problem.

Note that since each measurement is a coordinate of vector $\vec{a}$, a training measurement is given by an n-vector, and corresponds to one column of the matrix A. These columns (or training measurements) can be viewed as arriving in sequence, and we must predict the training vector $\vec{b}$ from these measurements. Let us call the ith column of the A matrix a^i, and define the matrix A_j as the submatrix consisting of the first j columns of A. In order to solve for the ith linear predictor, we solve the least squares problem $\hat{x}^i = \arg\min_{x \in \mathbb{R}^i} \|A_i x - \vec{b}\|_2^2$. The prediction on the training data is given by $\hat{b}^i = A_i \hat{x}^i$.

(b) Someone suggests that maybe the measurements themselves are partially predictable from the previous measurements. They suggest a two part strategy. First we predict the next measurement based on the measurements so far. And then we look at the difference (sometimes deemed the “innovation”) between the actual measurement we made and our prediction for it, and just use that to update our prediction for b.

Give a way to learn (from the training data) these best predictors of the next measurement from the previous measurements as well as learn (from the training data) the weights used to update the prediction for b based on the innovation.

Solution: Every time we get a new training measurement, there is a component of that measurement in the subspace spanned by the previous measurements, and another component that is orthogonal to this subspace. From the geometric intuition of least squares, we can find the projection of a^i onto the subspace formed by $a^1, \dots, a^{i-1}$ by solving $\hat{a}^i = \min_{x \in \mathbb{R}^{i-1}} \|a^i - A_{i-1}x\|_2^2$. This gives us the best linear predictor of a^i from the past. The innovation is the difference $a^i - \hat{a}^i$.

We now replace the column a^i with $a^i - \hat{a}^i$ to form another matrix $\tilde{A}$.

We can now predict the vector $\vec{b}$ from the matrix $\tilde{A}_i$. As before, we would like to solve the least squares problem $\min_{x \in \mathbb{R}^i} \|\tilde{A}_i x - b\|_2^2$. Note that our predictions $\hat{b}^i$ are exactly the same as in part (a), since the column space of the matrices A_i and $\tilde{A}_i$ are the same for each i.

An efficient way to solve the least squares problem in this case is to estimate the error $\vec{b} - \hat{b}^{i-1}$ by using just the innovation $a^i - \hat{a}^i$, and adding that error prediction to $\hat{b}^{i-1}$. Again, this is justified by the geometric interpretation of least squares: since $\hat{b}^{i-1}$ is the projection of $\vec{b}$ onto the subspace spanned by the first $i-1$ columns of A, and so the only useful new information is in the innovation sequence, which is orthogonal to this subspace. As a result, our regression vector at time i can be simply updated from the regression vector at time $i-1$ by simply adding a coefficient that multiplies the innovation $a^i - \hat{a}^i$.

(c) Is this two-part prediction strategy equivalent to the straightforward approach? Why or why not?

HINT: Think about what it might mean to orthonormalize the columns of the training matrix A above.

Solution: Note that by the definition of the projection operations noted above, we were able to make each column of the matrix $\tilde{A}$ orthogonal.

As explained in the previous parts, our prediction of the vector $\vec{b}^i$ is the same for both parts (a) and (b). However, the regression coefficients estimated from measurements (part (a)) are different from those estimated from the innovations (part (b)).

This is a form of feature preprocessing, that allows us to make the features in our problem orthogonal. With an additional normalization step, this is equivalent to performing Gram-Schmidt orthonormalization of the matrix A.

(d) **(BONUS — but worth doing)** A student complains to you that the case of learning a linear predictor for $\vec{b}$ given $\vec{a}$ seems too complicated. She suggests just setting up the problem directly as learning a matrix X so that $\vec{b} \approx X\vec{a}$. She takes the training data $(\vec{a}_i, \vec{b}_i)$ and stacks them **horizontally** into fat matrices A and B and computes the hypothetical difference matrix $B - XA$.

She would like to minimize an appropriate norm for this and chooses the simple entry-wise 2-norm (also known as the Frobenius norm) squared. So $\min_X \|B - XA\|_F^2$. Recall that the Frobenius norm of a real matrix (need not be square) L is $\|L\|_F^2 = \text{trace}(L^T L)$. Use this and vector calculus to directly solve for the minimizing X.

Solution: Recall the following trace identity for two matrices $P \in \mathbb{R}^{p\times q}$ and $Q \in \mathbb{R}^{q\times p}$.

$$\text{trace}(PQ) = \text{trace}(QP).$$

(You can show this by writing out the trace of both matrices and verifying that they are equal.)

Now expand the objective as

$$\begin{aligned}\|B - XA\|_F^2 &= \text{trace}((B - XA)^\top (B - XA)) \\ &= \text{trace}(B^\top B) - 2\text{trace}(B^\top XA) + \text{trace}(A^\top X^\top XA) \\ &= \text{trace}(B^\top B) - 2\text{trace}(XAB^\top) + \text{trace}(X^\top XAA^\top).\end{aligned}$$

Let us now use two useful identities from vector calculus, which we prove shortly.

$$\frac{\partial}{\partial P}\text{trace}(PQ) = Q^\top.$$
$$\frac{\partial}{\partial P}\text{trace}(P^\top PQ) = PQ^\top.$$

Let us now differentiate the objective and apply these identities to obtain

$$\frac{\partial}{\partial X}\|B - XA\|_F^2 = -2BA^\top + 2XAA^\top.$$

Setting the derivative to zero and moving terms around yields

$$X = BA^\top (AA^\top)^{-1}.$$

Proofs of identities

Let P be a $p \times q$ matrix and Q be a $q \times p$ matrix. From the definition of the trace, we have

$$\begin{aligned}\frac{\partial \text{trace}(PQ)}{\partial P_{k\ell}} &= \frac{\partial \sum_{i=1}^p \sum_{j=1}^q P_{ij} Q_{ji}}{\partial P_{k\ell}} \\ &= Q_{\ell k},\end{aligned}$$

where the last part follows since the term $P_{k\ell}$ appears in the sum once when $i = k$ and $j = \ell$, with a multiplicative scaling $Q_{\ell k}$. Collecting all of these partial derivatives into a matrix proves the first identity.

To prove the second identity, note that

$$\begin{aligned}\frac{\partial \text{trace}(P^\top PQ)}{\partial P_{ij}} &= \frac{\partial}{\partial P_{ij}} \sum_{k=1}^p \sum_{\ell=1}^q (P^\top P)_{k\ell} Q_{\ell k} \\ &= \frac{\partial}{\partial P_{ij}} \sum_{k=1}^p \sum_{\ell=1}^q (\sum_m P_{\ell m} P_{mk}) Q_{\ell k}.\end{aligned}$$

Within the large sum, let us collect all the terms that have P_{11} in them by setting the values k, ℓ, and m to 1 in turn. Writing these terms out, we have

$$P_{11}^2Q_{11} + (P_{11}P_{12}Q_{21} + P_{12}P_{11}Q_{21}) + (P_{11}P_{13}Q_{31} + P_{13}P_{11}Q_{31}) + \cdots + (P_{11}P_{1q}Q_{q1} + P_{1q}P_{q1}Q_{q1}).$$

Now computing the partial derivative, we have

$$\frac{\partial \text{trace}(P^\top PQ)}{\partial P_{11}} = 2\sum_k P_{1k}Q_{k1} = 2(PQ^\top)_{11}.$$

Repeating such an argument for each index P_{ij}, we obtain

$$\frac{\partial \text{trace}(P^\top PQ)}{\partial P_{ij}} = 2\sum_k P_{ik}Q_{kj} = 2(PQ^\top)_{ij}.$$

Collection the terms into a matrix proves the identity.

(e) Comment on why it makes sense that the computation of the best linear prediction of a vector just turns into a set of best linear predictions of each scalar component of the vector being predicted.

Solution: Let us answer the more general question, of why the above problem in part (d) decomposes into n independent problems. We have

$$\|B - XA\|_F^2 = \sum_{i=1}^n \|b_i^\top - X_i^\top A\|_2^2,$$

where b_i is the ith row of b, and X_i is the ith row of matrix X. Since the matrices A and B are fixed, the minimization of the above with respect to X decomposes as

$$\min_X \|B - XA\|_F^2 = \sum_{i=1}^n \min_{X_i} \|b_i^\top - X_i^\top A\|_2^2.$$

In other words, there are no terms linking X_i and X_j for $i \neq j$, and so each minimization can be performed independently.

When b_i is a scalar (and when B is a vector) we see that we are predicting each scalar component independently.

6 System Identification by ordinary least squares regression

Making autonomous vehicles involves using machine learning for many purposes. One of which is learning how the car actually behaves based on data about it.

Make sure to submit the code you write in this problem to "HW1 Code" on Gradescope.

(a) Consider a noisy time sequence of scalars $x[t]$. Let $x[t+1] = Ax[t] + Bu[t] + w$ where $A, B \in \mathbb{R}$ and w is the noise. Fit a line $x[t+1] = Ax[t] + Bu[t]$ to the values of x and u as provided by `system_identification_programming_a.mat`.

Solution:

$x[t+1] = 0.977552135184x[t] - 0.0877532187735u[t]$.

```
        import numpy as np
        np.set_printoptions(threshold='nan')
        import scipy.io

        mdict = scipy.io.loadmat("system_identification_programming_a.mat")

        x = mdict['x'][0]
        u = mdict['u'][0]

        b_solve = x[1:]
        A_solve = np.vstack((x[:-1], u[:-1])).T
        x_solve = np.linalg.inv(A_solve.T.dot(A_solve)).dot(A_solve.T).dot(b_solve)

        A, B = x_solve
        print("A: {}, B: {}".format(A, B))
```

(b) Consider a noisy time sequence of vectors $x[t]$. Let $x[t+1] = Ax[t] + Bu[t] + w$ where $A, B \in \mathbb{R}^{3\times 3}$ and w is the noise. Fit a line $x[t+1] = Ax[t] + Bu[t]$ to the values of x and u as provided by the `system_identification_programming_b.mat`.

Solution:

$$A = \begin{bmatrix} 0.15207406 & 0.93480864 & -0.00110243 \\ 0.03893567 & 0.30958727 & 0.87436511 \\ -0.52552959 & 0.0540906 & -0.47026217 \end{bmatrix}$$

;

$$B = \begin{bmatrix} 0.04894161 & 0.20568264 & -0.37090438 \\ -0.04524735 & -0.92861546 & 0.12756569 \\ 0.91096923 & -0.47124981 & -0.84222314 \end{bmatrix}.$$

These matrices get us a normalized squared error of 0.004.

```
import numpy as np
np.set_printoptions(threshold='nan')
import scipy.io

mdict = scipy.io.loadmat("system_identification_programming_b.mat")

X_raw = mdict['x']
U_raw = mdict['u']

X_raw = X_raw.reshape(X_raw.shape[:-1])
U_raw = U_raw.reshape(U_raw.shape[:-1])

X_curr = X_raw[:-1]
U_curr = U_raw[:-1]
Y = X_raw[1:]

X = np.hstack((X_curr, U_curr))
soln = np.linalg.inv(X.T.dot(X)).dot(X.T).dot(Y).T
A, B = soln[:, :3], soln[:, 3:]
```

```
print("A: \n{}, \nB: \n{}".format(A, B))

Y_hat = A.dot(X_curr.T) + B.dot(U_curr.T)
print("Error", np.linalg.norm(Y_hat - Y.T) / Y.shape[0])
```

(c) Consider a dynamical system consisting of a cars driving on a straight road. The dynamics (accelerations) for a car i are governed by a linear function of its own position and velocity, as well as the position and velocity of the car $i-1$ preceding it. This is called a linear *car following model*. That is, we may write the dynamical system as follows:

$$\ddot{x}_i = ax_i + b\dot{x}_i + cx_{i-1} + d\dot{x}_{i-1} + w(t)$$

where $w(t) \sim \mathcal{N}(0, \sigma)$. Re-write the dynamical system in matrix form, i.e. in the form $\dot{x} = Ax + Bu + w(t)$.

Solution: Correction: The use of $w(t)$ in the stated dynamical system is overloaded with its use in the final form. In the former case, it is a scalar, whereas in the latter form it is a vector.

Let's consider the continuous dynamical system for car i. The state of car i consists of its position x_i and velocity $\dot{x}_i$, so this information will comprise the state x of the dynamical system. Other exogenous variables which affect the state evolution of car i include the position and velocity of car $i-1$, denoted $x_{i-1}, \dot{x}_{i-1}$ respectively, so these will comprise the input u to the dynamical system. The dynamical system above can then be written as below:

$$x = \begin{bmatrix} x_i \\ \dot{x}_i \end{bmatrix}, u = \begin{bmatrix} x_{i-1} \\ \dot{x}_{i-1} \end{bmatrix}$$

$$A = \begin{bmatrix} 0 & 1 \\ a & b \end{bmatrix}, B = \begin{bmatrix} 0 & 0 \\ c & d \end{bmatrix}$$

This form $\dot{x} = Ax + Bu + w(t)$ is also called *state space form*.

(d) Given data measurements from a dynamical system, we wish to estimate the parameters of the system. This is called *system identification*. In the car following model example, given vehicle traces consisting of positions, velocities, and accelerations of vehicles and the vehicles preceding them (that is, given samples of $\ddot{x}_i, x_i, \dot{x}_i, x_{i-1}, \dot{x}_{i-1}$, denoted by *row vectors* $\hat{\ddot{x}}_i, \hat{x}_i, \hat{\dot{x}}_i, \hat{x}_{i-1}, \hat{\dot{x}}_{i-1}$, respectively), give the analytical solution to parameters (a, b, c, d) which gives the best linear fit to the data collected from the dynamical system.

Solution: Re-write the dynamical system as follows:

$$\dot{x} = \begin{bmatrix} A & B \end{bmatrix} \begin{bmatrix} x \\ u \end{bmatrix} + w(t)$$

Equivalently:

$$\dot{x}^T = \begin{bmatrix} x \\ u \end{bmatrix}^T \begin{bmatrix} A & B \end{bmatrix}^T + w(t)^T$$

Due to the noise term $w(t)$, we cannot solve for A, B exactly, but instead we want to solve for them approximately. We can however safely ignore the noise term $w(t)$ in the following derivation because its mean is known (and is zero); we will soon see in lecture formally why. Thus, we formulate The least squares problem in the familiar form:

$$min_{A,B}\frac{1}{2}\| \begin{bmatrix} x \\ u \end{bmatrix}^T \begin{bmatrix} A & B \end{bmatrix}^T - \dot{x}^T + w(t)^T \|_2^2$$

Then, solve the least squares problem. We take the derivative of the above and set it equal to zero[1].

$$0 = \begin{bmatrix} x \\ u \end{bmatrix} (\begin{bmatrix} x \\ u \end{bmatrix}^T \begin{bmatrix} A & B \end{bmatrix}^T - \dot{x}^T)$$

Then, we solve for $[AB]$:

$$\begin{bmatrix} A & B \end{bmatrix} = \dot{x} \begin{bmatrix} x^T & u^T \end{bmatrix} \left(\begin{bmatrix} x \\ u \end{bmatrix} \begin{bmatrix} x^T & u^T \end{bmatrix} \right)^{-1}$$

So far, we have written the least squares solution as if we are only regressing on a single data point x. Let's define the following notation for the data vectors.

$$\hat{x} = \begin{bmatrix} \hat{x}_i \\ \hat{\dot{x}}_i \end{bmatrix}$$

Then, the corresponding least squares solution regressing on data vectors $\hat{x}$ is as follows:

$$\begin{bmatrix} A & B \end{bmatrix} = \hat{\dot{x}} \begin{bmatrix} \hat{x}^T & \hat{u}^T \end{bmatrix} \left(\begin{bmatrix} \hat{x} \\ \hat{u} \end{bmatrix} \begin{bmatrix} \hat{x}^T & \hat{u}^T \end{bmatrix} \right)^{-1}$$

Finally, unlike part (b), we observe that we have *structured matrices*; we only need to solve for the second row of the A, B matrices, since that's where all the unknown parameters are. Recall that

$$\dot{x} = \begin{bmatrix} \dot{x}_i \\ \ddot{x}_i \end{bmatrix}.$$

Regressing on the first row $\dot{x}_i$ is easy since we in fact have an exact feature which predicts it (you guessed it, it's $\dot{x}_i$!); thus, the least squares solution will give a weight of 1 for that feature and 0 for other features, and this is what we see in the first row of the A,B matrices:

$$\begin{aligned} \begin{bmatrix} 0 & 1 & 0 & 0 \end{bmatrix} &= \hat{\dot{x}}_i \begin{bmatrix} \hat{x}^T & \hat{u}^T \end{bmatrix} (\begin{bmatrix} \hat{x} \\ \hat{u} \end{bmatrix} \begin{bmatrix} \hat{x}^T & \hat{u}^T \end{bmatrix})^{-1} \\ &= \hat{\dot{x}}_i \begin{bmatrix} \hat{x}_i^T & \hat{\dot{x}}_i^T & \hat{x}_{i-1}^T \hat{\dot{x}}_{i-1}^T \end{bmatrix} (\begin{bmatrix} \hat{x}_i \\ \hat{\dot{x}}_i \\ \hat{x}_{i-1} \\ \hat{\dot{x}}_{i-1} \end{bmatrix} \begin{bmatrix} \hat{x}_i^T & \hat{\dot{x}}_i^T & \hat{x}_{i-1}^T & \hat{\dot{x}}_{i-1}^T \end{bmatrix})^{-1} \end{aligned}$$

[1]For useful matrix identities, see `http://www.math.uwaterloo.ca/~hwolkowi/matrixcookbook.pdf`

Thus, the final solution, which only solves for the second row is:

$$\begin{aligned}\begin{bmatrix}a & b & c & d\end{bmatrix} &= \hat{\dot{x}}_i \begin{bmatrix}\hat{x}^T & \hat{u}^T\end{bmatrix} \left(\begin{bmatrix}\hat{x} \\ \hat{u}\end{bmatrix} \begin{bmatrix}\hat{x}^T & \hat{u}^T\end{bmatrix} \right)^{-1} \\ &= \hat{\ddot{x}}_i \begin{bmatrix}\hat{x}_i^T & \hat{\dot{x}}_i^T & \hat{x}_{i-1}^T \hat{\dot{x}}_{i-1}^T\end{bmatrix} \left(\begin{bmatrix}\hat{x}_i \\ \hat{\dot{x}}_i \\ \hat{x}_{i-1} \\ \hat{\dot{x}}_{i-1}\end{bmatrix} \begin{bmatrix}\hat{x}_i^T & \hat{\dot{x}}_i^T & \hat{x}_{i-1}^T & \hat{\dot{x}}_{i-1}^T\end{bmatrix} \right)^{-1}\end{aligned}$$

(e) Implement your estimator using data file `system_identification_programming_train.mat`, which contains a dictionary with several useful values: `ddot_x_i, x_i, dot_x_i, x_i-1, dot_x_i-1`. There are 40,000 data points. This data is a processed, filtered, and sampled form of data collected from the I-80 highway here in California; it is available from the Next Generation Simulation (NGSIM) dataset, commonly used in traffic simulation. The data is given in units of $m/s^2, m, m/s, m, m/s$, respectively. More precisely, the velocities given are deviations from the speed limit of $29.0576 m/s$ (65mph). Give the resulting parameters below. Describe qualitatively how the dynamics evolve with respect to the state x and input u variables. When are the accelerations positive?

Solution: Correction: The data file given contains `xdd, x, xd, xp, xdp` respectively.

The resulting dynamical system is: `xdd_i[t] = -0.007 x_i[t] + -0.305 xd_i[t] + 0.006 x_i-1[t] + 0.273 xd_i-1[t] + N(0.000, sigma)`

Notice that this system is pretty close to one that looks like the form:

$$\ddot{x}_i = e(x_{i-1} - x_i) + f(\dot{x}_{i-1} - \dot{x}_i) - g\dot{x}_i + w(t)$$

where the parameters e, f, g are non-negative.

That is, $x_{i-1} - x_i$, $\dot{x}_{i-1} - \dot{x}_i$, and $\dot{x}_i$ are predictive features of the dynamics of this system. The quantity $x_{i-1} - x_i$ is the headway of car i, or the spacing between car i to car $i-1$ preceding it. Similarly, $\dot{x}_{i-1} - \dot{x}_i$ is the relative velocity between the two cars. Finally, $\dot{x}_i$ is the deviation of car i's velocity from the speed limit. Thus, we can say that car i accelerates with larger headways, with speeds lower than that of the preceding vehicle, and with a speed lower than the speed limit. The close coupling of the dynamics of car i to the state of car $i-1$ (viewed as the input to the dynamical system for car i) is why these systems are called *car following models*.

Sample implementation:

```
# Load mat file
mdict = {}
scipy.io.loadmat("system identification programming train.mat",
    mdict=mdict)

# Assemble xu matrix
```

```
x = np.vstack([mdict["x"], mdict["xd"]])
u = np.vstack([mdict["xp"], mdict["xdp"]])
xdot = mdict["xdd"]
xu = np.vstack([x, u])

# Solve
soln = xdot.dot(xu.T).dot(np.linalg.pinv(xu.dot(xu.T)))
a, b, c, d = soln.squeeze()
print("Fitted dynamical system:")
print("xdd_i[t] = {:.3f} x_i[t] + {:.3f} xd_i[t] +
    {:.3f} x_i-1[t] + {:.3f} xd_i-1[t] + N({:.3f},
     sigma)".format(a,b,c,d,0))
```

(f) Use your estimated parameters and data file `system_identification_programming_eval.mat` to generate a vehicle trace and submit the evaluation results to Gradescope. You will be given `x_i(0), dot_x_i(0), x_i-1(t), dot_x_i-1(t)` and be asked to submit `x_i(t)` for 150 time steps. The provided data is given at 15Hz and remember that the given velocities are deviations from 29.0576m/s.

Instructions: Generate a file named `submission.txt`. Double check that this file is in plaintext. (i.e., `cat submission.txt` does not give you strange characters at the start of your file). Put only one float `x_i(t)` on each line, where the first line is `x_i(0)`. Submit the file to "HW1 Test Set" on Gradescope. You are allowed two submissions per every 24 hours, so start early! If the script catches a formatting error, the submission will **not** count towards your two daily submissions.

Solution: This section requires discretizing from the given continuous time system to a discrete time system. For common discretization schemes, see the Euler method[2] or the Backwards Euler method[3]. These solutions use the Euler method.

Conceptually, next position of car i can be computed from the current position of car i, the current velocity of car i, and the discretization step. The next velocity of car i can be computed from the current velocity of car i, the current acceleration of car i, and the discretization step. The next acceleration of car i can be computed using the current position and velocity of car i, along with the input from car $i-1$ and the model parameters, as given by the original equation representing the dynamical system.

The resulting ℓ_2-norm evaluation error is around 16.4, depending on your discretization scheme.

Sample implementation:

```
# Useful constants
mph65 = 29.0576  # conversion from m/s
dt = 1.0/15.0  # discretization step
L = 150  # length of desired trajectory
```

[2]See `https://en.wikipedia.org/wiki/Euler_method`
[3]See `https://en.wikipedia.org/wiki/Backward_Euler_method`

```
# Load mat file
mdict = {}
scipy.io.loadmat("system identification programming eval.mat",
    mdict=mdict)

# inputs
xp = mdict["xp"][0]; xdp = mdict["xdp"][0]
x = [mdict["x0"][0,0]]; xd = [mdict["xd0"][0,0]]

xdd = [a*x[0] + b*xd[0] + c*xp[0] + d*xdp[0] + mu]

# Use a, b, c, d from part (e)
# Forwards euler discretization scheme
for i in range(1, L):
    x.append(x[i-1] + dt*(xd[i-1] + mph65))
    xd.append(xd[i-1] + dt*xdd[i-1])
    xdd.append(a*x[i-1] + b*xd[i-1] + c*xp[i-1] + d*xdp[i-1]
        + mu)

print("Estimated x_i", np.array(x))
```

7 Your Own Question

Write your own question, and provide a thorough solution.

Writing your own problems is a very important way to really learn material. The famous "Bloom's Taxonomy" that lists the levels of learning is: Remember, Understand, Apply, Analyze, Evaluate, and Create. Using what you know to create is the top-level. We rarely ask you any HW questions about the lowest level of straight-up remembering, expecting you to be able to do that yourself. (e.g. make yourself flashcards) But we don't want the same to be true about the highest level.

As a practical matter, having some practice at trying to create problems helps you study for exams much better than simply counting on solving existing practice problems. This is because thinking about how to create an interesting problem forces you to really look at the material from the perspective of those who are going to create the exams.

Besides, this is fun. If you want to make a boring problem, go ahead. That is your prerogative. But it is more fun to really engage with the material, discover something interesting, and then come up with a problem that walks others down a journey that lets them share your discovery. You don't have to achieve this every week. But unless you try every week, it probably won't happen ever.