Problem 1: Getting Started

Read through this page carefully. You may typeset your homework in latex or submit neatly handwritten/scanned solutions. Please start each question on a new page. Deliverables:

1. Submit a PDF of your writeup, **with an appendix for your code**, to assignment on Gradescope, "HW1 Write-Up". If there are graphs, include those graphs in the correct sections. Do not simply reference your appendix.
2. If there is code, submit all code needed to reproduce your results, "HW1 Code".
3. If there is a test set, submit your test set evaluation results, "HW1 Test Set".

After you've submitted your homework, watch out for the self-grade form.

a. Who else did you you work with on this homework? In case of course events, just describe the group. How did you work on this homework? Any comments about the homework?

> **Solution** I used Yagna Patel's latex template after removing the annoying cover page. I worked with Weiran liu again.

b. Please copy the following statement and sign next to it. We just want to make it *extra* clear so that no one inadvertently cheats.

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.

> **Solution** 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.

Problem 2: "Sample Complexity" of coupon collecting

One of the books of Roald Dahl has the following problem. Willy Wonka has a chocolate factory where the produced chocolates have 40 different types of cards hidden under the chocolate wrappers. To encourage people to consume these chocolates, a game is announced: Whoever finds all the 40 types of cards will be allowed to enter Willy Wonka's factory and participate in a questionable experiment.

Charlie is a consumer of Willy's chocolates and he visits a particular local store every day to buy his chocolates. The store contains equal number of chocolates of each card type at any given time. Every time a chocolate with a particular card hidden beneath the wrapper is bought, another chocolate containing an identical card is put in its place immediately. Whenever Charlie buys a chocolate, he does that by picking up a chocolate uniformly at random from the store.

a. **What is the expected number of days it takes Charlie to qualify for Willy Wonka's game if he buys one random chocolate every day from the local store?** Please explain your computations precisely.

> **Solution** Let's assume Charlie already has x different cards ($1 \leq x \leq 39$), to get a new card is a Bernoulli process which the probability of success is $\frac{40-x}{40}$. Therefore, the expected days to get a new card is $\frac{40}{40-x}$. Sum up all these fractions gives us the expected number of days it takes Charlie to qualify for Wily Wonka's game.
>
> $$1 + \frac{40}{39} + ... + + \frac{40}{1} \approx 172$$
>
> (Note: this is called Harmonic series and there is no general formula for that)

b. *For all the following parts, the game has been changed.* Suppose there are d types of different cards. Instead of having to collect all d card types, at the end of the game Willy Wonka will draw a card uniformly at random and if someone has that card in their collection, they win a prize. Charlie still visits the local store everyday and buys a chocolate at random from it. If Charlie wants his probability of winning to be at least $1 - \delta$, **how many *distinct* card types should he have in his collection before Willy Wonka draws the card?**

> **Solution** This is very simple. If Charlie has x cards then his chance of winning is just x/d and this needs to be greater than $1 - \delta$. So we have $x \geq \lceil d(1-\delta) \rceil$

c. Suppose that Charlie visited the particular local store for n days and bought 1 chocolate at random each day before Willy's draw of the random card. **What is the probability that Charlie wins a prize from Willy's draw?**

> **Solution** Let's assume the number of card types Charlie has after $n-1$ days is $f(n-1)$. Then on n days, he has a probability of $\frac{d-f(n-1)}{d}$ to get one more card type and will have $f(n) = f(n-1) + \frac{d-f(n-1)}{d}$ after all. We end up with a iteration equation $f(n) - d = \frac{d-1}{d}(f(n-1) - d)$. Solve this gives us $f(n) = d - \frac{(d-1)^n}{d^{n-1}}$. Therefore the probability that Charlie wins a prize from Willy's draw after n days is
>
> $$f(n)/d = 1 - \frac{(d-1)^n}{d^n}$$

d. Now assume $n = \alpha d$ (i.e. Charlie always buys exactly α times the number of total types of cards before Wonka's draw of the random card). **What does Charlie's probability of winning converge to as d gets large?**

> **Solution** Substitute $n = \alpha d$ into the solution above gives us: $1 - (\frac{d-1}{d})^{\alpha d}$. We can see the limit $\lim_{d\to\infty}(1 - \frac{1}{d})^d = \frac{1}{e}$. Therefore, the probability is
>
> $$\lim_{d\to\infty} 1 - (\frac{d-1}{d})^{\alpha d} = 1 - \frac{1}{e^\alpha}$$

e. Now, consider the following function estimation problem, which at first sight might seem unrelated. We want to learn a completely unstructured function f on a finite domain $\mathcal{D}$ of size d. We collect a training data of size n from random samples, i.e., we have the dataset $\{(U_i, f(U_i)), i = 1, \ldots, n\}$ where each U_i is drawn uniformly at random from $\mathcal{D}$. **How big of a training set do we need to collect to ensure that with probability at least $1-\delta$, we will successfully estimate the function at a point which is drawn uniformly at random from the domain $\mathcal{D}$? Repeat the computations for the case when d gets large.**

Solution Consider the following analogue: domain $\mathcal{D} \sim$ card types, training data size $n \sim n$ days, if a data point has been collected into the training set we assume that point will be correctly estimated. The winning prize is the final number we want to estimate. Therefore we have

$$1 - (\frac{d-1}{d})^n \leq 1 - \delta, \Rightarrow n \geq \frac{\ln \delta}{\ln(\frac{d-1}{d})}$$

When d gets large, we will get $\ln \frac{d-1}{d} \approx -\frac{1}{d}$. So

$$n \geq \frac{\ln \delta}{\ln(\frac{d-1}{d})} \approx -d \ln \delta$$

And when $d \to \infty$, $n \to \infty$

Problem 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 to improve the accuracy of learning things.

For each $\theta \in (1/3, 2/3)$, 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 1.

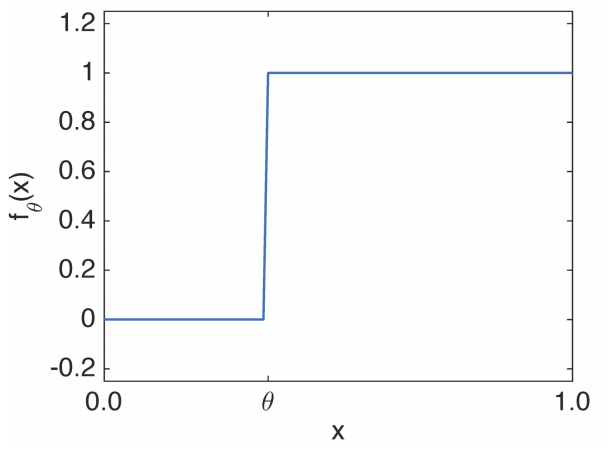


Figure 1: Plot of function $f_\theta(x)$ against x.

We draw samples $X_1, X_2, \ldots, X_n$ 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 $(X_1, f_\theta(X_1)), (X_2, f_\theta(X_2)), \ldots, (X_n, f_\theta(X_n))$.

Let $T_{min} = \max(\{\frac{1}{3}\} \cup \{X_i | f_\theta(X_i) = 0\})$. We know that the true θ must be larger than T_{min}.

Let $T_{max} = \min(\{\frac{2}{3}\} \cup \{X_i | f_\theta(X_i) = 1\})$. 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 $T_{max} - \theta > \epsilon$ as a function of ϵ? And what is the probability that $\theta - T_{min} > \epsilon$ as a function of ϵ?**

Solution First, we get the ranges of $T_{min} \in [\frac{1}{3}, \theta]$ and $T_{max} \in (\theta, \frac{2}{3}]$. I will derive T_{max} as an example. For all each point, we have a probability of $1 - \epsilon$ that point don't lie between θ and $\epsilon + \theta$. Just multiply them together for n points:

$$P_{T_{max}}(\epsilon) = \begin{cases} 0 \text{ if } \epsilon \geq \frac{2}{3} - \theta \text{ (TRAP!!!)} \\ (1-\epsilon)^n \text{ otherwise} \end{cases}$$

$$P_{T_{min}}(\epsilon) = \begin{cases} 0 \text{ if } \epsilon \geq \theta - \frac{1}{3} \text{ (TRAP!!!)} \\ (1-\epsilon)^n \text{ otherwise} \end{cases}$$

b. Suppose that you would like to have an estimate for θ that is ϵ-close (defined as $|\hat{\theta} - \theta_0| < \epsilon$, where $\hat{\theta}$ is the estimation and θ_0 is the true value) with probability at least $1 - \delta$. Both ϵ and δ are some small positive numbers. **Please bound or estimate how big of an n do you need?**

Solution Now we have a window of 2ϵ, the left and right must all have points. The complement set is either left or right have not points, which is the same as $\{T_{max} > \epsilon + \theta\} \cup \{T_{min} < \epsilon - \theta\}$:

$$\{T_{max} > \epsilon + \theta\} \cup \{T_{min} < \epsilon - \theta\} = \{T_{max} > \epsilon + \theta\} + \{T_{min} < \epsilon - \theta\} - \{T_{max} > \epsilon + \theta\} \cap \{T_{min} < \epsilon - \theta\}$$

$$2(1-\epsilon)^n - (1-2\epsilon)^n \leq \delta \Rightarrow \text{A (u)GSI told me to get this} \Rightarrow n \geq \frac{\ln(\delta/2)}{\ln(1-\epsilon)}$$

c. 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 as above?** (**Hint:** You need not use a randomized strategy.)

> **Solution** Because we know that $\frac{1}{3} < \theta < \frac{2}{3}$, we can just uniformly sample from this range with the same interval between points. This will give as bin sizes of $\frac{1}{3n}$. So we have $n \geq \frac{1}{6\epsilon}$

d. 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 as above?**

> **Solution** We have to be smarter this time so we need to conduct so-called binary search. We picked the middle point of $[\frac{1}{3}, \frac{2}{3}]$ and do the following iteration (x_i is the current sample point, $[a_i, b_i]$ is the possible range of θ for round i):
>
> $$x_{x+1} = \begin{cases} \frac{x_i+b_i}{2} \text{ if } f(x_i) = 0 \\ \frac{x_i+a_i}{2} \text{ otherwise} \end{cases}$$
>
> This will decrease the possible range of θ to be $\frac{1}{3\times 2^n}$. Therefore, we can get $n \geq \log_2(\frac{1}{6\epsilon})$

e. In the three sampling approaches above: random, deterministic, and adaptive, compare the scaling of n with ϵ (and d as well for the random case).

> **Solution**
> For random, it's $O(ln\delta)$ and $O(\frac{1}{\epsilon})$
> For deterministic, it's $O(\frac{1}{\epsilon})$
> For adaptive, it's $O(ln(\frac{1}{\epsilon}))$

f. **Why do you think we asked this series of questions? What are the implications of those results in a machine learning application?**

> **Solution** How am I supposed to know this? I don't know but I guess it must have something to do with training set size. You probably want us to compare different sampling strategies and see how sample size scales in each.

Problem 4: Eigenvalue and Eigenvector Review

A square matrix $\mathbf{A} \in \mathbb{R}^{d \times d}$ has a (right) eigenvalue $\lambda \in \mathbb{C}$ and (right) eigenvector $\mathbf{x} \in \mathbb{C}^d \backslash 0$ if $\mathbf{A}\mathbf{x} = \lambda \mathbf{x}$. Left eigenvalues and eigenvectors are defined analogously — $\mathbf{x}^T\mathbf{A} = \lambda \mathbf{x}^T$. Since the definition is scale invariant (if $\mathbf{x}$ is an eigenvector, then $t\mathbf{x}$ 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. You may use a computer for questions v and vi.

i. $\mathbf{A} = \begin{bmatrix} 2 & -4 \\ -1 & -1 \end{bmatrix}$

Solution

Right eigenvalues and eigenvectors:

$$\lambda_1 = 3, \lambda_2 = -2, x_1 = \begin{bmatrix} -\frac{4}{\sqrt{17}} \\ \frac{1}{\sqrt{17}} \end{bmatrix}, x_2 = \begin{bmatrix} \frac{1}{\sqrt{2}} \\ \frac{1}{\sqrt{2}} \end{bmatrix}$$

Left evigenvalues and eigenvectors:

$$\lambda_1 = 3, \lambda_2 = -2, x_1 = \begin{bmatrix} -\frac{1}{\sqrt{2}} \\ \frac{1}{\sqrt{2}} \end{bmatrix}, x_2 = \begin{bmatrix} \frac{1}{\sqrt{17}} \\ \frac{4}{\sqrt{17}} \end{bmatrix}$$

ii. $\mathbf{B} = \begin{bmatrix} 3 & 1 \\ 1 & 3 \end{bmatrix}$

Solution

Right eigenvalues and eigenvectors:

$$\lambda_1 = 4, \lambda_2 = 2, x_1 = \begin{bmatrix} \frac{1}{\sqrt{2}} \\ \frac{1}{\sqrt{2}} \end{bmatrix}, x_2 = \begin{bmatrix} -\frac{1}{\sqrt{2}} \\ \frac{1}{\sqrt{2}} \end{bmatrix}$$

Left eigenvalues and eigenvectors:

$$\lambda_1 = 4, \lambda_2 = 2, x_1 = \begin{bmatrix} \frac{1}{\sqrt{2}} \\ \frac{1}{\sqrt{2}} \end{bmatrix}, x_2 = \begin{bmatrix} -\frac{1}{\sqrt{2}} \\ \frac{1}{\sqrt{2}} \end{bmatrix}$$

iii. $\mathbf{A}^2$

Solution

Right eigenvalues and eigenvectors:

$$\lambda_1 = 9, \lambda_2 = 4, x_1 = \begin{bmatrix} -\frac{4}{\sqrt{17}} \\ \frac{1}{\sqrt{17}} \end{bmatrix}, x_2 = \begin{bmatrix} \frac{1}{\sqrt{2}} \\ \frac{1}{\sqrt{2}} \end{bmatrix}$$

Left eigenvalues and eigenvectors:

$$\lambda_1 = 9, \lambda_2 = 4, x_1 = \begin{bmatrix} -\frac{1}{\sqrt{2}} \\ \frac{1}{\sqrt{2}} \end{bmatrix}, x_2 = \begin{bmatrix} \frac{1}{\sqrt{17}} \\ \frac{4}{\sqrt{17}} \end{bmatrix}$$

iv. $\mathbf{B}^2$

Solution

$$\lambda_1 = 16, \lambda_2 = 4, x_1 = \begin{bmatrix} \frac{1}{\sqrt{2}} \\ \frac{1}{\sqrt{2}} \end{bmatrix}, x_2 = \begin{bmatrix} -\frac{1}{\sqrt{2}} \\ \frac{1}{\sqrt{2}} \end{bmatrix}$$

Left eigenvalues and eigenvectors:

$$\lambda_1 = 16, \lambda_2 = 4, x_1 = \begin{bmatrix} \frac{1}{\sqrt{2}} \\ \frac{1}{\sqrt{2}} \end{bmatrix}, x_2 = \begin{bmatrix} -\frac{1}{\sqrt{2}} \\ \frac{1}{\sqrt{2}} \end{bmatrix}$$

v. **AB**

Solution

$$\lambda_1 = -8, \lambda_2 = 6, x_1 = \begin{bmatrix} \frac{1}{\sqrt{2}} \\ \frac{1}{\sqrt{2}} \end{bmatrix}, x_2 = \begin{bmatrix} -\frac{5}{\sqrt{29}} \\ \frac{2}{\sqrt{29}} \end{bmatrix}$$

Left eigenvalues and eigenvectors:

$$\lambda_1 = -8, \lambda_2 = 6, x_1 = \begin{bmatrix} \frac{2}{\sqrt{29}} \\ \frac{5}{\sqrt{29}} \end{bmatrix}, x_2 = \begin{bmatrix} -\frac{1}{\sqrt{2}} \\ \frac{1}{\sqrt{2}} \end{bmatrix}$$

vi. **BA**

Solution

$$\lambda_1 = -8, \lambda_2 = 6, x_1 = \begin{bmatrix} \frac{1}{\sqrt{2}} \\ \frac{1}{\sqrt{2}} \end{bmatrix}, x_2 = \begin{bmatrix} -\frac{13}{\sqrt{170}} \\ \frac{1}{\sqrt{170}} \end{bmatrix}$$

Left eigenvalues and eigenvectors:

$$\lambda_1 = -8, \lambda_2 = 6, x_1 = \begin{bmatrix} \frac{1}{\sqrt{170}} \\ \frac{13}{\sqrt{170}} \end{bmatrix}, x_2 = \begin{bmatrix} -\frac{1}{\sqrt{2}} \\ \frac{1}{\sqrt{2}} \end{bmatrix}$$

b. **Compute the singular value decompositions** of the matrices above. In addition, please compute the SVD of:

$$\mathbf{C} = \begin{bmatrix} 3 & 1 \\ 1 & 3 \\ 2 & -4 \\ -1 & -1 \end{bmatrix}$$

For **B** $\mathbf{B}^2$ compute by hand. For the rest of the matrices, you may use a computer.

Solution Some answers are too complicated so the approximation has been taken

$$A = \begin{bmatrix} 2 & -4 \\ -1 & -1 \end{bmatrix} = \begin{bmatrix} -1.00 & 0.11 \\ -0.11 & -1.00 \end{bmatrix} \begin{bmatrix} 4.50 & 0 \\ 0 & 1.33 \end{bmatrix} \begin{bmatrix} -0.42 & 0.91 \\ 0.91 & 0.42 \end{bmatrix}^T$$

$$B = \begin{bmatrix} 3 & 1 \\ 1 & 3 \end{bmatrix} = \begin{bmatrix} \frac{1}{\sqrt{2}} & -\frac{1}{\sqrt{2}} \\ \frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} \end{bmatrix} \begin{bmatrix} 4 & 0 \\ 0 & 2 \end{bmatrix} \begin{bmatrix} \frac{1}{\sqrt{2}} & -\frac{1}{\sqrt{2}} \\ \frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} \end{bmatrix}^T$$

$$A^2 = \begin{bmatrix} 8 & -4 \\ -1 & 5 \end{bmatrix} = \begin{bmatrix} -0.92 & 0.39 \\ 0.39 & 0.92 \end{bmatrix} \begin{bmatrix} 9.59 & 0 \\ 0 & 3.76 \end{bmatrix} \begin{bmatrix} -0.81 & 0.59 \\ 0.59 & 0.81 \end{bmatrix}^T$$

$$B^2 = \begin{bmatrix} \frac{1}{\sqrt{2}} & -\frac{1}{\sqrt{2}} \\ \frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} \end{bmatrix} \begin{bmatrix} 16 & 0 \\ 0 & 4 \end{bmatrix} \begin{bmatrix} \frac{1}{\sqrt{2}} & -\frac{1}{\sqrt{2}} \\ \frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} \end{bmatrix}^T$$

$$AB = \begin{bmatrix} 2 & -10 \\ -4 & -4 \end{bmatrix} = \begin{bmatrix} -0.93 & 0.36 \\ -0.36 & -0.93 \end{bmatrix} \begin{bmatrix} 10.78 & 0 \\ 0 & 4.45 \end{bmatrix} \begin{bmatrix} -0.04 & 1.00 \\ 1.00 & 0.04 \end{bmatrix}^T$$

$$BA = \begin{bmatrix} 5 & -13 \\ -1 & -7 \end{bmatrix} = \begin{bmatrix} -0.91 & 0.42 \\ -0.42 & -0.91 \end{bmatrix} \begin{bmatrix} 15.30 & 0 \\ 0 & 3.14 \end{bmatrix} \begin{bmatrix} -0.27 & 0.96 \\ 0.96 & 0.27 \end{bmatrix}^T$$

$$C = \begin{bmatrix} 0.14 & 0.80 & 0.24 & -0.54 \\ 0.56 & 0.32 & 0.24 & 0.73 \\ -0.80 & 0.43 & 0.00 & 0.42 \\ -0.18 & -0.28 & 0.94 & -0.05 \end{bmatrix} \begin{bmatrix} 5.20 & 0 \\ 0 & 3.86 \\ 0 & 0 \\ 0 & 0 \end{bmatrix} \begin{bmatrix} -0.08 & 1.00 \\ 1.00 & 0.08 \end{bmatrix}^T$$

c. **Show** from the definition of a right eigenvalue that the quantity λ is an eigenvalue with associated eigenvector $\mathbf{x}$ iff for all $1 \leqslant i \leqslant d$, we have

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

Solution For the i-th ro of Axw, we have

$$\sum_j A_{ij}x_j = \sum_{j=i} A_{ij}x_j + \sum_{j \neq i} A_{ij}x_j.$$

Because we have $Ax = \lambda x$, we calculate the i-th row of λx

$$A_{ii}x_i + \sum_{j \neq i} A_{ij}x_j = \lambda x_i$$

Therefore

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

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

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

Solution

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

Because $|x_i| \geqslant |x_j|$ for all $j \neq i$

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

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 $\sum_{j \neq i} |A_{ij}|$.

Problem 5: Fun with Least Squares

In ordinary least squares we learn to predict a *target* scalar $y \in \mathbb{R}$ given a *feature* vector $\mathbf{x} \in \mathbb{R}^d$. Each element of $\mathbf{x}$ is called a feature, which could correspond to a scientific *measurement*. For example, the i-th element of $\mathbf{x}$, denoted by $(\mathbf{x})_i$, could correspond to the velocity of a car at time i. y could represent the final location (say just in one direction) of the car.

For the purpose of predicting y from $\mathbf{x}$ we are given n samples $(\mathbf{x}_i, y_i)$ with $i = 1, \ldots, n$ (where feature vectors and target scalars are observed in pairs), which we also call the training set. In this problem we want to predict the unobserved target y corresponding to a new $\mathbf{x}$ (not in the training set) by some linear prediction $\hat{y} = \mathbf{x}^\top \hat{\mathbf{w}}$ where the *weight* $\hat{\mathbf{w}} \in \mathbb{R}^d$ minimizes the least-squares training cost

$$\sum_{i=1}^{n} (\mathbf{x}_i^\top \mathbf{w} - y_i)^2 = \|\mathbf{X}\mathbf{w} - \mathbf{y}\|_2^2$$

where in the matrix $\mathbf{X} \in \mathbb{R}^{n \times d}$, the transposed sample feature vectors $\mathbf{x}_i^\top$ constitute the d-dimensional row vectors, and the n-dimensional vectors of training measurements $\mathbf{x}^j = ((x_1)_j, \ldots, (x_n)_j)^\top$ for $j = 1, \ldots, d$ correspond to the column vectors (see Figure 2) and $\mathbf{y} = (y_1, \ldots, y_n)^\top$.

$$\mathbf{X} = \begin{matrix} \\ n \downarrow \end{matrix} \overset{\overset{d}{\longrightarrow}}{\begin{bmatrix} — & x_1^\top & — \\ — & x_2^\top & — \\ & \vdots & \\ — & x_n^\top & — \end{bmatrix}} = \begin{bmatrix} | & | & & | \\ x^1 & x^2 & \cdots & x^d \\ | & | & & | \end{bmatrix}$$

Figure 2: Dimensions, column and row vector notation for matrix $\mathbf{X}$

Let us actually build on the example mentioned above and view the measurements $(x_i)_j$ of each sample $\mathbf{x}_i$ as a sequence of measurements, e.g. velocities of car i, over time $j = 1, \ldots, d$.

a. Is this problem in a supervised or unsupervised learning setting? **Please explain.**

> **Solution** It's supervised learning setting because we know the desired output value y.

b. Suppose that we want to learn (from our training set) to predict the location y from only the first t measurements. Denoting the prediction of y at time t by $\hat{y}^t$, we thus want to use $(x)_j, j = 1, \ldots, t$ to predict y. If we now learn how to obtain $\hat{y}^t$ for each $t = 1, \ldots, d$, we end up with a sequence of estimators $\hat{y}^1, \ldots, \hat{y}^d$ for each car.

Provide a method to obtain $\hat{y}^t$ for each t. Note that we will obtain a different model for each t.

> **Solution** If we obtain a different model for each t, we just need to do linear regression for the known j features of $(X)_j = [(x)_1 \quad (x)_2 \quad \ldots \quad (x)_j]^T$ to get the prediction $\hat{y}^t = (X)_t \hat{w}^t$, where $(X)_t$ denotes the first t measurements. $\hat{w}^t$ is a column vector with a length of t which minimize $\|(X)_t \hat{w}^t - y^t\|_2^2$

c. Someone suggests that maybe the measurements themselves are partially predictable from the previous measurements, which suggests employing a two stage strategy to solve the original prediction problem: First we predict the t-th measurement $(x)_t$ based on the previous measurements $(x)_1, \ldots, (x)_{t-1}$. Then we look at the differences (sometimes deemed the "innovation") between the actual t-th measurement we obtained and our prediction for it, i.e. $(\Delta\mathbf{x})_t := (\mathbf{x})_t - (\hat{\mathbf{x}})_t$. Finally, we use $(\Delta\mathbf{x})_1, \ldots, (\Delta\mathbf{x})_t$ to obtain a prediction $\tilde{y}^t$.

In order to learn the maps which allow us to (1) take $(x)_1, \ldots, (x)_{t-1}$ to obtain $(\Delta\mathbf{x})_1, \ldots, (\Delta\mathbf{x})_t$ and (2) take $(\Delta\mathbf{x})_1, \ldots, (\Delta\mathbf{x})_t$ to predict $\tilde{y}^t$, we again use our training set. Specifically for each t, in stage (1), we fit the vectors of training measurements $\mathbf{x}^1, \ldots, \mathbf{x}^{t-1}$ linearly to $\mathbf{x}^t$ using least squares for each t. In stage (2), we use the innovation vectors $(\Delta\mathbf{x}^1, \ldots, \Delta\mathbf{x}^t)$ to predict $\mathbf{y}^t$ again using least squares. Let's define the matrix $\tilde{\mathbf{X}}^t := (\Delta\mathbf{x}^1, \ldots, \Delta\mathbf{x}^t)$ and $\tilde{\mathbf{X}} = \tilde{\mathbf{X}}^d$.

Show how we can learn the best linear predictions $\hat{\mathbf{x}}^t$ from $\mathbf{x}^1, \ldots, \mathbf{x}^{t-1}$. Then **provide an expression** for $\tilde{\mathbf{y}}^t$ depending on the innovations $\Delta\mathbf{x}^1, \ldots, \Delta\mathbf{x}^t$.

When presented with a new feature vector $\mathbf{x}$, are the sequence of final predictions of the one-stage training $\hat{y}^t$ in (a) and two-stage training $\tilde{y}^t$ in (b) the same? **Explain your reasoning.**

Solution We use linear regression to get $\hat{\mathbf{x}}^t$ first:

$$(\hat{\mathbf{x}})_t = (X)_{t-1}(\hat{w})_{t-1} \text{ ,where } (\hat{w})_{t-1} \text{is a column vector with a length of t which minimizes } \|(X)_{t-1}(\hat{w})_{t-1} - (x)_t\|_2^2$$

We then get the innovation vector for t:

$$(\Delta\mathbf{x})_t := (\mathbf{x})_t - (\hat{\mathbf{x}})_t$$

Similarly, we have:

$$\tilde{\mathbf{y}}^t = \tilde{X}^t\hat{w}^t \text{ ,where } \hat{w}^t \text{ is a column vector with a length of t which minimizes } \|\hat{X}^t\hat{w}^t - y\|_2^2$$

We can write out an expression for $\tilde{\mathbf{y}}^t$ as:

$$\tilde{\mathbf{y}}^t = \hat{X}\hat{w}^t = \hat{X}(\hat{X}^T\hat{X})^{-1}\hat{X}^T y$$

To fully understand this question, let's draw it out

$$\tilde{X}^T = X^T - \hat{W}X^T = \begin{bmatrix} --- & (x^1)^T & --- \\ --- & (x^2)^T & --- \\ & \cdot & \\ & \cdot & \\ & \cdot & \\ --- & (x^d)^T & --- \end{bmatrix} - \begin{bmatrix} 0 & \dots & 0 & \dots & 0 \\ \tilde{w}_{21} & \dots & 0 & \dots & 0 \\ & & \cdot & & \\ & & \cdot & & \\ & & \cdot & & \\ \tilde{w}_{d1} & \dots & \tilde{w}_{d?} & \dots & 0 \end{bmatrix} \begin{bmatrix} --- & (x^1)^T & --- \\ --- & (x^2)^T & --- \\ & \cdot & \\ & \cdot & \\ & \cdot & \\ --- & (x^d)^T & --- \end{bmatrix} = (I - \tilde{W}^T)X^T$$

Therefore, $\tilde{X}$ is just a linear transformation of X and its predictions should be the same as in (b). To get the new coefficients, we just multiply the coefficients in (b) by $I - \tilde{W}$.

d. **Which well-known procedure do the steps to obtain $\tilde{\mathbf{X}}$ from $\mathbf{X}$ remind you of**? (**HINT:** Think about how the column vectors in $\tilde{\mathbf{X}}$ are geometrically related.)

Is there an efficient way to update the weight vector $\hat{\mathbf{w}}^t$ from $\hat{\mathbf{w}}^{t-1}$ when computing the sequence of predictions $\tilde{y}^t$?

Solution It doesn't remind me of anything. My friend says it's Gram–Schmidt process and I trust her. By reading it on Wikipedia, what we essentially did was to make the column vectors in X to be orthogonal vectors in $\tilde{X}$. Having known that, we can take a look at the expansion in (c). We only need to prove that any pair of column vectors are orthogonal. For simplicity, we assume $i < j$:

$$\begin{aligned} & \Delta x^i \Delta x^j \\ = \; & (x^i - (X)^{i-1}\hat{w}^i)^T(x^j - (X)^{j+1}\hat{w}^j) \\ = \; & (x^i)^T(x^j - (X)^{j-1}\hat{w}^j) - (\hat{w}^i)^T((X)^{i-1})^T(x^j - (X)^{j-1}\hat{w}^j) \end{aligned}$$

This is hard understand, but we can take a look at the definition of $\hat{w}^i$: $((X)^{j-1})^T(x^j - (X)^{j-1}\hat{w}^j) = 0$. We can draw $((X)^{j-1})^T$ out:

$$((X)^{j-1})^T = \begin{bmatrix} --- & (x^1)^T & --- \\ --- & (x^2)^T & --- \\ & \cdot & \\ & \cdot & \\ & \cdot & \\ --- & (x^j)^T & --- \end{bmatrix} = \begin{bmatrix} & ((X)^{i-1})^T & \\ --- & (x^i)^T & --- \\ & \cdot & \\ --- & (x^j)^T & --- \end{bmatrix}$$

Because we already know the whole matrix $((X)^{j-1})^T$ is orthogonal to $(x^j - (X)^{j-1}\hat{w}^j)$, a part of the matrix should also be orthogonal to that. This also tells us that all the old measurements before time t are all orthogonal to our new subspace at time t. The whole process is like projecting a new vector into a known subspace and replace the original base.
Prof. Sahai told me during office hour that we should try to make up some data and see what happened to $\hat{w}^t$. I did that and found out the first $t-1$ elements are exactly the same as $\hat{w}^{t-1}$. Knowing this, we can do matrix partitioning:

$$(\tilde{X}^t)^T(\tilde{X}^t)\hat{w}^t = (\tilde{X}^t)^T y$$

$$\Downarrow$$

$$\begin{bmatrix} (\tilde{X}^{t-1})^T \\ (\Delta x^t)^T \end{bmatrix} \begin{bmatrix} \tilde{X}^{t-1} & \Delta x^t \end{bmatrix} \begin{bmatrix} \tilde{w}^{t-1} \\ \tilde{w}_t^t \end{bmatrix} = \begin{bmatrix} (\tilde{X}^{t-1})^T \\ (\Delta x^t)^T \end{bmatrix} y$$

$$\Downarrow$$

$$\begin{bmatrix} (\tilde{X}^{t-1})^T\tilde{X}^{t-1}\tilde{w}^{t-1} + (\tilde{X}^{t-1})^T\Delta x^t\tilde{w}_t^t \\ (\Delta x^t)^T\tilde{X}^{t-1}\tilde{w}^{t-1} + (\Delta x^t)^T\Delta x^t\tilde{w}_t^t \end{bmatrix} = \begin{bmatrix} (\tilde{X}^{t-1})^T y \\ (\Delta x^t)^T y \end{bmatrix}$$

Solution (cont.)

We notice that $\Delta x^t$is orthogonal to$\tilde{X}^{t-1}$ so we have:

$$(\Delta x^t)^T \Delta x^t \tilde{w}_t^t = (\Delta x^t)^T y$$

Each time we just need to solve this equation and add a new element $\tilde{w}_t^t$ to the original $\tilde{w}^{t-1}$.

e. Now let's consider the more general setting where we now want to predict a target vector $\mathbf{y} \in \mathbb{R}^k$ from a feature vector $\mathbf{x} \in \mathbb{R}^d$, thus having a training set consisting of observations $(\mathbf{x}_i, \mathbf{y}_i)$ for $i = 1, \ldots, n$.

Instead of learning a weight vector $\mathbf{w} \in \mathbb{R}^d$, we now want a linear estimate $\hat{\mathbf{y}} = \hat{\mathbf{W}}\mathbf{x}$ with a weight matrix $\hat{\mathbf{W}} \in \mathbb{R}^{k \times d}$ instead. From our samples, we obtain wide matrices $\mathbf{Y} \in \mathbb{R}^{k \times n}$ with columns $\mathbf{y}_1, \ldots, \mathbf{y}_n$ and $\mathbf{X} \in \mathbb{R}^{d \times n}$ with columns $\mathbf{x}_1, \ldots, \mathbf{x}_n$ (note that this is the transpose of $\mathbf{X}$ in Figure 2). In order to learn $\hat{\mathbf{W}}$ we now want to minimize $\|\mathbf{Y} - \mathbf{W}\mathbf{X}\|_F^2$ where $\|\cdot\|_F$ denotes the Frobenius norm of matrices, i.e. $\|\mathbf{L}\|_F^2 = \text{trace}(\mathbf{L}^\top \mathbf{L})$.

Show how to find $\hat{\mathbf{W}} = \arg\min_{\mathbf{W} \in \mathbb{R}^{d \times d}} \|\mathbf{Y} - \mathbf{W}\mathbf{X}\|_F^2$ using vector calculus as reviewed in Discussion 0 and 1.

Solution

$$\begin{aligned}
& \nabla_W \|Y - WX\|_F^2 \\
= \; & \nabla_W (tr((Y - WX)^T (Y - WX))) \\
= \; & \nabla_W (tr(Y^T Y - X^T W^T Y - Y^T W X + X^T W^T W X)) \\
= \; & \nabla_W (tr(Y^T Y) - tr(X^T W^T Y) - tr(Y^T W X) + tr(X^T W^T W X)) \\
= \; & \nabla_W (-2tr(W X Y^T) + tr(X^T W^T W X)) \\
= \; & -2(X Y^T)^T + 2 W X X^T
\end{aligned}$$

Set the gradient to zero, we have

$$-2YX^T + 2\hat{W}XX^T = 0 \Rightarrow \hat{W} = YX^T(XX^T)^{-1}$$

(*Reference for the derivatives of trace : https://web.stanford.edu/~jduchi/projects/matrix_prop.pdf)

f. In the setting of problem (e), **argue why** the computation of the best linear prediction $\hat{\mathbf{y}}$ of a target vector $\mathbf{y}$ using a feature vector $\mathbf{x}$ can be solved by separately finding the best linear prediction for each measurement $(y)_j$ of the target vector $\mathbf{y}$.

Solution We take a look at the above solution in our regular OLS form, where $\tilde{Y}^T = Y$ and $\tilde{X}^T = X$

$$(YX^T(XX^T)^{-1})^T = (\tilde{Y}^T \tilde{X}(\tilde{X}^T \tilde{X})^{-1})^T = (\tilde{X}^T \tilde{X})^{-1} \tilde{X}^T \tilde{Y}$$

They are the exactly same as OLS. Of course, we can solve it by separately finding the best linear prediction for each measurement (it's for each measurement, because we take the transpose of X in 5e).

Problem 6: System Identification by Ordinary Least Squares Regression

Making autonomous vehicles involves machine learning for different purposes. One of which is learning how cars actually behave based on their data.

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

a. Consider the time sequence of scalars $x_t \in \mathbb{R}$ and $u_t \in \mathbb{R}$ in which $x_{t+1} \approx Ax_t + Bu_t$. In control theory, x_t usually represents the state, and u_t usually represents the control input. **Find the numbers A and B so that $\sum_t (x_{t+1} - Ax_t - Bu_t)^2$ is minimized.** The values of x_t and u_t are stored in `a.mat`.

Solution

$$A = 0.97755214, B = -0.08775322$$

```
import numpy as np
import scipy.io

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

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

# Your code to compute a and b
x = x[0]
u = u[0]

b = x[1:]
A = np.vstack([x[:-1], u[:-1]]).T
Result = np.linalg.lstsq(A, b)[0]
print(Result)
```

b. Consider the time sequences of vectors $\mathbf{x}_t \in \mathbb{R}^3$ and $\mathbf{u}_t \in \mathbb{R}^3$ in which $\mathbf{x}_{t+1} \approx \mathbf{A}\mathbf{x}_t + \mathbf{B}\mathbf{u}_t$. **Find the matrix $\mathbf{A} \in \mathbb{R}^{3\times 3}$ and $\mathbf{B} \in \mathbb{R}^{3\times 3}$ so that the sum of the squared ℓ^2-norms of the error, $\sum_t \|\mathbf{x}_{t+1} - \mathbf{A}\mathbf{x}_t - \mathbf{B}\mathbf{u}_t\|_2^2$, is minimized.** The values of $\mathbf{x}_t$ and $\mathbf{u}_t$ are stored in `b.mat`.

Solution

$$A = \begin{bmatrix} 0.15207406 & 0.93480864 & -0.00110243 \\ 0.03893567 & 0.30958727 & 0.87436511 \\ -0.52552959 & 0.0540906 & -0.47026217 \end{bmatrix}, B = \begin{bmatrix} 0.04894161 & 0.20568264 & -0.37090438 \\ -0.04524735 & -0.92861546 & 0.12756569 \\ 0.91096923 & -0.47124981 & -0.84222314 \end{bmatrix}$$

```
import numpy as np
import scipy.io

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

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


# Your code to compute a and b
# We know that x, u \in R^3
N = 3
x = np.reshape(x, (int(x.size / N), N))
u = np.reshape(u, (int(u.size / N), N))

B = x[1:]
A = np.hstack([x[:-1], u[:-1]])
Result = np.linalg.lstsq(A, B)[0].T

print(Result)
```

c. Consider a *car following model* that models how cars line up on a straight 1-d highway at a given time. The acceleration of a car can be approximated by a linear function of the positions and velocities of its own and the car in front of it. Mathematically, we can formulate this as

$$\ddot{x}_i \approx ax_i + b\dot{x}_i + cx_{i-1} + d\dot{x}_{i-1} + e,$$

where x_i, $\dot{x}_i$, and $\ddot{x}_i$ are the position, velocity, and acceleration of the ith car in the line.

Find a, b, c, d, and e that minimizes

$$\sum_i \| - \ddot{x}_i + ax_i + b\dot{x}_i + cx_{i-1} + d\dot{x}_{i-1} + e\|_2^2$$

using data file `train.mat`, which contains the status of 40 000 cars at a given point from the I-80 highway in California. The data were sampled from the Next Generation Simulation (NGSIM) dataset so that the i may not be continuous. For your convenience, we give you the profiles of each sampled car and the car that is in front of it.

Solution

$$xdd_i = -0.012x_i - 0.318xd_i + 0.011x_{i-1} + 0.275xd_{i-1} - 0.883$$

```
# Load mat file
import numpy as np
import scipy.io

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

# Assemble xu matrix
x = mdict["x"] # position of a car
v = mdict["xd"] # velocity of the car
xprev = mdict["xp"] # position of the car ahead
vprev = mdict["xdp"] # velocity of the car ahead

acc = mdict["xdd"] # acceleration of the car

a, b, c, d, e = 0, 0, 0, 0, 0

# Your code to compute a, b, c, d

if x.shape != v.shape or x.shape != xprev.shape \
or x.shape != vprev.shape or x.shape != acc.shape:
print("Dimensions mismatch")
else:
b = acc.T
A = np.vstack([x, v, xprev, vprev, np.ones(x.shape)]).T
Result = np.linalg.lstsq(A, b)[0]
a, b, c, d, e = np.squeeze(Result)
print("xdd_i = {:.3f} x_i + {:.3f} xd_i + {:.3f} x_i-1 + {:.3f} xd_i-1 + {:.3f}".format(a, b, c, d, e))
```

---

d. **Try to justify why your result in (c) is physically reasonable.** Hint: You can reorganize your equation to be

$$\ddot{x}_i = h(x_{i-1} - x_i) + f(\dot{x}_{i-1} - \dot{x}_i) - g(\dot{x}_i - L) + w_i,$$

and try to explain the physical meaning for each term, with L being the speed limit.

Solution

First of all, although I'm not physicists I still know that all units have to be the same and we should nondimensionalize the problem before solving it. Therefore, the equation ("Not my equation") makes zero sense.
However to get points for this question, I have to put myself into your shoes.
$h(x_{i-1} - x_i)$ means the acceleration is proportional to the distance between two cars. This makes sense because we don't want to see big gaps between cars.
$f(\dot{x}_{i-1} - \dot{x}_i)$ means the acceleration is proportional to the speed difference between two cars. This makes sense because we want all cars to be synchronized so the gaps between cars don't increase.
$-g(\dot{x}_i - L)$ means the acceleration is proportional to the velocity from speed limit. This makes sense because we don't want to get tickets. Therefore, it's negative. I've never learned control theory and I even know integral control is much better

Solution (cont.)

than linear control.
w_i is the only term that makes little sense to me because GSI told us it's just noise.

Problem 7: A Simple Classification Approach

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

Classification is an important problem in applied machine learning and is used in many different applications like image classification, object detection, speech recognition, machine translation and others.

In *classification*, we assign each datapoint a class from a finite set (for example the image of a digit could be assigned the value $0, 1, \ldots, 9$ of that digit). This is different from *regression*, where each datapoint is assigned a value from a continuous domain like $\mathbb{R}$ (for example features of a house like location, number of bedrooms, age of the house, etc. could be assigned the price of the house).

In this problem we consider the simplified setting of classification where we want to classify data points from $\mathbb{R}^d$ into *two* classes. For a linear classifier, the space $\mathbb{R}^d$ is split into two parts by a hyperplane: All points on one side of the hyperplane are classified as one class and all points on the other side of the hyperplane are classified as the other class.

The goal of this problem is to show that even a regression technique like linear regression can be used to solve a classification problem. This can be achieved by regressing the data points in the training set against -1 or 1 depending on their class and then using the level set of 0 of the regression function as the classification hyperplane (i.e. we use 0 as a threshold on the output to decide between the classes).

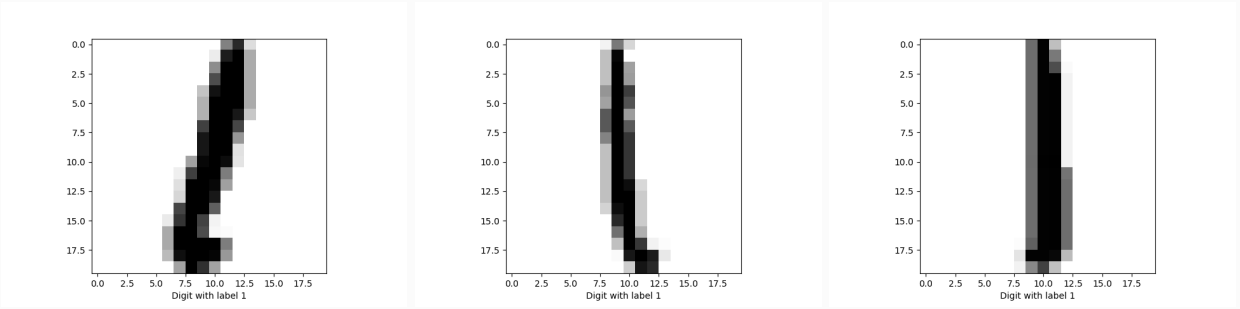
Later in lecture we will learn why linear regression is not the optimal approach for classification and we will study better approaches like logistic regression, SVMs and neural networks.

a. The dataset used in this exercise is a subset of the MNIST dataset. The MNIST dataset assigns each image of a handwritten digit their value from 0 to 9 as a class. For this problem we only keep digits that are assigned a 0 or 1, so we simplify the problem to a two-class classification problem.

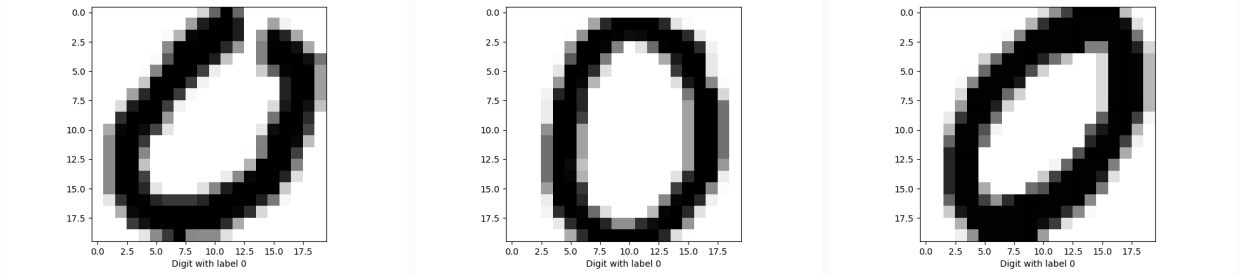
Download and visualize the dataset (example code included). Include three images that are labeled as 0 and three images that are labeled as 1 in your submission.

Solution

Sample images of 1:



Sample images of 0:



b. We now want to use linear regression for the problem, treating class labels as real values $y = -1$ for class "zero" and $y = 1$ for class "one". In the dataset we provide, the images have already been flattened into one dimensional vectors (by concatenating all pixel values of the two dimensional image into a vector) and stacked as rows into a feature matrix $\mathbf{X}$. We want to set up the regression problem $\min_{\mathbf{w}} \|\mathbf{X}\mathbf{w} - \mathbf{y}\|_2^2$ where the entry y_i is the value of the class (-1 or 1) corresponding to the image in row $\mathbf{x}_i^\top$ of the feature matrix. **Solving this regression problem for the training set and report the value of $\|\mathbf{X}\mathbf{w} - \mathbf{y}\|_2^2$ as well as the weights $\mathbf{w}$.** For this problem you may only use pure Python and NumPy (no machine learning libraries!).

Solution

$$||Xw - y||_2^2 = 20.56$$

Solution (cont.)

$$w = \begin{bmatrix} -3.30801139e-01 \\ 3.91726636e-01 \\ 1.48153634e-01 \\ -1.60602318e-01 \\ 1.03277284e-01 \\ -1.96941103e-02 \\ -1.27705114e-01 \\ 9.45889276e-03 \\ -1.71493893e-02 \\ -5.67518079e-03 \\ \dots \\ -4.06860536e-02 \\ -1.61140971e-02 \\ -6.13231934e-03 \\ -3.09129487e-02 \\ -5.01960458e-02 \\ 1.58634491e-02 \\ -1.27863763e-01 \\ 1.68131755e-01 \\ -2.77894244e-01 \\ 4.19035954e-02 \end{bmatrix}$$

c. Given a new flattened image $\mathbf{x}$, one natural rule to classify it is the following one: It is a zero if $\mathbf{x}^\top \mathbf{w} \leqslant 0$ and a one if $\mathbf{x}^\top \mathbf{w} > 0$. **Report what percentage of the digits in the training set are correctly classified by this rule. Report what percentage of the digits in the test set are correctly classified by this rule.**

Solution -1 and 1 no bias
11595 out of 11623 images (99.76%) are correctly classified in the training data set
2111 out of 2115 images (99.81%) are correctly classified in the test data set

d. **Why is the performance typically evaluated on a separate test set (instead of the training set) and why is the performance on the training and test set similar in our case?** We will cover these questions in more detail later in the class.

Solution
1. The performance is typically evaluated on a separate test set because our model is usually optimized based on the training set. If test set data have been included in the training set already then our test error will be arbitrarily small. This will fool us and make us think our model is good.
2. It's because we have lots of data. Recall the calculation we did in problem 3. In this case, for each feature $\delta = 0.01$, $n = 11595$, then we can tolerate above 0.04% of bias (ϵ). Because we only have 400 features, that means we can basically get the best model. If we have that much information, we can also predict our test set pretty well. Another reason is that our test set is relatively small compared to our training set so we might not get a complete picture of other possible test data.
There is another interpretation that because 0 and 1 are so different, there might be one subset of pixels are always dark in 0 and a different subset of pixels are always dark in 1. That is, we have some strong predictors that can readily distinguish between 0 and 1.

e. Somebody suggests to use 0 (for class 0) and 1 (for class 1) as the entries for the target vector $\mathbf{b}$. Try out how well this is doing (make sure to adapt the classification rule, i.e. the threshold set for the outputs). **Report what percentage of digits are correctly classified using this approach on the training set and test set.** How are the performances of the two approaches if you add a bias column to the feature matrix $\mathbf{X}$? A bias column is a column of all ones, i.e. the new feature matrix $\mathbf{X}'$ is

$$\mathbf{X}' = \begin{bmatrix} \mathbf{x}_0^\top & 1 \\ \mathbf{x}_1^\top & 1 \\ \vdots & \vdots \\ \mathbf{x}_n^\top & 1 \end{bmatrix}$$

Report what percentage of digits are correctly classified using regression targets $0/1$ and $-1/1$ with bias on the training set and test set. Try to explain the results!

Solution
-1 and 1 no bias
11595 out of 11623 images (99.76%) are correctly classified in the training data set
2111 out of 2115 images (99.81%) are correctly classified in the test data set
-1 and 1 with bias:
11555 out of 11623 images (99.41%) are correctly classified in the training data set
2107 out of 2115 images (99.62%) are correctly classified in the test data set
0 and 1 no bias:
11503 out of 11623 images (98.97%) are correctly classified in the training data set
2097 out of 2115 images (99.15%) are correctly classified in the test data set
0 and 1 with bias:
11555 out of 11623 images (99.41%) are correctly classified in the training data set
2107 out of 2115 images (99.62%) are correctly classified in the test data set

With bias, linear regression is scale-free meaning that we can manipulate y and preserve the relative prediction value $\hat{y}$. Without bias, we might get better or worse result depending on the scale of y. Including a bias term is essentially the same as centralize the data.

Solution Problem 7 Code attached:

```
import numpy as np
import matplotlib.pyplot as plt

# Load the training dataset
train_features = np.load("train_features.npy")
train_labels = np.load("train_labels.npy").astype("int8")

n_train = train_labels.shape[0]


def visualize_digit(features, label):
# Digits are stored as a vector of 400 pixel values. Here we
# reshape it to a 20x20 image so we can display it.
plt.imshow(features.reshape(20, 20), cmap="binary")
plt.xlabel("Digit with label " + str(label))
plt.show()


# Visualize a digit
# visualize_digit(train_features[0,:], train_labels[0])

# TODO: Plot three images with label 0 and three images with label 1

count0 = 0
count1 = 0
for i in range(1, train_labels.size):
if count0 >= 3 and count1 >= 3:
break
if count0 < 3 and train_labels[i] == 0:
visualize_digit(train_features[i, :], train_labels[i])
count0 += 1
if count1 < 3 and train_labels[i] == 1:
visualize_digit(train_features[i, :], train_labels[i])
count1 += 1
# print("label %d count0 %d count1 %d." % (train_labels[i], count0, count1))

# Linear regression

# TODO: Solve the linear regression problem, regressing
# X = train_features against y = 2 * train_labels - 1

X = train_features
y = 2 * train_labels - 1
w = np.linalg.lstsq(X, y)[0]
print(w)
```

Solution (cont.)

```
print("||Xw-y||_2^2 %.2f" % np.linalg.norm(X.dot(w) - y))

# TODO: Report the residual error and the weight vector

# Load the test dataset
# It is good practice to do this after the training has been
# completed to make sure that no training happens on the test
# set!


test_features = np.load("test_features.npy")
test_labels = np.load("test_labels.npy").astype("int8")

n_test = test_labels.size
n_train = train_labels.size

train_predicted = train_features.dot(w)
test_predicted = test_features.dot(w)

one_label = 1
zero_label = 0
one_value = 1
zero_value = -1
threshold = (one_value + zero_value)/2

train_correct = np.count_nonzero(np.logical_or((train_predicted > threshold) == (train_labels == one_label),
    (train_predicted <= threshold) == (train_labels == zero_label)))
test_correct = np.count_nonzero(np.logical_or((test_predicted > threshold) == (test_labels == one_label),
    (test_predicted <= threshold) == (test_labels == zero_label)))
print("-1 and 1 no bias")
print("%d out of %d images (%.2f\%%) are correctly classified in the training data set" % (train_correct,
    n_train, train_correct / n_train * 100))
print("%d out of %d images (%.2f\%%) are correctly classified in the test data set" % (test_correct, n_test,
    test_correct / n_test * 100))


# stupid code to make sure test data do not overlap with training data
"""
for i in range(1, test_labels.size):
img = test_features[i, :]
label = test_labels[i]
for j in range(1, train_labels.size):
if label == train_labels[j] and np.count_nonzero(img == train_features[j, :]) == img.size:
print("test image %d is the same as training image %d" %(i+1, j+1))
print("Done")
"""

# TODO: Implement the classification rule and evaluate it
# on the training and test set

# TODO: Try regressing against a vector with 0 for class 0
# and 1 for class 1

# TODO: Form a new feature matrix with a column of ones added
# and do both regressions with that matrix

train_features_bias = np.hstack([train_features, np.ones((train_features.shape[0], 1))])
test_features_bias = np.hstack([test_features, np.ones((test_features.shape[0], 1))])
one_label = 1
zero_label = 0

# -1 and 1 with bias

X = train_features_bias
y = 2 * train_labels - 1
w = np.linalg.lstsq(X, y)[0]
```

Solution (cont.)

```
train_predicted = train_features_bias.dot(w)
test_predicted = test_features_bias.dot(w)

one_value = 1
zero_value = -1
threshold = (one_value + zero_value)/2

train_correct = np.count_nonzero(np.logical_or((train_predicted > threshold) == (train_labels == one_label),
    (train_predicted <= threshold) == (train_labels == zero_label)))
test_correct = np.count_nonzero(np.logical_or((test_predicted > threshold) == (test_labels == one_label),
    (test_predicted <= threshold) == (test_labels == zero_label)))

print("-1 and 1 with bias: ")
print("%d out of %d images (%.2f\%%) are correctly classified in the training data set" % (train_correct,
    train_labels.size, train_correct / train_labels.size * 100))
print("%d out of %d images (%.2f\%%) are correctly classified in the test data set" % (test_correct,
    test_labels.size, test_correct / test_labels.size * 100))


# 0 and 1 no bias
X = train_features
y = train_labels
w = np.linalg.lstsq(X, y)[0]

train_predicted = train_features.dot(w)
test_predicted = test_features.dot(w)

one_value = 1
zero_value = 0
threshold = (one_value + zero_value)/2

train_correct = np.count_nonzero(np.logical_or((train_predicted > threshold) == (train_labels == one_label),
    (train_predicted <= threshold) == (train_labels == zero_label)))
test_correct = np.count_nonzero(np.logical_or((test_predicted > threshold) == (test_labels == one_label),
    (test_predicted <= threshold) == (test_labels == zero_label)))

print("0 and 1 no bias: ")
print("%d out of %d images (%.2f\%%) are correctly classified in the training data set" % (train_correct,
    train_labels.size, train_correct / train_labels.size * 100))
print("%d out of %d images (%.2f\%%) are correctly classified in the test data set" % (test_correct,
    test_labels.size, test_correct / test_labels.size * 100))

# 0 and 1 with bias
X = train_features_bias
y = train_labels
w = np.linalg.lstsq(X, y)[0]

train_predicted = train_features_bias.dot(w)
test_predicted = test_features_bias.dot(w)

one_value = 1
zero_value = 0
threshold = (one_value + zero_value)/2

train_correct = np.count_nonzero(np.logical_or((train_predicted > threshold) == (train_labels == one_label),
    (train_predicted <= threshold) == (train_labels == zero_label)))
test_correct = np.count_nonzero(np.logical_or((test_predicted > threshold) == (test_labels == one_label),
    (test_predicted <= threshold) == (test_labels == zero_label)))

print("0 and 1 with bias: ")
print("%d out of %d images (%.2f\%%) are correctly classified in the training data set" % (train_correct,
    train_labels.size, train_correct / train_labels.size * 100))
print("%d out of %d images (%.2f\%%) are correctly classified in the test data set" % (test_correct,
    test_labels.size, test_correct / test_labels.size * 100))

# Logistic Regression
```

Solution (cont.)

```
# You can also compare against how well logistic regression is doing.
# We will learn more about logistic regression later in the course.
"""
import sklearn.linear_model
lr = sklearn.linear_model.LogisticRegression()
lr.fit(X, train_labels)
test_error_lr = 1.0 * sum(lr.predict(test_features) != test_labels) / n_test
"""
```

Problem 8: Your Own Question

Write your own question, and provide a thorough solution.

Writing your own problems is a very important way to really learn the 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.

Solution

I will compare the performance of the linear regression in problem 7 to the logistic regression. The question is: which one is better? The logistic regression correctly predicts 99.95% of the test set. It's even better than the least square linear regression

```
train_features = np.load("train_features.npy")
train_labels = np.load("train_labels.npy").astype("int8")

test_features = np.load("test_features.npy")
test_labels = np.load("test_labels.npy").astype("int8")

import sklearn.linear_model
lr = sklearn.linear_model.LogisticRegression()
lr.fit(train_features, train_labels)
test_error_lr = 1.0 * sum(lr.predict(test_features) != test_labels) / test_labels.size

print(1-test_error_lr)
```