EECS 189 Introduction to Machine Learning
Fall 2020 HW0

This homework is due **Tuesday, September 1 at 11:59 p.m.** Since no material from 189/289A is in scope for this homework, the problems in this homework were taken verbatim from the freshman course EECS 16B, and the problems might reference the lectures and labs from that course, even though they should be self-contained. If you have seen these questions before, that is fine — this is a chance to bring these basics back into your memory. If you have not seen these questions before, that is also fine but you really need to be able to do such questions very easily since we will be assuming this background throughout the course.

You may notice that some of the notation has changed from 16B - for instance, vectors were written like $\vec{x}$ in 16B, but $\mathbf{x}$ in this homework. This is done in accordance with the notation convention used in 189, in order to keep course materials consistent, even though it means modifying the look of these questions.

Some questions will require you to run a Jupyter notebook. You can either download the notebook as a ZIP from the course website or load it from DataHub in your browser at `links.eecs189.org/hw0-notebook`.

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 the appropriate assignment on Gradescope. 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.
3. If there is a test set, submit your test set evaluation results.

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

(a) Who else did 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?

(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 nor have I looked at any online solutions to any of these problems. I have credited all external sources in this write up.

2 Sample Submission

Please submit a plain text file to the Gradescope programming assignment "Homework 0 Test Set":

1. Containing 5 rows, where each row has only one value "1".
2. No spaces or miscellaneous characters.
3. Name it "submission.txt".

3 Stability for information processing: solving least-squares via gradient descent with a constant step size

Although ideas of control were originally developed to understand how to control physical and electronic systems, they can be used to understand purely informational systems as well. Most of modern machine learning is built on top of fundamental ideas from control theory. This is a problem designed to give you some of this flavor.

In this problem, we will derive a dynamical system approach for solving a least-squares problem which finds the $\mathbf{x}$ that minimizes $\|A\mathbf{x} - \mathbf{y}\|^2$. We consider A to be tall and full rank — i.e. it has linearly independent columns.

As covered in EE16A, this has a closed-form solution:

$$\hat{\mathbf{x}} = (A^T A)^{-1} A^T \mathbf{y}.$$

Direct computation requires the "inversion" of $A^T A$, which has a complexity of $O(N^3)$ where $(A^T A) \in \mathbb{R}^{N \times N}$. This may be okay for small problems with a few parameters, but can easily become unfeasible if there are lots of parameters that we want to fit. Instead, we will solve the problem iteratively using something called "gradient descent" which turns out to fit into our perspective of state-space dynamic equations. Again, this problem is just trying to give you a flavor for this and connect to stability, "gradient descent" itself is not yet in scope for 16B.

(a) Let $\mathbf{x}(t)$ be the estimate of $\mathbf{x}$ at time step t. We can define the least-squares error $\boldsymbol{\epsilon}(t)$ to be:

$$\boldsymbol{\epsilon}(t) = \mathbf{y} - A\mathbf{x}(t)$$

Show that if $\mathbf{x}(t) = \hat{\mathbf{x}}$, then $\boldsymbol{\epsilon}(t)$ is orthogonal to the columns of A, i.e. show $A^T \boldsymbol{\epsilon}(t) = \mathbf{0}$.

This was shown to you in 16A, but it is important that you see this for yourself again.

(b) We would like to develop a "fictional" state space equation for which the state $\mathbf{x}(t)$ will converge to $\mathbf{x}(t) \to \hat{\mathbf{x}}$, the true least squares solution. The evolution of these states reflects what is happening computationally.

Here $A\mathbf{x}(t)$ represents our current reconstruction of the output $\mathbf{y}$. The difference $(\mathbf{y} - A\mathbf{x}(t))$ represents the current residual.

We define the following update:

$$\mathbf{x}(t+1) = \mathbf{x}(t) + \alpha A^T(\mathbf{y} - A\mathbf{x}(t)) \tag{1}$$

that gives us an updated estimate from the previous one. Here α is the step-size that we get to choose. For us in 16B, it doesn't matter where this iteration comes from. But if you want, this can be interpreted as a tentative sloppy projection. If A had orthonormal columns, then $A^T(\mathbf{y} - A\mathbf{x}(t))$ would take us exactly to where we need to be. It would update the parameters perfectly. But A doesn't have orthonormal columns, so we just move our estimate a little bit in that direction where α controls how much we move. You can see that if we ever reach $\mathbf{x}(t) = \hat{\mathbf{x}}$, the system reaches equilibrium — it stops moving. At that point, the residual is perfectly orthogonal to the columns of A. In a way, this is a dynamical system that was chosen based on where its equilibrium point is.

By the way, it is no coincidence that the gradient of $\|A\mathbf{x} - \mathbf{y}\|^2$ with respect to $\mathbf{x}$ is

$$\nabla\|A\mathbf{x} - \mathbf{y}\|^2 = 2A^T(A\mathbf{x} - \mathbf{y})$$

This can be derived directly by using vector derivatives (outside of 16B's class scope) or by carefully using partial derivatives as we will do for linearization, later in 16B. So, the heuristic update (1) is actually just taking a step along the negative gradient direction. This insight is what lets us adapt this heuristic for a kind of "linearization" applied to other optimization problems that aren't least-squares. (But all this is currently out-of-scope for 16B at this point, and is something discussed further in 127 and 189. Here, at this point in 16B, (1) is just some discrete-time linear system that we have been given.)

To show that $\mathbf{x}(t) \to \hat{\mathbf{x}}$, we define a new state variable $\Delta\mathbf{x}(t) = \mathbf{x}(t) - \hat{\mathbf{x}}$.

Derive the discrete-time state evolution equation for $\Delta\mathbf{x}(t)$, and show that it takes the form:

$$\Delta\mathbf{x}(t+1) = (I - \alpha G)\Delta\mathbf{x}(t). \tag{2}$$

(c) We would like to make the system such that $\Delta\mathbf{x}(t)$ converges to 0. As a first step, we just want to make sure that we have a stable system. To do this, we need to understand the eigenvalues of $I - \alpha G$. **Show that the eigenvalues of matrix $I - \alpha G$ are $1 - \alpha\lambda_{i\{G\}}$, where $\lambda_{i\{G\}}$ are the eigenvalues of G.**

(d) To be stable, we need all these eigenvalues to have magnitudes that are smaller than 1 (since this is a discrete-time system). Since the matrix G above has a special form, all of the eigenvalues of G are non-negative and real. **For what α would the eigenvalue $1 - \alpha\lambda_{max\{G\}} = 0$ where**

$\lambda_{max\{G\}}$ **is the largest eigenvalue of** G**. At this** α**, what would be the largest magnitude eigenvalue of** $I - \alpha G$**? Is the system stable?**

(Hint: Think about the smallest eigenvalue of G. What happens to it? Feel free to assume that this smallest eigenvalue $\lambda_{min\{G\}}$ *is strictly greater than* 0*.)*

(e) **Above what value of** α **would the system** (2) **become unstable?** This is what happens if you try to set the learning rate to be too high.

(f) Looking back at the part before last (where you moved the largest eigenvalue of G to zero), **if you slightly increased the** α**, would the convergence become faster or slower?**

(HINT: think about the dominant eigenvalue here. Which is the eigenvalue of $I - \alpha G$ *with the largest magnitude?)*

(g) **What is the** α **that would result in the system being stable, and converge fastest to** $\Delta \mathbf{x} = 0$**?**

(HINT: When would growing α *stop helping shrink the biggest magnitude eigenvalue of* $I - \alpha G$*?)*

(h) **Play with the given jupyter notebook and comment on what you observe.** Consider how the different step sizes relate to recurrence relations, and how a sufficiently small step size can approach a continuous solution.

4 Proving the SVD

In lecture, we thought about the SVD for a wide matrix M with n rows and $m > n$ columns by looking at the big $m \times m$ symmetric matrix $M^T M$ and its eigenbasis. This question is about seeing what happens when we look at the small $n \times n$ symmetric matrix $Q = MM^T$ and its orthonormal eigenbasis $U = [\mathbf{u}_1, \mathbf{u}_2, \cdots, \mathbf{u}_n]$ instead. Suppose we have sorted the eigenvalues so that the real eigenvalues $\widetilde{\lambda}_i$ are sorted in descending order where $\widetilde{\lambda}_i \mathbf{u}_i = Q\mathbf{u}_i$.

(a) **Show that** $\widetilde{\lambda}_i \geq 0$**.**

(HINT: You want to involve $\mathbf{u}_i^T \mathbf{u}_i$ *somehow.)*

(b) Suppose that we define $\mathbf{w}_i = \frac{M^T \mathbf{u}_i}{\sqrt{\widetilde{\lambda}_i}}$ for all i for which $\widetilde{\lambda}_i > 0$. Suppose that there are ℓ such eigenvalues. **Show that** $W = [\mathbf{w}_1, \mathbf{w}_2, \cdots, \mathbf{w}_\ell]$ **has orthonormal columns.**

(c) **Prove that** $M = \sum_{i=1}^{\ell} \sqrt{\widetilde{\lambda}_i} \mathbf{u}_i \mathbf{w}_i^T$**.**

(HINT: Take an arbitrary left-input $\mathbf{x}^T$ *and consider* $\mathbf{x}^T M$*. Decompose* $\mathbf{x}$ *into the U basis. See what happens when you multiply things out.)*

(d) Consider an arbitrary input $\mathbf{x} = \mathbf{x}_\perp + \sum_{i=1}^{\ell} \alpha_i \mathbf{w}_i$ where $\mathbf{x}_\perp$ is orthogonal to each of the $\mathbf{w}_i$. **Show that** $M\mathbf{x}_\perp = \mathbf{0}$**.**

(HINT: Use the previous part.)

5 Low Rank Approximation of a Matrix

In this question we will study the so called "low rank approximation" problem. As the name implies, consider an arbitrary matrix $X \in \mathbb{R}^{m\times n}$, with $n \geq m$. (It doesn't matter how we do this, but we will consider the matrix X as consisting of n columns of m-dimensional vectors.) We are interested in finding another matrix $\widehat{X}$ having specified lower rank k, such that $\widehat{X}$ is "closest" to X, i.e.,

$$\min_{\widehat{X}} \|X - \widehat{X}\|_F \tag{3}$$

$$\text{subject to} \quad \text{rank}(\widehat{X}) \leq k \tag{4}$$

This problem goes to the heart of how we use the SVD for dimensionality reduction and to look at data. If we view a data matrix as a collection of columns where each of the columns is a different data point, then a rank-k approximation to that matrix is a collection of columns all of which represent points that are all on a k-dimensional subspace. *This discovery of hidden subspace structure is what finding low-rank approximations is truly about.* (The analogous story could be told about rows, but we'll keep our focus on columns since you are all more comfortable working with columns from 16A and 16B so far.)

To understand this problem, we will have to also think about what it might mean to approximate a matrix and we use the natural matrix norm to do this. In the previous homework, you were introduced to the Frobenius norm — which involved treating a matrix as though it was just a big long vector filled with its entries.

(a) First, let's understand one of the simpler interpretations of why rank-r approximations to huge matrices are so useful. To specify an arbitrary $m\times n$ matrix, we have to choose $m\times n$ independent elements (entries). In other words, an arbitrary $m \times n$ matrix has $m \times n$ degrees of freedom. **How much information (independent elements/degree of freedom) do we have to know to specify a rank r matrix of the same $m \times n$ size. How is low-rank approximation a kind of lossy compression for a matrix?**

(HINT: Think about outer-product representations for a rank r matrix, for example, that given by the SVD.)

(b) Now, let us start into understanding the approximation itself. Before we get into the "hidden subspace" aspect, we need to think about what it means to approximate. Suppose we view a matrix A as a list of columns:

$$A = [\mathbf{a}_1, \mathbf{a}_2, \cdots, \mathbf{a}_n] \tag{5}$$

Show that the Frobenius norm $\|A\|_F = \sqrt{\sum_{i=1}^{m}\sum_{j=1}^{n}|A[i][j]|^2}$ for a matrix A can be understood in terms of the regular Euclidean norms of its columns as:

$$\|A\|_F = \sqrt{\sum_{j=1}^{n}\|\mathbf{a}_j\|^2} \tag{6}$$

This is useful because it justifies using Frobenius norm as a way to measure the length of a matrix when we are really viewing the matrix as a collection of columns. (It turns out the same thing is true for viewing it as a collection of rows.)

(c) Now we want to get into the "hidden subspace" aspect of this problem. To do this, we need a way of talking about a potential subspace. To define a subspace, we need a basis. For convenience, we might as well think of an orthonormal basis. Let the matrix B consist of k orthonormal columns. The matrix B defines a subspace S of dimension k. Our underlying goal is to find an optimal such subspace S for approximating the data in X and equivalently, to find an optimal basis B for it. (Note here that $B \in \mathbb{R}^{m \times k}$. There are k columns, each of which is an m-dimensional vector.)

Before we worry about finding such a basis or subspace, it is good to understand how we would approximate X using it. But we know how to approximate a column in a subspace! We can project into it. So we can project all of X into that subspace simultaneously by computing BB^TX. To understand the quality of this approximation, first **show that** $\|X - BB^TX\|_F^2 = \|X\|_F^2 - \|BB^TX\|_F^2$.

(HINT: Give things names and invoke the Pythagorean theorem. Let $\mathbf{x}_i$ *be the i-th column of X and let* $Y_B = BB^TX$*, and let* $\mathbf{y}_{B,i}$ *be the i-th column of* Y_B*. Let* $R_B = X - Y_B$ *be the residual that remains when estimating X using the basis B. Then show the desired result for each column using the properties of projection and put things together.)*

(d) Now, our goal is to find the optimal such basis B. We can see from the previous part that since $\|X\|_F^2$ is outside our control, minimizing $\|X - BB^TX\|_F^2$ is the same as finding a matrix B with k orthonormal columns that maximizes $\|BB^TX\|_F^2$. Homework 10 showed that for any matrix A, and a matrix with orthonormal columns U:

$$\|UA\|_F = \|A\|_F.$$

Using this fact, it immediately follows that: $\|BB^TX\|_F^2 = \|B^TX\|_F^2$. Consequently, it suffices to find a B with orthonormal columns that maximizes $\|B^TX\|_F^2$. This is what the rest of the problem is about.

Now, we are going to zoom in on a special case of our main theorem first. Consider the special case of $X = \Sigma$ matrices (with m rows and $n \geq m$ columns) that are already diagonal. Further suppose that the diagonal of Σ is non-negative and sorted so that it has $\sigma_1 \geq \sigma_2 \geq \cdots \geq \sigma_m \geq 0$ down the diagonal. (i.e. We are considering the kinds of Σ matrices that the SVD gives us.)

To warm up, further restrict attention to matrices B that are made up of only standard basis vectors (i.e. these are the columns of the identity — each of the k columns of B has at most one 1 in them and the rest of that column is zero.) Furthermore, to be orthonormal, no two columns can have a 1 in the same row — we have to pick a subset of the columns of the identity matrix as our B matrix.

Under that assumption, **show that for such a** B**, the** $\|B^TX\|_F^2$ **must be a sum of** k **different** σ_i^2**.**

(HINT: Realize that each of the columns of B basically will pick out exactly one of the σ_i*.)*

(e) Building on the previous part and its very special assumptions, **show that any resulting $Y_B = BB^T\Sigma$ is going to be a diagonal matrix and have the i-th diagonal entry equal to either 0 or σ_i.**

(HINT: The i-th standard basis vector is either in the basis B or not. What happens if it is in the basis B? What happens if it is not in the basis B?)

(f) Building on the previous part and its assumptions, **show that a best such $Y_B = BB^T\Sigma$ (for minimizing $\|\Sigma - Y_B\|_F^2$) has $\sigma_1, \sigma_2, \ldots, \sigma_k, 0, \ldots, 0$ down the diagonal.**

Further assume that all of the $\sigma_i < \sigma_k$ for $i > k$. (This is just to prevent ties that wouldn't change anything, but would slightly complicate writing the proof.)

(HINT: In a sense, this should feel a bit obvious. But how do you prove it? The previous parts tell you that we want to maximize $\|B^T\Sigma\|_F^2$, which in turn is the sum of the squares of any k distinct diagonal elements of Σ. Now, the claim is that choosing the largest k of the σ's will give the largest sum. As is often the case for obvious things, it is easiest to proceed by a contradiction argument. Suppose that you had selected a truly different subset of the σs. How do you show that this subset isn't the best possible subset? One way is to show how you can strictly improve on it. What would you do to improve on it?)

(g) Now that we have dealt with the very special case as a warmup, we want to relax our artificial restriction that B has to be made of standard basis vectors. From this point forward, B is generic and just has k orthonormal columns. They can be anything we want. Let's look at the columns $\mathbf{c}_i$ of $C = B^T$. We can immediately see that $B^T\Sigma = C\Sigma = [\sigma_1\mathbf{c}_1, \sigma_2\mathbf{c}_2, \cdots, \sigma_m\mathbf{c}_m, \mathbf{0}, \mathbf{0}, \cdots, \mathbf{0}]$.

So we need to get a handle on these columns. Since $\|C\|_F^2 = \|B\|_F^2 = k$, we know that $\sum_{i=1}^n \|\mathbf{c}_i\|^2 = k$. We also know that since they are norms, that each of the $\|\mathbf{c}_i\|^2 \geq 0$.

Show that $\|\mathbf{c}_i\|^2 \leq 1$ for every $i = 1, \ldots, m$.

(HINT: Invoke Gram-Schmidt to assert that you can extend B with $m - k$ more orthonormal vectors to get a square orthonormal matrix $\widetilde{B}$. Then, since $\widetilde{B}^T\widetilde{B} = \widetilde{B}\widetilde{B}^T = I$, what do you know about the norms of the columns of $\widetilde{B}^T$? What is the relationship of those norms to the norms of $\mathbf{c}_i$?)

(h) You know from above that $\|B^T\Sigma\|_F^2 = \sum_{i=1}^m \sigma_i^2\|\mathbf{c}_i\|^2$ and that further $0 \leq \|\mathbf{c}_i\|^2 \leq 1$ for each i and their sum $\sum_{i=1}^m \|\mathbf{c}_i\|^2 = k$.

Further assume that all of the $\sigma_i < \sigma_k$ for $i > k$. (This is just to prevent ties that wouldn't change anything, but would complicate writing the proof.)

Show that under those constraints, $\sum_{i=1}^m \sigma_i^2\|\mathbf{c}_i\|^2 \leq \sum_{i=1}^k \sigma_i^2$.

(Hint: For convenience in writing, give new names $f_i = \|\mathbf{c}_i\|^2$ and remember that $\sum f_i = k$ while $0 \leq f_i \leq 1$.

This is another thing that should feel like it is kind of obvious. It should feel like a more continuous version of what you already showed earlier (two parts ago) about the best possible subset of numbers to sum up if we want to maximize the sum — pick the biggest numbers. Now, you're allowed to take fractional parts of the numbers if you'd like. You probably have no

doubt that the best thing to do is to just take the biggest numbers. Why would you want to take half of a big number and half of a smaller number instead of all of the big number?

Anyway, you need to make this thinking into a proof. Getting an upper bound is about maximizing. You know how to reach that bound. Assume that you have some different allocation (i.e. it includes some positive allocation in an index $> k$) of the k total weight across the f_i that is claimed to be optimal and bigger than your claimed bound. Show that you can make it even bigger by redistributing the weight while keeping your constraints all satisfied. This contradicts the supposed optimality of the claimed optimal allocation. And so no such different allocation can exist.)

(i) Because you know this bound can be hit, you have actually proved that the best k-dimensional subspace for approximating Σ in Frobenius norm is just that spanned by the first k standard basis vectors. In other words, the best rank k approximation to a diagonal matrix Σ with non-negative elements $\sigma_i \geq 0$ on the diagonal that are non-decreasing (i.e. $\sigma_i \geq \sigma_j$ if $i < j$) is the diagonal matrix with $\sigma_1, \sigma_2, \ldots, \sigma_k$ on the diagonal at the beginning and zero everywhere else.

We have already proved in a previous homework that using the SVD, the Frobenius norm can be understood in terms of the singular values $\|A\|_F^2 = \sum_{i=0}^{\min(m,n)} \sigma_i^2$ (Recall that a $m \times n$ matrix can have at most $\min(m, n)$ nonzero singular values), where σ_i's are the singular values of A.

This was proved as a consequence of the fact that if U has orthonormal columns, then $\|UA\|_F^2 = \text{Tr}(A^T U^T UA) = \text{Tr}(A^T IA) = \text{Tr}(A^T A) = \|A\|_F^2$ even if U is not square, and similarly if V^T has orthonormal rows, then $\|AV^T\|_F^2 = \text{Tr}(AV^T VA^T) = \text{Tr}(AIA^T) = \text{Tr}(AA^T) = \|A\|_F^2$. That means that the SVD $A = U\Sigma V^T$ implies that $\|A\|_F = \|\Sigma\|_F$.

Now **please solve for** $\hat{X}$ in equation (3) using the results you developed so far in earlier parts of this problem for the diagonal case.

(HINT: The SVD of X is going to be useful here.)

Congratulations! You have now been walked (hopefully not dragged!) through an elementary proof of why the SVD gives you the best low-rank approximation to a matrix of data. This is the heart of PCA.

In the linear-algebraic (and machine learning) literature, this is called the Eckhart-Young-Mirsky Theorem but all of the proofs that are easily accessible online or in standard textbooks take a more challenging (but shorter) route to the result. The argument given here is in more elementary 16AB style — it is the figurative long "green circle" trail down from the top of the mountain as compared to the black diamond path taken by more experienced skiers. Anyway, this important result justifies many uses of the SVD in machine learning, control, and statistics.

6 Weighted minimum norm problems

(Merged problem across past 16B HW and exams.)

You saw in 16B lecture in the context of open-loop control, how we consider problems in which

we have a wide matrix A and solve $A\mathbf{x} = \mathbf{y}$ such that $\mathbf{x}$ is a minimum norm solution:

$$||\mathbf{x}|| \leq ||\mathbf{z}||$$

for all $\mathbf{z}$ such that $A\mathbf{z} = \mathbf{y}$. You then saw this idea again earlier in this HW where you saw how to compute the appropriate "pseudo-inverse" for such wide matrices.

But what if you weren't interested in just the norm of $\mathbf{x}$? What if you instead cared about minimizing the norm of a linear transformation $C\mathbf{x}$? For example, suppose that controls were more or less costly at different times.

The problem can be written out mathematically as:

Given a wide matrix A and a matrix C find $\mathbf{x}$ such that $A\mathbf{x} = \mathbf{y}$ and $||C\mathbf{x}|| \leq ||C\mathbf{z}||$ for all $\mathbf{z}$ such that $A\mathbf{z} = \mathbf{y}$.

(a) Let's start with the case of C being invertible. **Solve this problem (i.e. find the optimal x with the minimum $||C\mathbf{x}||$) for the specific matrices and y given below. Show your work.**

It is fine to leave your answer as an explicit product of matrices and vectors.

(HINT: You might want to change variables to solve this problem. Don't forget to change back!)

$$A = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 1 \end{bmatrix}, \quad C = \begin{bmatrix} 0 & 0 & 2 \\ 0 & 1 & 0 \\ 2 & 0 & 0 \end{bmatrix}, \quad \mathbf{y} = \begin{bmatrix} 2 \\ 1 \end{bmatrix}$$

For convenience, $C^{-1} = \begin{bmatrix} 0 & 0 & 0.5 \\ 0 & 1 & 0 \\ 0.5 & 0 & 0 \end{bmatrix}$ and you are also given some SVDs on the following page.

$$A = (U_A = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix})(\Sigma_A = \begin{bmatrix} \sqrt{2} & 0 & 0 \\ 0 & 1 & 0 \end{bmatrix})(V_A^T = \begin{bmatrix} 0 & \frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} \\ 1 & 0 & 0 \\ 0 & -\frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} \end{bmatrix}) \tag{7}$$

$$C = (U_C = \begin{bmatrix} -1 & 0 & 0 \\ 0 & 0 & 1 \\ 0 & -1 & 0 \end{bmatrix})(\Sigma_C = \begin{bmatrix} 2 & 0 & 0 \\ 0 & 2 & 0 \\ 0 & 0 & 1 \end{bmatrix})(V_C^T = \begin{bmatrix} 0 & 0 & -1 \\ -1 & 0 & 0 \\ 0 & 1 & 0 \end{bmatrix}) \tag{8}$$

$$AC = (U_{AC} = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}(\Sigma_{AC} = \begin{bmatrix} \sqrt{5} & 0 & 0 \\ 0 & 2 & 0 \end{bmatrix})(V_{AC}^T = \begin{bmatrix} \frac{2}{\sqrt{5}} & \frac{1}{\sqrt{5}} & 0 \\ 0 & 0 & 1 \\ -\frac{1}{\sqrt{5}} & \frac{2}{\sqrt{5}} & 0 \end{bmatrix}) \tag{9}$$

$$AC^{-1} = (U_{AC^{-1}} = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix})(\Sigma_{AC^{-1}} = \begin{bmatrix} \frac{\sqrt{5}}{2} & 0 & 0 \\ 0 & 0.5 & 0 \end{bmatrix})(V_{AC^{-1}}^T = \begin{bmatrix} \frac{1}{\sqrt{5}} & \frac{2}{\sqrt{5}} & 0 \\ 0 & 0 & 1 \\ -\frac{2}{\sqrt{5}} & \frac{1}{\sqrt{5}} & 0 \end{bmatrix}) \tag{10}$$

(b) What if C were a tall matrix with linearly independent columns? **Explicitly describe how you would solve this problem in that case, step by step.**

For convenience, we have copied the problem statement again here: Given a wide matrix A and a matrix C **find** $\mathbf{x}$ such that $A\mathbf{x} = \mathbf{y}$ and $\|C\mathbf{x}\| \leq \|C\mathbf{z}\|$ for all $\mathbf{z}$ such that $A\mathbf{z} = \mathbf{y}$.

Here, you can assume that the wide matrix A has linearly-independent rows but is otherwise generic. Similarly, $\mathbf{y}$ is a generic vector.

(HINT: Does C have a nullspace? Does C^TC have a nullspace? Does the SVD of C suggest any (invertible) change of coordinates from $\mathbf{x}$ to $\widetilde{\mathbf{x}}$ such that $\|\widetilde{\mathbf{x}}\| = \|C\mathbf{x}\|$?)

(c) The key issue in extending to wide C matrices is that they have nontrivial nullspaces — that means that there are "free" directions in which we can vary $\mathbf{x}$ while not having to pay anything. How do we best take advantage of these "free" directions?

Given a wide matrix A (with m columns and n rows) and a wide matrix C (with m columns and r rows), we want to solve:

$$\min_{\mathbf{x} \text{ such that } A\mathbf{x}=\mathbf{y}} \|C\mathbf{x}\| \tag{11}$$

As mentioned above, the key new issue is to isolate the "free" directions in which we can vary $\mathbf{x}$ so that they might be properly exploited. Consider the full SVD of $C = U\Sigma_c V^\top = \sum_{i=1}^{\ell} \sigma_{c,i}\mathbf{u}_i\mathbf{v}_i^\top$. Here, we write $V = [V_c, V_f]$ so that $V_c = [\mathbf{v}_1, \mathbf{v}_2, \cdots, \mathbf{v}_\ell]$ all correspond to singular values $\sigma_{c,i} > 0$ of C, and $V_f = [\mathbf{v}_{\ell+1}, \cdots, \mathbf{v}_m]$ form an orthonormal basis for the nullspace of C.

Change variables in the problem to be in terms of $\widetilde{\mathbf{x}} = \begin{bmatrix} \widetilde{\mathbf{x}}_c \\ \widetilde{\mathbf{x}}_f \end{bmatrix}$ where the ℓ-dimensional $\widetilde{\mathbf{x}}_c$ has components $\widetilde{x}_c[i] = \alpha_i \mathbf{v}_i^\top \mathbf{x}$, and the $(m-\ell)$-dimensional $\widetilde{\mathbf{x}}_f$ has components $\widetilde{x}_f[i] = \mathbf{v}_{\ell+i}^\top \mathbf{x}$. In vector/matrix form, $\widetilde{\mathbf{x}}_f = V_f^\top \mathbf{x}$ and $\widetilde{\mathbf{x}}_c = \begin{bmatrix} \alpha_1 & 0 & \cdots & 0 \\ 0 & \alpha_2 & \cdots & 0 \\ \vdots & \ddots & \ddots & \vdots \\ 0 & 0 & \cdots & \alpha_\ell \end{bmatrix} V_c^\top \mathbf{x}$. Or directly, $\widetilde{\mathbf{x}} = \begin{bmatrix} \widetilde{\mathbf{x}}_c \\ \widetilde{\mathbf{x}}_f \end{bmatrix} = \left[\begin{bmatrix} \alpha_1 & 0 & \cdots & 0 \\ 0 & \alpha_2 & \cdots & 0 \\ \vdots & \ddots & \ddots & \vdots \\ 0 & 0 & \cdots & \alpha_\ell \end{bmatrix} V_c^\top \right] \mathbf{x}$.

$$\left[\begin{bmatrix} \alpha_1 & 0 & \cdots & 0 \\ 0 & \alpha_2 & \cdots & 0 \\ \vdots & \ddots & \ddots & \vdots \\ 0 & 0 & \cdots & \alpha_\ell \end{bmatrix} V_c^\top \right] \mathbf{x}.$$

$$\begin{bmatrix} & V_f^\top & \end{bmatrix}$$

Express $\mathbf{x}$ in terms of $\widetilde{\mathbf{x}}_f$ and $\widetilde{\mathbf{x}}_c$. Assume the $\alpha_i \neq 0$ so the relevant matrix is invertible.

What is $\|C\mathbf{x}\|$ in terms of $\widetilde{\mathbf{x}}_f$ and $\widetilde{\mathbf{x}}_c$? Simplify as much as you can for full credit.

(HINT: If you get stuck on how to express $\mathbf{x}$ in terms of the new variables, think about the special case when $\ell = 1$ and $\alpha_1 = \frac{1}{2}$. How is this different from when $\alpha_1 = 1$? The SVD of C might be useful when looking at $\|C\mathbf{x}\|$.)

(d) Continuing the previous part, **give appropriate values for the α_i so that the problem** (11)

becomes

$$\min_{\widetilde{\mathbf{x}}=\begin{bmatrix}\widetilde{\mathbf{x}}_c\\ \widetilde{\mathbf{x}}_f\end{bmatrix} \text{such that } [A_c, A_f]\begin{bmatrix}\widetilde{\mathbf{x}}_c\\ \widetilde{\mathbf{x}}_f\end{bmatrix}=\mathbf{y}} \|\widetilde{\mathbf{x}}_c\| \tag{12}$$

Give explicit expressions for A_c and A_f in terms of the original A and terms arising from the SVD of C. Because you have picked values for the α_i, there should be no α_i in your final expressions for full credit.

(HINT: How do the singular values $\sigma_{c,i}$ interact with the α_i? Then apply the appropriate substitution to (11) *to get* (12)*.)*

(e) Let us focus on a simple case. *(You can do this even if you didn't get the previous parts.)* Suppose that $A = [A_c, A_f]$ where the columns of A_f are orthonormal, as well as orthogonal to the columns of A_c. The columns of A together span the entire n-dimensional space. We directly write $\mathbf{x} = \begin{bmatrix}\mathbf{x}_c\\ \mathbf{x}_f\end{bmatrix}$ so that $A\mathbf{x} = A_c\mathbf{x}_c + A_f\mathbf{x}_f$. Now suppose that we want to solve $A\mathbf{x} = \mathbf{y}$ and only care about minimizing $\|\mathbf{x}_c\|$. We don't care about the length of $\mathbf{x}_f$ — it can be as big or small as necessary.

In other words, we want to:

$$\min_{\mathbf{x}=\begin{bmatrix}\mathbf{x}_c\\ \mathbf{x}_f\end{bmatrix} \text{such that } [A_c, A_f]\begin{bmatrix}\mathbf{x}_c\\ \mathbf{x}_f\end{bmatrix}=\mathbf{y}} \|\mathbf{x}_c\| \tag{13}$$

Show that the optimal solution has $\mathbf{x}_f = A_f^\top \mathbf{y}$.

(HINT: Multiplying both sides of something by $A_f^\top$ might be helpful.)

(f) Continuing the previous part, **compute the optimal $\mathbf{x}_c$.** Show your work.

(HINT: What is the work that $\mathbf{x}_c$ needs to do? $\mathbf{y} - A_f A_f^\top \mathbf{y}$ might play a useful role, as will the SVD of $A_c = \sum_i \sigma_i \mathbf{t}_i \mathbf{w}_i^\top$.)

(g) Now suppose that A_c did not necessarily have its columns orthogonal to A_f. Continue to assume that A_f has orthonormal columns. *(You can do this part even if you didn't get any of the previous parts.)* Write the matrix $A_c = A_{c\perp} + A_{cf}$ where the columns of A_{cf} are all in the column span of A_f and the columns of $A_{c\perp}$ are all orthogonal to the columns of A_f. **Give an expression for A_{cf} in terms of A_c and A_f.**

(HINT: What does this have to do with projection and least squares?)

(h) Continuing the previous part, **compute the optimal $\mathbf{x}_c$** that solves (13): (copied below)

$$\min_{\mathbf{x}=\begin{bmatrix}\mathbf{x}_c\\ \mathbf{x}_f\end{bmatrix} \text{such that } [A_c, A_f]\begin{bmatrix}\mathbf{x}_c\\ \mathbf{x}_f\end{bmatrix}=\mathbf{y}} \|\mathbf{x}_c\|$$

Show your work. Feel free to call the SVD as a black box as a part of your computation.

(HINT: What is the work that $\mathbf{x}_c$ needs to do? The SVD of $A_{c\perp}$ might be useful.)

(i) Continuing the previous part, **compute the optimal $\mathbf{x}_f$.**

You can use the optimal $\mathbf{x}_c$ in your expression just assuming that you did the previous part correctly, even if you didn't. You can also assume a decomposition $A_c = A_{c\perp} + A_{cf}$ from further above without having to write what these are, just assume that you did them correctly, even if you didn't do them at all.

(HINT: What is the work that $\mathbf{x}_f$ needs to do? How is A_{cf} relevant here?

(j) The only question that remains is what to do if the A_f does not have orthonormal columns.

Suppose that we have a matrix A_f, but we don't know whether or not the columns of A_f are linearly independent or orthonormal. How can you use the results from the SVD $A_f = U_f \Sigma_f V_f^\top = \sum_{i=1}^{r} \sigma_{f,i} \mathbf{u}_{f,i} \mathbf{v}_{f,i}^\top$ to help project a vector $\mathbf{s}$ onto the subspace spanned by the columns of A_f? Here, the $\sigma_{f,1}, \ldots, \sigma_{f,r}$ are all strictly positive. **Give an expression for a matrix P_f such that $P_f\mathbf{s}$ is the projection of $\mathbf{s}$ in the subspace spanned by the columns of A_f.**

Also give an expression for an $\mathbf{x}_f$ so that $A_f\mathbf{x}_f = P_f\mathbf{s}$.

7 Linearization to help classification: discovering logistic regression and how to solve it

We saw in lecture how the problem of linear classification between two categories boiled down to discovering a vector that told us what to project onto to get a score, together with a threshold that told us where to draw the boundary between the two categories. The goal in linear classification problems is to learn those vectors and thresholds from data — examples that have been labeled as belonging to the two categories in question.

You can, in spirit, reduce the problem of linear multi-class classification to that of binary classification by picking vectors that correspond to each of the categories "X" as compared with all the other examples categorized into a hybrid synthetic category of "not-X". This will give rise to vectors corresponding to each category with the winner selected by seeing which one wins. However, we will focus here on the binary problem since that is the conceptual heart of this approach.

As was discussed in lecture, the naive straightforward way of picking the decision boundary (by looking at the mean example of each category and drawing the perpendicular bisector) is not always the best. The included Jupyter Notebook includes synthetic examples that illustrate the different things that can happen so that you can better appreciate the pathway that almost inexorably leads us to discover logistic regression as a natural approach to solve this problem based on the conceptual building blocks that you have already seen. (See if you can catch why this problem follows the previous one on approximating Bode plots.)

It is no exageration to say that logistic regression is the default starting method for doing classification in machine learning contexts, the same way that straightforward linear regression is the default starting method for doing regression. A lot of other approaches can be viewed as being built on top of logistic regression. Consequently, getting to logistic regression is a nice ending-point for this part of the 16AB story as pertains to classification.

Let's start by giving things some names. Consider trying to classify a set of measurements $\mathbf{x}_i$ with given labels ℓ_i. For the binary case of interest here, we will think of the labels as being "+" and "-" for reasons that should be clear from our discussion of multi-class classification above. For expository convenience, and because we don't want to have to carry it around separately, we will fold our threshold implicitly into the weights by augmenting our given measurements with the constant "1" in the first position of $\mathbf{x}_i$ for each i. Now, the classification rule becomes simple. We want to learn a vector of weights $\mathbf{w}$ so that we can deem any point with $\mathbf{x}_i^\top \mathbf{w} > 0$ as being a member of the "+" category and anything with $\mathbf{x}_i^\top \mathbf{w} < 0$ as being a member of the "-" category.

The way that we will do this, following the treatment in lecture, is to do a minimization in the spirit of least squares. Except, instead of necessarily using some sort of squared loss function, we will just consider a generic cost function that can depend on the label and the prediction for the point. For the i-th data point in our training data, we will incur a cost $c^{\ell_i}(\mathbf{x}_i^\top \mathbf{w})$ for a total cost that we want to minimize as:

$$\arg\min_{\mathbf{w}} \quad c_{\text{total}}(\mathbf{w}) = \sum_{i=1}^{m} c^{\ell_i}(\mathbf{x}_i^T \mathbf{w}) \tag{14}$$

Because this can be a nonlinear function, our goal is to solve this iteratively as a sequence of least-squares problems that we know how to solve.

Consider the following algorithm pattern

1: $\mathbf{w} = \mathbf{0}$ ▷ Initialize the weights to $\mathbf{0}$
2: **while** Not done **do** ▷ Iterate towards solution
3: Compute $p_i = \mathbf{w}^T \mathbf{x}_i$ ▷ Generate current estimated labels
4: Compute $(c^{\ell_i})'(p_i)$ ▷ Generate derivatives of the cost for update step
5: Compute $(c^{\ell_i})''(p_i)$ ▷ Generate second derivatives of the cost for update step
6: $\boldsymbol{\delta}\mathbf{w} = LeastSquares(\cdot, \cdot)$ ▷ We will derive what to call least squares on
7: $\mathbf{w} = \mathbf{w} + \boldsymbol{\delta}\mathbf{w}$ ▷ Update parameters
8: **end while**
9: Return $\mathbf{w}$

The key step above is figuring out with what arguments to call our good friend LeastSquares — who we know so well from 16A and from using it in 16B. We just have the labels ℓ_i and the points $\mathbf{x}_i$. We need to create new virtual inputs and outputs for least-squares to try to match.

Notice that this is the same algorithm pattern you saw in the last homework where you were trying to set the joint angles on a robot arm so that it reaches a desired target point. There, you solved a minimum-norm problem in each loop of the iteration, but decided to take only a cautious step partially towards that solution. Here, it is a least-squares problem, and we are going to accept taking bolder steps in that direction.

Recall from lecture and discussion that the derivative of a scalar function of a vector is row, and the second derivative can be represented by a matrix.

When the function $\mathbf{f}(\mathbf{x}, \mathbf{y}) : \mathbb{R}^n \times \mathbb{R}^k \to \mathbb{R}^m$ takes in vectors and outputs a vector, the relevant

derivatives for linearization are also represented by matrices:

$$D_{\mathbf{x}}\mathbf{f} = \begin{bmatrix} \frac{\partial f_1}{\partial x[1]} & \cdots & \frac{\partial f_1}{\partial x[n]} \\ \vdots & \ddots & \vdots \\ \frac{\partial f_m}{\partial x[1]} & \cdots & \frac{\partial f_m}{\partial x[n]} \end{bmatrix}.$$

$$D_{\mathbf{y}}\mathbf{f} = \begin{bmatrix} \frac{\partial f_1}{\partial y[1]} & \cdots & \frac{\partial f_1}{\partial y[k]} \\ \vdots & \ddots & \vdots \\ \frac{\partial f_m}{\partial y[1]} & \cdots & \frac{\partial f_m}{\partial y[k]} \end{bmatrix}.$$

Then, the linearization (first-order expansion) becomes

$$\mathbf{f}(\mathbf{x}, \mathbf{y}) \approx \mathbf{f}(\mathbf{x}_0, \mathbf{y}_0) + D_{\mathbf{x}}\mathbf{f} \cdot (\mathbf{x} - \mathbf{x}_0) + D_{\mathbf{y}}\mathbf{f} \cdot \left(\mathbf{y} - \mathbf{y}_0\right).$$

(a) Now, suppose we wanted to approximate the total cost function

$$c_{total}(\mathbf{w}) = \sum_{i=1}^{m} c^{\ell_i}(\mathbf{x}_i^\top \mathbf{w}) \tag{15}$$

in the neighborhood of a weight vector $\mathbf{w}_0$. **Write out the first-order expression for approximating the cost function** $c_{total}(\mathbf{w}_0 + \boldsymbol{\delta}\mathbf{w})$.

This should be something in vector/matrix form like you have seen for the approximation of nonlinear systems by linear systems. We don't want to take any second derivatives just yet — only first derivatives.

(HINT: Use the linearity of derivatives and sums to compute the partial derivatives with respect to each of the $w[g]$ terms. Don't forget the chain rule and the fact that $\mathbf{x}_i^\top \mathbf{w} = \sum_{j=1}^{n} x_i[j]w[j] = x_i[g]w[g] + \sum_{j \neq g} x_i[j]w[j]$. Then put them together into an appropriate row. Your final answer should involve an appropriately weighed sum of rows.)

(b) Now, we want a better approximation that includes second derivatives. For a general function, we would look for

$$f(\mathbf{x}_0 + \boldsymbol{\delta}\mathbf{x}) \approx f(\mathbf{x}_0) + f'(\mathbf{x}_0)\boldsymbol{\delta}\mathbf{x} + \frac{1}{2}\boldsymbol{\delta}\mathbf{x}^\top f(\mathbf{x}_0)''\boldsymbol{\delta}\mathbf{x} \tag{16}$$

where $f'(\mathbf{x}_0)$ is an appropriate row and, as you've seen in class, $f''(\mathbf{x}_0)$ is a matrix that represents the second derivatives.

Take the second derivatives of the total cost and write an expression for the second-order approximation for the total cost function.

(HINT: Once again, use the linearity of derivatives and sums to compute the partial derivatives with respect to each of the $w[h]$ terms. This will give you $\frac{\partial^2}{\partial w[g]\partial w[h]}$. Don't forget the chain rule and again use the fact that $\mathbf{x}_i^\top \mathbf{w} = \sum_{j=1}^{n} x_i[j]w[j] = x_i[h]w[h] + \sum_{j \neq h} x_i[j]w[j]$. Then put everything together into an appropriate matrix. Your final answer should involve an appropriately weighed sum of matrices. Each matrix will itself be representable in simple outer-product form.)

(c) **Complete the squares to reformat the cost approximation in the form of a sum of a constant plus the sum of $(\mathbf{q}_i^\top \delta\mathbf{w} - b_i)^2$.**

(d) **Interpret the result of the previous step as desiring for δw to be the solution to a standard least-squares problem. What is the A matrix? What is the y?**

(e) Consider the following cost functions:

squared error: $c^+_{sq}(p) = (p-1)^2,\ c^-_{sq}(p) = (p+1)^2$;

exponential: $c^+_{exp}(p) = e^{-p},\ c^-_{exp}(p) = e^{p}$;

and logistic: $c^+_{logistic}(p) = \ln(1+e^{-p}),\ c^-_{logistic}(p) = \ln(1+e^{p})$.

Compute the first and second derivatives of the above expressions with respect to p.

(f) **Fill in the missing code and examine how these cost functions behave in the Jupyter notebook for classification. What did you see?**

Congratulations! You now know the basic optimization-theoretic perspective on logistic regression. After you understand the probability material in 70 and 126, you can understand the probabilistic perspective on it as well. After you understand the optimization material in 127, you will understand even more about the optimization-theoretic perspective on the problem including why this approach actually converges.

Contributors:

- Aditya Arun
- Anant Sahai
- Kuan-Yun Lee
- Kyle Tanghe
- Miki Lustig
- Moses Won
- Nathan Lambert
- Nikhil Shinde
- Pavan Bhargava
- Rahul Arya
- Sidney Buchbinder
- Yuxun Zhou