CS 189 / 289A Introduction to Machine Learning
Fall 2023 Jennifer Listgarten, Jitendra Malik HW1

Due 09/08/22 11:59 PM PT

- Homework 1 consists of both written and coding questions.
- We prefer that you typeset your answers using LaTeX or other word processing software. If you haven't yet learned LaTeX, one of the crown jewels of computer science, now is a good time! Neatly handwritten and scanned solutions will also be accepted for the written questions.
- In all of the questions, **show your work**, not just the final answer.
- **Start early. This is a long assignment. Most of the material is prerequisite material not covered in lecture; you are responsible for finding resources to understand it.**

Deliverables:

1. Submit a PDF of your homework to the Gradescope assignment entitled "HW 1 Write-Up". **Please start each question on a new page.** If there are graphs, include those graphs in the correct sections. **Do not** put them in an appendix. We need each solution to be self-contained on pages of its own.
    - In your write-up, please state with whom you worked on the homework. This should be on its own page and should be the first page that you submit.
    - In your write-up, please copy the following statement and sign your signature underneath. If you are using LaTeX, you can type your full name underneath instead. We want to make it *extra* clear so that no one inadvertently cheats.

      "I certify that all solutions are entirely in my own words and that I have not looked at another student's solutions. I have given credit to all external sources I consulted."
    - **Replicate all of your code in an appendix**. Begin code for each coding question on a fresh page. Do not put code from multiple questions in the same page. When you upload this PDF on Gradescope, *make sure* that you assign the relevant pages of your code from the appendix to correct questions.

1 Gradients and Derivatives (13 points)

What is the derivative of the function $f : \mathbb{R} \to \mathbb{R}$ given by $f(x) = 2x$? From basic calculus, we know that the answer is 2 everywhere. Fundamentally, the derivative is a *best linear approximation*: for $f(x) = 2x$, anywhere we look, the linear transformation that best approximates f is "multiplication by 2" (in fact, f is exactly equal to this linear transformation in our current example). More precisely, for any differentiable function f, the derivative at a, denoted $\frac{df}{dx}(a)$, is the *best linear approximation of f at a*. That is,

$$f(x) \approx f(a) + \frac{df}{dx}(a) * (x - a)$$

for all x near a (the equation above is the line tangent to f at a). Equivalently, we can view $\frac{df}{dx}(a)$ as the slope or the rate of change of f at a. Thus, the derivative of $f(x) = 2x + 3$ is also equal to 2 everywhere; constant shifts do not change the derivative.

This perspective is also handy in higher dimensions. Take the function $\mathbf{f} : \mathbb{R}^n \to \mathbb{R}^k$ given by $\mathbf{f}(\mathbf{x}) = \mathbf{A}\mathbf{x}$ where $\mathbf{A} \in \mathbb{R}^{k \times n}$ (i.e., $\mathbf{A}$ is a real-valued $k \times n$ matrix) and $\mathbf{x} \in \mathbb{R}^n$. How can we compute the derivative now that one scalar has been replaced by a matrix, and the other by a vector? We can simply recognize the fact that $\frac{\mathrm{df}}{\mathrm{dx}}(\mathbf{x})$, also denoted by $D\mathbf{f}(\mathbf{x})$ in some texts, must be the linear transformation that best approximates $\mathbf{f}$ at $\mathbf{x}$. However, note that $\mathbf{f} : \mathbf{x} \mapsto \mathbf{A}\mathbf{x}$ is nothing more than multiplication by $\mathbf{A}$, a linear transformation already. Thus, $\frac{\mathrm{df}}{\mathrm{dx}} = \mathbf{A}$ for all $\mathbf{x} \in \mathbb{R}^n$.

The derivative and gradient of a function of a vector

When $f : \mathbb{R}^n \to \mathbb{R}$ maps a vector to a scalar, the derivative at a point $\mathbf{a} \in \mathbb{R}^n$ is a linear transformation from $\mathbb{R}^n$ to $\mathbb{R}$, represented by a row vector $\frac{\mathrm{d}f}{\mathrm{dx}}(\mathbf{a}) \in \mathbb{R}^{1 \times n}$, that gives the *best linear approximation* of $f(\mathbf{x})$ near $\mathbf{a}$. That is, for $\mathbf{x} - \mathbf{a}$ small,

$$f(\mathbf{x}) \approx f(\mathbf{a}) + \left[\frac{\mathrm{d}f}{\mathrm{dx}}(\mathbf{a})\right](\mathbf{x} - \mathbf{a})$$

The *gradient* is the transpose of the derivative, $\nabla_{\mathbf{x}} f(\mathbf{x}) = \left[\frac{\mathrm{d}f}{\mathrm{dx}}(\mathbf{x})\right]^\top \in \mathbb{R}^n$. Note that it is a column vector and not a row vector:

$$f(\mathbf{x}) \approx f(\mathbf{a}) + [\nabla_{\mathbf{x}} f(\mathbf{a})]^\top (\mathbf{x} - \mathbf{a})$$

Why do we bother to define the gradient? The fact that the gradient is the same shape as the input is convenient, and its ith entry is the partial derivative of f with respect to the ith entry of the input:

$$[\nabla_{\mathbf{x}} f(\mathbf{x})]_i = \frac{\partial f}{\partial x_i}(\mathbf{x})$$

$$\nabla_{\mathbf{x}} f(\mathbf{x}) = \begin{bmatrix} \frac{\partial f}{\partial x_1}(\mathbf{x}) \\ \vdots \\ \frac{\partial f}{\partial x_n}(\mathbf{x}) \end{bmatrix}$$

The derivative and gradient of a function of a matrix

Similarly, when $f : \mathbb{R}^{n\times m} \to \mathbb{R}$ maps a matrix to a scalar, its derivative at $\mathbf{A} \in \mathbb{R}^{n\times m}$ is a linear transformation from $\mathbb{R}^{n\times m}$ to $\mathbb{R}$ that gives the *best linear approximation* of $f(\mathbf{X})$ near $\mathbf{A}$. That is, for $\mathbf{X} - \mathbf{A}$ small,

$$f(\mathbf{X}) \approx f(\mathbf{A}) + \left[\frac{\mathrm{d}f}{\mathrm{d}\mathbf{X}}(\mathbf{A})\right](\mathbf{X} - \mathbf{A})$$

For any linear transformation $T : \mathbb{R}^{n\times m} \to \mathbb{R}$, there is a matrix $\mathbf{B} \in \mathbb{R}^{n\times m}$ associated with it such that:

$$\forall \mathbf{X} \in \mathbb{R}^{n\times m},\ T(\mathbf{X}) = \langle \mathbf{B}, \mathbf{X}\rangle := \mathrm{Tr}\left(\mathbf{B}^\top \mathbf{X}\right)$$

Thus, the transformation T can be represented by the matrix $\mathbf{B}$. Don't be intimidated if you haven't seen this before: $\mathrm{Tr}\left(\mathbf{B}^\top \mathbf{X}\right)$ is merely the matrix dot product between $\mathbf{B}$ and $\mathbf{X}$ in the same sense as a regular vector dot product. It is equivalent to multiplying $\mathbf{B}$ and $\mathbf{X}$ element-wise and summing up the entries of the resulting matrix, or flattening out both matrices and computing their vector dot product (in fact, you should verify this fact yourself by expanding $\mathbf{B}^\top \mathbf{X}$ and taking its trace).

We can now define the gradient $\nabla_{\mathbf{X}} f(\mathbf{X}) \in \mathbb{R}^{n\times m}$ as the matrix representing the derivative linear transformation from above, i.e., $\left[\frac{\mathrm{d}f}{\mathrm{d}\mathbf{X}}(\mathbf{X})\right](\mathbf{C}) = \langle \nabla_{\mathbf{X}} f(\mathbf{X}), \mathbf{C}\rangle$, for all $\mathbf{C} \in \mathbb{R}^{n\times m}$. Therefore, the best linear approximation of $f(\mathbf{X})$ near $\mathbf{A}$ can be re-written as

$$f(\mathbf{X}) \approx f(\mathbf{A}) + \langle \nabla_{\mathbf{X}} f(\mathbf{A}), \mathbf{X} - \mathbf{A}\rangle$$

The gradient, as in the vector case, is also expressible as the matrix of partial derivatives of f with respect to each entry of $\mathbf{X}$:

$$\nabla_{\mathbf{X}} f(\mathbf{X})]_{ij} = \frac{\partial f}{\partial X_{ij}}(\mathbf{X})$$

$$\nabla_{\mathbf{X}} f(\mathbf{X}) = \begin{bmatrix} \frac{\partial f}{\partial X_{11}}(\mathbf{X}) & \dots & \frac{\partial f}{\partial X_{1m}}(\mathbf{X}) \\ \vdots & \ddots & \vdots \\ \frac{\partial f}{\partial X_{n1}}(\mathbf{X}) & \dots & \frac{\partial f}{\partial X_{nm}}(\mathbf{X}) \end{bmatrix}$$

Note: just like the gradient with respect to a vector has the same dimension as said vector, the gradient with respect to a matrix has the same shape as the matrix. The importance of this fact will become clearer when we cover Gradient Descent.

The Hessian

Finally, we define the Hessian of a function $f : \mathbb{R}^n \to \mathbb{R}$ as the $n \times n$ matrix with elements

$$\left[\nabla^2_{\mathbf{x}} f(\mathbf{x})\right]_{ij} = \frac{\partial^2 f}{\partial x_i \partial x_j}(\mathbf{x}).$$

The Hessian equals the derivative of the gradient, $\nabla^2_{\mathbf{x}} f(\mathbf{x}) = \frac{\mathrm{d}}{\mathrm{d}\mathbf{x}}[\nabla_{\mathbf{x}} f(x)]$. If f has continuous second order partial derivatives (and most functions you see in this course will indeed), this matrix is symmetric.

Just like the derivative/gradient provides the "best linear approximation" to a vector function $f(\mathbf{x})$ near a point $\mathbf{a} \in \mathbb{R}^n$, the Hessian can be used to also define its "best quadratic approximation":

$$f(\mathbf{x}) \approx f(\mathbf{a}) + [\nabla_{\mathbf{x}} f(\mathbf{a})]^\top (\mathbf{x} - \mathbf{a}) + \frac{1}{2}(\mathbf{x} - \mathbf{a})^\top [\nabla_{\mathbf{x}}^2 f(\mathbf{a})](\mathbf{x} - \mathbf{a})$$

Let's get some practice with all of the concepts above through the following problems:

(a) (1 point) Let $\mathbf{w} \in \mathbb{R}^n$. Compute the gradient $\nabla_{\mathbf{x}} f(\mathbf{x})$ of

$$f : \mathbb{R}^n \to \mathbb{R},\ f(\mathbf{x}) = \mathbf{w}^\top \mathbf{x}$$

Solution: The function f is already linear: it is multiplication by $\mathbf{w}^\top$. Thus,

$$\nabla_{\mathbf{x}} f(\mathbf{x}) = \left[\frac{\mathrm{d}f}{\mathrm{d}\mathbf{x}}(\mathbf{x})\right]^\top = [\mathbf{w}^\top]^\top = \mathbf{w}$$

Alternatively, since $f(\mathbf{x}) = \sum_{j=1}^n w_j x_j$, we have $[\nabla_{\mathbf{x}} f(\mathbf{x})]_i = \frac{\partial f}{\partial x_i}(\mathbf{x}) = w_i$.

(b) (2 points) Let $\mathbf{A} \in \mathbb{R}^{n \times n}$. Compute the gradient $\nabla_{\mathbf{x}} f(\mathbf{x})$ of

$$f : \mathbb{R}^n \to \mathbb{R},\ f(\mathbf{x}) = \mathbf{x}^\top \mathbf{A} \mathbf{x}$$

Solution: Since the derivative $\frac{\mathrm{d}f}{\mathrm{d}\mathbf{x}}$ is the best linear approximation to $f(\mathbf{x})$, for small $\boldsymbol{\Delta} \in \mathbb{R}^n$,

$$\begin{aligned}
f(\mathbf{x} + \boldsymbol{\Delta}) &\approx f(\mathbf{x}) + \left[\frac{\mathrm{d}f}{\mathrm{d}\mathbf{x}}\right]\Delta \\
(\mathbf{x} + \boldsymbol{\Delta})^\top \mathbf{A}(\mathbf{x} + \boldsymbol{\Delta}) &\approx \mathbf{x}^T \mathbf{A}\mathbf{x} + \left[\frac{\mathrm{d}f}{\mathrm{d}\mathbf{x}}\right]\Delta \\
\mathbf{x}^\top \mathbf{A}\mathbf{x} + \boldsymbol{\Delta}^\top \mathbf{A}\mathbf{x} + \mathbf{x}^\top \mathbf{A}\boldsymbol{\Delta} + \boldsymbol{\Delta}^\top \mathbf{A}\boldsymbol{\Delta} &\approx \mathbf{x}^T \mathbf{A}\mathbf{x} + \left[\frac{\mathrm{d}f}{\mathrm{d}\mathbf{x}}\right]\Delta \\
\mathbf{x}^\top \mathbf{A}^\top \boldsymbol{\Delta} + \mathbf{x}^\top \mathbf{A}\boldsymbol{\Delta} + \boldsymbol{\Delta}^\top \mathbf{A}\boldsymbol{\Delta} &\approx \left[\frac{\mathrm{d}f}{\mathrm{d}\mathbf{x}}\right]\Delta \\
\mathbf{x}^\top (\mathbf{A}^\top + \mathbf{A})\boldsymbol{\Delta} + \boldsymbol{\Delta}^\top \mathbf{A}\boldsymbol{\Delta} &\approx \left[\frac{\mathrm{d}f}{\mathrm{d}\mathbf{x}}\right]\Delta
\end{aligned}$$

Note that $\boldsymbol{\Delta}^T \mathbf{A}\boldsymbol{\Delta}$ is a quadratic term that vanishes much quicker than any term linear in $\boldsymbol{\Delta}$ as $\|\boldsymbol{\Delta}\|_2 \to 0$; for small $\boldsymbol{\Delta}$, this term is negligible. Therefore, matching the non-negligible terms, we can see that

$$\frac{\mathrm{d}f}{\mathrm{d}\mathbf{x}} = \mathbf{x}^T(\mathbf{A}^\top + \mathbf{A}) \implies \nabla_{\mathbf{x}} f(\mathbf{x}) = \left[\mathbf{x}^T(\mathbf{A}^\top + \mathbf{A})\right]^T = (\mathbf{A} + \mathbf{A}^\top)\mathbf{x}$$

An alternative way would be to expand $\mathbf{x}^\top \mathbf{A}\mathbf{x}$ in terms of a_{ij} and x_k, take the partial derivatives individually and stack them into a column vector. Let $\mathbf{a}_i$ be the ith column of $\mathbf{A}$. Similarly, let $\boldsymbol{\alpha}_i$ be the ith row of $\mathbf{A}$. Then,

$$f(\mathbf{x}) = \mathbf{x}^\top \mathbf{A}\mathbf{x}$$

$$
\begin{aligned}
&= \sum_{i=1}^{n} \sum_{j=1}^{n} a_{ij} x_i x_j \\
&= \sum_{i=1}^{n} a_{ii} x_i^2 + \sum_{i \neq j} a_{ij} x_i x_j \\
\frac{\partial}{\partial x_i} f(\mathbf{x}) &= 2a_{ii} x_i + \sum_{\substack{j \in [1,\dots,n] \\ j \neq i}} a_{ij} x_j + \sum_{\substack{j \in [1,\dots,n] \\ j \neq i}} a_{ji} x_j \\
&= \sum_{j=1}^{n} a_{ij} x_j + \sum_{j=1}^{n} a_{ji} x_j \\
&= \boldsymbol{\alpha}_i^\top \mathbf{x} + \mathbf{a}_i^\top \mathbf{x} \\
\nabla_{\mathbf{x}} f(\mathbf{x}) &= \begin{bmatrix} \frac{\partial}{\partial x_i} f(\mathbf{x}) \\ \vdots \\ \frac{\partial}{\partial x_n} f(\mathbf{x}) \end{bmatrix} = \begin{bmatrix} \boldsymbol{\alpha}_1^\top \mathbf{x} + \mathbf{a}_1^\top \mathbf{x} \\ \vdots \\ \boldsymbol{\alpha}_n^\top \mathbf{x} + \mathbf{a}_n^\top \mathbf{x} \end{bmatrix} \\
&= \begin{bmatrix} \boldsymbol{\alpha}_1^\top \\ \vdots \\ \boldsymbol{\alpha}_n^\top \end{bmatrix} \mathbf{x} + \begin{bmatrix} \mathbf{a}_1^\top \\ \vdots \\ \mathbf{a}_n^\top \end{bmatrix} \mathbf{x} \\
&= (\mathbf{A} + \mathbf{A}^\top)\mathbf{x}
\end{aligned}
$$

Both methods result in the same solution!

(c) (1 point) Compute the Hessian $\nabla_{\mathbf{x}}^2 f(\mathbf{x})$ of the function above.

Solution: The Hessian is the derivative of the gradient:

$$\nabla_{\mathbf{x}}^2 f(\mathbf{x}) = \frac{\mathrm{d}}{\mathrm{d}\mathbf{x}}[(\mathbf{A}^\top + \mathbf{A})\mathbf{x}] = \mathbf{A}^\top + \mathbf{A}$$

using the fact that the derivative of a matrix times a vector with respect to the vector is just the matrix.

(d) (2 points) Let $\mathbf{A} \in \mathbb{R}^{m \times n}$, $y \in \mathbb{R}^m$. Compute the gradient $\nabla_{\mathbf{x}} f(\mathbf{x})$ of

$$f : \mathbb{R}^n \to \mathbb{R},\ f(\mathbf{x}) = \|\mathbf{A}\mathbf{x} - \mathbf{y}\|_2^2.$$

Hint: you can use the chain rule of derivatives. Given vector spaces U, V, W and functions $\mathbf{f} : U \to W$, $\mathbf{z} \mapsto \mathbf{f}(\mathbf{z})$ and $\mathbf{g} : V \to U$, $\mathbf{x} \mapsto \mathbf{g}(\mathbf{x})$ differentiable, we have

$$\frac{\mathrm{d}(\mathbf{f} \circ \mathbf{g})}{\mathrm{d}\mathbf{x}}(\mathbf{x}) = \left[\frac{\mathrm{d}\mathbf{f}}{\mathrm{d}\mathbf{z}}(\mathbf{g}(\mathbf{x}))\right] \cdot \left[\frac{\mathrm{d}\mathbf{g}}{\mathrm{d}\mathbf{x}}(\mathbf{x})\right]$$

Solution: Note that $f = h \circ \mathbf{g}$ where $h(\mathbf{z}) = \mathbf{z}^\top \mathbf{z}$ and $\mathbf{g}(\mathbf{x}) = \mathbf{A}\mathbf{x} - \mathbf{y}$. We observe that $g(\mathbf{x})$ is a linear transformation already (but with an offset $-\mathbf{y}$) so $\frac{\mathrm{d}\mathbf{g}}{\mathrm{d}\mathbf{x}}(\mathbf{x}) = \mathbf{A}$. Moreover, from part (b), we know that $\frac{\mathrm{d}h}{\mathrm{d}\mathbf{z}}(\mathbf{z}) = 2\mathbf{z}^\top$ (substitute $\mathbf{A} = \mathbf{I}$). Therefore,

$$\frac{\mathrm{d}f}{\mathrm{d}\mathbf{x}}(\mathbf{x}) = \left[\frac{\mathrm{d}h}{\mathrm{d}\mathbf{z}}(\mathbf{A}\mathbf{x} - \mathbf{y})\right] \cdot \left[\frac{\mathrm{d}\mathbf{g}}{\mathrm{d}\mathbf{x}}(\mathbf{x})\right]$$

$$= 2(\mathbf{A}\mathbf{x} - \mathbf{y})^\top \mathbf{A}$$

Therefore $\nabla_{\mathbf{x}} f(\mathbf{x}) = \left[\frac{\mathrm{d}f}{\mathrm{d}\mathbf{x}}(\mathbf{x})\right]^\top = 2\mathbf{A}^\top(\mathbf{A}\mathbf{x} - \mathbf{y})$.

(e) (1 point) Let $\mathbf{u} \in \mathbb{R}^m$, $\mathbf{v} \in \mathbb{R}^n$. Compute the gradient $\nabla_{\mathbf{A}} f(\mathbf{A})$ of

$$f : \mathbb{R}^{m \times n} \to \mathbb{R},\ f(\mathbf{A}) = \mathbf{u}^\top \mathbf{A}\mathbf{v}.$$

Hint: use the cyclic property of trace to write $\mathbf{u}^T\mathbf{A}\mathbf{v} = \mathrm{Tr}\left(\mathbf{u}^T\mathbf{A}\mathbf{v}\right) = \mathrm{Tr}\left(\mathbf{v}\mathbf{u}^T\mathbf{A}\right)$.

Solution: Note that $f(\mathbf{A}) = \mathrm{Tr}(\mathbf{u}^\top\mathbf{A}\mathbf{v}) = \mathrm{Tr}(\mathbf{v}\mathbf{u}^\top\mathbf{A}) = \langle \mathbf{u}\mathbf{v}^\top, \mathbf{A}\rangle$ is linear, therefore $\nabla_{\mathbf{A}} f(\mathbf{A}) = \mathbf{u}\mathbf{v}^\top$.

(f) (3 points) Let $\mathbf{x} \in \mathbb{R}^n$ and $\mathbf{y} \in \mathbb{R}^m$. Compute the gradient $\nabla_{\mathbf{A}} f(\mathbf{A})$ of

$$f : \mathbb{R}^{m \times n} \to \mathbb{R},\ f(\mathbf{A}) = \|\mathbf{A}\mathbf{x} - \mathbf{y}\|_2^2.$$

Hint: $\mathrm{Tr}(\mathbf{A}\mathbf{B}) = \mathrm{Tr}(\mathbf{B}\mathbf{A})$.

Solution: Note that $f = h \circ \mathbf{g}$ where $h(\mathbf{z}) = \mathbf{z}^\top\mathbf{z}$ and $\mathbf{g}(\mathbf{A}) = \mathbf{A}\mathbf{x} - \mathbf{y}$. Note that if we ignore the constant shift by $-\mathbf{y}$, the function $\mathbf{g}(\mathbf{A})$ is already linear. So,

$$\mathbf{g}(\mathbf{A} + \boldsymbol{\Delta}) = \mathbf{g}(\mathbf{A}) + \left[\frac{\mathrm{d}\mathbf{g}}{\mathrm{d}\mathbf{A}}(\mathbf{A})\right]\boldsymbol{\Delta}$$
$$(\mathbf{A} + \boldsymbol{\Delta})\mathbf{x} - \mathbf{y} = \mathbf{A}\mathbf{x} - \mathbf{y} + \left[\frac{\mathrm{d}\mathbf{g}}{\mathrm{d}\mathbf{A}}(\mathbf{A})\right]\boldsymbol{\Delta}$$
$$\boldsymbol{\Delta}\mathbf{x} = \left[\frac{\mathrm{d}\mathbf{g}}{\mathrm{d}\mathbf{A}}(\mathbf{A})\right]\boldsymbol{\Delta}$$

Thus, given a dummy matrix $\mathbf{B} \in \mathbb{R}^{m \times n}$,

$$\begin{aligned}
\left[\frac{\mathrm{d}f}{\mathrm{d}\mathbf{A}}(\mathbf{A})\right](\mathbf{B}) &= \left[\frac{\mathrm{d}h}{\mathrm{d}\mathbf{z}}(\mathbf{A}\mathbf{x} - \mathbf{y})\right] \cdot \left[\frac{\mathrm{d}\mathbf{g}}{\mathrm{d}\mathbf{A}}(\mathbf{A})\right](\mathbf{B}) \\
&= \left[\frac{\mathrm{d}h}{\mathrm{d}\mathbf{z}}(\mathbf{A}\mathbf{x} - \mathbf{y})\right]\mathbf{B}\mathbf{x} \\
&= 2(\mathbf{A}\mathbf{x} - \mathbf{y})^\top\mathbf{B}\mathbf{x} \\
&= 2\,\mathrm{Tr}\left(\mathbf{x}(\mathbf{A}\mathbf{x} - \mathbf{y})^\top\mathbf{B}\right) \\
&= 2\,\mathrm{Tr}\left((\mathbf{x}\mathbf{x}^\top\mathbf{A}^\top - \mathbf{x}\mathbf{y}^\top)\mathbf{B}\right) \\
&= 2\langle \mathbf{A}\mathbf{x}\mathbf{x}^\top - \mathbf{y}\mathbf{x}^\top, \mathbf{B}\rangle \\
&= \langle \nabla_{\mathbf{A}} f(\mathbf{A}), \mathbf{B}\rangle
\end{aligned}$$

Therefore $\nabla_{\mathbf{A}} f(\mathbf{A}) = 2(\mathbf{A}\mathbf{x}\mathbf{x}^\top - \mathbf{y}\mathbf{x}^\top)$.

An alternative approach would be to apply the "best linear approximation" definition of a derivative directly to $f(\mathbf{A})$. Let $\boldsymbol{\Delta} \in \mathbb{R}^{m \times n}$ be a small perturbation. Then,

$$f(\mathbf{A} + \boldsymbol{\Delta}) \approx f(\mathbf{A}) + \left[\frac{\mathrm{d}f}{\mathrm{d}\mathbf{A}}(\mathbf{A})\right]\boldsymbol{\Delta}$$

$$\left\|(\mathbf{A}+\boldsymbol{\Delta})\mathbf{x}-\mathbf{y}\right\|_2^2 \approx \left\|\mathbf{A}\mathbf{x}-\mathbf{y}\right\|_2^2 + \left[\frac{\mathrm{d}f}{\mathrm{d}\mathbf{A}}(\mathbf{A})\right]\boldsymbol{\Delta}$$

$$\mathbf{x}^T(\mathbf{A}+\boldsymbol{\Delta})^T(\mathbf{A}+\boldsymbol{\Delta})\mathbf{x} - 2\mathbf{y}^T(\mathbf{A}+\boldsymbol{\Delta})\mathbf{x} + \mathbf{y}^T\mathbf{y} \approx \mathbf{x}^T\mathbf{A}^T\mathbf{A}\mathbf{x} - 2\mathbf{y}^T\mathbf{A}\mathbf{x} + \mathbf{y}^T\mathbf{y} + \left[\frac{\mathrm{d}f}{\mathrm{d}\mathbf{A}}(\mathbf{A})\right]\boldsymbol{\Delta}$$

$$2\mathbf{x}^T\mathbf{A}^T\boldsymbol{\Delta}\mathbf{x} + \mathbf{x}^T\boldsymbol{\Delta}^T\boldsymbol{\Delta}\mathbf{x} - 2\mathbf{y}^T\boldsymbol{\Delta}\mathbf{x} \approx \left[\frac{\mathrm{d}f}{\mathrm{d}\mathbf{A}}(\mathbf{A})\right]\boldsymbol{\Delta}$$

$$2\,\mathrm{Tr}\Big((\mathbf{x}\mathbf{x}^T\mathbf{A}^T - \mathbf{x}\mathbf{y}^T)\boldsymbol{\Delta}\Big) + \mathbf{x}^T\boldsymbol{\Delta}^T\boldsymbol{\Delta}\mathbf{x} \approx \left[\frac{\mathrm{d}f}{\mathrm{d}\mathbf{A}}(\mathbf{A})\right]\boldsymbol{\Delta}$$

Note that $\mathbf{x}^T\boldsymbol{\Delta}^T\boldsymbol{\Delta}\mathbf{x} = \|\boldsymbol{\Delta}\mathbf{x}\|_2^2$ is a quadratic term that vanishes much quicker than any term linear in $\boldsymbol{\Delta}$ as $\|\boldsymbol{\Delta}\| \to 0$; for small $\boldsymbol{\Delta}$, this term is negligible. Therefore, matching the non-negligible terms, we can see that

$$2\,\mathrm{Tr}\Big((\mathbf{A}\mathbf{x}\mathbf{x}^T - \mathbf{y}\mathbf{x}^T)^T\boldsymbol{\Delta}\Big) = \left[\frac{\mathrm{d}f}{\mathrm{d}\mathbf{A}}(\mathbf{A})\right]\boldsymbol{\Delta} = \langle \nabla_A f(\mathbf{A}), \boldsymbol{\Delta}\rangle$$

Thus, $\nabla_{\mathbf{A}} f(\mathbf{A}) = 2(\mathbf{A}\mathbf{x}\mathbf{x}^T - \mathbf{y}\mathbf{x}^T) = 2(\mathbf{A}\mathbf{x}-\mathbf{y})\mathbf{x}^\top$.

(g) (3 points) Consider the function that maps a vector to its maximum entry, $\mathbf{x} \mapsto \max_i x_i$. While this function is non-smooth, a common trick in machine learning is to use a smooth approximation, *LogSumExp*, defined as follows:

$$\mathrm{LSE} : \mathbb{R}^n \to \mathbb{R},\ \mathrm{LSE}(\mathbf{x}) = \ln\left(\sum_{i=1}^n e^{x_i}\right).$$

One of the nice properties of this function is that it is convex, which can be proved by showing that its Hessian matrix is positive semi-definite. To that end, compute its gradient and Hessian. You do not need to prove that the Hessian is PSD.

Solution: We can first compute the gradient by finding each of the partials by applying the chain rule. Notice that the gradient is actually the softmax function, which we will encounter soon in class.

$$\begin{aligned}\frac{\partial}{\partial x_k}\mathrm{LSE}(\mathbf{x}) &= \frac{1}{\sum_{i=1}^n e^{x_i}} \cdot \frac{\partial}{\partial x_k}\sum_{i=1}^n e^{x_i} \\ &= \frac{e^{x_k}}{\sum_{i=1}^n e^{x_i}} \\ \therefore \nabla_{\mathbf{x}}\mathrm{LSE}(\mathbf{x}) &= \frac{1}{\sum_{i=1}^n e^{x_i}}\begin{bmatrix} e^{x_1} \\ e^{x_2} \\ \vdots \\ e^{x_n}\end{bmatrix}\end{aligned}$$

Now we compute each entry of the Hessian. To that end, notice that the diagonal and off-diagonal entries will have different values:

$$\begin{aligned}
\frac{\partial}{\partial x_l}\frac{\partial}{\partial x_k}\operatorname{LSE}(x) &= \frac{\partial}{\partial x_l}\frac{e^{x_k}}{\sum_{i=1}^n e^{x_i}} \\
&= \frac{1}{\left(\sum_{i=1}^n e^{x_i}\right)^2}\left[\sum_{i=1}^n e^{x_i}\cdot\frac{\partial}{\partial x_l}e^{x_k} - e^{x_k}\cdot\frac{\partial}{\partial x_l}\sum_{i=1}^n e^{x_i}\right] \\
&= \frac{1}{\left(\sum_{i=1}^n e^{x_i}\right)^2}\left[\sum_{i=1}^n e^{x_i}\cdot\frac{\partial}{\partial x_l}e^{x_k} - e^{x_k}\cdot e^{x_l}\right] \\
&= \frac{1}{\sum_{i=1}^n e^{x_i}}\cdot\frac{\partial}{\partial x_l}e^{x_k} - \frac{e^{x_k}}{\sum_{i=1}^n e^{x_i}}\cdot\frac{e^{x_l}}{\sum_{i=1}^n e^{x_i}}
\end{aligned}$$

To simplify this, notice that that the derivative of e^{x_k} with respect to x_l is 0 when $l \neq k$, since it is a constant. Hence the first term is non-zero only along the diagonal. Therefore:

$$\begin{aligned}
\left[\nabla_{\mathbf{x}}^2 \operatorname{LSE}(\mathbf{x})\right]_{kl} &= \begin{cases} \frac{e^{x_k}}{\sum_{i=1}^n e^{x_i}} - \frac{e^{x_k}}{\sum_{i=1}^n e^{x_i}}\cdot\frac{e^{x_k}}{\sum_{i=1}^n e^{x_i}} & : k = l \\ -\frac{e^{x_k}}{\sum_{i=1}^n e^{x_i}}\cdot\frac{e^{x_l}}{\sum_{i=1}^n e^{x_i}} & : k \neq l \end{cases} \\
&= \begin{cases} \frac{e^{x_k}}{\sum_{i=1}^n e^{x_i}} - \left(\frac{e^{x_k}}{\sum_{i=1}^n e^{x_i}}\right)^2 & : k = l \\ -\frac{e^{x_k}}{\sum_{i=1}^n e^{x_i}}\cdot\frac{e^{x_l}}{\sum_{i=1}^n e^{x_i}} & : k \neq l \end{cases}
\end{aligned}$$

If we let $\mathbf{z} = \nabla_{\mathbf{x}} \operatorname{LSE}(\mathbf{x})$, we can compactly express the hessian as the difference between a diagonal matrix and the outer product below.

$$\nabla_{\mathbf{x}}^2 \operatorname{LSE}(\mathbf{x}) = \operatorname{diag}(\mathbf{z}) - \mathbf{z}\mathbf{z}^\top$$

2 Linear Algebra Review (14 points)

1. (3 points) Let $A \in \mathbb{R}^{n \times n}$ be a symmetric matrix. Prove equivalence between these three different definitions of positive semidefiniteness (PSD).

   (a) For all $x \in \mathbb{R}^n$, $x^\top A x \geq 0$.

   (b) All the eigenvalues of A are nonnegative.

   (c) There exists a matrix $U \in \mathbb{R}^{n \times n}$ such that $A = UU^\top$.

   Mathematically, we write positive semidefiniteness as $A \succeq 0$.

2. (5 points) Now that we're equipped with different definitions of positive semidefiniteness, use them to prove the following properties of PSD matrices.

   (a) If A and B are PSD, then $2A + 3B$ is PSD.

   (b) If A is PSD, all diagonal entries of A are nonnegative: $A_{ii} \geq 0, \forall i \in [n]$.

   (c) If A is PSD, the sum of all entries of A is nonnegative: $\sum_{j=1}^{n} \sum_{i=1}^{n} A_{ij} \geq 0$.

   (d) If A and B are PSD, then $\text{Tr}(AB) \geq 0$, where $\text{Tr}(M)$ denotes the *trace* of M.

   (e) If A and B are PSD, then $\text{Tr}(AB) = 0$ if and only if $AB = 0$.

3. (2 points) Let $A \in \mathbb{R}^{n \times n}$ be a symmetric, PSD matrix. Write $\|A\|_F$ as a function of the eigenvalues of A.
   Hint: Recall that $\|A\|_F = \sqrt{\text{Tr}(A^\top A)}$. If you haven't seen this before, you should try to prove it. However, you can accept this as a given fact for this homework assignment.

4. (4 points) Let $A \in \mathbb{R}^{n \times n}$ be a symmetric matrix. Prove that the largest eigenvalue of A is

$$\lambda_{\max}(A) = \max_{\|x\|_2 = 1} x^\top A x$$

Solution:

1. We will prove the following cycle of implications:

   - (a) $\Rightarrow$ (b): Let λ be an eigenvalue of A with corresponding eigenvector v. Then:

$$v^\top A v = \lambda v^\top v = \lambda \|v\|^2$$

     By part (a), we know that $\lambda \|v\|^2 \geq 0$, so $\lambda \geq 0$.

   - (b) $\Rightarrow$ (c): Consider the eigendecomposition of A given by $A = V \Lambda V^\top$, where Λ is a diagonal matrix with entries equal to the eigenvalues of A, $\lambda_1, \ldots, \lambda_n$ and V is orthogonal — this follows from the Spectral Theorem. Define $U := V\sqrt{\Lambda}$, where $\sqrt{\Lambda}$ is diagonal with entries equal to $\sqrt{\lambda_1}, \ldots, \sqrt{\lambda_n}$; notice that this choice is justified because, by assumption, the eigenvalues are non-negative. Clearly, $A = UU^\top$.

- (c) $\Rightarrow$ (a): Let $x \in \mathbb{R}^n$, then:

$$x^\top A x = x^\top U U^\top x = (U^\top x)^\top (U^\top x) = \|U^\top x\|^2 \geq 0.$$

2. (a) $x^\top (2A + 3B)x = 2x^\top Ax + 3x^\top Bx \geq 0$.

   (b) Fix $i \in [n]$. Take $x = e_i$ in the first definition of PSD, where e_i is a canonical vector, i.e. it has zeros everywhere but at coordinate i, where it is equal to 1. Then $e_i^\top A e_i = A_{ii} \geq 0$.

   (c) Take $x = \mathbf{1}$ to be the all-ones vector in the first definition of PSD. Then $\mathbf{1}^\top A \mathbf{1} = \sum_{i=1}^n \sum_{j=1}^n A_{ij} \geq 0$.

   (d) By the third definition of PSD, let $A = UU^\top$ and $B = VV^\top$. Then:

$$\mathrm{Tr}(AB) = \mathrm{Tr}\Big(UU^\top VV^\top\Big) = \mathrm{Tr}\Big(U^\top VV^\top U\Big) = \mathrm{Tr}\Big(U^\top V(U^\top V)^\top\Big) \geq 0,$$

   which follows because $M := U^\top V(U^\top V)^\top$ is PSD by the third definition, and $\mathrm{Tr}(M) \geq 0$ by part (b).

   (e) If $AB = 0$, then clearly $\mathrm{Tr}(AB) = 0$. To prove the other direction, by the third definition of PSD, let $A = UU^\top$ and $B = VV^\top$, for some U and V. Then:

$$\mathrm{Tr}(AB) = \mathrm{Tr}\Big(UU^\top VV^\top\Big) = \mathrm{Tr}\Big(V^\top UU^\top V\Big) = \mathrm{Tr}\Big((U^\top V)^\top U^\top V\Big),$$

   Since $M := (U^\top V)^\top U^\top V$ is PSD, $\mathrm{Tr}(M) = \sum_i \lambda_i(M) = 0$ only if $\lambda_i(M) = 0$ for all $i \in [n]$. From the eigendecomposition of M, it follows that $M = 0$, and moreover this implies $U^\top V = 0$. With this, we have $AB = U(U^\top V)V^\top = U(0)V^\top = 0$.

3. We know the frobenius norm of A can be written as

$$\|A\|_F = \sqrt{\mathrm{Tr}(A^\top A)}$$

   Now, we use that A is PSD, so

$$A^\top A = U\Lambda U^\top U\Lambda U^\top = U\Lambda^2 U^\top$$

   By the cyclic property of trace,

$$\mathrm{Tr}\Big(U\Lambda^2 U^\top\Big) = \mathrm{Tr}\Big(\Lambda^2 U^\top U\Big) = \mathrm{Tr}\Big(\Lambda^2\Big)$$

   We know Λ is a diagonal matrix containing the eigenvalues of A, i.e., $\mathrm{Tr}\Big(\Lambda^2\Big) = \sum \lambda^2$ therefore,

$$\|A\|_F = \sqrt{\sum \lambda^2}$$

4. Let $A = V\mathrm{diag}(\lambda_1, \ldots, \lambda_n)V^\top$ be the eigendecomposition of A given by the spectral theorem. Since $V^\top$ is invertible, for every $y \in \mathbb{R}^n$ there is a $x \in \mathbb{R}^n$ such that $V^\top x = y$, and also V is orthogonal so $\|Vy\|_2 = \|y\|_2$. Therefore:

$$\max_{\|x\|_2=1} x^\top Ax = \max_{\|x\|_2=1} (V^\top x)^\top \mathrm{diag}(\lambda_1, \ldots, \lambda_n)(V^\top x) = \max_{\|Vy\|_2=1} y^\top \mathrm{diag}(\lambda_1, \ldots, \lambda_n)y$$

$$= \max_{\|y\|_2=1} \sum_{i=1}^{n} y_i^2 \lambda_i$$
$$= \lambda_{\max}(A)$$

The last equality follows because in the optimization problem $\max_{\|y\|_2=1} \sum_{i=1}^{n} y_i^2 \lambda_i$, our best choice is to place all weight on the coefficient y_i which corresponds to the largest eigenvalue of A.

Here is another cute solution; first, write out the Lagrangian for this optimization problem.

$$\mathcal{L}(x, \nu) = x^\top A x + \nu(1 - \|x\|_2^2)$$

Now we differentiate with respect to x and ν and set this equal to zero to write out the first-order optimality conditions:

$$\nabla_x \mathcal{L}(x, \nu) = 2Ax - 2\nu x = 0 \implies Ax = \nu x$$
$$\frac{\mathrm{d}}{\mathrm{d}\nu} \mathcal{L}(x, \nu) = 1 - \|x\|_2^2 = 0 \implies \|x\|_2 = 1$$

Notice the first condition tells us that the optimal x^* is an eigenvalue of A, while the second one confirms its norm is 1. This immediately implies that the optimum is a unit eigenvector of A corresponding to its largest eigenvalue. Plugging this in to the objective, we immediately find that it is $\lambda_{\max}$.

3 Probability Potpourri (11 points)

1. (2 points) Recall the covariance of two random variables X and Y is defined as $\text{Cov}(X, Y) = \mathbb{E}[(X - \mathbb{E}[X])(Y - \mathbb{E}[Y])]$. For a multivariate random variable Z (i.e., each index of Z is a random variable), we define the covariance matrix Σ such that $\Sigma_{ij} = \text{Cov}(Z_i, Z_j)$. Concisely, $\Sigma = \mathbb{E}[(Z - \mu)(Z - \mu)^\top]$, where μ is the mean value of the random column vector Z. Prove that the covariance matrix is always positive semidefinite (PSD).
   Hint: Use linearity of expectation.

2. (4 points) The probability that an archer hits her target when it is windy is 0.4; when it is not windy, her probability of hitting the target is 0.7. On any shot, the probability of a gust of wind is 0.3. Find the probability that

   (i) on a given shot there is a gust of wind and she hits her target.

   (ii) she hits the target with her first shot.

   (iii) she hits the target exactly once in two shots.

   (iv) there was no gust of wind on an occasion when she missed.

3. (2 points) An archery target is made of 3 concentric circles of radii $1/\sqrt{3}$, 1 and $\sqrt{3}$ feet. Arrows striking within the inner circle are awarded 4 points, arrows within the middle ring are awarded 3 points, and arrows within the outer ring are awarded 2 points. Shots outside the target are awarded 0 points.

   Consider a random variable X, the distance of the strike from the center (in feet), and let the probability density function of X be

   $$f(x) = \begin{cases} \frac{2}{\pi(1+x^2)} & x > 0 \\ 0 & \text{otherwise} \end{cases}$$

   What is the expected value of the score of a single strike?

4. (3 points) Let $X \sim \text{Pois}(\lambda)$, $Y \sim \text{Pois}(\mu)$. given that $X \perp\!\!\!\perp Y$, derive an expression for $\mathbb{P}(X \mid X + Y = n)$. What well-known probability distribution is this? What are its parameters?

Solution:

1. For any $v \in \mathbb{R}^n$, note that $v^\top \mathbb{E}[(X - \mu)(X - \mu)^\top]v = \Sigma_{i=1}^n \Sigma_{j=1}^n \mathbb{E}[(X_i - \mu_i)(X_j - \mu_j)]v_i v_j = \mathbb{E}[v^\top (X - \mu)(X - \mu)^\top v] = \mathbb{E}[((X - \mu)^\top v)^2] \geq 0$. Note the second identity comes from linearity of expectation.

2. Denote with H the event that she hits her target, and with W the event that there is a gust of wind. Then we know that: $P(H \mid W) = 0.4$, $P(H \mid W^c) = 0.7$ and $P(W) = 0.3$.

   (i) $P(H \cap W) = P(H \mid W)P(W) = 0.12$

   (ii) $P(H) = P(H \mid W)P(W) + P(H \mid W^c)P(W^c) = 0.61$

(iii) This probability is equal to $\binom{2}{1}P(H)P(H^c) = 0.4758$

(iv) $P(W^c \mid H^c) = \frac{P(H^c|W^c)P(W^c)}{P(H^c)} = 0.538$

3. The expected value is

$$\begin{aligned}
&\int_0^{1/\sqrt{3}} 4\frac{2}{\pi(1+x^2)}\,\mathrm{d}x + \int_{1/\sqrt{3}}^{1} 3\frac{2}{\pi(1+x^2)}\,\mathrm{d}x + \int_1^{\sqrt{3}} 2\frac{2}{\pi(1+x^2)}\,\mathrm{d}x \\
= \;& \frac{2}{\pi}\left[4\left(\arctan\frac{1}{\sqrt{3}} - \arctan 0\right) + 3\left(\arctan 1 - \arctan\frac{1}{\sqrt{3}}\right) + 2\left(\arctan\sqrt{3} - \arctan 1\right)\right] \\
= \;& \frac{13}{6}
\end{aligned}$$

4. To derive this conditional distribution, we can write

$$P(X = k|X + Y = n) = \frac{P(X = k \cap X + Y = n)}{P(X + Y = n)}$$

using the definition of conditional probability. The event $X = k \cap X + Y = n$ can equivalently be expressed as $X = k \cap Y = n - k$ and we can express this using that $X \perp\!\!\!\perp Y$, i.e.,

$$\begin{aligned}
P(X = k \cap Y = n - k) &= \frac{e^{-\lambda}\lambda^k}{k!}\frac{e^{-\mu}\mu^{n-k}}{(n-k)!} \\
&= \frac{1}{n!}e^{-(\lambda+\mu)}\binom{n}{k}\lambda^k\mu^{n-k}
\end{aligned}$$

Now, we note that we can use the law of total probability with the above to get an expression for the denominator

$$\begin{aligned}
P(X + Y = n) &= \sum_{k=0}^{n} P(X = k \cap Y = n - k) \\
&= \sum_{k=0}^{n} \frac{1}{n!}e^{-(\lambda+\mu)}\binom{n}{k}\lambda^k\mu^{n-k} \\
&= \frac{1}{n!}e^{-(\lambda+\mu)}\sum_{k=0}^{n}\binom{n}{k}\lambda^k\mu^{n-k} \\
&= \frac{1}{n!}e^{-(\lambda+\mu)}(\lambda+\mu)^n
\end{aligned}$$

where the last equality comes from binomial expansion. Lastly, we plug these in to get

$$P(X = k|X + Y = n) = \binom{n}{k}\frac{\lambda^k\mu^{n-k}}{(\lambda+\mu)^n}$$

This is exactly the PMF for a binomial distribution with parameters n and $p = \frac{\lambda}{\lambda+\mu}$.

4 Gaussian basics (11 points)

The multivariate Gaussian distribution with mean $\mu \in \mathbb{R}^d$ and positive semidefinite covariance $\Sigma \in \mathbb{R}^{d\times d}$, denoted $N(\mu, \Sigma)$, has the probability density function

$$f(x; \mu, \Sigma) = \frac{1}{\sqrt{(2\pi)^d |\Sigma|}} \exp\left\{-\frac{1}{2}(x-\mu)^\top \Sigma^{-1}(x-\mu)\right\}.$$

Here $|\Sigma|$ denotes the determinant of Σ. In this problem, we assume that the covariance Σ is invertible and, therefore, positive definite, although there are multivariate Gaussians with non-invertible covariance matrices. You may use the following facts without proof:

1. The Gaussian pdf integrates to 1:

$$\int_{\mathbb{R}^d} f(x; \mu, \Sigma)\, \mathrm{d}x = \int_{\mathbb{R}^d} \frac{1}{\sqrt{(2\pi)^d |\Sigma|}} \exp\left\{-\frac{1}{2}(x-\mu)^\top \Sigma^{-1}(x-\mu)\right\} \mathrm{d}x = 1$$

2. Change of variables formula: let f be a smooth function from $\mathbb{R}^d \to \mathbb{R}$, $b \in \mathbb{R}^d$, and $A \in \mathbb{R}^{d\times d}$ be an invertible matrix. Then, performing the change of variable $x \mapsto z = Ax + b$,

$$\int_{\mathbb{R}^d} f(x)\, \mathrm{d}x = \int_{\mathbb{R}^d} f(A^{-1}z - A^{-1}b)|A^{-1}|\, \mathrm{d}z.$$

   You don't need to worry about smoothness when applying this fact; rest assured that polynomials, exponentials, and products and compositions of smooth functions are smooth.

(a) (2 points) Let $X \sim N(\mu, \Sigma)$. Show that $\mathbb{E}[X] = \mu$.

Solution: We perform the change of variable $x \mapsto z = x - \mu$.

$$\begin{aligned}
\mathbb{E}X &= \int_{\mathbb{R}^d} \frac{x}{\sqrt{(2\pi)^d |\Sigma|}} \exp\left\{-\frac{1}{2}(x-\mu)^\top \Sigma^{-1}(x-\mu)\right\} \mathrm{d}x \\
&= \int_{\mathbb{R}^d} \frac{\mu + z}{\sqrt{(2\pi)^d |\Sigma|}} \exp\left\{-\frac{1}{2} z^\top \Sigma^{-1} z\right\} \mathrm{d}z \\
&= \int_{\mathbb{R}^d} \mu f(z; 0, \Sigma)\, \mathrm{d}z + \int_{\mathbb{R}^d} z f(z; 0, \Sigma)\, \mathrm{d}z = \mu + 0 = \mu.
\end{aligned}$$

The left integral equals μ because μ is constant and the Gaussian pdf integrates to 1. The right integral integrates to 0 because $zf(z; 0, \Sigma)$ is an odd function; the contributions of z and $-z$ cancel each other out.

(b) (4 points) Show that $\mathrm{Cov}(X) = \Sigma$.

Solution: Using the change of variable $x \mapsto \sqrt{\Sigma^{-1}}(x - \mu)$ (any PSD matrix has a square root),

$$\mathrm{Cov}(X) = \int_{\mathbb{R}^d} (x-\mu)(x-\mu)^\top \frac{1}{\sqrt{(2\pi)^d |\Sigma|}} \exp\left\{-\frac{1}{2}(x-\mu)^\top \Sigma^{-1}(x-\mu)\right\} \mathrm{d}x$$

$$
\begin{aligned}
&= \int_{\mathbb{R}^d} \sqrt{\Sigma} z z^\top \sqrt{\Sigma} \frac{1}{\sqrt{(2\pi)^d}} \exp\left\{\frac{1}{2} z^\top z\right\} \mathrm{d}z \\
&= \mathbb{E}_{Z\sim N(0,I)}[\sqrt{\Sigma} Z Z^\top \sqrt{\Sigma}] = \sqrt{\Sigma} \mathbb{E}_{Z\sim N(0,I)}[Z Z^\top] \sqrt{\Sigma} \\
&= \sqrt{\Sigma} \mathrm{Cov}(Z) \sqrt{\Sigma}.
\end{aligned}
$$

Thus if we can show that $\mathrm{Cov}(Z) = I$ where $Z \sim N(0, I)$, we are done. To that end, for $i \neq j$, we have

$$
\int_{\mathbb{R}^d} z_i z_j \frac{1}{\sqrt{(2\pi)^d}} e^{-\frac{1}{2} z^\top z} \mathrm{d}z = \frac{1}{2\pi} \int_{-\infty}^{\infty} z_i e^{-\frac{1}{2} z_i^2} \mathrm{d}z_i \int_{-\infty}^{\infty} z_j e^{-\frac{1}{2} z_j^2} \mathrm{d}z_j = 0.
$$

$$
\int_{\mathbb{R}^d} z_i^2 \frac{1}{\sqrt{(2\pi)^d}} e^{-\frac{1}{2} z^\top z} \mathrm{d}z = \int_{-\infty}^{\infty} z_i^2 \frac{1}{\sqrt{2\pi}} e^{-\frac{1}{2} z_i^2} \mathrm{d}z_i = \left[-z_i \frac{1}{\sqrt{2\pi}} e^{-\frac{1}{2} z_i^2}\right]_{-\infty}^{\infty} + \int_{-\infty}^{\infty} \frac{1}{\sqrt{2\pi}} e^{-\frac{1}{2} z_i^2} \mathrm{d}z_i = 1
$$

where we used integration by parts in the second equality on the second line. This suffices to prove that $\mathrm{Cov}(Z) = I$. Therefore, $\mathrm{Cov}(X) = \Sigma$.

(c) (2 points) Compute the moment generating function (MGF) of X: $M_X(\lambda) = \mathbb{E}[e^{\lambda^\top X}]$, where $\lambda \in \mathbb{R}^d$. Note: moment generating functions have several interesting and useful properties, one being that M_X characterizes the distribution of X: if $M_X = M_Y$, then X and Y have the same distribution.

Solution: Notice that by completing the square,

$$
-\frac{1}{2}(x-\mu)^\top \Sigma^{-1}(x-\mu) + \lambda^\top x = -\frac{1}{2}(x-\mu-\Sigma\lambda)^\top \Sigma^{-1}(x-\mu-\Sigma\lambda) + \mu^\top \lambda + \frac{1}{2}\lambda^\top \Sigma \lambda.
$$

Therefore,

$$
\begin{aligned}
\mathbb{E}\left[\exp\left\{\lambda^\top X\right\}\right] &= \frac{1}{\sqrt{(2\pi)^d |\Sigma|}} \int_{\mathbb{R}^d} \exp\left\{\lambda^\top x\right\} \exp\left\{-\frac{1}{2}(x-\mu)^\top \Sigma^{-1}(x-\mu)\right\} \mathrm{d}x \\
&= \frac{\exp\left\{\mu^\top \lambda + \frac{1}{2}\lambda^\top \Sigma \lambda\right\}}{\sqrt{(2\pi)^d |\Sigma|}} \int_{\mathbb{R}^d} \exp\left\{-\frac{1}{2}(x-\mu-\Sigma\lambda)^\top \Sigma^{-1}(x-\mu-\Sigma\lambda)\right\} \mathrm{d}x \\
&= \exp\left\{\lambda^\top \mu + \frac{1}{2}\lambda^\top \Sigma \lambda\right\}.
\end{aligned}
$$

(d) (2 points) Using the fact that MGFs determine distributions, given $A \in \mathbb{R}^{k\times d}$, $b \in \mathbb{R}^k$ identify the distribution of $AX + b$ (don't worry about covariance matrices being invertible).

Solution: We compute the MGF of $AX + b$:

$$
\mathbb{E} e^{\lambda^\top (AX+b)} = e^{\lambda^\top b} \mathbb{E} e^{(A^\top \lambda)^\top X} = \exp\left\{\lambda^\top b + \lambda^\top A\mu + \frac{1}{2}\lambda^\top A \Sigma A^\top \lambda\right\}.
$$

Thus, $AX + b \sim N(A\mu + b, A\Sigma A^\top)$. In other words, an affine transformation of a multivariate Gaussian returns another Gaussian.

(e) (1 point) Show that there exists an affine transformation of X that is distributed as the standard multivariate Gaussian, $N(0, I_d)$. (Assume Σ is invertible.)

Solution: Consider $Z = \sqrt{\Sigma^{-1}}(X - \mu)$. Then, by the above part, $Z \sim N(0, I_d)$.

5 NumPy Intro (8 points)

NumPy is a library in Python that allows for efficient computation on matrices and vectors. Given that machine learning's foundations are in linear algebra, this library is widely used in ML research and industry to develop models.

The following questions will help you get your bearings in NumPy, which will be useful for the next questions and future homeworks. You're allowed to look through NumPy documentation and don't need to cite it. You must use a NumPy-based implementation for each of the following questions and it must be vectorized (e.g. you cannot implement dot product by hand instead of using a NumPy primitive like `np.dot` or the `@` operation).

Note: This is the only auto-graded question for this homework. Download `hw1.py` from Edstem and fill out the functions. When done, you may submit to the HW 1 Code assignment on Gradescope. Please do not change the filename or function names, since these are required for the autograder to reference the file and functions correctly.

Hint: The staff solution for each subpart was done in 1-2 lines.

We will primarily use PyTorch for future assignments, but most of the functions we have here will be identical in PyTorch.

(a) (2 points) Implement `special_reshape`, which takes an ndarray with an arbitrary number of dimensions and reduces it to 2 dimensions, so that the first $n-1$ dimensions of the input get combined into the first output dimension, and the last dimension of the input gets preserved in the output. For example, an input ndarray of shape (3, 7, 2, 9) will result in an output ndarray of shape (42, 9). More examples are given in the function signature.

(b) (2 points) Implement `linear`, which takes in an input 1-D ndarray (which we will call vector from now on) x, weight matrix W, and bias vector b. Perform a linear transformation on x using W and b using the formula $y = Wx + b$.

(c) (2 points) Implement `sigmoid`, which takes in an input vector and performs the sigmoid operation on each element. The output ndarray should have the same shape as the input ndarray. Recall that the sigmoid function on a scalar input is:

$$\sigma(x) = \frac{1}{1 + e^{-x}}$$

.

(d) (2 points) Implement `two_layer_nn`, which simulates the forward-propagation of a two-layer neural network, given the weight matrices and bias vectors for layers 1 (W_1 and b_1) and 2 (W_2 and b_2) and an input vector x. For this neural network, you should perform a linear transformation AND sigmoid activation after both layers. Note that you must use `linear` and `sigmoid` in your implementation for this question.

Solution:

(a) Here is the code for part (a).

```
def special_reshape(x: ndarray) -> ndarray:
    last_dim = x.shape[-1]
    return np.reshape(x, (-1, last_dim))
```

(b) Here is the code for part (b).

```
def linear(x: ndarray, W: ndarray, b) -> ndarray:
    result = (W @ x) + b
    return result
```

(c) Here is the code for part (c).

```
def sigmoid(x: ndarray) -> ndarray:
    return 1 / (np.ones_like(x) + np.exp(-x))
```

(d) Here is the code for part (d).

```
def two_layer_nn(x: ndarray, W1: ndarray, W2: ndarray, b1: ndarray, b2: ndarray) -> ndarray:
    hidden = sigmoid(linear(x, W1, b1))
    output = sigmoid(linear(hidden, W2, b2))
    return output
```

6 Isocontours of Normal Distributions (6 points)

Let $f(\mu, \Sigma)$ be the probability density function of a normally distributed random variable in $\mathbb{R}^2$.

(a) (3 points) The spectral theorem allows us to factorize

$$\Sigma = UDU^\top, \quad D = \begin{bmatrix} \lambda_1 & 0 \\ 0 & \lambda_2 \end{bmatrix}, \quad U = \begin{bmatrix} u_1 & u_2 \end{bmatrix} \in \mathbb{R}^{2\times 2},$$

where U has orthonormal columns u_1 and u_2 and D has positive diagonal entries (assume Σ is invertible). Consider the level set

$$S = \{x \in \mathbb{R}^2 : f(x; \mu, \Sigma) = c\},$$

i.e., the set of points $x \in \mathbb{R}^2$ such that the probability density of the Gaussian evaluates to c at those points (c is some value $0 < c < (\sqrt{(2\pi)^d |\Sigma|})^{-1}$). Show that S is an ellipse, and compute the direction of each axis and its semi-length in terms of $u_1, u_2, \lambda_1, \lambda_2$. For background, an ellipse has two perpendicular axes. We consider the direction of an axis to be a unit vector that is a scalar multiple of the vector pointing from the center of the ellipse to either endpoint of the axis. By axis semi-length, we mean half the length of the line segment connecting the axis endpoints. For more info, see `https://en.wikipedia.org/wiki/Ellipse`.

For parts (b) and (c), write code to plot the isocontours of the following functions, each on its own separate figure. Plot at least 5 contours, enough to get a rough sense of the probability density. Default settings of commonly used contour plotting functions probably suffice for this. You are free to use Matplotlib, NumPy, and SciPy.

(b) (2 points) $f(\mu, \Sigma)$, where $\mu = \begin{bmatrix} 1 \\ 1 \end{bmatrix}$ and $\Sigma = \begin{bmatrix} 1 & 0 \\ 0 & 2 \end{bmatrix}$.

(c) (1 point) $f(\mu, \Sigma)$, where $\mu = \begin{bmatrix} -1 \\ 2 \end{bmatrix}$ and $\Sigma = \begin{bmatrix} 2 & 1 \\ 1 & 4 \end{bmatrix}$.

Solution:

(a) First we simplify the level set equation:

$$\begin{aligned} f(x; \mu, \Sigma) = c &\iff \frac{1}{2\pi\sqrt{|\Sigma|}} e^{-\frac{1}{2}(x-\mu)^\top \Sigma^{-1}(x-\mu)} = c \\ &\iff (x-\mu)^\top \Sigma^{-1}(x-\mu) = -2\log(c) - 2\log\left(2\pi\sqrt{|\Sigma|}\right). \end{aligned}$$

Note that $|\Sigma| = \lambda_1\lambda_2$. We can rename $-2\log(c) - 2\log\left(2\pi\sqrt{|\Sigma|}\right)$ to some constant C, so we are left with the equation

$$(x-\mu)^\top \Sigma^{-1}(x-\mu) = C.$$

Let us parameterize $x = \mu + z_1 u_1 + z_2 u_2$, where z_1 and z_2 are scalars. This is possible because $\text{Span}\{u_1, u_2\} = \mathbb{R}^2$ and μ is just some vector offset. We can denote $z = \begin{bmatrix} z_1 \\ z_2 \end{bmatrix}$. Then, our equation becomes

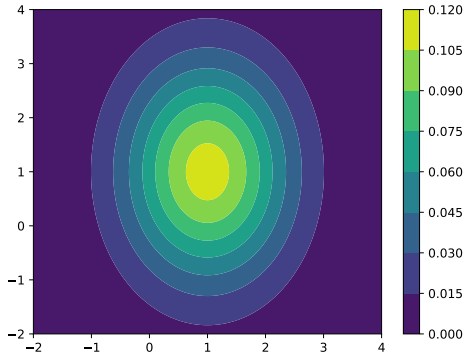
$$z^\top U^\top \Sigma^{-1} U z = z^\top U^\top U D^{-1} U^\top U z = z^\top D^{-1} z = C.$$

This corresponds to the equation

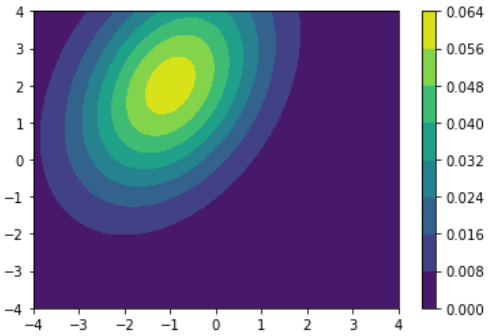
$$\frac{z_1^2}{\lambda_1} + \frac{z_2^2}{\lambda_2} = C.$$

This describes an axis-aligned ellipse in z-space with axis semi-lengths $\sqrt{C\lambda_1}$ and $\sqrt{C\lambda_2}$. To get the points of the level set in the original x-space, we transform each level set point in z-space according to $x = \mu + Uz$. This describes an ellipse in x-space centered at μ with the same semi-lengths, but with axes pointing in the directions of u_1 and u_2.

(b) Isocontours with $\mu = \begin{bmatrix} 1 \\ 1 \end{bmatrix}$ and $\Sigma = \begin{bmatrix} 1 & 0 \\ 0 & 2 \end{bmatrix}$:



(c) Isocontours with $\mu = \begin{bmatrix} -1 \\ 2 \end{bmatrix}$ and $\Sigma = \begin{bmatrix} 2 & 1 \\ 1 & 4 \end{bmatrix}$:



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

def plot_contours():
    fig = plt.figure(figsize=(10,10))
    ax0 = fig.add_subplot(111)
    ax0.contour(rv.pdf(pos).reshape(500,500))
    plt.show()

# Part b

# Generate grid of points at which to evaluate pdf
x = np.linspace(-2, 4, 500)
y = np.linspace(-2, 4, 500)
X,Y = np.meshgrid(x, y)
pos = np.array([Y, X]).T
rv = scipy.stats.multivariate_normal([1, 1], [[1, 0], [0, 2]])
Z = rv.pdf(pos)

plt.contourf(X, Y, Z)
plt.colorbar()
plt.show()

# Part c
x = np.linspace(-4, 4, 500)
y = np.linspace(-4, 4, 500)
X,Y = np.meshgrid(x, y)
pos = np.array([Y, X]).T
rv = scipy.stats.multivariate_normal([-1, 2], [[2, 1], [1, 4]])
Z = rv.pdf(pos)

plt.contourf(X, Y, Z)
plt.colorbar()
plt.show()
```

7 Hands-on with data (10 points)

In the following problem, you will use two simple datasets to walk through the steps of a standard machine learning workflow: inspecting your data, choosing a model, implementing it, and verifying its accuracy. We have provided two datasets in the form of numpy arrays: `dataset_1.npy` and `dataset_2.npy`. You can load each using NumPy's `np.load` method; see https://numpy.org/doc for more information if you are unfamiliar with the numpy library.

Each dataset is a two-column array with the first column consisting of n scalar inputs $X \in \mathbb{R}^{n\times 1}$ and the second column consisting of n scalar labels $Y \in \mathbb{R}^{n\times 1}$. We denote each entry of X and Y with subscripts:

$$X = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix} \qquad Y = \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_n \end{bmatrix}$$

and assume that y_i is a (potentially stochastic) function of x_i.

(a) (2 points) It is often useful to visually inspect your data and calculate simple statistics; this can detect dataset corruptions or inform your method. For both datasets:

(i) Plot the data as a scatter plot.

(ii) Calculate the correlation coefficient between X and Y:

$$\rho_{X,Y} = \frac{\text{Cov}(X, Y)}{\sigma_X \sigma_Y}$$

in which $\text{Cov}(X, Y)$ is the covariance between X and Y and σ_X is the standard deviation of X.

Your solution may make use of the NumPy library only for arithmetic operations, matrix-vector or matrix-matrix multiplications, matrix inversion, and elementwise exponentiation. It may not make use of library calls for calculating means, standard deviations, or the correlation coefficient itself directly.

(b) (1 point) We would like to design a function that can predict y_i given x_i and then apply it to new inputs. This is a recurring theme in machine learning, and you will soon learn about a general-purpose framework for thinking about such problems. As a preview, we will now explore one of the simplest instantiations of this idea using the class of linear functions:

$$\hat{Y} = Xw. \tag{1}$$

The parameters of our function are denoted by $w \in \mathbb{R}$. It is common to denote predicted variants of quantities with a hat, so $\hat{Y}$ is a predicted label whereas Y is a ground truth label.

We would like to find a w^* that minimizes the **squared error** $\mathcal{J}_{\text{SE}}$ between predictions and labels:

$$w^* = \underset{w}{\text{argmin}}\ \mathcal{J}_{\text{SE}}(w) = \underset{w}{\text{argmin}}\ \|Xw - Y\|_2^2.$$

Derive $\nabla_w \mathcal{J}_{\text{SE}}(w)$ and set it equal to 0 to solve for w^*. (Note that this procedure for finding an optimum relies on the convexity of $\mathcal{J}_{\text{SE}}$. You do not need to show convexity here, but it is a useful exercise to convince yourself this is valid.)

(c) (1 point) Your solution w^* should be a function of X and Y. Implement it and report its **mean squared error** (MSE) for **dataset 1**. The mean squared error is the objective $\mathcal{J}_{\text{SE}}$ from part (b) divided by the number of datapoints:

$$\mathcal{J}_{\text{MSE}}(w) = \frac{1}{n}\|Xw - Y\|_2^2.$$

Also visually inspect the model's quality by plotting a line plot of predicted $\hat{y}$ for uniformly-spaced $x \in [0, 10]$. Keep the scatter plot from part (a) in the background so that you can compare the raw data to your linear function. Does the function provide a good fit of the data? Why or why not?

(d) (1 point) We are now going to experiment with constructing new *features* for our model. That is, instead of considering models that are linear in the inputs, we will now consider models that are linear in some (potentially nonlinear) transformation of the data:

$$\hat{Y} = \Phi w = \begin{bmatrix} \phi(x_1)^\top \\ \phi(x_2)^\top \\ \vdots \\ \phi(x_n)^\top \end{bmatrix} w,$$

where $\phi(x_i), w \in \mathbb{R}^m$. Repeat part (c), providing both the mean squared error of your predictor and a plot of its predictions, for the following features on **dataset 1**:

$$\phi(x_i) = \begin{bmatrix} x_i \\ 1 \end{bmatrix}.$$

How do the plotted function and mean squared error compare? (A single sentence will suffice.)

Hint: the general form of your solution for w^* is still valid, but you will now need to use features Φ where you once used raw inputs X.

(e) (1 point) Now consider the quadratic features:

$$\phi(x_i) = \begin{bmatrix} x_i^2 \\ x_i \\ 1 \end{bmatrix}.$$

Repeat part (c) with these features on **dataset 1**, once again providing short commentary on any changes.

(f) (2 points) Repeat parts (c)-(e) with **dataset 2**.

(g) (2 points) Finally, we would like to understand which features Φ provide us with the best model. To that end, you will implement a method known as k-fold cross validation. The following are instructions for this method; deliverables for part (g) are at the end.

(i) Split **dataset 2** randomly into $k = 4$ equal sized subsets. Group the dataset into 4 distinct training / validation splits by denoting each subset as the validation set and the remaining subsets as the training set for that split.

(ii) On each of the 4 training / validation splits, fit linear models using the following 5 polynomial feature sets:

$$\phi_1(x_i) = \begin{bmatrix} x_i \\ 1 \end{bmatrix} \quad \phi_2(x_i) = \begin{bmatrix} x_i^2 \\ x_i \\ 1 \end{bmatrix} \quad \phi_3(x_i) = \begin{bmatrix} x_i^3 \\ x_i^2 \\ x_i \\ 1 \end{bmatrix} \quad \phi_4(x_i) = \begin{bmatrix} x_i^4 \\ x_i^3 \\ x_i^2 \\ x_i \\ 1 \end{bmatrix} \quad \phi_5(x_i) = \begin{bmatrix} x_i^5 \\ x_i^4 \\ x_i^3 \\ x_i^2 \\ x_i \\ 1 \end{bmatrix}$$

This step will produce 20 distinct w^* vectors: one for each dataset split and featurization ϕ_j.

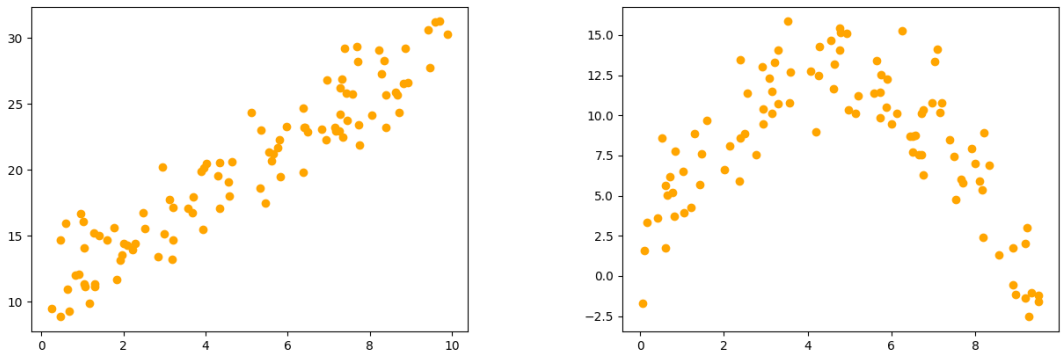
(iii) For each feature set ϕ_j, average the training and validation mean squared errors over all training splits.

It is worth thinking about what this extra effort has bought us: by splitting the dataset into subsets, we were able to use all available datapoints for model fitting while still having held-out datapoints for evaluation for any particular model.

Deliverables for part (g): Plot the training mean squared error and the validation mean squared error on the same plot as a function of the largest exponent in the feature set. Use a log scale for the y-axis. Which model does the training mean squared error suggest is best? Which model does the validation mean squared error suggest is best?

Solution:

(a) Correlation coefficient of dataset 1: 0.939; dataset 2: −0.179.



(a) (b)

Figure 1: Dataset plots

(b)

$$\begin{aligned}\nabla_w \|\hat{Y} - Y\|_2^2 &= \nabla_w \frac{1}{n} \|Xw - Y\|_2^2 \\ &= \nabla_w (Xw - Y)^\top (Xw - Y) \\ &= \nabla_w w^\top X^\top X w - 2w^\top X^\top Y + Y^\top Y \\ &= 2\left(X^\top X w - X^\top Y\right).\end{aligned}$$

Setting the gradient equal to zero and solving for w^* gives:

$$\begin{aligned}2\left(X^\top X w^* - X^\top Y\right) &= 0 \\ \Rightarrow X^\top X w^* &= X^\top Y \\ \Rightarrow w^* &= (X^\top X)^{-1} X^\top Y.\end{aligned}$$

(c) MSE: 32.027. The fit is poor because our parameterization does not have an offset, so the function must pass through the origin even though the data do not.

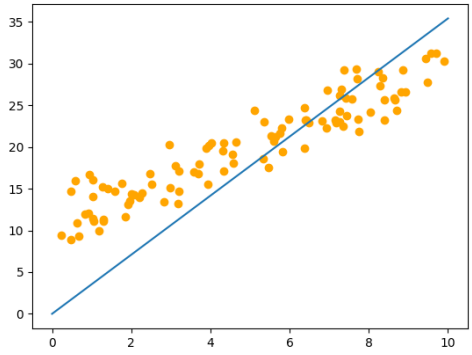


Figure 2: Dataset 1; raw features

(d) MSE: 4.020. The fit is now much better (both quantitatively and visually) because the function has an offset.

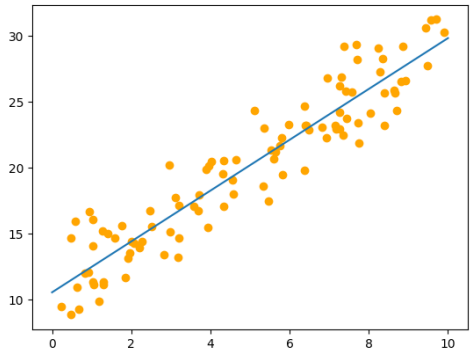


Figure 3: Dataset 1; offset features

(e) MSE: 4.009. There are no new meaningful changes because the function class is already expressive enough to capture the deterministic parts of our the data. (**Also acceptable:** we are now adding unneeded features that increase the expressivity of the function class, so our function is beginning to overfit slightly.)

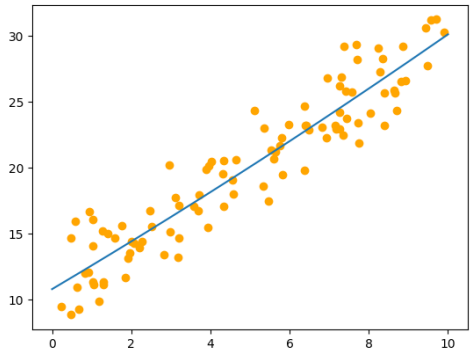


Figure 4: Dataset 1; quadratic features

(f) MSE for raw features: 42.556; for offset features: 19.807; for quadratic features: 4.287. Because the underlying data is quadratic, we now need the second-order features to fit the data well.

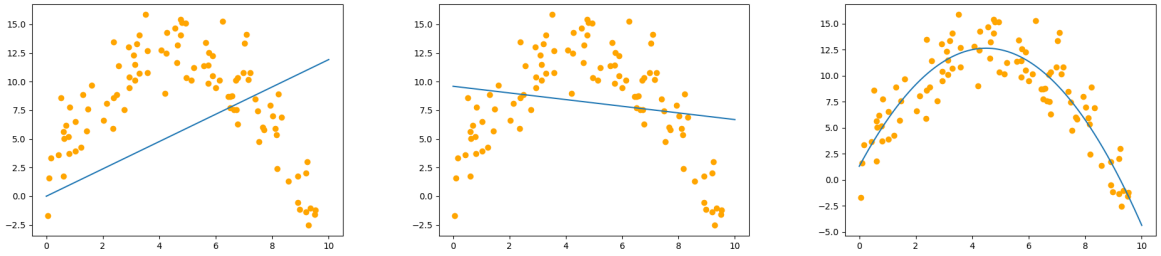


Figure 5: Dataset 2; raw, offset, and quadratic features

(g) The training MSE decreases with more features, so suggests that the quintic features are best. The validation MSE is lowest when the model class matches the underlying data-generating process, so suggests that the quadratic features are best. The validation MSE trend suggests that higher-order features lead to overfitting.

Grading note: There is more stochasticity in this solution because of the randomness introduced by dataset splitting. As long as the answer mentions training loss improving more than validation loss with more features, the answer should be counted as correct.

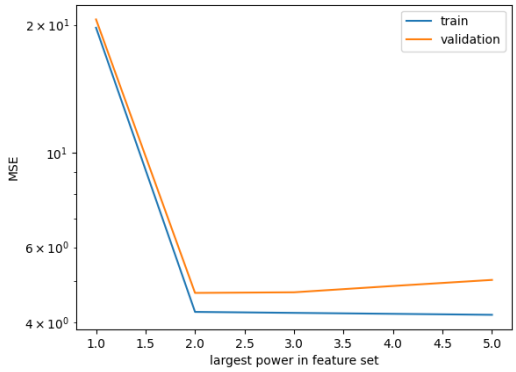


Figure 6: 4-fold cross-validation

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

def plot(X, Y, X_linspace=None, Y_linspace=None, savepath=''):
    plt.clf()
    plt.plot()
    plt.scatter(X, Y, c='orange')
    if X_linspace is not None and Y_linspace is not None:
        plt.plot(X_linspace, Y_linspace)
    plt.savefig(savepath)


def cov_fn(X, Y):
    assert len(X) == len(Y)
    X_mean = X.sum() / len(X)
    Y_mean = Y.sum() / len(Y)
    cov = ((X - X_mean) * (Y - Y_mean)).sum() / (len(X) - 1)
    return cov

def corrcoef(X, Y, validate=True):
    X, Y = X.squeeze(), Y.squeeze()
    numer = cov_fn(X, Y)
    denom = (cov_fn(X, X) * cov_fn(Y, Y))**.5
    corr = numer / denom

    if validate:
        corr_ = np.corrcoef(X, Y)[0,1]
        assert np.isclose(corr, corr_)
    return corr

def lstsq(A, b, validate=True):
    w = np.linalg.inv(A.T @ A) @ A.T @ b
    if validate:
        w_, *_ = np.linalg.lstsq(A, b, rcond=None)
        assert np.allclose(w, w_)
    return w

def mse_fn(preds, targets, validate=True):
    mse = ((preds - targets)**2).sum() / len(preds)
    if validate:
        mse_ = np.linalg.norm(preds - targets, 2)**2 / len(preds)
        assert np.isclose(mse, mse_)
    return mse

def make_poly_features(X, p=2):
```

```
    phi = np.concatenate([
        X**power for power in range(p+1)
    ], axis=-1)
    return phi

def k_fold(X, Y, k):
    assert len(X) == len(Y)
    split_size = len(X) // k
    indices = np.arange(len(X))
    ## shuffle in place
    np.random.shuffle(indices)
    for i in range(k):
        val_inds = indices[np.arange(split_size * i, split_size * (i+1))]
        train_inds = np.setdiff1d(np.arange(len(X)), val_inds)
        yield X[train_inds], Y[train_inds], X[val_inds], Y[val_inds]


featurizers = [
    lambda x: x,  # raw features
    functools.partial(make_poly_features, p=1),  # offset
    functools.partial(make_poly_features, p=2),  # quadratic
]

X_linspace = np.linspace(0, 10, 1000)[:, None]

for i in range(2):

    with open(f'dataset_{i}.npy', 'rb') as f:
        Z = np.load(f)
    X, Y = Z[:,0:1], Z[:,1:2]

    ## plot raw data
    plot(X, Y, savepath=f'{i}_data.png')

    ## quick statistics
    corr = corrcoef(X, Y)
    print(f'Dataset {i} | Correlation coefficient: {corr:4f}')

    for j, featurizer in enumerate(featurizers):
        phi = featurizer(X)
        w = lstsq(phi, Y)

        preds = phi @ w
        mse = mse_fn(preds, Y)
        print(f'Dataset {i} | featurizer {j} | MSE: {mse:.4f}')

        ## use linspace for plotting
        phi_linspace = featurizer(X_linspace)
        preds_linspace = phi_linspace @ w
        plot(X, Y, X_linspace, preds_linspace, f'{i}_preds_{j}.png')

    print()

## cross validation on dataset 2

featurizers = [functools.partial(make_poly_features, p=p) for p in range(1, 6)]

k = 4
plot_train = []
plot_val = []
for j, featurizer in enumerate(featurizers):
    mse_trains = []
    mse_vals = []
    for X_train, Y_train, X_val, Y_val in k_fold(X, Y, k=k):
        phi_train = featurizer(X_train)
        phi_val = featurizer(X_val)
        w = lstsq(phi_train, Y_train)
```

```
        preds_train = phi_train @ w
        preds_val = phi_val @ w

        mse_train = mse_fn(preds_train, Y_train)
        mse_val = mse_fn(preds_val, Y_val)

        mse_trains.append(mse_train)
        mse_vals.append(mse_val)

    mse_train_mean = np.mean(mse_trains)
    mse_val_mean = np.mean(mse_vals)
    plot_train.append(mse_train_mean)
    plot_val.append(mse_val_mean)

    print(f'featurizer {j} | split {k} | MSE train: {mse_train_mean:.4f} | MSE val: {mse_val_mean:.4f}')


plt.clf()
plt.plot(np.arange(1, 6), plot_train, label='train')
plt.plot(np.arange(1, 6), plot_val, label='validation')

plt.yscale('log')
plt.legend()
plt.ylabel('MSE')
plt.xlabel('largest power in feature set')
plt.savefig('cross_validation.png')
```

A Appendix

This appendix contains many ways to manipulate block matrices. Since each fact in here is something you can derive yourself using definitions (e.g. of matrix multiplication), you may use any of them without proof.

A.1 Transposes of Block Matrices

$$\begin{bmatrix} \mathbf{x}_1 & \cdots & \mathbf{x}_n \end{bmatrix}^\top = \begin{bmatrix} \mathbf{x}_1^\top \\ \vdots \\ \mathbf{x}_n^\top \end{bmatrix}$$

$$\begin{bmatrix} \mathbf{x}_1^\top \\ \vdots \\ \mathbf{x}_n^\top \end{bmatrix} = \begin{bmatrix} \mathbf{x}_1 & \cdots & \mathbf{x}_n \end{bmatrix}^\top$$

$$\begin{bmatrix} \mathbf{A} & \mathbf{B} \end{bmatrix}^\top = \begin{bmatrix} \mathbf{A}^\top \\ \mathbf{B}^\top \end{bmatrix}$$

$$\begin{bmatrix} \mathbf{A} \\ \mathbf{B} \end{bmatrix}^\top = \begin{bmatrix} \mathbf{A}^\top & \mathbf{B}^\top \end{bmatrix}$$

$$\begin{bmatrix} \mathbf{A} & \mathbf{B} \\ \mathbf{C} & \mathbf{D} \end{bmatrix}^\top = \begin{bmatrix} \mathbf{A}^\top & \mathbf{C}^\top \\ \mathbf{B}^\top & \mathbf{D}^\top \end{bmatrix}$$

A.2 Block Matrix Products

In the following, $\mathbf{e}_i$ is the i^{th} standard basis vector – it has a 1 in the i^{th} coordinate and 0 in all other coordinates.

$$\begin{bmatrix} \mathbf{x}_1 & \cdots & \mathbf{x}_n \end{bmatrix} \begin{bmatrix} \mathbf{y}_1^\top \\ \vdots \\ \mathbf{y}_n^\top \end{bmatrix} = \sum_{i=1}^{n} \mathbf{x}_i \mathbf{y}_i^\top$$

$$\begin{bmatrix} \mathbf{x}_1^\top \\ \vdots \\ \mathbf{x}_n^\top \end{bmatrix} \begin{bmatrix} \mathbf{y}_1 & \cdots & \mathbf{y}_n \end{bmatrix} = \begin{bmatrix} \mathbf{x}_1^\top \mathbf{y}_1 & \cdots & \mathbf{x}_1^\top \mathbf{y}_n \\ \vdots & \ddots & \vdots \\ \mathbf{x}_n^\top \mathbf{y}_1 & \cdots & \mathbf{x}_n^\top \mathbf{y}_n \end{bmatrix}$$

$$\mathbf{e}_i^\top \begin{bmatrix} \mathbf{x}_1^\top \\ \vdots \\ \mathbf{x}_n^\top \end{bmatrix} = \mathbf{x}_i^\top$$

$$\begin{bmatrix} \mathbf{x}_1 & \cdots & \mathbf{x}_n \end{bmatrix} \mathbf{e}_i = \mathbf{x}_i$$

$$\mathbf{A} \begin{bmatrix} \mathbf{x}_1 & \cdots & \mathbf{x}_n \end{bmatrix} = \begin{bmatrix} \mathbf{A}\mathbf{x}_1 & \cdots & \mathbf{A}\mathbf{x}_n \end{bmatrix}$$

$$\begin{bmatrix} \mathbf{x}_1^\top \\ \vdots \\ \mathbf{x}_n^\top \end{bmatrix} \mathbf{A} = \begin{bmatrix} \mathbf{x}_1^\top \mathbf{A} \\ \vdots \\ \mathbf{x}_n^\top \mathbf{A} \end{bmatrix}$$

$$\mathbf{A} \begin{bmatrix} \mathbf{B} & \mathbf{C} \end{bmatrix} = \begin{bmatrix} \mathbf{AB} & \mathbf{AC} \end{bmatrix}$$

$$\begin{bmatrix} \mathbf{A} \\ \mathbf{B} \end{bmatrix} \mathbf{C} = \begin{bmatrix} \mathbf{AC} \\ \mathbf{BC} \end{bmatrix}$$

A.3 Block Diagonal Matrices

$$\begin{bmatrix} d_1 & 0 & \cdots & 0 \\ 0 & d_2 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & d_n \end{bmatrix} \begin{bmatrix} \mathbf{x}_1^\top \\ \mathbf{x}_2^\top \\ \vdots \\ \mathbf{x}_n^\top \end{bmatrix} = \begin{bmatrix} d_1 \mathbf{x}_1^\top \\ \vdots \\ d_n \mathbf{x}_n^\top \end{bmatrix}$$

$$\begin{bmatrix} \mathbf{A}_1 & 0 & \cdots & 0 \\ 0 & \mathbf{A}_2 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & \mathbf{A}_n \end{bmatrix} \begin{bmatrix} \mathbf{B}_1 \\ \mathbf{B}_2 \\ \vdots \\ \mathbf{B}_n \end{bmatrix} = \begin{bmatrix} \mathbf{A}_1 \mathbf{B}_1 \\ \mathbf{A}_2 \mathbf{B}_2 \\ \vdots \\ \mathbf{A}_n \mathbf{B}_n \end{bmatrix}$$

A.4 Quadratic Forms

$$\mathbf{x}^\top \mathbf{A} \mathbf{y} = \sum_i \sum_j a_{ij} x_i y_j$$

$$\begin{bmatrix} \mathbf{x} \\ \mathbf{y} \end{bmatrix}^\top \begin{bmatrix} \mathbf{A} & \mathbf{B} \\ \mathbf{C} & \mathbf{D} \end{bmatrix} \begin{bmatrix} \mathbf{x} \\ \mathbf{y} \end{bmatrix} = \mathbf{x}^\top \mathbf{A} \mathbf{x} + \mathbf{x}^\top \mathbf{B} \mathbf{y} + \mathbf{y}^\top \mathbf{C} \mathbf{x} + \mathbf{y}^\top \mathbf{D} \mathbf{y}.$$

Contributors:

- Aryan Jain
- Druv Pai