Machine Learning Basics

Instructors: Mohammadreza A. Dehaqani, Babak n. Arabi, Mostafa Tavassolipour

Saman Davachi Tousi, Paria Pasehvarz

Fall 2025

Homework 2

Question 1: Parzen Windows

Let $p(x) \sim U(0, a)$ be uniform from 0 to a, and let a Parzen window be defined as

$$\phi(x) = e^{-x} \quad \text{for } x > 0 \quad \text{and} \quad 0 \text{ for } x \leq 0.$$

1. Show that the mean of such a Parzen-window estimate is given by:

$$\bar{p}_n(x) = \begin{cases} 0, & x < 0, \\ \frac{1}{a}\left(1 - e^{-x/h_n}\right), & 0 \leq x \leq a, \\ \frac{1}{a}\left(e^{a/h_n} - 1\right) e^{-x/h_n}, & a \leq x. \end{cases}$$

2. Plot $\bar{p}_n(x)$ versus x for $a = 1$ and $h_n = 1,\ 0.25,$ and 0.0625.
3. How small does h_n have to be to have less than 1% bias over 99% of the range $0 < x < a$?
4. Find h_n for this condition if $a = 1$, and plot $\bar{p}_n(x)$ in the range $0 \leq x \leq 0.05$.

Solution:

1) Mean of the Parzen-window estimate

By linearity and i.i.d. sampling,

$$\bar{p}_n(x) \equiv \mathbb{E}[\hat{p}_n(x)] = \frac{1}{h_n}\,\mathbb{E}\left[\phi\left(\frac{x - X}{h_n}\right)\right] = \frac{1}{h_n}\int_{-\infty}^{\infty} p(t)\,\phi\left(\frac{x - t}{h_n}\right) dt.$$

Since $p(t) = \frac{1}{a}\mathbf{1}_{[0,a]}(t)$,

$$\bar{p}_n(x) = \frac{1}{ah_n}\int_0^a \phi\left(\frac{x - t}{h_n}\right) dt.$$

Because $\phi(u) = e^{-u}\mathbf{1}_{(0,\infty)}(u)$, we have

$$\phi\left(\frac{x - t}{h_n}\right) = e^{-(x-t)/h_n}\,\mathbf{1}_{\{t<x\}}.$$

Hence the integrand is nonzero only for $t < x$, so the effective upper limit is $m = \min\{a, x\}$:

$$\bar{p}_n(x) = \frac{1}{ah_n}\int_0^m e^{-(x-t)/h_n}\,dt = \frac{e^{-x/h_n}}{ah_n}\int_0^m e^{t/h_n}\,dt.$$

Compute the integral:

$$\int_0^m e^{t/h_n}\,dt = h_n\left(e^{m/h_n} - 1\right),$$

so

$$\bar{p}_n(x) = \frac{1}{a}e^{-x/h_n}\left(e^{m/h_n} - 1\right).$$

Now split by cases.

Case 1: $x < 0$. Then $m = \min\{a, x\} = x < 0$, and the integration region is empty (since $t \in [0, a]$), so

$$\bar{p}_n(x) = 0.$$

Case 2: $0 \le x \le a$. Then $m = x$, hence

$$\bar{p}_n(x) = \frac{1}{a} e^{-x/h_n} \left(e^{x/h_n} - 1 \right) = \frac{1}{a} \left(1 - e^{-x/h_n} \right).$$

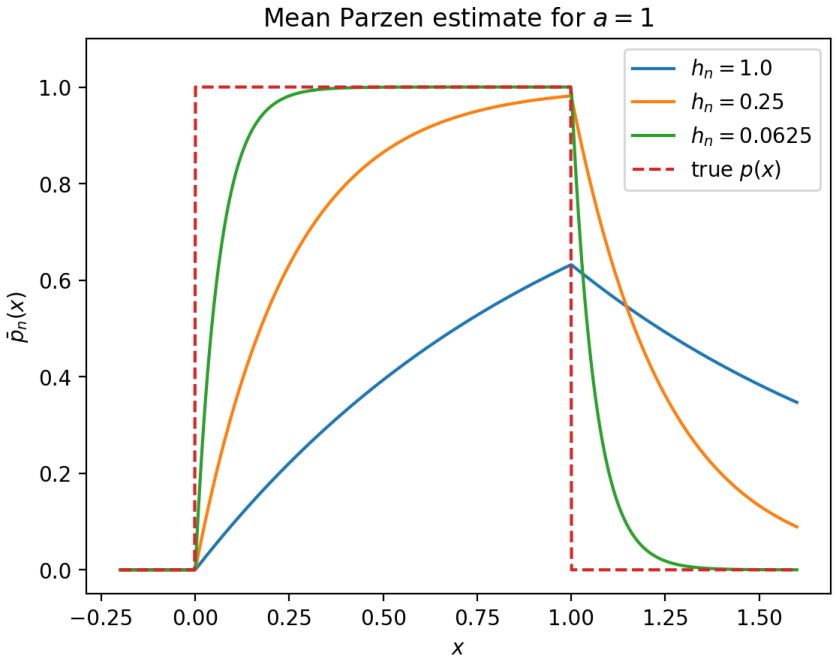
Case 3: $x \ge a$. Then $m = a$, hence

$$\bar{p}_n(x) = \frac{1}{a} e^{-x/h_n} \left(e^{a/h_n} - 1 \right).$$

Therefore,

$$\bar{p}_n(x) = \begin{cases} 0, & x < 0, \\ \frac{1}{a} \left(1 - e^{-x/h_n} \right), & 0 \le x \le a, \\ \frac{1}{a} \left(e^{a/h_n} - 1 \right) e^{-x/h_n}, & x \ge a. \end{cases}$$

2) Plot $\bar{p}_n(x)$ versus x for $a = 1$ and $h_n \in \{1, 0.25, 0.0625\}$



3) How small must h_n be for $< 1\%$ bias over 99% of $0 < x < a$?

For interior points $0 < x < a$, the true density is $p(x) = \frac{1}{a}$ and

$$\bar{p}_n(x) = \frac{1}{a} \left(1 - e^{-x/h_n} \right).$$

So the bias is

$$\text{Bias}(x) = \bar{p}_n(x) - p(x) = -\frac{1}{a} e^{-x/h_n},$$

and the *relative* bias magnitude (relative to $p(x) = 1/a$) is

$$\frac{|\text{Bias}(x)|}{p(x)} = \frac{(1/a) e^{-x/h_n}}{1/a} = e^{-x/h_n}.$$

The bias is largest near $x = 0$. Interpreting "over 99% of the range $0 < x < a$" as the sub-interval $x \in [0.01a,\, a]$, we require

$$e^{-x/h_n} \le 0.01 \quad \text{for all } x \in [0.01a, a].$$

The worst case is at the smallest x, namely $x = 0.01a$, so

$$e^{-0.01a/h_n} \le 0.01 \iff -\frac{0.01a}{h_n} \le \ln(0.01) = -\ln(100) \iff \frac{0.01a}{h_n} \ge \ln(100).$$

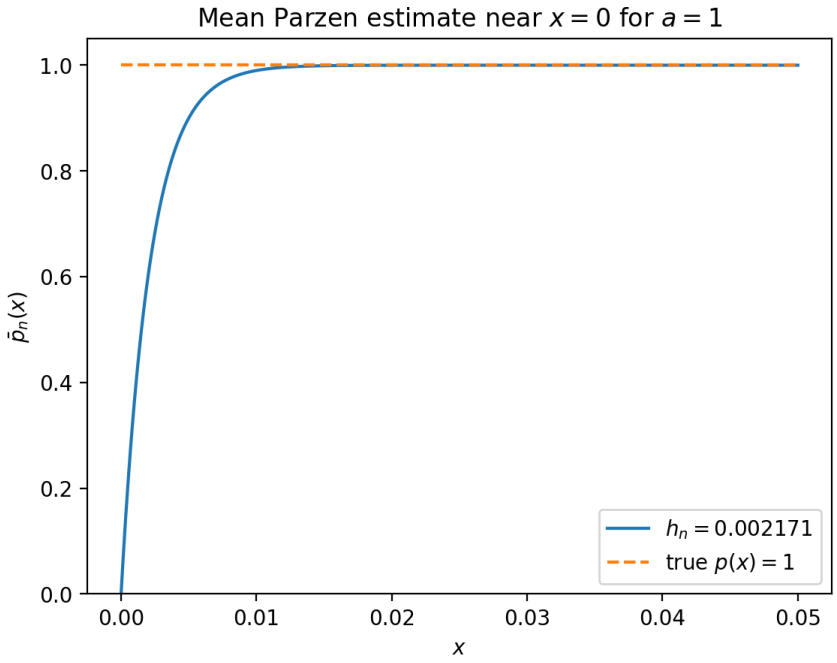
Thus,

$$\boxed{h_n \le \frac{0.01a}{\ln(100)} \approx 0.00217147\, a.}$$

4) Find h_n for $a = 1$ and plot $\bar{p}_n(x)$

on $0 \le x \le 0.05$ With $a = 1$,

$$\boxed{h_n \le \frac{0.01}{\ln(100)} \approx 0.0021714724.}$$



Question 2: Shift of the Eigenvalue Spectrum

In this problem, analyzing the regularization mechanism of ridge regression will serve as an excuse to do some linear algebra.

1. Let $A, B \in \mathbb{R}^{n\times n}$ be symmetric matrices. Assume that $\xi \in \mathbb{R}^n$ is an eigenvector for *both* matrices, with eigenvalues λ_A, λ_B respectively. Please show that ξ is an eigenvector of $A+B$. What is the corresponding eigenvalue?

2. Now consider the ridge regression solution

$$\hat{\beta}^{\text{ridge}} = (X^\top X + \lambda I)^{-1} X^\top y.$$

   Please use the result to explain why computing $\hat{\beta}^{\text{ridge}}$ is more stable than $\hat{\beta}$, i.e., why the solution can be reliably computed (for a suitable λ) even if $X^\top X$ is numerically singular.

3. Compare linear regression and ridge regression in terms of the bias–variance trade-off.

Solution:

1) Common eigenvector of A and B implies eigenvector of $A + B$

Let $A, B \in \mathbb{R}^{n\times n}$ be symmetric. Assume $\xi \in \mathbb{R}^n$ is an eigenvector of both:

$$A\xi = \lambda_A \xi, \qquad B\xi = \lambda_B \xi.$$

Then

$$(A + B)\xi = A\xi + B\xi = \lambda_A \xi + \lambda_B \xi = (\lambda_A + \lambda_B)\xi.$$

Hence ξ is an eigenvector of $A + B$, with corresponding eigenvalue

$$\boxed{\lambda_{A+B} = \lambda_A + \lambda_B.}$$

(Notice symmetry is not needed for this particular statement; it is included because in later parts we rely on real eigenvalues and orthogonal diagonalization of symmetric matrices.)

2) Why ridge is more numerically stable than OLS

The ridge solution is

$$\hat{\beta}^{\text{ridge}} = (X^\top X + \lambda I)^{-1} X^\top y, \qquad \lambda > 0.$$

Since $X^\top X$ is symmetric positive semidefinite, it admits an orthonormal eigen-decomposition

$$X^\top X = Q\Lambda Q^\top, \quad Q^\top Q = I, \quad \Lambda = \text{diag}(\mu_1, \ldots, \mu_p),\ \mu_i \geq 0.$$

The identity matrix shares *every* eigenvector with $X^\top X$ and has eigenvalue 1 for each eigenvector. By Part (1), each eigenvector q_i of $X^\top X$ is also an eigenvector of $X^\top X + \lambda I$ with eigenvalue $\mu_i + \lambda$:

$$(X^\top X + \lambda I)q_i = (\mu_i + \lambda)q_i.$$

Equivalently,

$$X^\top X + \lambda I = Q(\Lambda + \lambda I)Q^\top, \qquad \Lambda + \lambda I = \text{diag}(\mu_1 + \lambda, \ldots, \mu_p + \lambda).$$

Therefore, the inverse (when $\lambda > 0$) is

$$(X^\top X + \lambda I)^{-1} = Q(\Lambda + \lambda I)^{-1} Q^\top, \qquad (\Lambda + \lambda I)^{-1} = \text{diag}\left(\frac{1}{\mu_1 + \lambda}, \ldots, \frac{1}{\mu_p + \lambda}\right).$$

Key stability point (spectrum shift).

If $X^\top X$ is singular, then some $\mu_i = 0$, so $(X^\top X)^{-1}$ does not exist and the OLS normal-equation system

$$X^\top X\, \hat{\beta} = X^\top y$$

is ill-posed / numerically unstable. Ridge adds λI, shifting *every* eigenvalue by $+\lambda$:

$$\mu_i \mapsto \mu_i + \lambda \geq \lambda > 0,$$

so $X^\top X + \lambda I$ is strictly positive definite and invertible. Moreover, its condition number becomes

$$\kappa(X^\top X + \lambda I) = \frac{\mu_{\max} + \lambda}{\mu_{\min} + \lambda},$$

which is finite even if $\mu_{\min} = 0$, in which case

$$\kappa(X^\top X + \lambda I) = \frac{\mu_{\max} + \lambda}{\lambda}.$$

Thus, the inversion is well-defined and numerically more stable for suitable $\lambda > 0$.

3) Linear regression vs. ridge: bias–variance trade-off

Assume the standard linear model

$$y = X\beta^\star + \varepsilon, \qquad \mathbb{E}[\varepsilon] = 0, \quad \text{Var}(\varepsilon) = \sigma^2 I.$$

OLS (linear regression). When $X^\top X$ is invertible,

$$\hat{\beta}^{\text{ols}} = (X^\top X)^{-1} X^\top y.$$

It is unbiased:

$$\mathbb{E}[\hat{\beta}^{\text{ols}}] = \beta^\star.$$

Its variance is

$$\text{Var}(\hat{\beta}^{\text{ols}}) = \sigma^2 (X^\top X)^{-1}.$$

If $X^\top X$ has small eigenvalues (nearly singular), then $(X^\top X)^{-1}$ has large eigenvalues, inflating variance and making estimates unstable.

Ridge regression.

$$\hat{\beta}^{\text{ridge}} = (X^\top X + \lambda I)^{-1} X^\top y.$$

Its expectation is

$$\mathbb{E}[\hat{\beta}^{\text{ridge}}] = (X^\top X + \lambda I)^{-1} X^\top X \, \beta^\star \neq \beta^\star \quad (\lambda > 0),$$

so ridge is biased. The bias vector is

$$\text{Bias}(\hat{\beta}^{\text{ridge}}) = \mathbb{E}[\hat{\beta}^{\text{ridge}}] - \beta^\star = \Big((X^\top X + \lambda I)^{-1} X^\top X - I \Big) \beta^\star = -\lambda (X^\top X + \lambda I)^{-1} \beta^\star.$$

The variance is

$$\text{Var}(\hat{\beta}^{\text{ridge}}) = \sigma^2 (X^\top X + \lambda I)^{-1} X^\top X (X^\top X + \lambda I)^{-1}.$$

Using $X^\top X = Q \Lambda Q^\top$, write $\beta^\star = Q\theta$ and note

$$\hat{\beta}^{\text{ridge}} = Q \, \text{diag}\left(\frac{\mu_i}{\mu_i + \lambda} \right) Q^\top \beta^\star \; + \; \text{noise term}.$$

So ridge shrinks components in directions with small μ_i via the *shrinkage factors*

$$s_i(\lambda) = \frac{\mu_i}{\mu_i + \lambda} \in [0, 1).$$

- As λ increases, $s_i(\lambda)$ decreases $\Rightarrow$ **more bias** (stronger shrinkage toward 0).
- As λ increases, denominators $\mu_i + \lambda$ grow $\Rightarrow$ **less variance** (noise is damped, especially along small-μ_i directions).

Thus ridge trades a controlled increase in bias for a potentially large reduction in variance; this often reduces mean squared error

$$\text{MSE} = \|\text{Bias}\|^2 + \text{trace}(\text{Var})$$

in ill-conditioned or high-dimensional settings (p large, multicollinearity, or near singular $X^\top X$), which is the practical reason ridge can generalize better and be computed more reliably than OLS.

Question 3: Optimizing a Kernel

Kernel-based regression techniques are similar to nearest-neighbor regressors: rather than fit a parametric model, they predict values for new data points by interpolating values from existing points in the training set. In this problem, we will consider a kernel-based regressor of the form

$$f(x^*) = \frac{\sum_n K(x_n, x^*) y_n}{\sum_n K(x_n, x^*)},$$

where (x_n, y_n) are the training data points, and $K(x, x')$ is a kernel function that defines the similarity between two inputs x and x'. Assume that each x_i is represented as a row vector, i.e., a $1 \times D$ vector where D is the number of features for each data point. A popular choice of kernel is a function that decays as the distance between the two points increases, such as

$$K(x, x') = \exp\big(-\|x - x'\|_2^2\big) = \exp\big(-(x - x')(x - x')^\top\big).$$

However, the squared Euclidean distance $\|x - x'\|_2^2$ may not always be the right choice. In this problem, we will consider optimizing over squared Mahalanobis distances

$$K(x, x') = \exp\big(-(x - x')W(x - x')^\top\big),$$

where W is a symmetric $D \times D$ matrix. Intuitively, introducing the weight matrix W allows for different dimensions to matter differently when defining similarity.

1. Let $\{(x_n, y_n)\}_{n=1}^N$ be our training data set. Suppose we are interested in minimizing the residual sum of squares. Write down this loss over the training data $L(W)$ as a function of W.

   Hint: For each point (x_i, y_i) in the training dataset, consider for which subset of training data you should sum kernel function K over for each $f(x_i)$.

2. In the following, let us assume that $D = 2$. That means that W has three parameters: W_{11}, W_{22}, and $W_{12} = W_{21}$. Expand the formula for the loss function to be a function of these three parameters.

Solution:

1) Residual-sum-of-squares loss over the training set

To avoid the trivial "self-fit" for training points (since $K(x_i, x_i) = 1$), we use a leave-one-out (LOO) prediction for each training input x_i:

$$\hat{y}_i(W) \;=\; f_{-i}(x_i) \;=\; \frac{\sum\limits_{\substack{n=1\\ n\neq i}}^{N} K(x_n, x_i)\, y_n}{\sum\limits_{\substack{n=1\\ n\neq i}}^{N} K(x_n, x_i)}.$$

Define the residual-sum-of-squares loss

$$\boxed{L(W) \;=\; \sum_{i=1}^{N} \Big(y_i - \hat{y}_i(W)\Big)^2 \;=\; \sum_{i=1}^{N} \left(y_i - \frac{\sum\limits_{n\neq i} \exp\big(-(x_n - x_i)W(x_n - x_i)^\top\big)\, y_n}{\sum\limits_{n\neq i} \exp\big(-(x_n - x_i)W(x_n - x_i)^\top\big)}\right)^2.}$$

2) Expand $L(W)$ for $D = 2$ in terms of $W_{11}, W_{22}, W_{12} = W_{21}$

Let

$$W = \begin{pmatrix} W_{11} & W_{12} \\ W_{12} & W_{22} \end{pmatrix}, \qquad x_i = (x_{i1}, x_{i2}),\;\; x_n = (x_{n1}, x_{n2}),$$

and define coordinate differences

$$\Delta_{ni}^{(1)} = x_{n1} - x_{i1}, \qquad \Delta_{ni}^{(2)} = x_{n2} - x_{i2}.$$

Then the quadratic form expands as

$$(x_n - x_i)W(x_n - x_i)^\top = W_{11}\big(\Delta_{ni}^{(1)}\big)^2 + 2W_{12}\Delta_{ni}^{(1)}\Delta_{ni}^{(2)} + W_{22}\big(\Delta_{ni}^{(2)}\big)^2,$$

so

$$K(x_n, x_i) = \exp\Big(-W_{11}\big(\Delta_{ni}^{(1)}\big)^2 - 2W_{12}\Delta_{ni}^{(1)}\Delta_{ni}^{(2)} - W_{22}\big(\Delta_{ni}^{(2)}\big)^2\Big).$$

Therefore the LOO predictor becomes

$$\hat{y}_i(W_{11}, W_{22}, W_{12}) = \frac{\sum\limits_{n\neq i} \exp\Big(-W_{11}(\Delta_{ni}^{(1)})^2 - 2W_{12}\Delta_{ni}^{(1)}\Delta_{ni}^{(2)} - W_{22}(\Delta_{ni}^{(2)})^2\Big)\, y_n}{\sum\limits_{n\neq i} \exp\Big(-W_{11}(\Delta_{ni}^{(1)})^2 - 2W_{12}\Delta_{ni}^{(1)}\Delta_{ni}^{(2)} - W_{22}(\Delta_{ni}^{(2)})^2\Big)}.$$

Plugging into the RSS objective yields

$$\boxed{L(W_{11}, W_{22}, W_{12}) = \sum_{i=1}^{N} \left(y_i - \frac{\sum\limits_{n\neq i} e^{-\left[W_{11}(\Delta_{ni}^{(1)})^2 + 2W_{12}\Delta_{ni}^{(1)}\Delta_{ni}^{(2)} + W_{22}(\Delta_{ni}^{(2)})^2\right]} y_n}{\sum\limits_{n\neq i} e^{-\left[W_{11}(\Delta_{ni}^{(1)})^2 + 2W_{12}\Delta_{ni}^{(1)}\Delta_{ni}^{(2)} + W_{22}(\Delta_{ni}^{(2)})^2\right]}}\right)^2.}$$

Question 4: KNN and the Curse of Dimensionality

When the number of features p is large, there tends to be a deterioration in the performance of KNN and other *local* approaches that perform prediction using only observations that are near the test observation for which a prediction must be made. This phenomenon is known as the *curse of dimensionality*, and it ties into the fact that parametric approaches often perform poorly when p is large. We will now investigate this curse.

In each of the following scenarios, we assume that each predictor is uniformly (evenly) distributed on $[0, 1]$, and that each observation is associated with a response value.

(a) Suppose that we have a set of observations, each with measurements on $p = 1$ feature, X. Suppose that we wish to predict a test observation's response using only observations that are within 10% of the range of X closest to that test observation. For instance, in order to predict the response for a test observation with $X = 0.6$, we will use observations in the range $[0.55, 0.65]$. On average, **what fraction of the available observations** will we use to make the prediction?

(b) Now suppose that we have a set of observations, each with measurements on $p = 2$ features, X_1 and X_2. We wish to predict a test observation's response using only observations that are within 10% of the range of X_1 and within 10% of the range of X_2 closest to that test observation. On average, **what fraction of the available observations** will we use to make the prediction?

(c) Now suppose that we have a set of observations on $p = 100$ features. We wish to predict a test observation's response using observations within the 10% of each feature's range that is closest to that test observation. **What fraction of the available observations** will we use to make the prediction?

(d) Using your answers to parts (a)–(c), argue that a drawback of KNN when p is large is that there are very few training observations "near" any given test observation.

Solution:

(a) $p = 1$ feature

We keep points with X in an interval of length 0.1 around the test value (ignoring negligible boundary effects; more precisely, the expected length is still 0.1 under a uniform random test point). Thus

$$\Pr(\text{training point is used}) = 0.1,$$

so the average fraction used is

$$\boxed{0.1 \;=\; 10\%.}$$

(b) $p = 2$ features

We require the training point to be within 0.1 of the range in *each* coordinate. The selected region is a square of side length 0.1, hence area $(0.1)^2$ inside the unit square. With independent uniforms,

$$\Pr(\text{used}) = 0.1 \times 0.1 = (0.1)^2 = 0.01,$$

so the average fraction used is

$$\boxed{(0.1)^2 = 0.01 \;=\; 1\%.}$$

(c) $p = 100$ features

Analogously, the selected region is a p-dimensional hypercube of side length 0.1 and volume $(0.1)^p$ inside $[0, 1]^p$. Therefore,

$$\Pr(\text{used}) = (0.1)^{100} = 10^{-100},$$

so the fraction used is

$$\boxed{(0.1)^{100} = 10^{-100}.}$$

(d) Implication for KNN when p is large

From (a)–(c), the expected fraction of training points within a fixed *per-coordinate* closeness requirement decays *exponentially* in the dimension p:

$$\text{fraction used} = (0.1)^p.$$

Hence, if there are N training observations, the expected number of observations "near" a test point is

$$N(0.1)^p.$$

For moderate N and large p, $N(0.1)^p$ is essentially 0 (e.g. for $p = 100$, it is $N \cdot 10^{-100}$), meaning there are very few (often no) training observations in the local neighborhood of a test point. This makes local methods like KNN unreliable: to obtain enough neighbors one must enlarge the neighborhood substantially, but then the neighborhood is no longer "local," increasing bias and degrading predictive performance.

Question 5: Choosing Metrics

Suppose you wanted to classify a mysterious drink as coffee, energy drink, or soda based on the amount of caffeine and amount of sugar per 1-cup serving. Coffee typically contains a large amount of caffeine, lemonade typically contains a large amount of sugar, and energy drinks typically contain a large amount of both. Which distance metric would work best, and why? Answer 1, 2, 3, or 4:

1. Euclidean distance, because you're comparing standard numeric quantities
2. Manhattan distance, because it makes sense to array the training data points in a grid
3. Hamming distance, because the features "contains caffeine" and "contains sugar" are boolean values
4. Cosine distance, because you care more about the ratio of caffeine to sugar than the actual amount

Suppose you add a fourth possible classification, water, which contains no caffeine or sugar. Now which distance metric would work best, and why? Answer 1, 2, 3, or 4:

1. Euclidean distance, because you're still comparing standard numeric quantities, and it's now more difficult to compare ratios
2. Manhattan distance, because there are now four points, so it makes even more sense to array them in a grid
3. Hamming distance, because the features "contains caffeine" and "contains sugar" are still boolean values
4. Cosine distance, because you still care more about the ratio of caffeine to sugar than the actual amount

While visiting Manhattan, you discover that there are 3 different types of taxis, each of which has a distinctive logo. Each type of taxi comes in many makes and models of cars. Which distance metric would work best for classifying additional taxis based on their logos, makes, and models? Answer 1, 2, 3, or 4:

1. Euclidean distance, because there's a lot of variability in make and model
2. Manhattan distance, because the streets meet at right angles (and because you're in Manhattan and classifying taxicabs)
3. Hamming distance, because the features are non-numeric
4. Cosine distance, because it will separate taxis by how different their makes and models are

Why might an ID tree work better for classifying the taxis from previous question ? Answer 1, 2, 3, or 4:

1. k-nearest neighbors requires data to be graphed on a coordinate system, but the training data isn't quantitative
2. Some taxis may have the same make and model, and k-nearest neighbors can't handle identical training points
3. An ID tree can ignore irrelevant features, such as make and model
4. Taxis aren't guaranteed to stay within their neighborhood in Manhattan

Solution:

1) Distance Metric

We want to classify a drink into {coffee, energy drink, soda} using two numeric features:

$$x = (\text{caffeine},\ \text{sugar}).$$

The description emphasizes typical *patterns*:

- Coffee: high caffeine, low sugar
- Soda (e.g. lemonade): low caffeine, high sugar
- Energy drink: high caffeine, high sugar

A metric that captures *relative composition* (the balance/ratio between caffeine and sugar) is often more appropriate than one that depends primarily on absolute scale. In particular, if serving sizes vary (or overall concentration changes), two drinks with similar caffeine-to-sugar proportions should still be considered similar even if both amounts scale up or down.

Cosine distance compares the *angle* between vectors:

$$d_{\cos}(x, x') = 1 - \frac{x^\top x'}{\|x\|_2 \|x'\|_2},$$

so it is largely invariant to multiplying a vector by a positive scalar (i.e. changing the overall magnitude). Thus it focuses on the *ratio* between caffeine and sugar, which directly separates "coffee-like" (mostly caffeine direction) from "soda-like" (mostly sugar direction), while placing "energy-drink-like" points in a direction that has substantial components in both.

Therefore the best answer among the options is:

$$\boxed{4.}$$

2) Adding "water" (0 caffeine, 0 sugar)

Let the feature vector be

$$x = (c, s) = (\text{caffeine},\ \text{sugar}).$$

Adding water introduces a class at

$$x_{\text{water}} = (0, 0).$$

Cosine distance is

$$d_{\cos}(x, x') = 1 - \frac{x^\top x'}{\|x\|_2 \|x'\|_2},$$

which is *undefined* when either vector has zero norm. In particular,

$$\|x_{\text{water}}\|_2 = 0,$$

so cosine distance cannot directly compare water to any other drink (division by zero). This makes cosine distance a poor choice once the zero vector is a valid class prototype.

Euclidean distance, on the other hand, is well-defined for all points, including the origin. Moreover, it naturally captures that water should be close to drinks with small absolute caffeine and sugar and far from drinks with large amounts of either:

$$\|x - (0, 0)\|_2 = \sqrt{c^2 + s^2}.$$

Therefore, among the provided options, the best choice is:

$$\boxed{1.}$$

3) Taxi types by logo, make, and model

We want to classify taxis into 3 types based on *logo*, *make*, and *model*. These features are categorical / non-numeric labels (unless we first embed them into a numeric space). A natural notion of dissimilarity for such attributes is whether they match or not, which is exactly what Hamming distance measures for discrete features.

If we encode a taxi as a vector of categorical entries (e.g. (logo type, make, model)), then Hamming distance is

$$d_H(x, x') = \sum_j \mathbf{1}\{x_j \neq x'_j\},$$

counting how many fields differ. This aligns with the goal: taxis of the same type share the same logo even if make/model vary, while taxis of different types have different logos. Hamming distance can be paired with an encoding where the logo field is included (and can be given extra weight if desired), but among the provided choices it is the only metric intended for non-numeric attributes.

Therefore, the best answer is:

$$\boxed{3.}$$

Question 6: Kernel Ridge Regression

In contrast to ordinary least squares which has a cost function

$$J(\theta) = \frac{1}{2}\sum_{i=1}^{m}(\theta^\top x^{(i)} - y^{(i)})^2,$$

we can also add a term that penalizes large weights in θ. In *ridge regression*, our least squares cost is regularized by adding a term $\lambda\|\theta\|^2$, where $\lambda > 0$ is a fixed (known) constant. The ridge regression cost function is then

$$J(\theta) = \frac{1}{2}\sum_{i=1}^{m}(\theta^\top x^{(i)} - y^{(i)})^2 + \frac{\lambda}{2}\|\theta\|^2.$$

(a) Use the vector notation to find a closed-form expression for the value of θ which minimizes the ridge regression cost function.

(b) Suppose that we want to use kernels to implicitly represent our feature vectors in a high-dimensional (possibly infinite dimensional) space. Using a feature mapping ϕ, the ridge regression cost function becomes

$$J(\theta) = \frac{1}{2}\sum_{i=1}^{m}(\theta^\top \phi(x^{(i)}) - y^{(i)})^2 + \frac{\lambda}{2}\|\theta\|^2.$$

Making a prediction on a new input x_{new} would now be done by computing $\theta^\top\phi(x_{\text{new}})$. Show how we can use the "kernel trick" to obtain a closed form for the prediction on the new input without ever explicitly computing $\phi(x_{\text{new}})$. You may assume that the parameter vector θ can be expressed as a linear combination of the input feature vectors; i.e.,

$$\theta = \sum_{i=1}^{m}\alpha_i\phi(x^{(i)})$$

for some set of parameters α_i.

Hint: You may find the following identity useful:

$$(\lambda I + BA)^{-1}B = B(\lambda I + AB)^{-1}.$$

If you want, you can try to prove this as well, though this is not required for the problem.

Solution:

Let the data matrix be

$$X \triangleq \begin{bmatrix} - - - (x^{(1)})^\top - - - \\ \vdots \\ - - - (x^{(m)})^\top - - - \end{bmatrix} \in \mathbb{R}^{m\times d}, \qquad y \triangleq \begin{bmatrix} y^{(1)} \\ \vdots \\ y^{(m)} \end{bmatrix} \in \mathbb{R}^m.$$

Then the ridge objective can be written compactly as

$$J(\theta) = \frac{1}{2}\|X\theta - y\|_2^2 + \frac{\lambda}{2}\|\theta\|_2^2.$$

(a) **Closed form for ridge regression.**

Differentiate and set the gradient to zero:

$$\nabla_\theta J(\theta) = X^\top (X\theta - y) + \lambda\theta.$$

Setting $\nabla_\theta J(\theta) = 0$ gives

$$X^\top X\theta - X^\top y + \lambda\theta = 0 \quad \Longrightarrow \quad (X^\top X + \lambda I)\theta = X^\top y.$$

Since $\lambda > 0$, the matrix $X^\top X + \lambda I$ is positive definite and hence invertible, so the unique minimizer is

$$\boxed{\theta^\star = (X^\top X + \lambda I)^{-1} X^\top y.}$$

(b) **Kernel trick prediction formula.**

Define the feature matrix

$$\Phi \triangleq \begin{bmatrix} - - - \phi(x^{(1)})^\top - - - \\ \vdots \\ - - - \phi(x^{(m)})^\top - - - \end{bmatrix} \in \mathbb{R}^{m \times p},$$

(where p may be very large or infinite), so that the objective becomes

$$J(\theta) = \frac{1}{2}\|\Phi\theta - y\|_2^2 + \frac{\lambda}{2}\|\theta\|_2^2.$$

By the same calculation as in part (a),

$$\theta^\star = (\Phi^\top \Phi + \lambda I)^{-1} \Phi^\top y.$$

We now rewrite this in a form involving only inner products in feature space. Using the provided identity with $B = \Phi^\top$ and $A = \Phi$:

$$(\lambda I + \Phi^\top \Phi)^{-1} \Phi^\top = \Phi^\top (\lambda I + \Phi\Phi^\top)^{-1}.$$

Hence

$$\theta^\star = (\lambda I + \Phi^\top \Phi)^{-1} \Phi^\top y = \Phi^\top (\lambda I + \Phi\Phi^\top)^{-1} y.$$

This exhibits $\theta^\star$ as a linear combination of training feature vectors:

$$\theta^\star = \Phi^\top \alpha \quad \text{where} \quad \boxed{\alpha = (\Phi\Phi^\top + \lambda I)^{-1} y.}$$

Now consider predicting at a new point x_{new}:

$$\hat{y}_{\text{new}} = (\theta^\star)^\top \phi(x_{\text{new}}) = (\Phi^\top \alpha)^\top \phi(x_{\text{new}}) = \alpha^\top \Phi \phi(x_{\text{new}}).$$

Define the kernel function $k(x, z) = \phi(x)^\top \phi(z)$. Then the vector

$$k_{\text{new}} \triangleq \begin{bmatrix} k(x^{(1)}, x_{\text{new}}) \\ \vdots \\ k(x^{(m)}, x_{\text{new}}) \end{bmatrix} = \begin{bmatrix} \phi(x^{(1)})^\top \phi(x_{\text{new}}) \\ \vdots \\ \phi(x^{(m)})^\top \phi(x_{\text{new}}) \end{bmatrix} = \Phi\, \phi(x_{\text{new}}),$$

so the prediction becomes

$$\hat{y}_{\text{new}} = \alpha^\top k_{\text{new}}.$$

Also define the Gram matrix $K \in \mathbb{R}^{m \times m}$ by

$$K_{ij} = k(x^{(i)}, x^{(j)}) = \phi(x^{(i)})^\top \phi(x^{(j)}), \qquad \text{so that} \qquad K = \Phi\Phi^\top.$$

Therefore

$$\alpha = (K + \lambda I)^{-1} y,$$

and the final kernel ridge regression prediction is

$$\boxed{\hat{y}_{\text{new}} = k_{\text{new}}^\top (K + \lambda I)^{-1} y,}$$

which depends only on kernel evaluations $k(x^{(i)}, x^{(j)})$ and $k(x^{(i)}, x_{\text{new}})$ and never requires explicitly computing $\phi(x)$.

Question 7: Deriving Linear Regression

We noted that the solution for the least squares linear regressions "looked" like a ratio of covariance and variance terms. In this problem, we will derive that. Let us assume the following generative process for our data:

$$x \sim N(0, \Sigma_x)$$

$$\epsilon \sim N(0, \sigma^2)$$

$$y \mid x, \epsilon \;=\; w^\top x + \epsilon$$

Assume scalar x, c, w, and y, and that x is independent of ϵ.

1. Provide a formula for $\Sigma_{yx} = E_{x,y}[yx]$ based on the above generative model.
2. Provide a formula to estimate $E_{x,y}[yx]$ given observed data $\{(x_n, y_n)\}_{n=1}^N$.
3. Moment terms like $E_{x,y}[yx]$, $E_{x,y}[xx^\top]$, etc. can easily be estimated from the data (like you did above). Write down an expression for the optimal w^* which minimizes expected squared residual loss in terms of moments (e.g. $\mu_x, \Sigma_x, \Sigma_{yx}, \sigma$).
4. Now, suppose the data x were generated from $N(\mu_x, \Sigma_x)$. Write an expression for w^* in terms of moments (as above).
5. Would the formula for w^* derived in (part 4) hold if the process generating the x's were no longer Gaussian, but still had the same means, variances, and covariances? That is, $E[x] = \mu_x$, $\mathrm{Var}[x] = \Sigma_x$, $\mathrm{Var}[\epsilon] = \sigma^2$, and $E[yx] = \Sigma_{yx}$, but the distribution of x is not Gaussian?

Solution

Throughout, assume scalar random variables. The generative model is

$$x \sim \mathcal{N}(0, \Sigma_x), \qquad \varepsilon \sim \mathcal{N}(0, \sigma^2), \qquad y = wx + \varepsilon,$$

and x is independent of ε. (The independence is what matters for the moment calculations.)

1. **Compute $\Sigma_{yx} = \mathbb{E}_{x,y}[yx]$.**
   Using $y = wx + \varepsilon$,
   $$\Sigma_{yx} = \mathbb{E}[yx] = \mathbb{E}[(wx + \varepsilon)x] = w\,\mathbb{E}[x^2] + \mathbb{E}[\varepsilon x].$$
   Since $x \perp \varepsilon$ and $\mathbb{E}[\varepsilon] = 0$,
   $$\mathbb{E}[\varepsilon x] = \mathbb{E}[\varepsilon]\mathbb{E}[x] = 0.$$
   Also $x \sim \mathcal{N}(0, \Sigma_x)$ implies $\mathbb{E}[x^2] = (x) = \Sigma_x$. Hence
   $$\boxed{\Sigma_{yx} = w\,\Sigma_x.}$$
2. **Estimate $\mathbb{E}[yx]$ from observed data $\{(x_n, y_n)\}_{n=1}^N$.**
   A natural (plug-in) estimator of the moment $\mathbb{E}[yx]$ is the sample average:
   $$\boxed{\widehat{\mathbb{E}[yx]} \;=\; \frac{1}{N}\sum_{n=1}^{N} y_n x_n.}$$
   (If you also needed $\mathbb{E}[x^2]$, you would use $\frac{1}{N}\sum_{n=1}^N x_n^2$.)
3. **Derive the optimal $w^\star$ minimizing expected squared residual loss.**
   Consider the population risk for linear prediction:
   $$\mathcal{L}(w) \;\triangleq\; \frac{1}{2}\,\mathbb{E}\big[(wx - y)^2\big]\,.$$
   Differentiate w.r.t. w (scalar calculus):
   $$\frac{d}{dw}\mathcal{L}(w) = \frac{1}{2}\,\mathbb{E}[2(wx - y)\,x] = \mathbb{E}[(wx - y)x] = w\,\mathbb{E}[x^2] - \mathbb{E}[xy].$$

Setting to zero yields

$$w^\star \mathbb{E}[x^2] = \mathbb{E}[xy] \quad \Longrightarrow \quad \boxed{w^\star = \frac{\mathbb{E}[xy]}{\mathbb{E}[x^2]}}$$

and under the given mean-zero model ($\mathbb{E}[x] = 0$), we have $\mathbb{E}[xy] = \Sigma_{yx}$ and $\mathbb{E}[x^2] = \Sigma_x$, so

$$\boxed{w^\star = \frac{\Sigma_{yx}}{\Sigma_x}.}$$

(Notice σ^2 affects the *minimum achievable loss* but does not change the optimizer $w^\star$ for linear least squares.)

4. **Now suppose $x \sim \mathcal{N}(\mu_x, \Sigma_x)$. Express $w^\star$ in terms of moments.**

   If x is not mean-zero, the best linear predictor typically includes an intercept. Write the affine predictor

   $$\hat{y} = wx + c, \qquad \mathcal{L}(w, c) = \frac{1}{2}\mathbb{E}\left[(wx + c - y)^2\right].$$

   Set partial derivatives to zero:

   $$\frac{\partial}{\partial c}\mathcal{L}(w, c) = \mathbb{E}[wx + c - y] = 0 \implies c^\star = \mathbb{E}[y] - w^\star\mathbb{E}[x] = \mu_y - w^\star\mu_x.$$

   For w,

   $$\frac{\partial}{\partial w}\mathcal{L}(w, c) = \mathbb{E}[(wx + c - y)x] = 0.$$

   Substitute $c = c^\star$ and simplify (standard normal equations) to obtain the covariance/variance form:

   $$\boxed{w^\star = \frac{(x, y)}{(x)} = \frac{\mathbb{E}[xy] - \mathbb{E}[x]\mathbb{E}[y]}{\mathbb{E}[x^2] - \mathbb{E}[x]^2} = \frac{\mathbb{E}[xy] - \mu_x\mu_y}{\Sigma_x}.}$$

   Equivalently, if you define Σ_{yx} now as the covariance (y, x), then $w^\star = \Sigma_{yx}/\Sigma_x$.

5. **Would the formula for $w^\star$ from (4) still hold if x were not Gaussian, but had the same moments?**

   Yes. The minimizer of the mean squared error over *affine* predictors depends only on the first and second moments of (x, y) (i.e., $\mu_x, \mu_y, \mathbb{E}[x^2], \mathbb{E}[xy]$), because $\mathcal{L}(w, c)$ expands into a quadratic function of (w, c) whose coefficients are exactly those moments. Therefore the optimality conditions (normal equations) and the solution

   $$w^\star = \frac{(x, y)}{(x)}, \qquad c^\star = \mu_y - w^\star\mu_x$$

   do *not* require Gaussianity of x; they hold for any distribution with finite second moments.

Question 8: Logistic Regression with Newton's Method

Given examples $\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_n \in \mathbb{R}^d$ and associated labels $y_1, y_2, \ldots, y_n \in \{0, 1\}$, the cost function for *unregularized* logistic regression is

$$J(\mathbf{w}) \triangleq -\sum_{i=1}^{n} \left(y_i \ln s_i + (1 - y_i)\ln(1 - s_i)\right)$$

where $s_i \triangleq s(\mathbf{x}_i \cdot \mathbf{w})$, $\mathbf{w} \in \mathbb{R}^d$ is a weight vector, and

$$s(y) \triangleq \frac{1}{1 + e^{-y}}$$

is the logistic function.

Define the $n \times d$ design matrix X (whose ith row is $\mathbf{x}_i^\top$), the label n-vector $\mathbf{y} \triangleq [y_1 \ \ldots \ y_n]^\top$, and $\mathbf{s} \triangleq [s_1 \ \ldots \ s_n]^\top$. For an n-vector $\mathbf{a}$, let

$$\ln \mathbf{a} \triangleq [\ln a_1 \ \ldots \ \ln a_n]^\top.$$

The cost function can be rewritten in vector form as

$$J(\mathbf{w}) = -\mathbf{y} \cdot \ln \mathbf{s} - (1 - \mathbf{y}) \cdot \ln(1 - \mathbf{s}).$$

Further, recall that for a real symmetric matrix $A \in \mathbb{R}^{d \times d}$, there exist U and Λ such that $A = U\Lambda U^\top$ is the eigendecomposition of A. Here Λ is a diagonal matrix with entries $\{\lambda_1, \ldots, \lambda_d\}$. An alternative notation is $\Lambda = \text{diag}(\lambda_i)$, where diag() takes as input the list of diagnal entries, and constructs the corresponding diagonal matrix. This notation is widely used in libraries like numpy and is useful for simplifying some of the expressions when written in matrix–vector form. For example, we can write $\mathbf{s} = \text{diag}(s_i)\mathbf{1}$.

Hint: See page two for notational conventions used here.

Hint: Recall matrix calculus identities. The elements in ***bold*** *indicate vectors.*

$$\nabla_x \alpha y = (\nabla_x \alpha)\, y^\top + \alpha \nabla_x y \qquad \nabla_x (y \cdot z) = (\nabla_x y) z + (\nabla_x z) y;$$

$$\nabla_x f(y) = (\nabla_x y)(\nabla_y f(y)); \qquad \nabla_x g(y) = (\nabla_x y)(\nabla_y g(y));$$

$$\text{and} \qquad \nabla_x C y(x) = (\nabla_x y(x)) C^\top, \quad \text{where } C \text{ is a constant matrix.}$$

1. Derive the gradient $\nabla_{\mathbf{w}} J(\mathbf{w})$ of cost $J(\mathbf{w})$ as a matrix–vector expression. Also derive *all intermediate derivatives* in matrix–vector form. Do *not* specify them (including the intermediates) in terms of their individual components (e.g., w_i). You are *only* allowed to use individual components if and only if they are inside a diag function.

2. Derive the Hessian $\nabla^2_{\mathbf{w}} J(\mathbf{w})$ for the cost function $J(\mathbf{w})$ as a matrix–vector expression.

3. Write the matrix–vector update law for one iteration of Newton's method, substituting the gradient and Hessian of $J(\mathbf{w})$.

4. You are given four examples

$$\mathbf{x}_1 = [0.2 \;\; 3.1]^\top, \quad \mathbf{x}_2 = [1.0 \;\; 3.0]^\top, \quad \mathbf{x}_3 = [-0.2 \;\; 1.2]^\top, \quad \mathbf{x}_4 = [1.0 \;\; 1.1]^\top$$

   with labels $y_1 = 1, y_2 = 1, y_3 = 0, y_4 = 0$. These points cannot be separated by a line passing through the origin. Hence, as described in lecture, append a 1 to each $\mathbf{x}_i$ and use a weight vector $\mathbf{w} \in \mathbb{R}^3$ whose last component is the bias term (called α in lecture).

   Begin with initial weight

$$\mathbf{w}^{(0)} = [-1 \;\; 1 \;\; 0]^\top.$$

   For the following, state only the final answer with four digits after the decimal point. You may use a calculator or write a program to solve for these, but **do NOT** submit any code for this part.

Solution

Let

$$\mathbf{z} \triangleq X\mathbf{w} \in \mathbb{R}^n, \qquad \mathbf{s} \triangleq s(\mathbf{z}) \in \mathbb{R}^n, \qquad s(t) = \frac{1}{1 + e^{-t}}$$

(where $s(\mathbf{z})$ is applied elementwise). The cost is

$$J(\mathbf{w}) = -\mathbf{y}^\top \ln \mathbf{s} - (\mathbf{1} - \mathbf{y})^\top \ln(\mathbf{1} - \mathbf{s}).$$

1. **Gradient $\nabla_{\mathbf{w}} J(\mathbf{w})$ (with all intermediate derivatives).**

   Intermediate 1: $\nabla_{\mathbf{w}} \mathbf{z}$. Since $\mathbf{z} = X\mathbf{w}$,

$$\nabla_{\mathbf{w}} \mathbf{z} = X.$$

   Intermediate 2: $\nabla_{\mathbf{z}} \mathbf{s}$. Since $s'(t) = s(t)(1 - s(t))$, in vector form

$$\nabla_{\mathbf{z}} \mathbf{s} = \text{diag}\big(s_i(1 - s_i)\big).$$

   Intermediate 3: $\nabla_{\mathbf{s}} J$. Using $\nabla_{\mathbf{s}}(\mathbf{a}^\top \ln \mathbf{s}) = \text{diag}(1/s_i)\,\mathbf{a}$ and $\nabla_{\mathbf{s}}(\mathbf{a}^\top \ln(\mathbf{1} - \mathbf{s})) = -\,\text{diag}\big(1/(1 - s_i)\big)\mathbf{a}$, we get

$$\nabla_{\mathbf{s}} J = -\,\text{diag}\Big(\frac{1}{s_i}\Big)\mathbf{y} \;+\; \text{diag}\Big(\frac{1}{1 - s_i}\Big)(\mathbf{1} - \mathbf{y}).$$

Chain rule to $\nabla_{\mathbf{w}} J$. First,

$$\nabla_{\mathbf{w}} \mathbf{s} = (\nabla_{\mathbf{w}} \mathbf{z})(\nabla_{\mathbf{z}} \mathbf{s}) = X \operatorname{diag}\big(s_i(1 - s_i)\big).$$

Then

$$\nabla_{\mathbf{w}} J = (\nabla_{\mathbf{w}} \mathbf{s})\,(\nabla_{\mathbf{s}} J) = X \operatorname{diag}\big(s_i(1 - s_i)\big) \left[-\operatorname{diag}\Big(\frac{1}{s_i}\Big)\mathbf{y} + \operatorname{diag}\Big(\frac{1}{1 - s_i}\Big)(\mathbf{1} - \mathbf{y}) \right].$$

Use the simplifications

$$\operatorname{diag}\big(s_i(1 - s_i)\big)\operatorname{diag}\Big(\frac{1}{s_i}\Big) = \operatorname{diag}(1 - s_i), \qquad \operatorname{diag}\big(s_i(1 - s_i)\big)\operatorname{diag}\Big(\frac{1}{1 - s_i}\Big) = \operatorname{diag}(s_i),$$

to obtain

$$\nabla_{\mathbf{w}} J = X \left[-\operatorname{diag}(1 - s_i)\mathbf{y} + \operatorname{diag}(s_i)(\mathbf{1} - \mathbf{y}) \right] = X(\mathbf{s} - \mathbf{y}).$$

Finally, expressing as a d-vector,

$$\boxed{\nabla_{\mathbf{w}} J(\mathbf{w}) = X^\top (\mathbf{s} - \mathbf{y}).}$$

2. **Hessian $\nabla^2_{\mathbf{w}} J(\mathbf{w})$.**

Differentiate the gradient:

$$\nabla_{\mathbf{w}} J(\mathbf{w}) = X^\top (\mathbf{s} - \mathbf{y}),$$

and note $\mathbf{y}$ is constant, so

$$\nabla^2_{\mathbf{w}} J(\mathbf{w}) = X^\top (\nabla_{\mathbf{w}} \mathbf{s}).$$

From above,

$$\nabla_{\mathbf{w}} \mathbf{s} = \operatorname{diag}\big(s_i(1 - s_i)\big)\, X \quad \text{(shape } n \times d\text{)}.$$

Therefore,

$$\boxed{\nabla^2_{\mathbf{w}} J(\mathbf{w}) = X^\top \operatorname{diag}\big(s_i(1 - s_i)\big)\, X.}$$

(Equivalently, letting $R \triangleq \operatorname{diag}(s_i(1 - s_i))$, we have $\nabla^2 J = X^\top R X$.)

3. **Newton update (one iteration).**

Newton's method updates

$$\mathbf{w}^{(t+1)} = \mathbf{w}^{(t)} - \left[\nabla^2_{\mathbf{w}} J(\mathbf{w}^{(t)}) \right]^{-1} \nabla_{\mathbf{w}} J(\mathbf{w}^{(t)}).$$

Substituting the expressions from (1) and (2), with $\mathbf{s}^{(t)} = s(X\mathbf{w}^{(t)})$,

$$\boxed{\mathbf{w}^{(t+1)} = \mathbf{w}^{(t)} - \left(X^\top \operatorname{diag}\big(s_i^{(t)}(1 - s_i^{(t)})\big) X \right)^{-1} X^\top \left(\mathbf{s}^{(t)} - \mathbf{y} \right).}$$

4. **Numerical Newton step for the given data (with bias).**

Append 1 to each $\mathbf{x}_i$, so the design matrix is

$$X = \begin{bmatrix} 0.2 & 3.1 & 1 \\ 1.0 & 3.0 & 1 \\ -0.2 & 1.2 & 1 \\ 1.0 & 1.1 & 1 \end{bmatrix}, \qquad \mathbf{y} = \begin{bmatrix} 1 \\ 1 \\ 0 \\ 0 \end{bmatrix}, \qquad \mathbf{w}^{(0)} = \begin{bmatrix} -1 \\ 1 \\ 0 \end{bmatrix}.$$

Compute $\mathbf{z}^{(0)} = X\mathbf{w}^{(0)}$ and $\mathbf{s}^{(0)} = s(\mathbf{z}^{(0)})$:

$$\mathbf{s}^{(0)} \approx \begin{bmatrix} 0.9478 \\ 0.8808 \\ 0.8022 \\ 0.5250 \end{bmatrix}.$$

Gradient and Hessian at $\mathbf{w}^{(0)}$:

$$\nabla J(\mathbf{w}^{(0)}) = X^\top (\mathbf{s}^{(0)} - \mathbf{y}) \approx \begin{bmatrix} 0.2349 \\ 1.0208 \\ 1.1558 \end{bmatrix},$$

$$\nabla^2 J(\mathbf{w}^{(0)}) = X^\top \operatorname{diag}\big(s_i^{(0)}(1-s_i^{(0)})\big) X \approx \begin{bmatrix} 0.3627 & 0.5819 & 0.3325 \\ 0.5819 & 1.9503 & 0.9330 \\ 0.3325 & 0.9330 & 0.5625 \end{bmatrix}.$$

One Newton update gives

$$\boxed{\mathbf{w}^{(1)} = \mathbf{w}^{(0)} - \left[\nabla^2 J(\mathbf{w}^{(0)})\right]^{-1} \nabla J(\mathbf{w}^{(0)}) \approx \begin{bmatrix} 1.3247 \\ 3.0499 \\ -6.8291 \end{bmatrix}.}$$

Question 9: A Bayesian Interpretation of Lasso

Suppose you are aware that the labels y_i corresponding to sample points $x_i \in \mathbb{R}^d$ follow the density law

$$f(y_i \mid x_i, \mathbf{w}) = \frac{1}{\sigma\sqrt{2\pi}} \exp\left(-\frac{(y_i - \mathbf{w}\cdot x_i)^2}{2\sigma^2}\right)$$

where $\sigma > 0$ is a known constant and $\mathbf{w} \in \mathbb{R}^d$ is a random parameter. Suppose further that experts have told you that:

- Each component of $\mathbf{w}$ is independent of the others, and
- Each component of $\mathbf{w}$ has the Laplace distribution with location 0 and scale being a known constant b. That is, each component w_j obeys the density law

$$f(w_j) = \frac{1}{2b} \exp\left(-\frac{|w_j|}{b}\right)$$

Assume the outputs y_i are independent from each other.

Your goal is to find the choice of parameter $\mathbf{w}$ that is *most likely* given the input-output examples (x_i, y_i). This method of estimating parameters is called *maximum a posteriori* (MAP); Latin for *"maximum [odds] from what follows."*

1. Derive the *posterior* probability density law $f(\mathbf{w} \mid (x_i, y_i))$ for $\mathbf{w}$ *up to a proportionality constant* by applying Bayes' Theorem and substituting for the densities $f(y_i \mid x_i, \mathbf{w})$ and $f(\mathbf{w})$. Don't try to derive an exact expression for $f(\mathbf{w} \mid (x_i, y_i))$, as the denominator is very involved and irrelevant to maximum likelihood estimation.

2. Define the log-likelihood for MAP as

$$\ell(\mathbf{w}) \triangleq \ln f(\mathbf{w} \mid (x_i, y_i))$$

Show that maximizing the MAP log-likelihood over all choices of $\mathbf{w}$ is the same as minimizing

$$\sum_{i=1}^{n} (y_i - \mathbf{w}\cdot x_i)^2 + \lambda \|\mathbf{w}\|_1$$

where $\|\mathbf{w}\|_1 = \sum_{j=1}^{d} |w_j|$ and λ is a constant. Also give a formula for λ as a function of the distribution parameters.

Solution

Let the dataset be $\mathcal{D} = \{(x_i, y_i)\}_{i=1}^n$, where $x_i \in \mathbb{R}^d$ and $y_i \in \mathbb{R}$. Assume conditional independence of outputs given $(X, \mathbf{w})$, and an i.i.d. Laplace prior on the coordinates of $\mathbf{w}$.

1. **Posterior density up to proportionality.**

   By Bayes' theorem,

$$f(\mathbf{w} \mid \mathcal{D}) = \frac{f(\mathcal{D} \mid \mathbf{w})\, f(\mathbf{w})}{f(\mathcal{D})} \;\propto\; f(\mathcal{D} \mid \mathbf{w})\, f(\mathbf{w}),$$

   where the evidence $f(\mathcal{D})$ does not depend on $\mathbf{w}$ and is irrelevant for MAP.

Likelihood. Using independence of the outputs y_i conditioned on $\mathbf{w}$,

$$f(\mathcal{D} \mid \mathbf{w}) = \prod_{i=1}^{n} f(y_i \mid x_i, \mathbf{w}) = \prod_{i=1}^{n} \frac{1}{\sigma\sqrt{2\pi}} \exp\left(-\frac{(y_i - \mathbf{w}^\top x_i)^2}{2\sigma^2}\right).$$

Thus,

$$f(\mathcal{D} \mid \mathbf{w}) = \left(\frac{1}{\sigma\sqrt{2\pi}}\right)^n \exp\left(-\frac{1}{2\sigma^2}\sum_{i=1}^{n}(y_i - \mathbf{w}^\top x_i)^2\right).$$

Prior. The coordinates are independent and each w_j is Laplace$(0, b)$:

$$f(\mathbf{w}) = \prod_{j=1}^{d} f(w_j) = \prod_{j=1}^{d} \frac{1}{2b} \exp\left(-\frac{|w_j|}{b}\right) = \left(\frac{1}{2b}\right)^d \exp\left(-\frac{1}{b}\sum_{j=1}^{d}|w_j|\right).$$

Posterior up to proportionality. Multiplying likelihood and prior,

$$f(\mathbf{w} \mid \mathcal{D}) \propto \exp\left(-\frac{1}{2\sigma^2}\sum_{i=1}^{n}(y_i - \mathbf{w}^\top x_i)^2\right) \exp\left(-\frac{1}{b}\sum_{j=1}^{d}|w_j|\right).$$

Equivalently,

$$\boxed{f(\mathbf{w} \mid \mathcal{D}) \propto \exp\left(-\frac{1}{2\sigma^2}\sum_{i=1}^{n}(y_i - \mathbf{w}^\top x_i)^2 - \frac{1}{b}\|\mathbf{w}\|_1\right).}$$

2. **MAP as Lasso; identify λ.**

Define the MAP log-posterior

$$\ell(\mathbf{w}) \triangleq \ln f(\mathbf{w} \mid \mathcal{D}).$$

Using the proportional form above, we may ignore additive constants in $\ell(\mathbf{w})$ that do not depend on $\mathbf{w}$:

$$\ell(\mathbf{w}) = -\frac{1}{2\sigma^2}\sum_{i=1}^{n}(y_i - \mathbf{w}^\top x_i)^2 - \frac{1}{b}\|\mathbf{w}\|_1 + \text{const}.$$

Therefore, maximizing $\ell(\mathbf{w})$ is equivalent to minimizing its negative (and we may multiply by any positive constant without changing the minimizer). Multiply $-\ell(\mathbf{w})$ by $2\sigma^2 > 0$:

$$\arg\max_{\mathbf{w}} \ell(\mathbf{w}) = \arg\min_{\mathbf{w}} \left\{\sum_{i=1}^{n}(y_i - \mathbf{w}^\top x_i)^2 + \frac{2\sigma^2}{b}\|\mathbf{w}\|_1\right\}.$$

Hence the MAP estimator under a Gaussian noise model and Laplace prior is exactly the Lasso objective with

$$\boxed{\lambda = \frac{2\sigma^2}{b}.}$$

Question 10: Parzen Window Classification with Gaussian Kernel

Consider Parzen-window estimates and classifiers for points in the table below. Let your window function be a spherical Gaussian, i.e.,

$$\phi\left(\frac{\mathbf{x} - \mathbf{x}_i}{h}\right) \propto \exp\left[-\frac{(\mathbf{x} - \mathbf{x}_i)^\top(\mathbf{x} - \mathbf{x}_i)}{2h^2}\right].$$

(a) Write a program to classify an arbitrary test point $\mathbf{x}$ based on the Parzen window estimates. Train your classifier using the three-dimensional data from your three categories in the table above. Set $h = 1$ and classify the following three points:

$$(0.50, 1.0, 0.0)^\top, \quad (0.31, 1.51, -0.50)^\top, \quad \text{and} \quad (-0.3, 0.44, -0.1)^\top.$$

(b) Repeat with $h = 0.1$.

Sample	ω_1 (x_1, x_2, x_3)	ω_2 (x_1, x_2, x_3)	ω_3 (x_1, x_2, x_3)
1	0.28, 1.31, -6.2	0.011, 1.03, -0.21	1.36, 2.17, 0.14
2	0.07, 0.58, -0.78	1.27, 1.28, 0.08	1.41, 1.45, -0.38
3	1.54, 2.01, -1.63	0.13, 3.12, 0.16	1.22, 0.99, 0.69
4	-0.44, 1.18, -4.32	-0.21, 1.23, -0.11	2.46, 2.19, 1.31
5	-0.81, 0.21, 5.73	-2.18, 1.39, -0.19	0.68, 0.79, 0.87
6	1.52, 3.16, 2.77	0.34, 1.96, -0.16	2.51, 3.22, 1.35
7	2.20, 2.42, -0.19	-1.38, 0.94, 0.45	2.60, 2.44, 0.92
8	0.91, 1.94, 6.21	-0.12, 0.82, 0.17	0.64, 0.13, 0.97
9	0.65, 1.93, 4.38	-1.44, 2.31, 0.14	0.85, 0.58, 0.99
10	-0.26, 0.82, -0.96	0.26, 1.94, 0.08	0.66, 0.51, 0.88

Table 1: Three-dimensional data samples for categories ω_1, ω_2, and ω_3

Question 11: Maximum likelihood estimation

1. The file normal.data.txt has a random sample from a normal distribution.

   (a) Find the maximum likelihood estimates of $\hat{\mu}$ and $\hat{\sigma}^2$ numerically. Compare the answer to your closed-form solution.

   (b) Show that the minus log-likelihood is indeed minimized at $(\hat{\mu}, \hat{\sigma}^2)$ for this data set.

   (c) Calculate the estimated asymptotic covariance matrix of the MLEs.

   (d) Give a "better" estimated asymptotic covariance matrix based on your closed-form solution.

   (e) Calculate a large-sample 95% confidence interval for σ^2.

   (f) Test $H_0 : \mu = 103$ with a

      i. Z-test.

      ii. Likelihood ratio chi-squared test. Compare the closed-form version.

      iii. Wald chi-squared test.

      Give the test statistic and the p-value for each test.

   (g) The coefficient of variation (used in sample surveys and business statistics) is the standard deviation divided by the mean.

      i. Show that multiplication by a positive constant does not affect the coefficient of variation. This is a paper-and-pencil calculation.

      ii. Give a numerical point estimate of the coefficient of variation for the normal data of this question. Actually, it's the maximum likelihood estimate, because the invariance principle of maximum likelihood estimation says that the MLE of a function is that function of the MLE.

      iii. Using the delta method, give a 95% confidence interval for the coefficient of variation. Start with a paper-and-pencil calculation of $g(\theta) = \left(\frac{\partial g}{\partial \theta_1}, \ldots, \frac{\partial g}{\partial \theta_k}\right)$.

Question 12: Non-Linear Regression using basis functions

In this problem, we consider a way of extending the linear regression technique to non-linear problems. A simple (but often effective) approach is linear regression with basis expansion. Instead of actually fitting a non-linear function to the data, the data is preprocessed by a non-linear mapping, and linear regression is then applied to the transformed problem. This method is equivalent to fitting a regression function of the form

$$\hat{f}(x) = \sum_{j=0}^{d} \beta_j h_j(x), \tag{12.1}$$

where the basis functions h_j are fixed. The linear coefficients β_j are the target parameters of the learning problem. The parameter β_0 accounts for the bias, and the corresponding basis function is the constant function $h_0(x) = 1$.

To apply linear regression in the one-dimensional case, we transform each observation $x^{(i)}$ into a vector $h(x^{(i)}) := (h_0(x^{(i)}), \ldots, h_d(x^{(i)}))$. The input data vector $x = (x^{(1)}, \ldots, x^{(n)})^\top$ is then substituted by a matrix H, containing the vectors $h(x^{(i)})$ as its rows:

$$\begin{bmatrix} x^{(1)} \\ x^{(2)} \\ \vdots \\ x^{(n)} \end{bmatrix} \quad \text{becomes} \quad \begin{bmatrix} h_0(x^{(1)}) & h_1(x^{(1)}) & \cdots & h_d(x^{(1)}) \\ h_0(x^{(2)}) & h_1(x^{(2)}) & \cdots & h_d(x^{(2)}) \\ \vdots & \vdots & & \vdots \\ h_0(x^{(n)}) & h_1(x^{(n)}) & \cdots & h_d(x^{(n)}) \end{bmatrix}.$$

Both standard linear regression and ridge regression are now applicable.

Here is what we would ask you to do:

1. Generate values from a sinc function
$$f(x) = \text{sinc}(3x),$$
by uniformly sampling the input domain and adding zero-mean Gaussian noise to the function values, to obtain:
$$y = f(x) + \epsilon, \tag{12.2}$$
where $x \sim \text{Uniform}(0, 1)$ and $\epsilon \sim \mathcal{N}(0, 0.01)$. The input vector x consists of $n = 25$ samples and the corresponding output vector is y.

2. Program a regression estimator. The basis functions we consider are Gaussians of the form
$$h_j(x) = \exp\left(-\frac{(x - \mu_j)^2}{2\sigma_j^2}\right), \tag{12.3}$$
for $j = 1, \ldots, d$. Program an estimation routine
$$\beta = \text{regress.gauss}(x, y, \text{means}, \text{var}, \lambda)$$
The arguments are:
   - x — a single observation vector
   - y — the corresponding outputs
   - `means` — vector of basis function means μ_j
   - `var` — the (scalar) variance parameter of the basis functions
   - `lambda` — the ridge regression regularization parameter
   - `beta` — the estimate $\hat{\beta}$ or $\hat{\beta}^{\text{ridge}}$, respectively.

3. Apply your estimator to the data you generated. Use the following values for the parameters:
$$d = 21 \quad \text{(a total of 22 functions including the bias)},$$
$$\mu_j \text{ equally spaced over the interval } [0, 1],$$
$$\sigma_j = 0.04,$$
$$\lambda \in \{0.001, 0.05, 5\}.$$
For each λ, plot a graph that shows:
   - the function estimated from training vector x,
   - the function estimated by averaging over the $\beta^{(i)}$ obtained from repeating the above experiment 100 times (i.e. generating pairs $(x^{(i)}, y^{(i)})$, estimating coefficients $\beta^{(i)}$, and averaging over all runs to obtain β^{avg}).

The estimated function is given by

$$\hat{f}(x) = \hat{\beta}_0^{\text{avg}} + \sum_{j=1}^{d} h_j(x)\, \hat{\beta}_j^{\text{avg}}. \tag{12.4}$$

To plot the function, compute $\hat{f}(x)$ for $x \in [0, 1]$ equally spaced (e.g. using `linspace`). Comment about the plots and the quality of solutions.

4. The estimation error can be computed by looking at the squared error of the estimator to the true function. Please plot the average error as a function of λ. To this end, repeat the estimation process for the 100 random vectors $x^{(i)}, i = 1, \ldots, 100$, for the regularizations

$$\lambda = \exp(\{-5 : 0.3 : 2\}).$$

Instead of computing the L_2 distance between the functions by integration, we approximate it by

$$\text{err} = \frac{1}{m} \sum_{i=1}^{m} \left(f(z_i) - \hat{f}(z_i) \right)^2, \tag{12.5}$$

where $z_1, \ldots, z_m$ are equally spaced points ($m = 100$). Plot the average errors to show how it varies over λ. Comment on what happens on the two ends of the plot.