CSCI-SHU 360: Machine Learning - Homework 2
Due: Mar 12 23:59, 2025
100 points + 20 bonus points

Instructions

- **Collaboration policy:** Homework must be done individually, except where otherwise noted in the assignments. "Individually" means each student must hand in their own answers, and you must write and use your own code in the programming parts of the assignment. It is acceptable for you to collaborate in figuring out answers and to help each other solve the problems, and you must list the names of students you discussed this with. We will assume that, as participants in an undergraduate course, you will be taking the responsibility to make sure you personally understand the solution to any work arising from such collaboration.
- **Online submission:** You must submit your solutions online via **Gradescope**. You need to submit (1) a PDF that contains the solutions to all questions to the Gradescope **HW2 PaperWork** assignment, (2) `x.py` or `x.ipynb` files for the programming questions to the Gradescope **HW2 CodeFiles** assignment. We recommend that you type the solution (e.g., using LaTeX or Word), but we will accept scanned/pictured solutions as well (clarity matters).
- **Generative AI Policy:** You are free to use any generative AI, but you are required to document the usage: which AI do you use, and what's the query. You are responsible for checking the correctness.
- **Late Policy: No late submission is allowed.**

1 Linear Regression and Convexity [10 points]

Show detailed derivations that the linear regression loss function is convex in the parameters w: $L(w) = ||y - Xw||_2^2$. Here y is an n-dimensional vector, X is an $n \times d$ matrix and w is a d-dimensional vector. X is of full rank. There are multiple ways to show it, but we force you to take the following approach: compute $\nabla L(w)$ using directional derivative and then compute $\nabla^2 L(w)$.

2 Gaussian Distribution and the Curse of Dimensionality [40 points]

In this problem, we will investigate the Gaussian distribution in high dimensional space, and develop intuitions and awareness about the **curse of dimensionality**, a critical concept that everyone who wishes to pursue study in machine learning should understand.

For a random variable x of m dimensions (i.e., $x \in \mathbb{R}^m$) drawn from a multivariate Gaussian distribution, recall that the Gaussian density function takes the form:

$$p(x) = \frac{1}{|2\pi C|^{1/2}} \exp\left(-\frac{1}{2}(x-\mu)^T C^{-1}(x-\mu)\right) \tag{1}$$

where $\mu \in \mathbb{R}^m$ is an m-dimensional mean vector and $C \in \mathbb{R}^{m \times m}$ is an order $m \times m$ symmetric positive definite covariance matrix. When C is a diagonal matrix, then the covariance between different dimensions is zero, which is known as a axis-aligned Gaussian, and when $C = \sigma^2 I$, I being the $m \times m$ identity matrix,

we already saw this in Homework 1 where the $m = 2$ (2D) Gaussian in this case has spherical (circle) shaped contours. A spherical Gaussian in m dimensions thus has the following equation form:

$$p(x) = \frac{1}{(2\pi\sigma^2)^{m/2}} \exp\left(-\frac{\|x\|^2}{2\sigma^2}\right) \tag{2}$$

We start by examining some basic geometric properties of the sphere in m dimensional space. A sphere is generally a collection of points such that the distance of any point to the center of the sphere (we always center the sphere on origin $\vec{0}$ for simplicity) is equal to r, the radius. In other words, we define an m-dimensional sphere as $\mathcal{S}_{m-1}(r) = \{x \in \mathbb{R}^m : \|x\|_2 = r\}$, the set of points in m-dimensional space that are distance r from the origin (note that $\mathcal{S}_{m-1}(r)$ is the equation for the surface of the sphere, although in some fields, such as physics, they define the sphere as the surface and interior, as in $\{x \in \mathbb{R}^m : \|x\|_2 \leq r\}$). We also use $V_m(r)$ for the volume of an m-dimensional sphere.

$S_{m-1}(r)$ represents the surface area of the m-dimensional sphere (meaning, e.g., that $m = 2$ dimensional sphere of radius r has surface area $S_1(r)$, this convention is used since the surface area is a curved $m-1$ dimensional manifold embedded in m-dimensional ambient space).

Please make sure you answer every question:

1. *[1 points]* Before we move to m dimensions, let's talk about 2D and 3D cases. Write down the equations, in terms of the radius r, of $S_{m-1}(r)$ for $m \in \{2, 3\}$ and $V_m(r)$ for $m \in \{2, 3\}$.

2. *[5 points]* Intuitively explain the following equation:

$$S_{m-1}(r) = \frac{d}{dr} V_m(r). \tag{3}$$

   Why does this equation make sense and why should it be true? You may help to convince yourself and improve intuition, by verifying the equations of $S_1(r)$, $V_2(r)$, $S_2(r)$ and $V_3(r)$ from the previous question.

3. *[4 points]* As you may have guessed, $V_m(r)$'s only dependence on r and m is via m's power of r, or specifically r^m. Suppose for a unit sphere ($r = 1$) in m dimensional space, the surface area is $\bar{S}_{m-1}$. Write $S_{m-1}(r)$ in terms of r and $\bar{S}_{m-1}$.

4. *[5 points]* Now consider all the points on the sphere $\mathcal{S}_{m-1}(r)$ (which, because of our definition of $\mathcal{S}_{m-1}(r)$ really means the surface). We wish to integrate over all of those points weighted by the Gaussian probability density $p(x)$ of each point, where $p(x)$ is defined as given in Equation (2). That is, we integrate over all points points $x \in \mathcal{S}_{m-1}(r)$ weighted by $p(x)$. Indeed, this is an integration, but you can avoid doing the mathematical integration by utilizing the results from previous questions. Write the equation $\rho_m(r)$ for the integrated density of sampled points from the Gaussian distribution lying on the surface of $\mathcal{S}_{m-1}(r)$.

5. *[5 points]* For large m, show that $\rho_m(r)$ has a single maximum value at $\hat{r}$ such that $\hat{r} \approx \sqrt{m}\sigma$.

6. *[10 points]* For large m, consider a small value $\epsilon \ll \hat{r}$ (the symbol "$\ll$" means "much less than"), and show that

$$\rho(\hat{r} + \epsilon) \approx \rho(\hat{r}) e^{-\frac{\epsilon^2}{\sigma^2}}. \tag{4}$$

   Hint: during your derivation, first get the expression simplified and close to the desired form. Then use Taylor expansion to get the approximation.

7. *[3 points]* The previous problem shows that $\hat{r}$ is the radius where most of the probability mass is in a high dimension Gaussian, and moreover, as we move away from this radius, say going from $\hat{r}$ to $\hat{r} + \epsilon$, then the total mass becomes smaller exponentially quickly in ϵ. Also, note that since $\hat{r} \approx \sqrt{m}\sigma$, for large m we have $\sigma \ll \hat{r}$ — since σ (the standard deviation) is in low dimensions usually used to indicate where much of the probability mass is, indeed $p(x) > p(x')$ whenever $\|x\| < \|x'\|$ with the highest density value being $p(0)$ at the origin 0. When we get to high dimensions, however, most of the mass is far away from the σ neighborhood around the origin. This means that most of the probability

mass, in a high dimensional Gaussian, is concentration in a thin skin (e.g., think of the skin of an m-dimensional apple or synthetic leather layer of an m-dimensional soccer ball) of large radius.

If we only get a finite number of samples from a high-dimensional Gaussian distribution, therefore, where do most of the points reside? At what radius do they reside? For a low dimensional Gaussian distribution, where do most points reside?

8. *[7 points]* The conclusion from the previous questions may seem highly counterintuitive, but so can be the curse of dimensionality. Calculate and compare the probability density at the origin and at one point on sphere $\mathcal{S}_{m-1}(\hat{r})$. The curse of dimensionality comes from the extremely high growth rate of volume as the dimensionality of the space increases (there's just a lot of room in high dimensions).

   Write a python script that samples from an m-dimensional Gaussian. For each $m \in \{1, 2, \ldots, 40\}$, produce 100 samples and compute the mean and standard deviation of the radii of the samples, and plot this as a function of m. Is your plot consistent with the above? Why or why not? Fully understand what you see, and clearly explain to us that you understand it and how you justify it.

3 Ridge Regression [15 points]

Recall that linear regression solves

$$\min_w \|Xw - y\|_2^2, \tag{5}$$

Where X is the data matrix, and every row of X corresponds to a data point, y refers to the vector of labels, and w is the weight vector we aim to optimize. Specifically, X is an $n \times d$ data matrix with n data samples (rows) and d features (columns), and y is an n-dimensional column vector of labels, one for each sample.

Ridge regression is very similar, and it is defined as

$$\min_w \|Xw - y\|_2^2 + \frac{\eta}{2}\|w\|_2^2, \tag{6}$$

where we add an additional regularization term $\frac{\eta}{2}\|w\|_2^2$ to encourage the weights to be small and has other benefits as well which we will discuss in class.

Please make sure you answer every question

1. *[3 points]* Describe (with drawings and intuitive description) one setting for (X, y), where standard linear regression is preferred over ridge regression. The drawing should show: (1) the data points (X, y); (2) the expected linear regression solution (e.g., a line); (3) expected ridge regression solution (also, e.g., a line). You need to explain the reason for why the standard linear regression is preferred. You need not do any actual calculation here.

2. *[3 points]* Describe (with drawings and intuitive description) one setting for (X, y), where ridge regression is preferable to linear regression. Your answer should fulfill the same requirements in part 1.

3. *[4 points]* Solve for the closed-form solution for ridge regression. To get the closed-form solution, you can set the gradient of the objective function $F(w)$ in the above minimization problem to be zero and solve this equation of w. If you are not familiar with how to compute the gradient (or such derivatives), please refer to Section 2.4 of the Matrix Cookbook (https://www.math.uwaterloo.ca/~hwolkowi/matrixcookbook.pdf) The Matrix Cookbook is extremely helpful for linear algebra.

4. *[5 points]* Now consider two cases: (1) where the columns (or features) of X are more than the rows (or samples); and (2) where the columns (or features) of X are highly correlated (an extreme case is where many features are identical to each other). For each of the above:

   (a) Can you still compute the closed-form solution of the vanilla linear regression?

   (b) With the closed form solution of the previous question and compared to the solution for standard linear regression, do you discover other benefits of ridge regression?

4 Programming Problem: Draw an Ellipsoid [10 points]

General instruction for python: please install anaconda python (python 3.x version is recommended) by following the instructions at https://www.anaconda.com/download/ if you have not done so, and then install scikit-learn, numpy, matplotlib, seaborn, pandas and jupyter notebook in anaconda, for example, by running command "conda install seaborn". Note some of the above packages may have already been installed in anaconda, depending on which version of anaconda you just installed. You may also use Colab directly.

We provide an ipython notebook "draw_ellipsoid.ipynb" for you to complete. You can use any functions from numpy, scikit-learn, and plotting libraries.

1. *[2 points]* Write a function that takes in a list of 3D points and plot them. After you finish the function, draw the following list of points $[(0,0,0),(0,1,0),(0,0,1),(0,1,1)]$.

2. *[8 points]* Draw half of an ellipsoid. An ellipsoid is a high-dimensional ellipse. Note that an ellipsoid is mathematically defined as $x^T Mx = c$, with $c > 0$ and M positive definite. We've provided a randomly generated positive definite matrix M of dimension 3×3 in the python notebook. Please draw the 3D ellipse with M and $c = 300$. You are required to generate at least 2500 points that satisfy the ellipsoid equation. Moreover, we ask you to draw only half of the ellipsoid (any half is fine). Feel free to choose the points as long as the point cloud you draw can clearly illustrate the 3D shape of half of the ellipsoid. In addition, it is suggested to annotate your code properly.

5 Programming Problem: Linear Regression [25 points]

In this problem, you will implement the closed-form solvers of linear regression and ridge regression from scratch (which means that you cannot use built-in linear/ridge regression modules in scikit-learn or any other packages). Then you will try your implementation on a small dataset, the Boston housing price dataset, to predict the house prices in Boston ("MEDV") based on some related feature attributes.

We provide an ipython notebook "linear_regression_boston.ipynb" for you to complete. In your terminal, please go to the directory where this file is located, and run the command "jupyter notebook". A local webpage will be automatically opened in your web browser, click the above file to open the notebook. You need to complete the scripts below the "TODOs" (please search for every "TODO"), and submit the completed ipynb file (inside your .zip file). In your write-up, you also need to include the plots and answers to the questions required in this session.

The first part of this notebook serves as a quick tutorial of loading dataset, using pandas to get a summary and statistics of the dataset, using seaborn and matplotlib for visualization, and some commonly used functionalities of scikit-learn. You can explore more functionalities of these tools by yourself. You will use these tools in future homeworks.

1. *[5 points]* Below "1 how does each feature relate to the price" in the ipynb file, we show a 2D scatter plot for each feature, where each point associates with a sample, and the two coordinates are the feature value and the house price of the sample. Please find the top-3 features that are mostly (linearly) related to the house price ("MEDV").

2. *[5 points]* Below "2 correlation matrix", we compute the correlation matrix by pandas, and visualize the matrix by using a heatmap of seaborn. Please find the top-3 features that are mostly (linearly) related to the house price ("MEDV") according to the correlation matrix. Are they the same as the ones in the previous question (sub-question 1)?

3. *[5 points]* Below "3 linear regression and ridge regression", please implement the closed-form solver of linear regression and ridge regression (linear regression with L2 regularization). You can use numpy here. Recap: linear regression solves

$$\min_w F(w) = \|Xw - y\|_2^2, \tag{7}$$

while ridge regression solves

$$\min_w F(w) = \|Xw - y\|_2^2 + \frac{\eta}{2}\|w\|_2^2, \tag{8}$$

where X is an $n \times d$ data matrix with n data samples and d features, and y is an n-dim vector storing the prices of the n data samples, and $F(w)$ is the objective function. Run the linear regression and ridge regression on the randomly split training set, and report the obtained coefficients w. For ridge regression, it is recommended to try different η values.

4. *[5 points]* Below "4 evaluation", implement prediction function and root mean square error

$$RMSE = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2}, \tag{9}$$

where $\hat{y}_i$ is the predicted price and y_i is the true price of sample i. Apply the implementation and report the RMSE of linear regression and ridge regression on the training set and test set. Compare the training RMSE of linear regression and ridge regression, what do you find? How about the comparison of their test RMSE? Can you explain the difference?

5. *[5 points]* Below "5 linear models of top-3 features", train a linear regression model and a ridge regression model by using the top-3 features you achieved in sub-question 2, and then report the RMSE on the training set and test set. Compare the RMSE of using all the 13 features: what is the difference? What does this indicate?

6 Bonus: Locality Sensitive Hashing (LSH) [20 points]

We haven't talked about LSH in detail in class, and this problem serves as a tutorial for LSH. This problem requires a lot of reading and thinking, though the math involved is not particularly hard. **As this is a bonus problem, we suggest you leave it to the end if you have extra time and are interested in the topic.** In general, locality sensitive hashing refers to a special property of hash functions and is closely related to *approximate* nearest neighbor search (NNS), i.e., we find close points to the query instead of exactly the nearest neighbor.

For simplicity, suppose the design matrix $\mathbf{X}$ (which is $n \times m$) consists of only binary features. This means that every data point (i.e., every row of the design matrix) has the form $x_i \in \{0,1\}^m$. Define $d(x_i, x_j)$ as the hamming distance between two data points x_i and x_j, i.e. $d(x_i, x_j) = \sum_{a \in \{0,1,\ldots,m-1\}} |x_i[a] - x_j[a]|$. Hamming distance simply counts the number of positions where the two binary vectors differ, and hamming distance is a proper distance metric (see https://en.wikipedia.org/wiki/Metric_(mathematics) for the full definition of a distance metric).

1. *[5 points]* Imagine you have the following magical oracle. Given a query point $q \in \{0,1\}^m$, and two parameters $r > 0$ and $c \geq 1$,

   (a) If $\exists x \in \mathbf{X}$ such that $d(x, q) \leq r$, the oracle returns to you some point $x' \in \mathbf{X}$ such that $d(x', q) \leq cr$.

   (b) If $\nexists x \in \mathbf{X}$ such that $d(x, q) \leq cr$, the oracle returns to you nothing.

   (c) Otherwise ($\exists x \in \mathbf{X}$ such that $d(x, q) \leq cr$ but $\nexists x \in \mathbf{X}$ such that $d(x, q) \leq r$), the oracle is not stable, and can either return to you some point x' such that $d(x', q) \leq cr$ or return to you nothing.

   Suppose you want to find exactly the nearest neighbor point in $\mathbf{X}$ of the query point q. Using as few times as possible of calling the oracle, how do you appropriately set the values r and c in order to get back the nearest neighbor of q?

2. *[2 points]* Unfortunately, the magical oracle is not free. Suppose we set r to be slightly larger than the distance to the nearest neighbor, i.e. $r = \min_{x \in \mathbf{X}} d(x, q) + \epsilon$, if we set a very large c, the oracle may return to us any data point, which is cheap but not useful. However, if we set a small c, the oracle becomes expensive and returns to us a point with distance at most cr, which may serve as a good approximation to the true nearest neighbor. By setting values of c, we are controlling the trade-off between the quality of the approximate nearest neighbor and the oracle's running time. We will then show how to implement the oracle using locality sensitive hashing functions.

Consider a family of hash functions H, where each function h is associated with a random integer a from $\{0, 1, \ldots, m-1\}$, and given the input x_i, $h(x_i) = x_i[a]$. This hash function is very simple, and merely returns coordinate a of the data point x_i.

Suppose we randomly pick a hash function h from H. For two inputs x_i and x_j, if $d(x_i, x_j) \leq r$ ($r > 0$), what's a lower bound for the probability that h maps x_i and x_j to the same value (i.e. $Pr(h(x_i) = h(x_j))$)? We name this lower bound as p_1.

Similarly, if $d(x_i, x_j) \geq cr$ ($c \geq 1$), what's an upper bound for the probability that h maps x_i and x_j to the same value (i.e. $Pr(h(x_i) = h(x_j))$)? We name this upper bound as p_2.

3. *[2 points]* LSH refers to the following properties. A family of hash functions is (r, c, p_1, p_2)-LSH ($1 \geq p_1 \geq p_2 > 0$, $c > 1$, $r > 0$) if:

   (a) $Pr(h(x_i) = h(x_j)) \geq p_1$ when $d(x_i, x_j) \leq r$;

   (b) $Pr(h(x_i) = h(x_j)) \leq p_2$ when $d(x_i, x_j) \geq cr$.

   Note h is a randomly sampled hash function from the family of hash functions. Intuitively, when two data points are close ($d(x_i, x_j) \leq r$), the hash function should map them to the same output value with (relatively) high probability at least equal to p_1. If two data points are faraway ($d(x_i, x_j) \geq cr$), the hash function should map them to the same output value with (relatively) low probability at most equal to p_2.

   The very simple hash functions from H introduced above indeed satisfies the LSH property (you can verify by comparing your answers to the two previous questions). With the dataset $\mathbf{X}$, we can first hash all data points using a single function h randomly sampled from H. Then, given a query point q, we hash q with the same function h, and retrieve all data points in $\mathbf{X}$ that get hashed to the same value as $h(q)$ (if any). Finally, we can iterate over the retrieved points (if not empty), and return the point closest to the query point q. As $p_1 \geq p_2$, the hash function is more likely to hash close points into the same value than faraway points.

   However, the difference between p_1 and p_2 might be not significant enough. One simple trick to make close points more probable relative to the faraway points is to sample multiple hash functions from H, and two data points have the same hashed value if all the sampled hash functions give the same value. In other words, we define a new hash function g via the use of k randomly sampled hash functions from H, and we do this by concatenating their output together into a vector of length k.

$$g(x_i) = (h_1(x_i), h_2(x_i), h_3(x_i), \ldots, h_k(x_i)). \tag{10}$$

   Give a lower bound for the probability that $g(x_i) = g(x_j)$ if $d(x_i, x_j) \leq r$ in terms of p_1 and k.
   Give an upper bound for the probability that $g(x_i) = g(x_j)$ if $d(x_i, x_j) \geq cr$ in terms of p_2 and k.
   (Note that $g(x_i) = g(x_j)$ if they are equal on every coordinate of the output vectors. You can also think the binary vector output of g as a binary encoding of an integer, so the output of g becomes an integer value.)

4. *[2 points]* As we increase the value of k, the close data points are more probable relative to the faraway data points. However, we also get lower probability to have two data points hashed to the same value. For large value of k, the probability can be negligible, no two points may get hashed to the same value, and as a result, our algorithm may always return nothing. To alleviate such problems, we may have l instances of g functions, where each g has independent k sampled hash functions from H. Then we hash q with every g function, and collect all data points (if any) that share the same hashed value as $g(q)$. Finally, we iterate over the collected data points from all g functions, and return the one with the least distance.

   Give a lower bound for he probability that $\exists b \in \{0, 1, \ldots, l-1\}$ such that $g_b(x_i) = g_b(x_j)$ if $d(x_i, x_j) \leq r$.
   Give an upper bound for the probability that $\exists b \in \{0, 1, \ldots, l-1\}$ such that $g_b(x_i) = g_b(x_j)$ if $d(x_i, x_j) \geq cr$.

5. *[7 points]* Again, assume we set r appropriately so that there exists some data point $x \in \mathbf{X}$ with $d(x, q) \leq r$.
   Let $\rho = \frac{ln(p_1)}{ln(p_2)}$, $l = n^{\rho}$ and $k = \frac{ln(n)}{ln(1/p_2)}$. Consider the following two events:

   (a) For some $x' \in \mathbf{X}$, $d(x', q) \leq r$, $\exists b \in \{0, 1, \ldots, l-1\}$ such that $g_b(x') = g_b(q)$.

   (b) There are at most $4l$ items in $\mathbf{X}$ where each x in those $4l$ items has $d(x, q) \geq cr$ and for some b, $g_b(x) = g_b(q)$.

   Show the first event happens with probability at least $1 - e^{-1}$ (note that $\lim_{k\to\infty}(1 - 1/k)^k = e^{-1}$ and $(1 - 1/k)^k < e^{-1}$). Show the second event happens with probability at least $3/4$ (HINT: use Markov's inequality https://en.wikipedia.org/wiki/Markov%27s_inequality). Give a lower bound on the probability that both events happen.

6. *[2 points]* The two events from the previous question happen with a constant probability. We can then boost the probability of success by running multiple independent instances so that the probability that the two events fail for all instances is negligible. For this problem, assume that the previous two events happen with certainty.

   Recall that we construct l hash functions, and each hash function g consists of k sampled functions from H. Given a query q, we collect all data points in $\mathbf{X}$ if they share the same hashed value for any g_b with the query point. Now we want to iterate over the collected points and report one point as the nearest neighbor. If we only want to return a point that has distance at most cr to the query point q, how many points do we need to check? Are we guaranteed that there is a point with distance at most cr and why?