This homework is due at 11 PM on 18 November 2022.
Submission Format: Your homework submission should consist of a single PDF file that contains all of your answers (any handwritten answers should be scanned).

1. Project Logistics

Fill out this form to let us know whether:

- you plan on doing the project in a group;
- you plan on doing the project by yourself; or
- you are not planning to do the project.

Even if you do not plan to do the project, you must fill out the form.

To get credit for the problem, attach a screenshot of the filled-out form to the PDF you submit to Gradescope.
2. Newton’s Method, Coordinate Descent and Gradient Descent

In this question, we will compare three different optimization methods: Newton’s method, coordinate descent and gradient descent. We will consider the simple set-up of unconstrained convex quadratic optimization; i.e. we will consider the following problem:

$$
\min_{\vec{x} \in \mathbb{R}^d} \vec{x}^T A \vec{x} - 2\vec{b}^T \vec{x} + c
$$

where $A > 0$ and $\vec{b} \in \mathbb{R}^d$.

(a) How many steps does Newton’s method take to converge to the optimal solution? Recall that the update rule for Newton’s method is given by the equation:

$$
\vec{x}_{t+1} = \vec{x}_t - (\nabla^2 f(\vec{x}_t))^{-1} \nabla f(\vec{x}_t).
$$

when optimizing a function $f$.

(b) Now, consider the simple two variable quadratic optimization problem for $\sigma > 0$:

$$
\min_{\vec{x} \in \mathbb{R}^2} f(\vec{x}) = \sigma x_1^2 + x_2^2.
$$

How many steps does coordinate descent take to converge on this problem? Assume that we start by updating the variable $x_1$ in the first step, $x_2$ in step two and so on; therefore, we will update $x_1$ and $x_2$ in odd and even iterations respectively:

$$
(x_{t+1})_1 = \begin{cases} 
\arg\min_{x_1} f(x_1, (x_t)_2) & \text{for odd } t \\
(x_t)_1 & \text{otherwise}
\end{cases} \\
(x_{t+1})_2 = \begin{cases} 
\arg\min_{x_2} f((x_t)_1, x_2) & \text{for even } t \\
(x_t)_2 & \text{otherwise}
\end{cases}
$$

Here, $(x_t)_2$ represents $x_2$ at time $t$ and so on.

(c) We will now analyze the performance of coordinate descent on another quadratic optimization problem:

$$
\min_{\vec{x} \in \mathbb{R}^2} f(\vec{x}) = \sigma (x_1 + x_2)^2 + (x_1 - x_2)^2.
$$

where we have, as before, $\sigma > 0$. Note that $(0, 0)$ is the optimal solution to this problem. Now, starting from the point $(1, 1)$, how many steps does coordinate descent take to converge to $(0, 0)$? What happens when $\sigma$ grows large? HINT: First find the update rule for $x_1$, i.e. keep $x_2$ fixed and figure out how $x_1$ changes when $t$ is odd. Then do the same for $x_2$ when $x_1$ is fixed.

(d) Finally, for the objective function from the previous part, how long does gradient descent take to converge to $(0, 0)$ starting from the point $(1, -1)$? Assume for this part that $\sigma > 1$ and reason about how many steps it takes for gradient descent to converge when $\sigma$ grows large. HINT: What is the step size for gradient descent? HINT: Also note that $f$ is given by:

$$
f(\vec{x}) = \vec{x}^T A \vec{x} \text{ where } A = 2 \begin{pmatrix} \frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} \\
\frac{1}{\sqrt{2}} & -\frac{1}{\sqrt{2}} \end{pmatrix} \begin{pmatrix} \frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} \frac{1}{\sqrt{2}} & -\frac{1}{\sqrt{2}} \end{pmatrix}.
$$
3. Gradient Descent vs Newton Method

Run the Jupyter notebook `gradient_vs_newton.ipynb` which demonstrates differences between gradient descent and Newton’s method.
4. LASSO vs. Ridge

Say that we have the data set \( \{(\vec{x}^{(i)}, y^{(i)})\}_{i=1,...,n} \) of features \( \vec{x}^{(i)} \in \mathbb{R}^d \) and values \( y^{(i)} \in \mathbb{R} \).

Define \( X = [\vec{x}^{(1)} \ldots \vec{x}^{(n)}]^\top \) and \( y = [y^{(1)} \ldots y^{(n)}]^\top \).

For the sake of simplicity, assume that the data has been centered and whitened so that each feature has mean 0 and variance 1 and the features are uncorrelated, i.e. \( X^\top X = nI \). Consider the linear least squares regression with regularization in the \( \ell_1 \)-norm, also known as LASSO:

\[
\vec{w}^* = \arg\min_{\vec{w} \in \mathbb{R}^d} \|X\vec{w} - \vec{y}\|_2^2 + \lambda \|\vec{w}\|_1. \tag{7}
\]

This problem will compare \( \ell_1 \)-regularization with \( \ell_2 \)-regularization (ridge regression) to understand their similarities and differences. We will do this by looking at the elements of \( \vec{w}^* \) in the solution to each problem.

(a) First, we decompose this optimization problem into \( d \) univariate optimization problems over each element of \( \vec{w} \). Let \( X = [\vec{x}_1 \ldots \vec{x}_d] \) and recall that \( X^\top X = nI \).

(b) If \( w_i^* > 0 \), then what is the value of \( w_i^* \)? What is the condition on \( \vec{y}^\top \vec{x}_i \) for this to be possible?

(c) If \( w_i^* < 0 \), then what is the value of \( w_i^* \)? What is the condition on \( \vec{y}^\top \vec{x}_i \) for this to be possible?

(d) What can we conclude about \( w_i^* \) if \( |\vec{y}^\top \vec{x}_i| \leq \frac{\lambda}{2} \)? How does the value of \( \lambda \) impact the individual entries \( w_i^* \)?

(e) Now consider the case of ridge regression, which uses the the \( \ell_2 \) regularization \( \lambda \|\vec{w}\|_2^2 \).

\[
\vec{w}^* = \arg\min_{\vec{w} \in \mathbb{R}^d} \|X\vec{w} - \vec{y}\|_2^2 + \lambda \|\vec{w}\|_2^2. \tag{8}
\]

Write down the new condition for \( w_i^* \) to be 0. How does this differ from the condition obtained in part (4) and what does this suggest about LASSO?
5. More Fun with Lasso and Ridge

Complete the Jupyter notebook `ridge_vs_lasso.ipynb` which demonstrates differences between ridge regression and LASSO.
6. Wasserstein distance between distributions

The Wasserstein distance is a measure of distance between probability distributions. The Wasserstein distance can roughly be thought of as the cost of turning one distribution to another distribution by moving probability mass around from one location to another. It is also sometimes called the earth-mover distance, because it may be visualized as the cost of moving a pile of dirt from one configuration to another.

![Visualization of μ histogram on left and ν histogram on right.](image)

**Figure 1:** Visualization of μ histogram on left and ν histogram on right.

Let \( n \in \mathbb{N} \). We define two discrete probability distributions \( \mu = (\mu_1, \cdots, \mu_n) \) and \( \nu = (\nu_1, \cdots, \nu_n) \); that is, \( \mu_i, \nu_i \geq 0 \) and \( \sum_i \mu_i = \sum_i \nu_i = 1 \).

We define \( C \in \mathbb{R}^{n \times n} \) to be a cost matrix where \( c_{ij} \geq 0 \) is the cost of transporting one unit of probability mass from location \( i \in \{1, \cdots, n\} \) to location \( j \in \{1, \cdots, n\} \). We define a matrix \( M \in \mathbb{R}^{n \times n} \) where \( m_{ij} \geq 0 \) denotes the quantity of probability mass to be moved from location \( i \) to location \( j \). In summary, if we move \( m_{ij} \) units of probability mass from location \( i \) to location \( j \), we incur cost \( c_{ij}m_{ij} \).

In addition, the \( M \) matrix satisfies the following conditions. Row \( i \) of \( M \) indicates where all the probability mass in location \( i \) in the \( \mu \) distribution ends up. Hence, the sum of all the entries in row \( i \) must equal \( \mu_i \). Similarly, column \( j \) indicates where all the probability mass in location \( j \) in the \( \nu \) distribution came from. Hence, the sum of all the entries in column \( j \) must equal \( \nu_j \). We can summarize these conditions in math:

\[
\begin{align*}
M\vec{1} &= \mu, \\
M^\top \vec{1} &= \nu,
\end{align*}
\]  

\( \vec{1} \) is a vector of 1s.

(a) What is the total cost of transporting the mass \( \mu \) into \( \nu \) by following the transportation plan dictated by the matrix \( M \)?

(b) Given the cost matrix \( C \), write the optimization problem of finding the transportation plan \( M^* \) with minimal total cost. What type of optimization problem is it? (LP, QP, \cdots?)

Now, we apply the idea of Wasserstein distance to document similarity as illustrated in Fig. 2. Here, our application is that we want to identify words in two different documents that are most similar. This is mostly just a fun application, but may be of interest if you are trying to compare documents that are identical but in different languages. Here we consider a contrived example.
Natural Language Processing techniques have standard tools for converting words into vectors and embedding them in vector spaces, so that we can use machine learning and optimization tools on them. One such embedding is called word2vec. Assume we are provided with a word2vec embedding for the words in two documents. The word travel cost $c_{ij}$ between word $i$ and word $j$ is the Euclidean distance $\|x_i - x_j\|_2$ in the word embedding space. We can compute the similarity between two documents as the minimum cumulative cost required to move all non-stop words from one document to the other.

**Figure 2:** An illustration of the Wasserstein distance. All non-stop words (bold) of both documents are embedded into a word embedding space. The similarity between the two documents is the minimum cumulative distance that all words in document 1 need to travel to exactly match document 2.

(c) Using the `text_kantorovich.ipynb` Jupyter notebook, implement the calculation of the Wasserstein distance in the notebook and use the provided code to visualize the resulting matrix $M$. Comment on the results.
7. **Homework Process**

With whom did you work on this homework? List the names and SIDs of your group members.

*NOTE*: If you didn’t work with anyone, you can put “none” as your answer.