CS 189/289A Introduction to Machine Learning
Spring 2025 Jonathan Shewchuk HW4

Due: Wednesday, March 12 at 11:59 PM PST

- Homework 4 consists of coding assignments and math problems.
- 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.
- In all of the questions, **show your work**, not just the final answer.
- **We will not provide points back with respect to homework submission errors.** This includes, but is not limited to: 1) not assigning pages to problems; 2) not including code in the write-up appendix; 3) not including code in the "HW4 Code" Gradescope assignment; 4) not including Kaggle scores; 5) submitting code that only partially works; 6). submitting late regrade requests. **Please carefully read and follow the HW submission guidelines/reminders on Pages 1, 2, and 11 of HW 4.**
- **Start early; you can submit models to Kaggle only twice a day!**

Deliverables:

1. Submit your predictions for the test sets to Kaggle as early as possible. Include your Kaggle scores in your write-up. The Kaggle competition, the data for this assignment, AND a helper script for generating a submission CSV file can be found at
   - WINE: https://www.kaggle.com/t/248cd3234ea0486288e649be6cf75512
2. Write-up: Submit your solution in **PDF** format to "Homework 4 Write-Up" in Gradescope.
   - On the first page of your write-up, please list students with whom you collaborated
   - Start each question on a new page. If there are graphs, include those graphs on the same pages as the question write-up. DO NOT put them in an appendix. We need each solution to be self-contained on pages of its own.
   - **Only PDF uploads to Gradescope will be accepted**. You are encouraged use LaTeX or Word to typeset your solution. You may also scan a neatly handwritten solution to produce the PDF.
   - **Replicate all your code in an appendix**. Begin code for each coding question in 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 appendix to correct questions.

- While collaboration is encouraged, *everything* in your solution must be your (and only your) creation. Copying the answers or code of another student is strictly forbidden. Furthermore, all external material (i.e., *anything* outside lectures and assigned readings, including figures and pictures) should be cited properly. We wish to remind you that consequences of academic misconduct are *particularly severe*!

3. Code: Submit your code as a .zip file to "Homework 4 Code".
   - **Set a seed for all pseudo-random numbers generated in your code.** This ensures your results are replicated when readers run your code. For example, you can seed `numpy` with `np.random.seed(189)`.
   - Include a README with your name, student ID, the values of random seed (above) you used, and instructions for running (and compiling, if appropriate) your code.
   - Do NOT provide any data files. Supply instructions on how to add data to your code.
   - Code requiring exorbitant memory or execution time might not be considered.
   - Code submitted here must match that in the PDF Write-up. The Kaggle score will not be accepted if the code provided a) does not compile or b) compiles but does not produce the file submitted to Kaggle.

Notation: In this assignment we use the following conventions.

- Symbol "defined equal to" ($\triangleq$) *defines* the quantity to its left to be the expression to its right and is equivalent to :=.
- Scalars are lowercase non-bold: x, u_1, α_i. Matrices are uppercase alphabets: A, B_1, C_i. Vectors (column vectors) are in bold: $\mathbf{x}, \boldsymbol{\alpha_1}, \mathbf{X}, \mathbf{Y_j}$.
- $\|\mathbf{v}\|$ denotes the Euclidean norm (length) of vector $\mathbf{v}$: $\|\mathbf{v}\| \triangleq \sqrt{\mathbf{v} \cdot \mathbf{v}}$. $\|A\|$ denotes the (operator) norm of matrix A, the magnitude of its largest singular value: $\|A\| = \max_{\|v\|=1} \|Av\|$.
- $[n] \triangleq \{1, 2, 3, \ldots, n\}$. $\mathbf{1}$ and $\mathbf{0}$ denote the vectors with all-ones and all-zeros, respectively.

1 Honor Code

Declare and sign the following statement (Mac Preview, PDF Expert, and FoxIt PDF Reader, among others, have tools to let you sign a PDF file):

"I certify that all solutions are entirely my own and that I have not looked at anyone else's solution. I have given credit to all external sources I consulted."

Signature: _____

2 Multiclass Asymmetric Bayes Decision Theory

Let's apply Bayes decision theory to three-class classification with an asymmetric loss function. Consider the Giga-Gauss system of exoplanets that we newly discovered, where we want to classify each exoplanet as a gas giant, super-Earth, or terrestrial. Based on our expert scientists' previous classifications of exoplanets, we must predict the exoplanet type based on their radial velocity. Concretely:

- The input X is a scalar value representing the radial velocity of an exoplanet, with five discrete levels: 20, 40, 60, 80, and 100 (meters/second).
- We must predict one of three classes Y corresponding to the type of exoplanet. $Y = y_0$ means gas giant, y_1 means super-Earth, and y_2 means terrestrial.
- The priors for each class are: $P(Y = y_0) = 0.3$, $P(Y = y_1) = 0.6$, and $P(Y = y_2) = 0.1$.
- Our scientists have measured the radial velocity for closer exoplanets, with data for 100 gas giants, 100 super-Earths, and 100 terrestrials. From this analysis, they estimated the class-conditional probability mass functions $P(X|Y)$:

Radial Velocity (X)	Gas Giant, $P(X\|Y = y_0)$	Super-Earth, $P(X\|Y = y_1)$	Terrestrial, $P(X\|Y = y_2)$
20	0.6	0.3	0.1
40	0.2	0.2	0.1
60	0.1	0.2	0.1
80	0.1	0.2	0.2
100	0	0.1	0.5

- We use an asymmetric loss. Let $\hat{y}$ by the predicted class and y be the true class (label).

$$L(\hat{y}, y) = \begin{cases} 0 & \hat{y} = y, \\ 1 & y = y_0 \text{ and } \hat{y} \neq y_0, \\ 3 & y = y_1 \text{ and } \hat{y} \neq y_1, \\ 6 & y = y_2 \text{ and } \hat{y} \neq y_2. \end{cases}$$

1. Consider the constant decision rule $r_0(x) = y_0$, which *always* predicts y_0 (gas giant). What is the risk $R(r_0)$ of the decision rule r_0? Your answer should be a number, but **show all your work**.

2. Derive the Bayes optimal decision rule $r^*(x)$—the rule that minimizes the risk $R(r^*)$.

   Hint: Write down a table calculating $L(\hat{y}, y_i)P(X|Y = y_i)P(Y = y_i)$ for each class y_i and each possible value of X (15 values total), in the cases where the prediction $\hat{y}$ is wrong. Then figure out how to use it to minimize R. This problem can be solved without wasting time computing $P(X)$.

Problem 2

1) Constant Decision Rule $r_0(x) = \gamma_0$ always predicts γ_0 (gyas giant)

$$R(r_0) = \sum_{\gamma} L(\gamma_0, \gamma)\, P(\gamma)$$

Prior Dist for each class:

$$P(\gamma = \gamma_0) = \frac{3}{10} \qquad P(\gamma = \gamma_1) = \frac{6}{10} \qquad P(\gamma = \gamma_2) = \frac{1}{10}$$

Loss Function $L(\gamma_0, \gamma)$:

$$L(\gamma_0, \gamma_0) = 0 \qquad L(\gamma_0, \gamma_1) = 1 \qquad L(\gamma_0, \gamma_2) = 3$$

Expected Loss or $R(r_0)$ for decision rule r_0

$$= (0 \cdot \tfrac{3}{10}) + (1 \cdot \tfrac{6}{10}) + (3 \cdot \tfrac{1}{10})$$

$$= \frac{9}{10}$$

↳ $R(r_0) = \frac{9}{10}$

2) Bayes optimal decision rule $r^*(x)$ min risk $R(r^*)$

Bayes optimal rule:

$$R(r|x) = \sum_{r} L(r, r') P(r'|x)$$

Posterior Probabilities:

$$P(r|x) = \frac{P(x|r) P(r)}{P(x)}$$

Total probabilities:

$$P(x) = P(x|r_0) P(r_0) + P(x|r_1) P(r_1) + P(x|r_2) P(r_2)$$

$x = 20$:

$$P(x=20) = \frac{6}{10}\left(\frac{3}{10}\right) + \frac{3}{10}\left(\frac{6}{10}\right) + \frac{1}{10}\left(\frac{1}{10}\right) = 0.37$$

L_0

$$P(r_0|x=20) = \frac{\frac{6}{10}\left(\frac{3}{10}\right)}{0.37} = 0.486$$

$$P(r_1|x=20) = \frac{\frac{3}{10}\left(\frac{6}{10}\right)}{0.37} = 0.486$$

$$P(r_2|x=20) = \frac{\frac{1}{10}\left(\frac{1}{10}\right)}{0.37} = 0.027$$

$X = 40$:

$$P(X=40) = \frac{2}{10}\left(\frac{3}{10}\right) + \frac{2}{10}\left(\frac{6}{10}\right) + \frac{1}{10}\left(\frac{1}{10}\right) = 0.14$$

L_0 $\quad P(Y_0 \mid X=20) = \dfrac{\frac{2}{10}\left(\frac{3}{10}\right)}{0.14} = 0.314$

$$P(Y_1 \mid X=20) = \frac{\frac{2}{10}\left(\frac{6}{10}\right)}{0.14} = 0.672$$

$$P(Y_2 \mid X=20) = \frac{\frac{1}{10}\left(\frac{1}{10}\right)}{0.14} = 0.053$$

$X = 60$:

$$P(X=60) = \frac{1}{10}\left(\frac{3}{10}\right) + \frac{2}{10}\left(\frac{6}{10}\right) + \frac{1}{10}\left(\frac{1}{10}\right) = 0.16$$

L_0 $\quad P(Y_0 \mid X=20) = \dfrac{\frac{1}{10}\left(\frac{3}{10}\right)}{0.16} = 0.1875$

$$P(Y_1 \mid X=20) = \frac{\frac{2}{10}\left(\frac{6}{10}\right)}{0.16} = 0.75$$

$$P(Y_2 \mid X=20) = \frac{\frac{1}{10}\left(\frac{1}{10}\right)}{0.16} = 0.0625$$

$X = 80$:

$$P(X=80) = \frac{1}{10}\left(\frac{3}{10}\right) + \frac{2}{10}\left(\frac{6}{10}\right) + \frac{2}{10}\left(\frac{1}{10}\right) = 0.17$$

L_0 $\quad P(Y_0 \mid X=20) = \dfrac{\frac{1}{10}\left(\frac{3}{10}\right)}{0.17} = 0.171$

$$P(Y_1 \mid X=20) = \frac{\frac{2}{10}\left(\frac{6}{10}\right)}{0.17} = 0.702$$

$$P(Y_2 \mid X=20) = \frac{\frac{3}{10}\left(\frac{1}{10}\right)}{0.17} = 0.118$$

$X = 100$:

$$P(X=100) = 0\left(\frac{3}{10}\right) + \frac{1}{10}\left(\frac{6}{10}\right) + \frac{5}{10}\left(\frac{1}{10}\right) = 0.11$$

↳ $$P(Y_0 \mid X=20) = \frac{\frac{0}{10}\left(\frac{3}{10}\right)}{0.11} = 0$$

$$P(Y_1 \mid X=20) = \frac{\frac{1}{10}\left(\frac{6}{10}\right)}{0.11} = 0.545$$

$$P(Y_2 \mid X=20) = \frac{\frac{5}{10}\left(\frac{1}{10}\right)}{0.11} = 0.455$$

Expected Loss !

$X = 20$:

$$R(Y_0 \mid X=20) = 0(0.480) + 1(0.486) + 3(0.027) = 0.567$$

$$R(Y_1 \mid X=20) = 0.648 \qquad R(Y_2 \mid X=20) = 4.374$$

↳ $R(Y_0 \mid X)$ lowest → $r^*(20) = Y_0$ gas giving

$X = 40$:

$$R(Y_0 \mid X=40) = 0(0.316) + 1(0.632) + 3(0.05) = 0.791$$

$$R(Y_1 \mid X=40) = 0.634 \qquad R(Y_2 \mid X=40) = 4.74$$

↳ $R(Y_0 \mid X)$ lowest → $r^*(40) = Y_1$ (super-Bowl)

$X = 60$:

$R(Y_0 | x=60) = 0(0.1875) + 1(0.75) + 3(0.0625) = 0.9375$

$R(Y_1 | x=60) = 0.5625$ $\quad$ $R(Y_2 | x=60) = 5.0625$

↳ $R(Y_0|X)$ lowest → $r^*(60) = Y_1$ (Super Ecuth)

$X = 80$:

$R(Y_0 | x=80) = 0(0.176) + 1(0.706) + 3(0.118) = 1.06$

$R(Y_1 | x=80) = 0.824$ $\quad$ $R(Y_2 | x=80) = 4.764$

↳ $R(Y_0|X)$ lowest → $r^*(80) = Y_1$ (Super-Ecuth)

$X = 100$:

$R(Y_0 | x=100) = 0(0) + 1(0.545) + 3(0.455) = 1.91$

$R(Y_1 | x=100) = 2.73$ $\quad$ $R(Y_2 | x=100) = 3.27$

↳ $R(Y_0|X)$ lowest → $r^*(100) = Y_0$ (gas-giant)

3 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

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

where $s_i \triangleq s(\mathbf{x}_i \cdot \mathbf{w})$, $\mathbf{w} \in \mathbb{R}^d$ is a weight vector, and $s(\gamma) \triangleq 1/(1 + e^{-\gamma})$ is the logistic function.

Define the $n \times d$ design matrix X (whose i^{th} 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

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

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 $\{\lambda_1, ..., \lambda_d\}$. An alternative notation is $\Lambda = \text{diag}(\lambda_i)$, where diag() takes as input the list of diagonal 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.

$$\nabla_{\mathbf{x}} \alpha \mathbf{y} = \big(\nabla_{\mathbf{x}} \alpha\big)\mathbf{y}^\top + \alpha \nabla_{\mathbf{x}} \mathbf{y} \qquad \nabla_{\mathbf{x}} \big(\mathbf{y} \cdot \mathbf{z}\big) = \big(\nabla_{\mathbf{x}} \mathbf{y}\big)\mathbf{z} + \big(\nabla_{\mathbf{x}} \mathbf{z}\big)\mathbf{y};$$
$$\nabla_{\mathbf{x}} \mathbf{f}(\mathbf{y}) = \big(\nabla_{\mathbf{x}} \mathbf{y}\big)\big(\nabla_{\mathbf{y}} \mathbf{f}(\mathbf{y})\big); \qquad \nabla_{\mathbf{x}} g(\mathbf{y}) = \big(\nabla_{\mathbf{x}} \mathbf{y}\big)\big(\nabla_{\mathbf{y}} g(\mathbf{y})\big);$$

and $\nabla_{\mathbf{x}} C\mathbf{y}(\mathbf{x}) = \big(\nabla_{\mathbf{x}} \mathbf{y}(\mathbf{x})\big) C^\top$, *where C 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 for vector $\mathbf{w}$). 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 $\mathbf{x}_1 = [0.2 \ \ 3.1]^\top, \mathbf{x}_2 = [1.0 \ \ 3.0]^\top, \mathbf{x}_3 = [-0.2 \ \ 1.2]^\top, \mathbf{x}_4 = [1.0 \ \ 1.1]^\top$ with labels $y_1 = 1, y_2 = 1, y_3 = 0, y_4 = 0$. These points cannot be separated by a line passing through origin. Hence, as described in lecture, append a 1 to each $\mathbf{x}_{i \in [4]}$ 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 $w^{(0)} = \begin{bmatrix} -1 & 1 & 0 \end{bmatrix}^\top$. 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.

(a) State the value of $\mathbf{s}^{(0)}$ (the initial value of $\mathbf{s}$).

(b) State the value of $\mathbf{w}^{(1)}$ (the value of $\mathbf{w}$ after 1 iteration of Newton's method).

(c) State the value of $\mathbf{s}^{(1)}$ (the value of $\mathbf{s}$ after 1 iteration of Newton's method).

(d) State the value of $\mathbf{w}^{(2)}$ (the value of $\mathbf{w}$ after 2 iterations of Newton's method).

Problem 3

Cost Function Unregularized Log R:

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

Vectorized:

$$J(w) = -y \ln(s) - (1-y) \cdot \ln(1-s)$$

1) Derive Gradient $\nabla_w J(w)$

Established X is a design matrix

$$\nabla_x (y \cdot z) = (\nabla_x y) z + (\nabla_x z) y$$

$$= (\nabla_x z) y$$

$$\rightarrow \nabla_w J(w) = -(\nabla_w \ln s(Xw)) y - (\nabla_w \ln(1-s(Xw))) (1-y)$$

Chain rule: $\nabla_w \ln s(Xw)$ and $\nabla_w \ln(1-s(Xw))$:

i) $\nabla_w \ln s(Xw) = \underbrace{(\nabla_w Xw)}_{= X^T} \underbrace{(\nabla_x s(w))}_{= \text{diagonal}(s_i(1-s_i))} \underbrace{(\nabla_s \ln(s))}_{= \text{diagonal}(\frac{1}{s_i})}$

ii) $\nabla_w \ln(1-s(Xw)) = \underbrace{(\nabla_w Xw)}_{= X^T} \underbrace{(\nabla_x s(w))}_{= \text{diagonal}(s_i(1-s_i))} \underbrace{(\nabla_s \ln(1-s))}_{= \text{diagonal}\left(-\frac{1}{1-s_i}\right)}$

Putting it together:

$$\nabla_w S(w) = \underbrace{-\left(\nabla_w \ln s(xw)\right) y}_{= x^T \cdot \text{diagnol}(s_i(1-s_i)) \cdot \text{diagnol}\left(\frac{1}{s_i}\right) = x^T \cdot \text{diagnol}(s_i(1-s_i)) \cdot \text{diagnol}\left(-\frac{1}{1-s_i}\right)} \underbrace{- \left(\nabla_w \ln(1-s(xw))\right)(1-y)}$$

$$= -\left(x^T \cdot \text{diagnol}\left(1-s_i\right) y - x^T \cdot \text{diagnol}\left(s_i\right) \cdot (1-y)\right)$$

$$= -x^T \cdot \text{diagnol}\left(1-s_i\right) y + x^T \cdot \text{diagnol}(s_i) \cdot (1-y)$$

$$= \underbrace{-x^T \cdot \text{diagnol}\left(1-s_i\right) y}_{-x^T I y} + \underbrace{x^T \cdot \text{diagnol}(s_i) \cdot 1}_{x^T \cdot S}$$

$$= -x^T y + x^T S$$

$$= x^T (S - y)$$

Scratch work

Start

$$y_i \ln(s_i) - (1-y_i) \ln(1-s_i)$$

$$y_i \left(\frac{1}{s_i}\right)\left(\frac{ds_i}{dw}\right) + (1-y_i)\frac{1}{1-s_i}\left(\frac{ds_i}{dw}\right)$$

$$s_i = \frac{1}{1+e^{-x_i^T w}}$$

$$\frac{ds_i}{dw} = \frac{+x_i^T e^{-x_i^T w}}{(1+e^{-x_i^T w})^2}$$

$$\frac{ds_i}{dw} = x_i^T \frac{e^{-x_i^T w}}{(1+e^{-x_i^T w})^2}$$

$$(s_i) \cdot \frac{e^{-x_i^T w}}{(1+e^{-x_i^T w})}$$

Formula as $\frac{1}{1 + e^{-x_i^T w}}$ S

Bound in (0, 1)

$$\frac{e^{-x_i^T w}}{(1 - e^{-x_i^T w})} = \frac{z}{1 + z}$$

we have $\frac{1}{1 + z}$

$$\frac{z}{1 + z} = 1 - \frac{1}{1 + z}$$

$$\frac{1 + z}{1 + z} - \frac{1}{1 + z}$$

$$\downarrow \qquad \frac{+2}{1+2}$$

$$\frac{1-s}{}$$

$$\text{L}_7 \quad \frac{ds_i}{dw} = s(1-s)$$

$$y_i \left(\frac{1}{s_i}\right)\left(\frac{ds_i}{dw}\right) - (1-y_i)\frac{1}{1-s_i}\left(\frac{ds_i}{dw}\right)$$

$$\sum \left(y_i \left(\frac{1}{s_i}\right)(s_i(1-s_i)) - \left((1-y_i)\frac{1}{1-s_i}(s_i(1-s_i))\right)\right.$$

$$\times \left(y_i(1-s_i) - (1-y_i)s_i \right)$$

$$x_i^T \left[y_i - s_i y_i - s_i + y_i s_i \right]$$

$$x_1^T \left(y_1 - s_1 \right)$$

$$S(\text{a.w})$$

$$x^T (2-1)$$

2) Hessian $\nabla_w^2 J(w)$

$$\nabla_w J(w) = X^T(S - Y) = \underbrace{X^T S}_{} - \underbrace{X^T Y}_{0}$$

$$\nabla_w^2 J(w) = \nabla_w X^T S$$

* $\nabla_x (y(x)) = (\nabla_x y(x)) C^T \quad ; \quad C = X^T \quad y = S$

$$\nabla_w^2 J(w) = (\nabla_w S(Xw)) X$$

$$= X^T \text{diagm}(S_i(1 - S_i)) X$$

$$= X^T \left[\text{diagm}(S_i(1 - S_i))\right] X$$

3) Matrix - vector update law

$$w^{t+1} = w^t - (\nabla_w^2 J)^{-1} (\nabla_w J)$$

$$= w^t - \left(X^T \left[\text{diagm}(S_i(1 - S_i))\right] X\right)^{-1} \cdot X^T (Y - S)$$

from parts 3.1 and 3.2

4) Code : q3.py

```
Question 3

Part 4: Calculate Values
For this part, I simply coded what I discovered in part 3, which utilized the collection of parts
1 and 2. After finding all the derivations, I was able to find the formulas for calculating weight
and sigmoid, which ultimately were used to find the desired values for weight and sigmoid. The
values are down below

a) The value(s) of s0 is: [0.94784644 0.88079708 0.80218389 0.52497919]
b) The value(w) of w1 is: [ 1.32465198  3.04991697 -6.82910388]
c) The value(s) of s1 is: [0.94737826 0.97455097 0.03124556 0.10437391]
d) The value(w) of w2 is: [ 1.36602464  4.15753654 -9.19961627]
```

4 Wine Classification with Logistic Regression

The wine dataset `data.mat` (included in the Kaggle competition) consists of 6,000 sample points, each having 12 features. The description of these features is provided in `data.mat`. The dataset includes a training set of 5,000 sample points and a test set of 1,000 sample points. Your classifier needs to predict whether a wine is white (class label 0) or red (class label 1).

Begin by normalizing the data with each feature's mean and standard deviation. You should use training data statistics to normalize both training and validation/test data. Then add a fictitious dimension. Whenever required, it is recommended that you tune hyperparameter values with cross-validation.

Please set a random seed whenever needed and **report it**.

Use of automatic logistic regression libraries/packages is prohibited for this question. If you are coding in python, it is better to use `scipy.special.expit` for evaluating logistic functions as its code is numerically stable, and doesn't produce `NaN` or `MathOverflow` exceptions.

1. *Batch Gradient Descent Update*. State the batch gradient descent update rule for logistic regression **with ℓ_2 regularization**. You must write your rule in vector/matrix notation with no summations. As this is a "batch" algorithm, each iteration should use *every training example*. You don't have to show your derivation. You may reuse results from your solution to question 3.1.

   Hint: Recall that the batch gradient descent rule is

   $$\mathbf{w}^{(t+1)} = \mathbf{w}^{(t)} - \epsilon \nabla f(\mathbf{w}^{(t)}),$$

   where ϵ is the step size, and $f(\mathbf{w})$ is the loss function.

2. *Batch Gradient Descent Code*. Implement your batch gradient descent algorithm for logistic regression and include your code here. Choose reasonable values for the regularization parameter and step size (learning rate), specify your chosen values in the write-up, and train your model from part 1. Shuffle and split your data into training/validation sets and mention the random seed used in the write-up. Plot the value of the cost function versus the number of iterations spent in training.

3. *Stochastic Gradient Descent (SGD) Update*. State the SGD update law for logistic regression with ℓ_2 regularization. Since this is not a "batch" algorithm anymore, each iteration uses *just one* training example. You don't have to show your derivation.

4. *Stochastic Gradient Descent Code*. Implement your stochastic gradient descent algorithm for logistic regression and include your code here. Choose a suitable value for the step size (learning rate), specify your chosen value in the write-up, and run your SGD algorithm from part 3. Shuffle and split your data into training/validation sets and mention the random seed used in the write-up. Plot the value of the cost function versus the number of iterations spent in training.

   Compare your plot here with that of part 2. Which method converges more quickly? Briefly describe what you observe.

5. Instead of using a constant step size (learning rate) in SGD, you could use a step size that slowly shrinks from iteration to iteration. In modern machine learning literature, this kind of decaying learning rate is typically called "learning rate scheduling." Run your SGD algorithm from part 3 with a step size $\epsilon_t = \delta/t$ where t is the iteration number and δ is a hyperparameter you select empirically. Mention the value of δ chosen. Plot the value of cost function versus the number of iterations spent in training.

   How does this compare to the convergence of your previous SGD code?

6. *Kaggle*. Train your *best* classifier on the entire training set and submit your prediction on the test sample points to Kaggle. As always for Kaggle competitions, you are welcome to add or remove features, tweak the algorithm, and do pretty much anything you want to improve your Kaggle leaderboard performance **except** that you may not replace or augment logistic regression with a wholly different learning algorithm. Your code should output the predicted labels in a CSV file.

   Report your Kaggle username and your best score, and briefly describe what your best classifier does to achieve that score.

Problem 4

1) Batch gradient descent update rule

X: Design matrix w/ ith row corresponding to sample points x_i^T.

w^t: weight at step t

$$w^{t+1} = w^t - \varepsilon\left(\lambda w^t - X^T(y - s(Xw^t))\right)$$

2) Implementing code

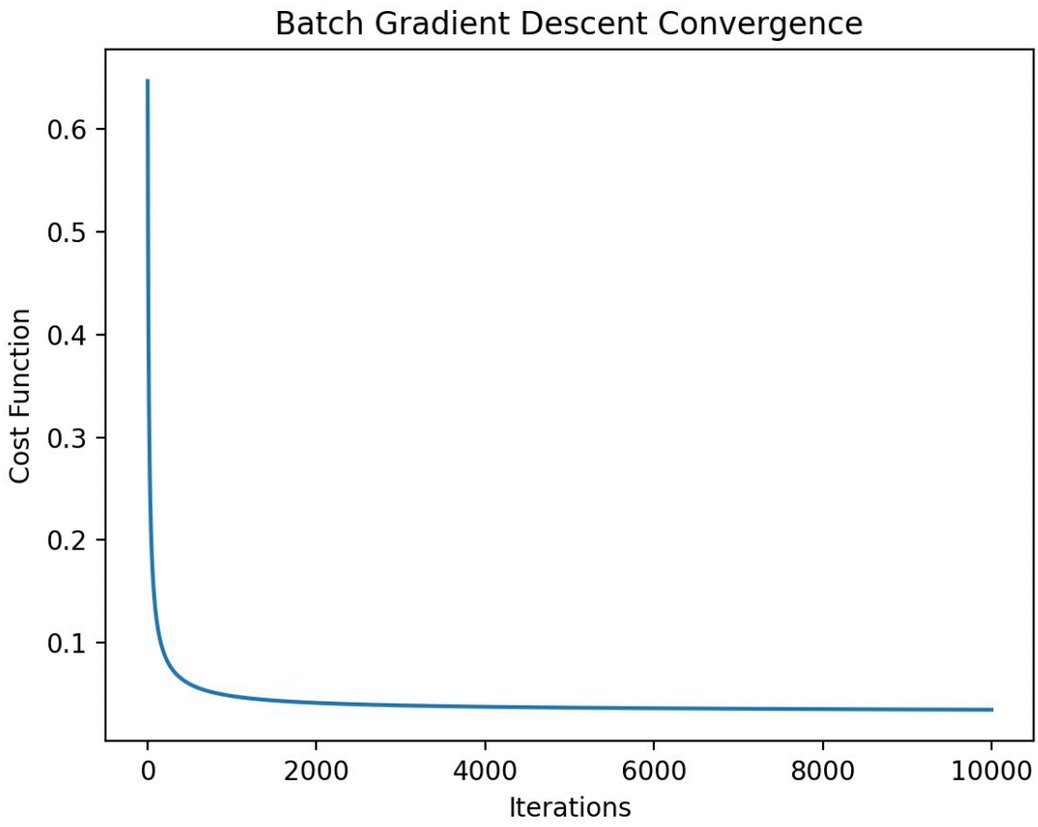
```
Question 4
Note: Set the random seed to be 7, my favorite number
Note2: Used the same Kaggle function given from homework 1


Part 2: Batch Gradient Descent

Utilizing the knowledge from questions 2 and 3, I was able to use the theory of the BGD and code it from scratch. Based on the code I was able to create, I found
the following data about the validation accuracy and the plot of the error.

Validation Accuracy =  0.994
p2 graph: ![Part 2 Graph](q4_plots/p2_batch_gd.png)

The visuals matched the eye test as I was looking for the data to plateu at a low training error for the cost function and for the validation accuracy to be higher
than 93% based on the Kaggle Submission
```



3) Stochastic Gradient Descent

X: Design matrix w/ ith row corresponding to sample points x_i^T.

w^t: weight at step t

* SGD picks random point and updates weights based on it in comparison to BGD which looks at it all as a whole

$$\rightarrow \quad w^{t+1} = w^t - \varepsilon\left(\lambda w^t - n\,(y_j - s(X_j \cdot w^t))\,x_j^T\right)$$

$j \sim \text{Uniform}(1, \ldots n)$

4g) Implementing code

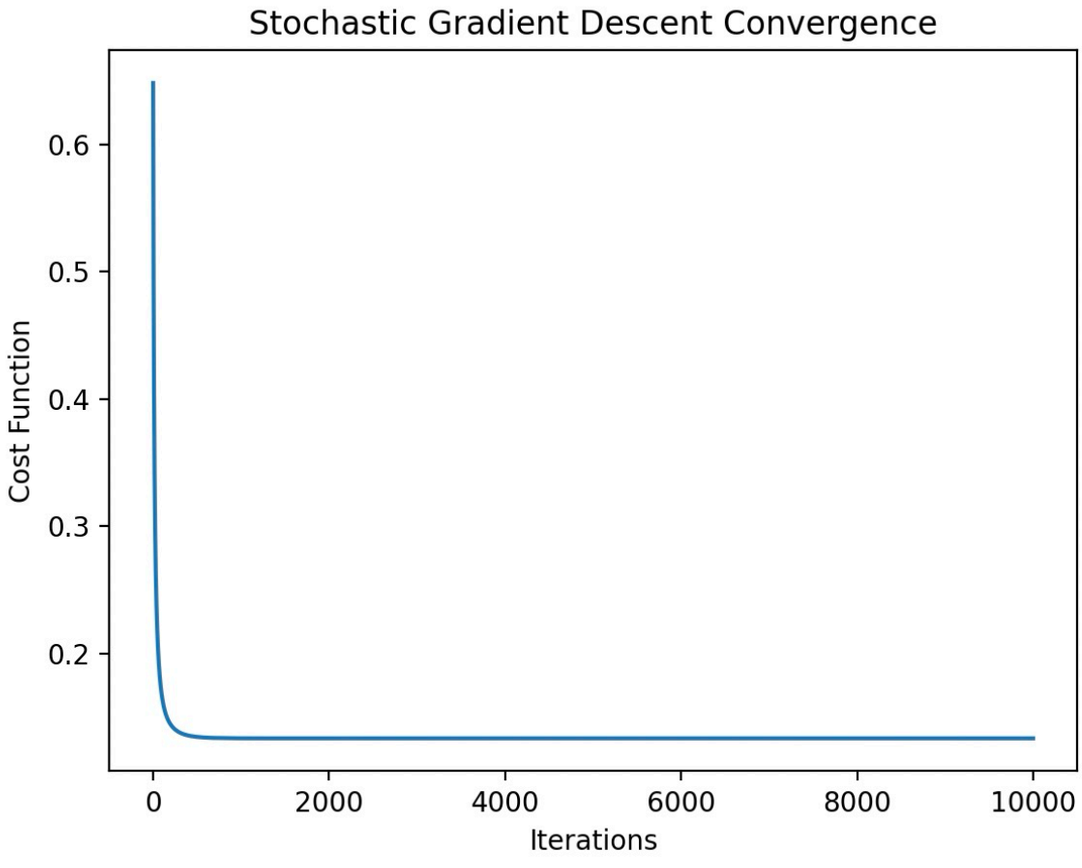
Part 4: Stochastic Gradient Descent

While the set up for the Design matrix and the weight are similar, the derivation of the Stochastic Gradient Descent was found for me slightly different to BGD. SGD picks random points and updates the weights based on it while BGD looks at it as a whole. With this discrepancy, I kept the code for SGD largely similar to the BGD and found the following results.

Validation Accuracy = 0.97875
p4 graph: ![Part 4 Graph](q4_plots/p4_stochastic_gd.png)

The visuals matched the eye test as I was looking for the data to plateu at a low training error for the cost function and for the validation accuracy to be higher than 93% based on the Kaggle Submission

When comparing the two graphs, I found that Batch Gradient Descent converges to a lower training loss than Stochastic Gradient Descent. This is also numerically provable through the validation accuracy of the two gradient descent methods. The validation accuracy for BGD ~ 0.994 and for SGD ~ 0.97875. In total, both the visual graph and the numerical accuracy proves this statement.

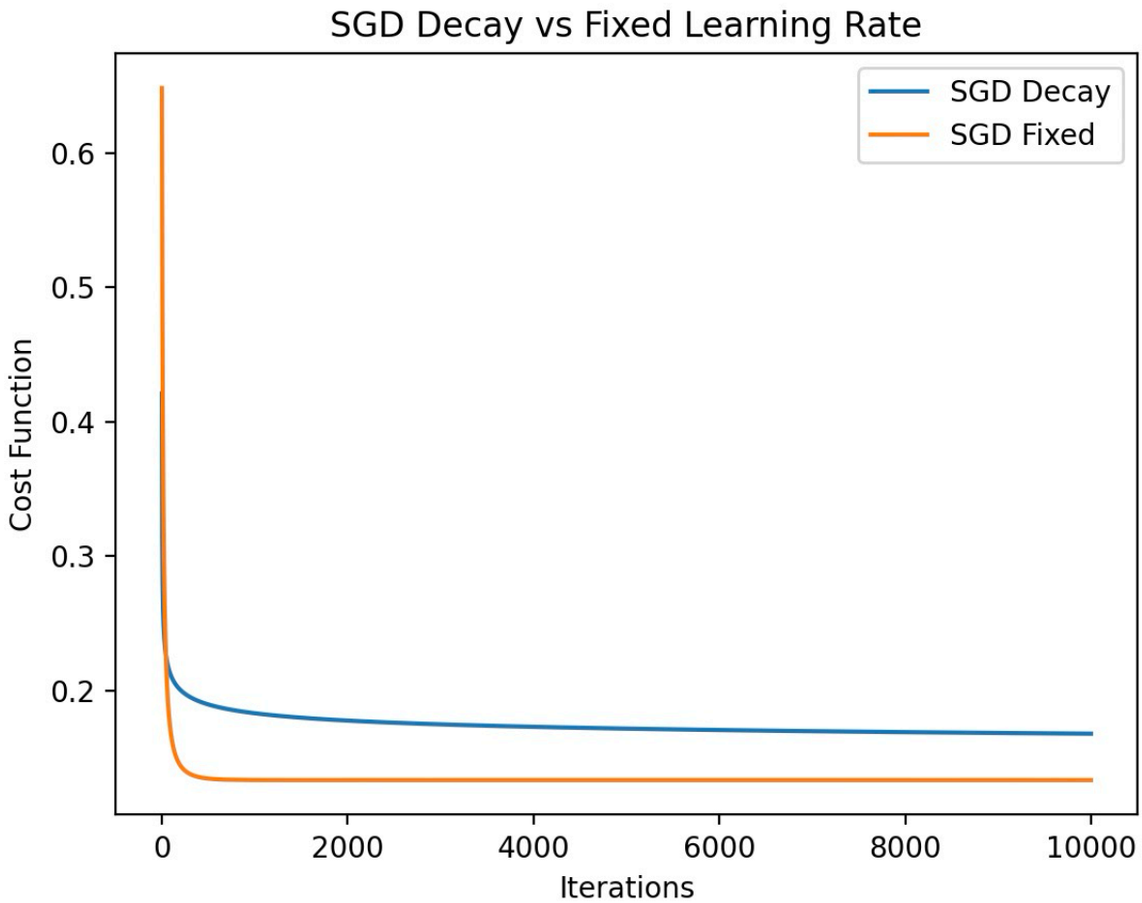


5) Stochastic Decay

```
Part 5: Stochastic Decay vs Fixed

Similarily to Stochastic GD, I used the same class for the decay portion however, creating a different means to fit by instead of using the fixed learning rate,
utilized a decaying learning rate defined by the question. After empirical tuning, I found that 0.001 was a strong example to use that demonstrated a key
discrepancy between fixed and decaying rates. Primarily, through a visual analysis of the graphs, I found that the decay SGD seemed to plateau at a lower training
error than the fixed learning rate. This is additionally reinforced numerically through the validation accuracy comparisons, with Decay being at ~ 0.99025 >> 0.
97875 for the fixed learning rates. However, the convergence of the plots showed that the fixed learning rates converged faster and this is due to the nature of how
these two methods are implemented with decay rates being updated while the learning rate being fixed.

p4: Validation Accuracy =  0.97875
p5: Validation Accuracy with Learning Rate Decay = 0.99025
p5 graph v p4 graph: ![Part 5 Graph](q4_plots/p5_decay_v_fixed.png)
```



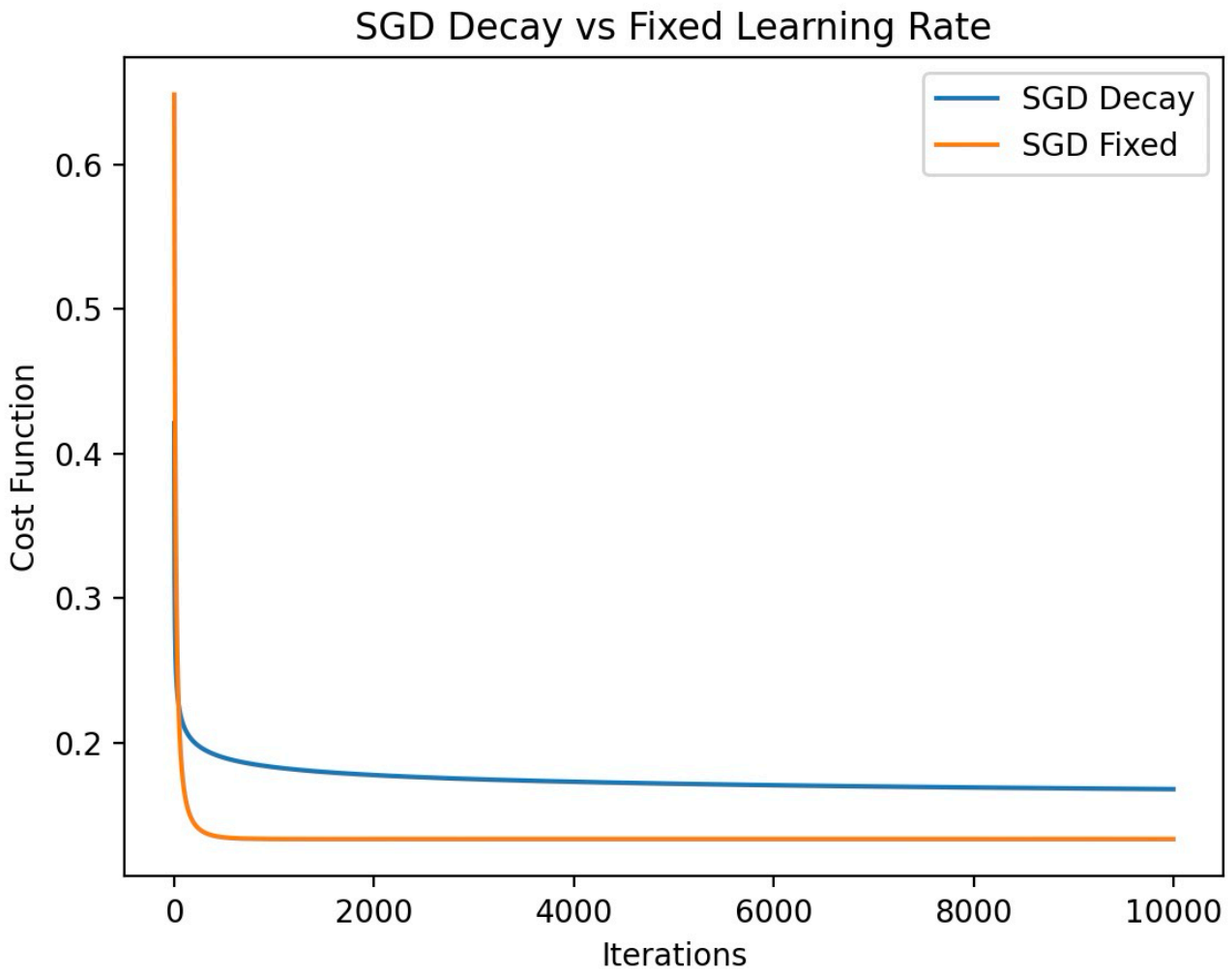
6) Kaggle Submission

```
Part 6: Kaggle
For the Kaggle Submission, I ultimately decided that the Batch Gradient Descent was the most effective means to accomplish test predictions. The reason why is again
stated in part 4 where I compared stochastic and batch gradient descents and their corresponding validation accuracies and graphulated error.

Formally: When comparing the two graphs, I found that Batch Gradient Descent converges to a lower training loss than Stochastic Gradient Descent. This is also
numerically provable through the validation accuracy of the two gradient descent methods. The validation accuracy for BGD ~ 0.994 and for SGD ~ 0.97875. In total,
both the visual graph and the numerical accuracy proves this statement.


Kaggle Submission View:
user: dankimchi0430
Image: ![Part 6 Kaggle Submission](Kaggle_Submission/hw4_Kaggle_Submission.png)
```

Kaggle_Submission_Grad_Desc_Batch.csv Complete · 27s ago · Daniel Kim Batch Gradient Descent Kaggle Submission #1	**1.0000**	☑



5 A Bayesian Interpretation of Lasso

Suppose you are aware that the labels $y_{i\in[n]}$ corresponding to sample points $\mathbf{x}_{i\in[n]} \in \mathbb{R}^d$ follow the density law

$$f(y_i|\mathbf{x}_i, \mathbf{w}) = \frac{1}{\sigma\sqrt{2\pi}} e^{-(y_i - \mathbf{w}\cdot\mathbf{x}_i)^2/(2\sigma^2)}$$

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 $\mathbf{w}_i$ obeys the density law $f(\mathbf{w}_i) = e^{-|\mathbf{w}_i|/b}/(2b)$.

Assume the outputs $y_{i\in[n]}$ 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 $(\mathbf{x}_i, y_i)_{i\in[n]}$. 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 $f(\mathbf{w}|(\mathbf{x}_i, y_i)_{i\in[n]})$ for $\mathbf{w}$ *up to a proportionality constant* by applying Bayes' Theorem and substituting for the densities $f(y_i|\mathbf{x}_i, \mathbf{w})$ and $f(\mathbf{w})$. Don't try to derive an exact expression for $f(\mathbf{w}|(\mathbf{x}_i, y_i)_{i\in[n]})$, as the denominator is very involved and irrelevant to maximum likelihood estimation.

2. Define the log-likelihood for MAP as $\ell(\mathbf{w}) \triangleq \ln f(\mathbf{w}|\mathbf{x}_{i\in[n]}, y_{i\in[n]})$. Show that maximizing the MAP log-likelihood over all choices of $\mathbf{w}$ is the same as minimizing $\sum_{i=1}^n (y_i - \mathbf{w}\cdot\mathbf{x}_i)^2 + \lambda\|\mathbf{w}\|_1$ 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.

Problem 5

Bayes theorem:

$$f(w \mid (x_i, y_i)_{i \in n}) = \frac{f(\{y_i\}_{i \in n} \mid w, \{x_i\}_{i \in n}) f(w)}{f(\{y_i\}_{i \in n} \mid \{x_i\}_{i \in n})}$$

since y_i's are independent

$$f(\{y_i\}_{i \in n} \mid w, \{x_i\}_{i \in n}) = \prod f(y_i \mid w, x_i)$$

$$= (\sigma\sqrt{2\pi})^{-n} \exp\left(- \sum_{i=1}^{n} \frac{(y_i - w \cdot x_i)^2}{2\sigma^2} \right)$$

since weights are independent

$$f(w) = \prod_{j \in J} f(w_j)$$

$$= (2b)^{-2} e^{-\sum_{j=0}^{2} |w_j|/b}$$

Denominator is a constant.

$$f(w \mid x_i, y_i)_{i \in n} \propto e^{-\sum_{i=1}^{n} (y_i - w \cdot x_i)^2 / 2\sigma^2} \cdot e^{-\sum_{j=1}^{2} |w_j|/b}$$

2) Det Log-likelihood for MAP

* utilizing 3.1 :

$$\ln(w) = \ln(f(w \mid x_{i:n}, y_{i:n}))$$

$$= -\frac{\sum_{i=1}^{n}(y_i - w \cdot x_i)^2}{2\sigma^2} - \frac{\|w\|_1}{b} + \ln(C)$$

proportionality constant

Formula ↗

$$= -\sum_{i=1}^{n}(y_i - w \cdot x_i)^2 - \frac{2\sigma^2}{b}\|w\|_1 + \ln(C)$$

$$\lambda$$

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

6 ℓ_1-regularization, ℓ_2-regularization, and Sparsity

You are given a design matrix X (whose i^{th} row is sample point $\mathbf{x}_i^\top$) and an n-vector of labels $\mathbf{y} \triangleq [y_1 \ \ldots \ y_n]^\top$. For simplicity, assume X is whitened, so $X^\top X = nI$. Do not add a fictitious dimension/bias term; for input $\mathbf{0}$, the output is always 0. Let $\mathbf{x}_{*i}$ denote the i^{th} column of X.

1. The ℓ_p-norm for $w \in \mathbb{R}^d$ is defined as $\|w\|_p = (\sum_{i=1}^d |w_i|^p)^{1/p}$, where $p > 0$. Plot the isocontours with $w \in \mathbb{R}^2$, for the following norms.
   (a) $\ell_{0.5}$ (b) ℓ_1 (c) ℓ_2
   Use of automatic libraries/packages for computing norms is prohibited for the question.

2. Show that the cost function for ℓ_1-regularized least squares, $J_1(\mathbf{w}) \triangleq \|X\mathbf{w} - \mathbf{y}\|^2 + \lambda\|\mathbf{w}\|_1$ (where $\lambda > 0$), can be rewritten as $J_1(\mathbf{w}) = \|\mathbf{y}\|^2 + \sum_{i=1}^d f(\mathbf{x}_{*i}, \mathbf{w}_i)$ where $f(\cdot,\cdot)$ is a suitable function whose first argument is a vector and second argument is a scalar.

3. Using your solution to part 2, derive necessary and sufficient conditions for the i^{th} component of the optimizer $\mathbf{w}^*$ of $J_1(\cdot)$ to satisfy each of these three properties: $w_i^* > 0, w_i^* = 0$, and $w_i^* < 0$.

4. For the optimizer $\mathbf{w}^\#$ of the ℓ_2-regularized least squares cost function $J_2(\mathbf{w}) \triangleq \|X\mathbf{w} - \mathbf{y}\|^2 + \lambda\|\mathbf{w}\|^2$ (where $\lambda > 0$), derive a necessary and sufficient condition for $\mathbf{w}_i^\# = 0$, where $\mathbf{w}_i^\#$ is the ith component of $\mathbf{w}^\#$.

5. A vector is called *sparse* if most of its components are 0. From your solution to part 3 and 4, which of $\mathbf{w}^*$ and $\mathbf{w}^\#$ is more likely to be sparse? Why?

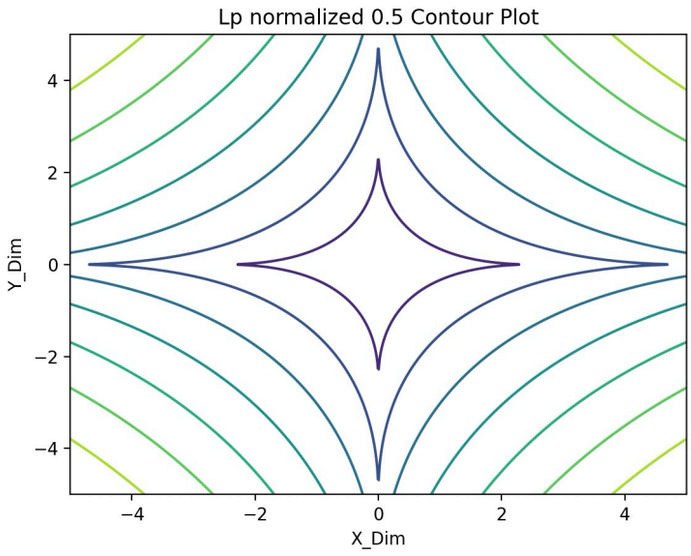
Problem 6

1) Coding + Plots:

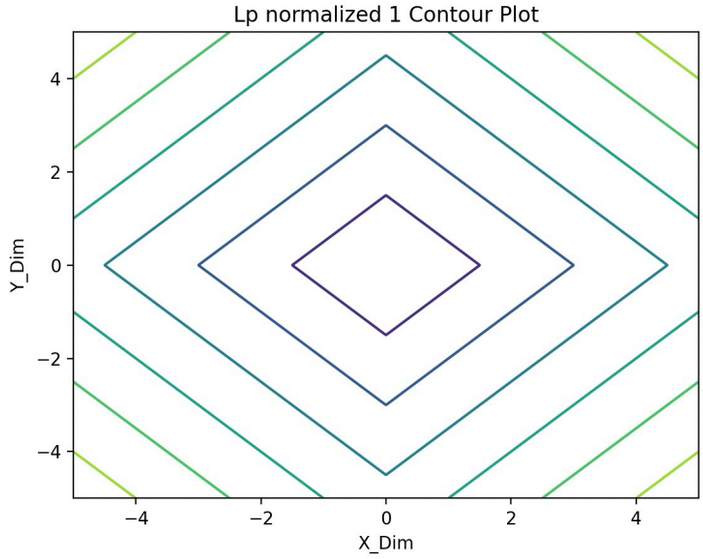
```
Question 6: Regularization + Sparisty
Given the lp norm equation, I simply coded this through the function l_p_norm and built a grid in the X and Y
direction. For each contour plot, the Z dimension would dictate the shapes of the contours through the value
of p asked for each part.

pa: ![Part 1a Graph](q6_plots/q6a.png)
pb: ![Part 1b Graph](q6_plots/q6b.png)
pc: ![Part 1c Graph](q6_plots/q6c.png)
```

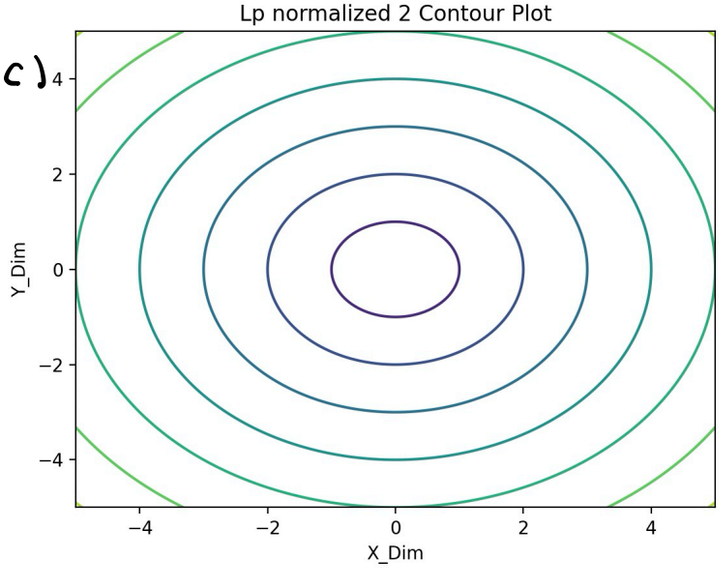
a)



b)



c)



2] ℓ_1-reg LS Rewritten

$$J_1(w) \triangleq \|Xw - y\|^2 + \lambda \|w\|_1$$

$$= w^T X^T X w - 2 y^T X w + \|y\|^2 + \lambda \|w\|_1$$

$$= \|y\|^2 + n\|w\|^2 - 2 y^T X w + \lambda \|w\|_1$$

$$\sum_{j=1}^{d} n w_j^2 \qquad 2 \sum_{j=1}^{d} (x_{\cdot,j}^T y) w_j \qquad \lambda \sum_{j=1}^{d} |w_j|$$

$$= \|y\|^2 + \sum_{j=1}^{d} f(x_{\cdot,j}, w_j)$$

(vec, scalar)

3) Derive for i^{th}

Using the solution from 6.2:

to find w_i^*:

$$g_i(w) = \mathbb{I}(w > 0)(n w^2 - 2 b_i w + \lambda w) + \mathbb{I}(w < 0)(n w^2 - 2 b_i w - \lambda w)$$

a) Condition: $w_i^* > 0$

Given: $g_i(0) = 0$ and $w_i^* > 0$ if:

i) $g_i(w) = -\left(\frac{1}{n}\right)\left(b_i - \frac{\lambda}{2}\right)^2 \left[(b_i - \lambda/2) / w > 0\right]$

ii) $g_i(w) = -\left(\frac{1}{n}\right)\left(b_i + \frac{\lambda}{2}\right)^2 \left[(b_i - \lambda/2) / w < 0\right]$

↳

$$-\left(\frac{1}{u}\right)\left(b_i-\frac{\lambda}{2}\right)^2\left[(b_i-\lambda/2)/u>0\right] \;<\; -\left(\frac{1}{u}\right)\left(b_i-\frac{\lambda}{2}\right)^2\left[(b_i-\lambda/2)/u<0\right]$$

$$\left(b_i-\frac{\lambda}{2}\right)^2\left[(b_i-\lambda/2)/u>0\right] \;<\; \left(b_i-\frac{\lambda}{2}\right)^2\left[(b_i+\lambda/2)/u<0\right]$$

↳ True iff $|b_i-\lambda/2|>0$

b) Condition $w_i^* < 0$

Similar set up to a)

Given: $g_i(0)=0$ and $w_i^*>0$ if:

i) $g_i(u) = -\left(\frac{1}{u}\right)\left(b_i-\frac{\lambda}{2}\right)^2\left[(b_i-\lambda/2)/u>0\right]$

ii) $g_i(u) = -\left(\frac{1}{u}\right)\left(b_i+\frac{\lambda}{2}\right)^2\left[(b_i-\lambda/2)/u<0\right]$

↳

$$-\left(\frac{1}{u}\right)\left(b_i-\frac{\lambda}{2}\right)^2\left[(b_i+\lambda/2)/u>0\right] \;<\; -\left(\frac{1}{u}\right)\left(b_i-\frac{\lambda}{2}\right)^2\left[(b_i-\lambda/2)/u<0\right]$$

$$\left(b_i+\frac{\lambda}{2}\right)^2\left[(b_i+\lambda/2)/u>0\right] \;<\; \left(b_i-\frac{\lambda}{2}\right)^2\left[(b_i-\lambda/2)/u<0\right]$$

↳ True iff $|b_i+\lambda/2|<0$

Conditions 1 and 2 prove Condition 3 $w_i=0$ iff $-\lambda/2 \le b_i \le \lambda/2$

4) optimizer w^*

$$J_2(w) \triangleq \|Xw - y\|^2 + \lambda \|w\|^2$$

$$\nabla_w J_2 = 2nw^* - 2X^T y + 2\lambda w^*$$

$$2X^T y = 2nw^* + 2\lambda w^*$$

$$w^* = \frac{2X^T y}{2(n+\lambda)}$$

$$w^* = \frac{X^T y}{n+\lambda}$$

5) what is more sparse ?

A w^* is more likely. Since $y \cdot x_i$ is more likely to lie in an interval than be a specific value.

Submission Checklist

Please ensure you have completed the following before your final submission.

At the beginning of your writeup...

1. Have you copied and hand-signed the honor code specified in Question 1?
2. Have you listed all students (Names and ID numbers) that you collaborated with?

In your writeup for Question 4...

1. Have you included your **Kaggle Score** and **Kaggle Username**?

At the end of the writeup...

1. Have you provided a code appendix including all code you wrote in solving the homework?

Executable Code Submission

1. Have you created an archive containing all ".py" files that you wrote or modified to generate your homework solutions?
2. Have you removed all data and extraneous files from the archive?
3. Have you included a `README` file in your archive containing any special instructions to reproduce your results?

Submissions

1. Have you submitted your written solutions to the Gradescope assignment titled **HW4 Write-Up** and selected pages appropriately?
2. Have you submitted your executable code archive to the Gradescope assignment titled **HW4 Code**?
3. Have you submitted your test set predictions for **Wine** dataset to the appropriate Kaggle challenge?

Congratulations! You have completed Homework 4.