CS 189 Introduction to Machine Learning
Spring 2024 Jonathan Shewchuk

HW4

Due: Friday, March 8 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 designated 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 for Pages 1, 2, and 10 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 for this assignment can be found at
   - WINE: https://www.kaggle.com/competitions/cs189-hw4-wine-spring-2024/
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.

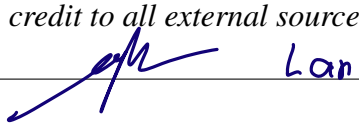
Notation: In this assignment we use the following conventions.

- 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 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 $\mathbf{y} \triangleq [y_1 \quad \ldots \quad y_n]^\top$, and $\mathbf{s} \triangleq [s_1 \quad \ldots \quad s_n]^\top$. For an n-vector $\mathbf{a}$, let $\ln \mathbf{a} \triangleq [\ln a_1 \quad \ldots \quad \ln a_n]^\top$. 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 $\mathbf{w}$).

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

Q2.1

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 $\mathbf{w}$).

Derivative of Logistic function:

$$s(\gamma) = \frac{1}{1+e^{-\gamma}} \; ; \; \frac{ds(\gamma)}{\gamma} = (1-s(\gamma))\, s(\gamma) \quad \text{(from lecture)}$$

- $\nabla_w s(Xw) = \nabla_w (Xw)\, \nabla_\gamma s(\gamma) = X^T \cdot \text{diag}\{s_i(1-s_i)\}$
- $\nabla_w \ln s(Xw) = \nabla_w s(Xw) \cdot \nabla_z \ln(z)$ $\qquad s_i = s(X_i w)$

$$= X^T \text{diag}\{s_i(1-s_i)\} \cdot \text{diag}\left\{\frac{1}{s_i}\right\}$$

$$= X^T \text{diag}\{(1-s_i)\} = X^T[1 - s(Xw)]$$

- $\nabla_w \ln(1-s(Xw)) = \nabla_w (1-s(Xw)) \cdot \nabla_z \ln(1-z)$

$$= -X^T \text{diag}\{s_i(1-s_i)\} \cdot \text{diag}\left\{\frac{1}{1-s_i}\right\}$$

$$= -X^T \text{diag}(s_i) = -X^T s(Xw)$$

We have:

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

$$\nabla_w J(w) = -\{[\nabla_w \ln s(Xw)]\, y + [\nabla_w \ln(1-s(Xw))](1-y)\}$$

$$= -[X^T(1-s(Xw))\, y - X^T \cdot s(Xw)(1-y)]$$

$$= -X^T[y - s(Xw)\cdot y - s(Xw) + s(Xw)\cdot y]$$

$$= -X^T[y - s(Xw)]$$

$$= X^T[s(Xw) - y]$$

Q2.2+Q2.3

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

$$\nabla_w J(w) = X^T(s(Xw) - y) = X^T s(Xw) - X^T y = X^T \operatorname{diag}\{s_i\}\mathbf{1} - X^T y$$

$$\nabla_w^2 J(w) = \nabla_w s(Xw) \cdot X = X^T \operatorname{diag}\{s_i(1-s_i)\} X$$

let $\Omega = \operatorname{diag}\{s_i(1-s_i)\}$

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

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

Newton's method:

$$w^{k+1} \leftarrow w^k + e$$

where $\nabla_w^2 J(w) e = -\nabla_w J(w)$

$$X^T \Omega X e = X^T(y - s(Xw))$$

Assume X has full rank $\Rightarrow X^T \Omega X$ is invertible

$$e = (X^T \Omega X)^{-1} X^T (y - s(Xw))$$

Thus our Newton's method update is:

$$w^{k+1} \leftarrow w^k + (X^T \Omega X)^{-1} X^T (y - s(Xw))$$

Q2.4

4 You are given four examples $\mathbf{x}_1 = [0.2 \quad 3.1]^\top, \mathbf{x}_2 = [1.0 \quad 3.0]^\top, \mathbf{x}_3 = [-0.2 \quad 1.2]^\top, \mathbf{x}_4 = [1.0 \quad 1.1]^\top$ 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).

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

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

$X = [x_1 \quad x_2 \quad x_3 \quad x_4]^T \qquad y = [1 \quad 1 \quad 0 \quad 0]^T$

a) $s^{(0)} = [0.9478 \quad 0.8808 \quad 0.8022 \quad 0.5250]^T$

b) $w^{(1)} = [1.3246 \quad 3.0499 \quad -6.8291]^T$

c) $s^{(1)} = [0.9474 \quad 0.9745 \quad 0.0312 \quad 0.1044]^T$

d) $w^{(2)} = [1.3660 \quad 4.1575 \quad -9.1996]^T$

(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).

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

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

3 Wine Classification with Logistic Regression

The wine dataset `data.mat` 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 law for logistic regression **with ℓ_2 regularization**. 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 2.1.

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 question 3.1. 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 question 3.3. 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 question 3.2. 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. Run your SGD algorithm from question 3.3 with a step size $\epsilon_t = \delta/t$ where t is the iteration number and δ is a hyperparameter you select

Q3.1

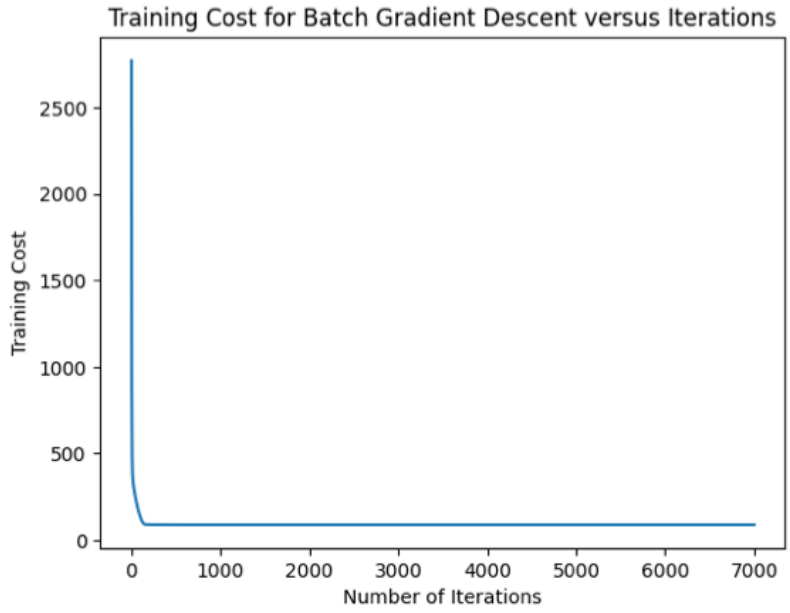
1 *Batch Gradient Descent Update*. State the batch gradient descent update law for logistic regression **with** ℓ_2 **regularization**. As this is a "batch" algorithm, each iteration should use *every training example*. You don't have to show your derivation. You may reuse results from your solution to question 2.1.

$$J(w) := -y \ln s(Xw) - (1-y) \ln(1 - s(Xw)) + \lambda \|w\|_2^2$$

- Batch gradient descent rule:

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

Q3.2



Learning rate: $\epsilon = 0.01$

Regularization parameter: $\lambda = 0.1$

random seed used to split data: 42

Q3.3

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.

Stochastic Gradient Descent (SGD) update:

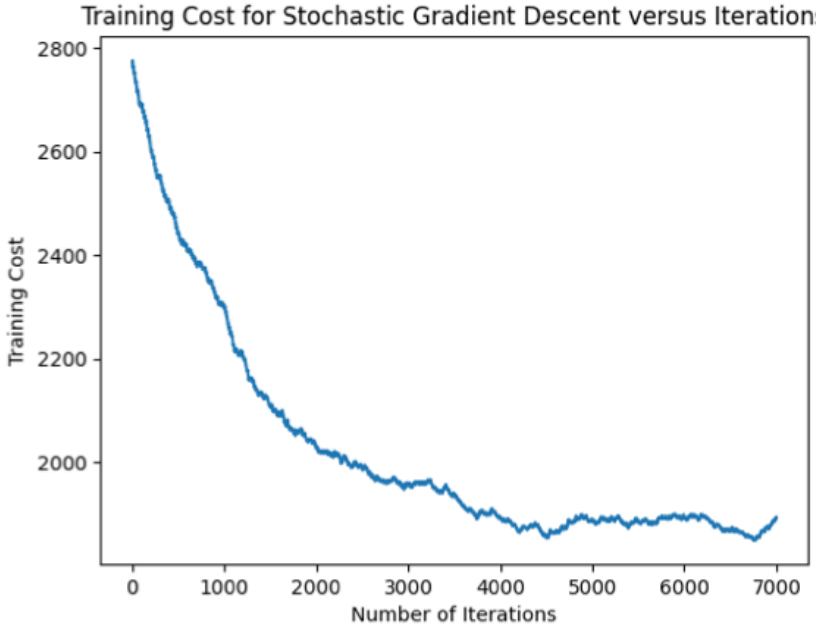
$$w^{t+1} \leftarrow w^t - \varepsilon\left(\left(s(X_i w) - y_i\right)X_i^T + \varrho\lambda w^t\right)$$

for $i \sim \text{Uniform}[\{1, \ldots, n\}]$

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 question 3.3. 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 question 3.2. Which method converges more quickly? Briefly describe what you observe.

3.4

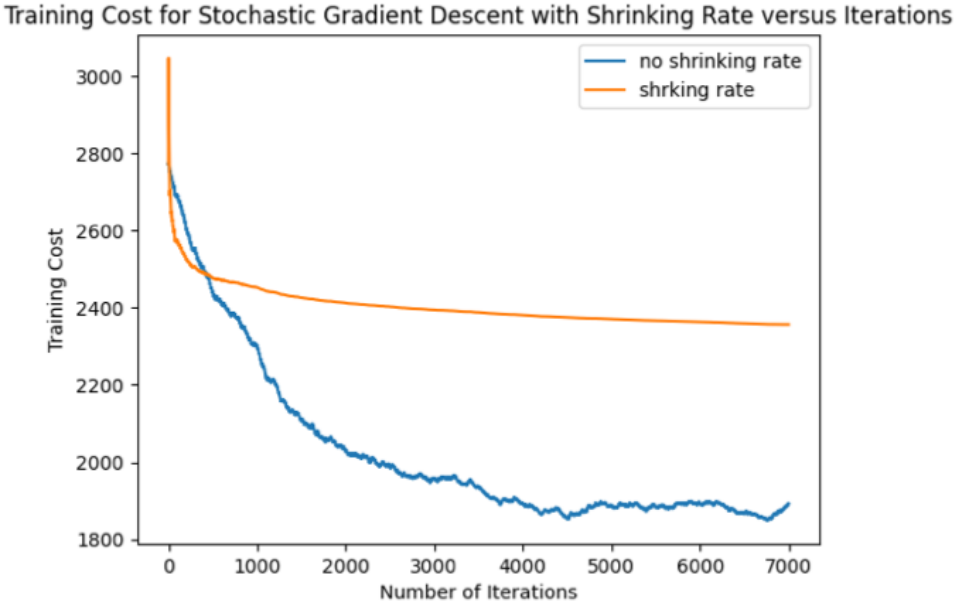


Learning rate: $\epsilon = 0.001$
Regularization parameter: $\lambda = 0.1$
Random seed = 42 , 1000 for validation set, 4000 for training
Compared with question 3.2, the Batch Gradient Descent converges more quickly than the Stochastic Gradient Descent. On first plot, Batch Gradient Descent converges at early iteration ($i < 1000$) while Stochastic Gradient Descent converges at iteration more than 7000.
Additionally, Batch Gradient Descent converges to cost value lower than 100 while Stochastic Gradient Descent converges to cost value around 1800, which is significanly higher

3.5

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. Run your SGD algorithm from question 3.3 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?



Initial learning rate: $\delta = 0.1$
Regularization parameter: $\lambda = 0.1$
Random seed = 42
Compared to my previous SGD code, this new SGD converges to a higher cost, which is around 2400 compared to 1800 cost of the previous SGD). However, it seems that this new SGD converges more quickly than the previous SGD (at around iteration 400)

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.

Q3.6

Kaggle username: hiamlan

Best score: 0.993

I use the learning rate $\epsilon = 0.01$, Batch Gradient Descent, and cross-validation to find the best regularization parameter λ in range $(0.0001; 0.1; \text{step}=0.002)$. The best λ I found is 0.022. Then, I use the optimal weight vector w^* from the Batch Gradient Descent corresponding to $\lambda = 0.022$ for my Logistic Regression model and apply that model to predict labels y in the test set. Lastly, I round up the values to make sure my $\hat{y}$ value is 0 and 1.

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.

4 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.

Q4

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.

By Bayes' Theorem:

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

Sin $y_{i\in[n]}$ are independent:

$$f(\{y_i\}_{i\in[n]}|\{x_i\}_{i\in[n]},w)$$

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

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

And w_j are also independent.

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

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

Because w is not present in denominator, so we will drop it.

→ $f(\{y_i\}_{i\in[n]}|\{x_i\}_{i\in[n]})$ is a constant

Thus, $f(w|(x_i,y_i)_{i\in[n]} \propto f(\{y_i\}_{i\in[n]}|\{x_i\}_{i\in[n]},w)f(w)$

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

$$\propto \exp\left\{-\sum_{i=1}^{n}\frac{(y_i - w\cdot x_i)^2}{2\sigma^2}\right\} \exp\left\{-\sum_{j=1}^{d}\frac{|w_j|}{b}\right\}$$

Q4.2

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.

$$\ell(w) = \ln f(w \mid x_{i\in[n]}, y_{i\in[n]})$$

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

where C is the constant part

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

We have:

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

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

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

$$= \min_w \left(\sum_{i=1}^{n}(y_i - wx_i)^2 + \lambda\|w\|_1\right)$$

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

5 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 $\mathbf{y} \triangleq [y_1 \quad \dots \quad y_n]^\top$. 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 $\mathbf{w}^\#$.

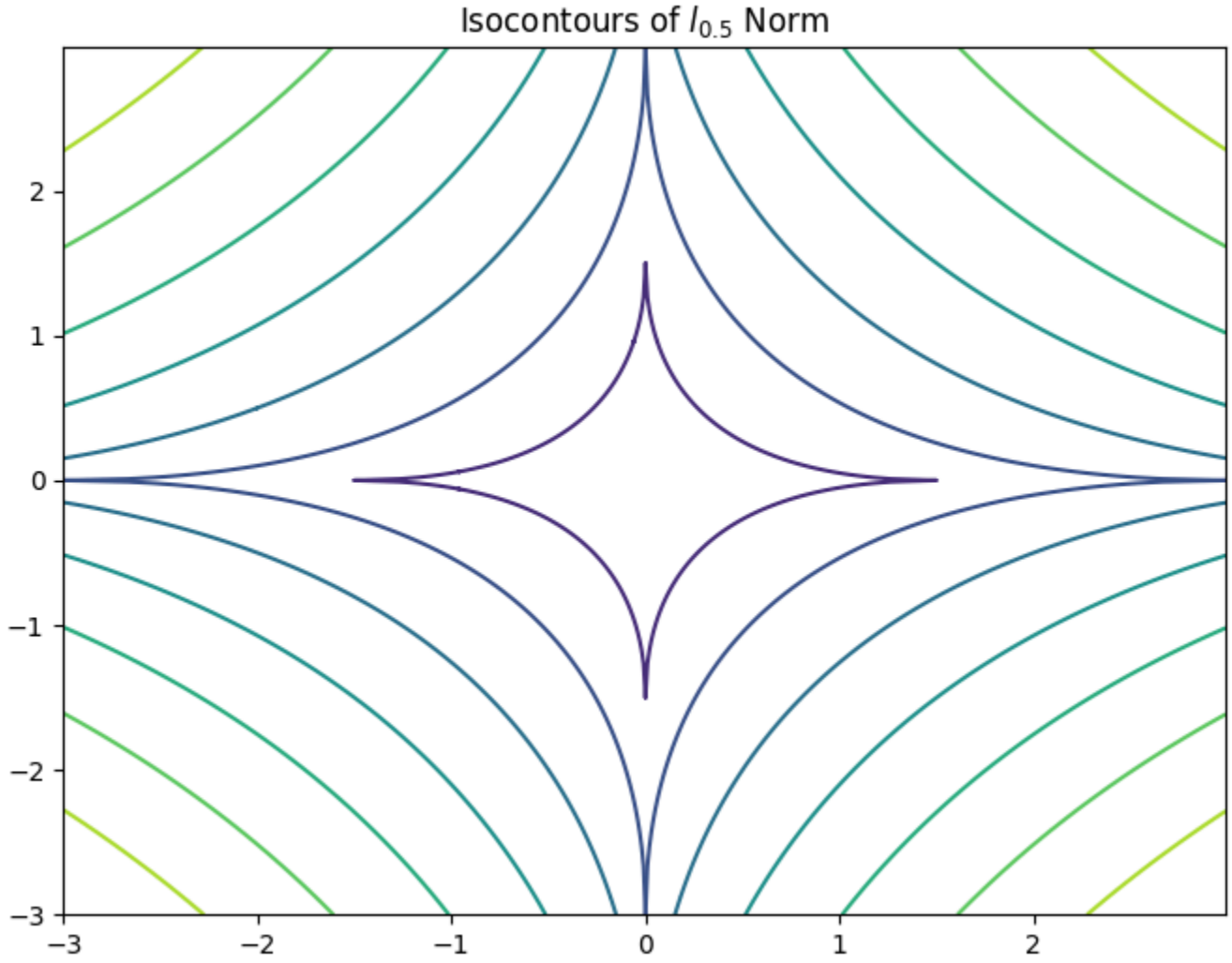
5. A vector is called *sparse* if most of its components are 0. From your solution to part 3 and 4, which of $\mathbf{w}^*$ and $\mathbf{w}^\#$ is more likely to be sparse? Why?

Q5.1

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


def compute_norm(w1, w2, p):
  return (np.abs(w1)**p + np.abs(w2)**p)**(1/p)
```

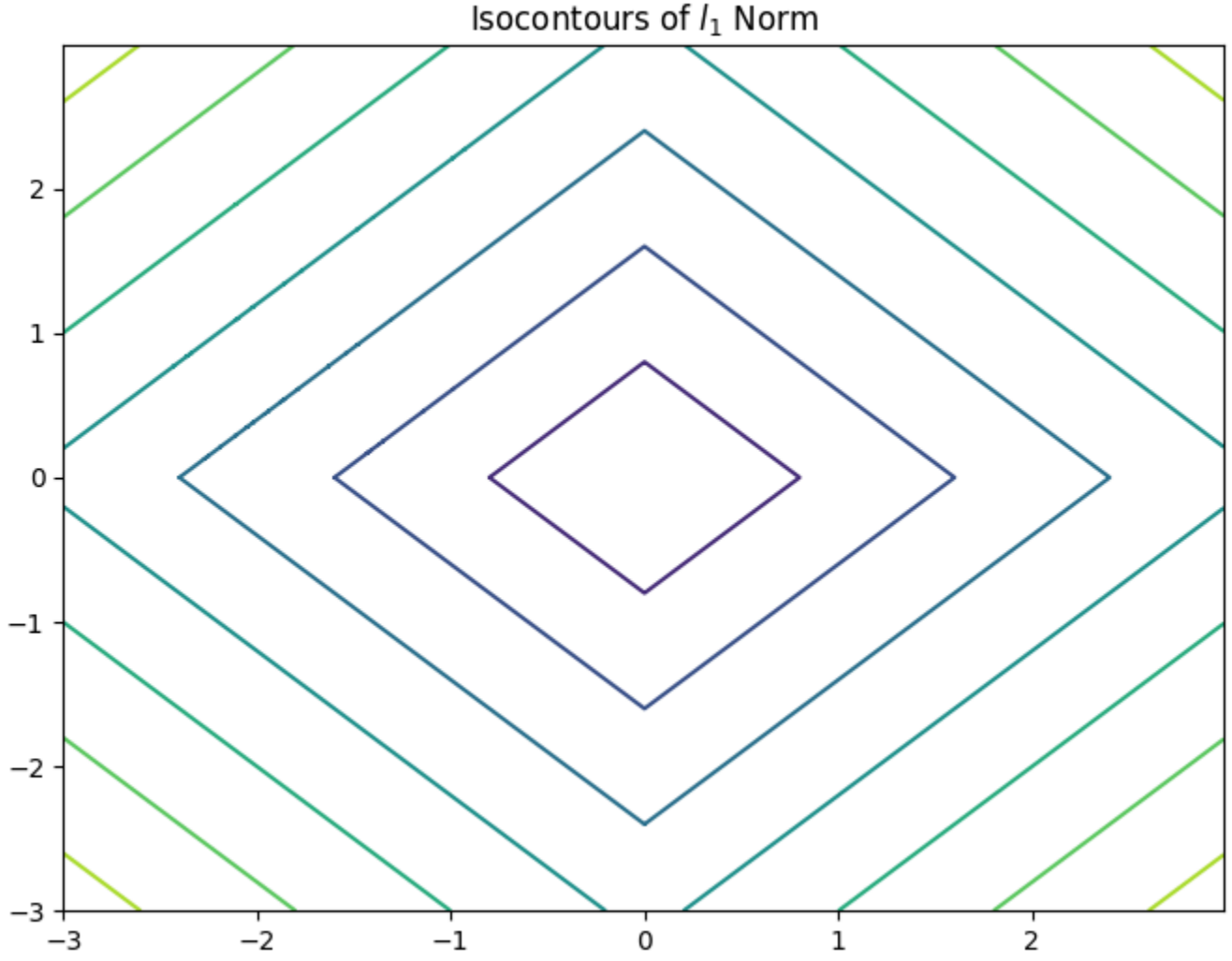
(a) ℓ0.5

```
## l-0.5 norm
w1, w2 = np.mgrid[-3:3:.01, -3:3:.01]
norm_w = compute_norm(w1,w2,0.5)

plt.figure(figsize=(8, 6))
plt.title("Isocontours of $l_{0.5}$ Norm")
plt.contour(w1, w2, norm_w)
plt.show()
```



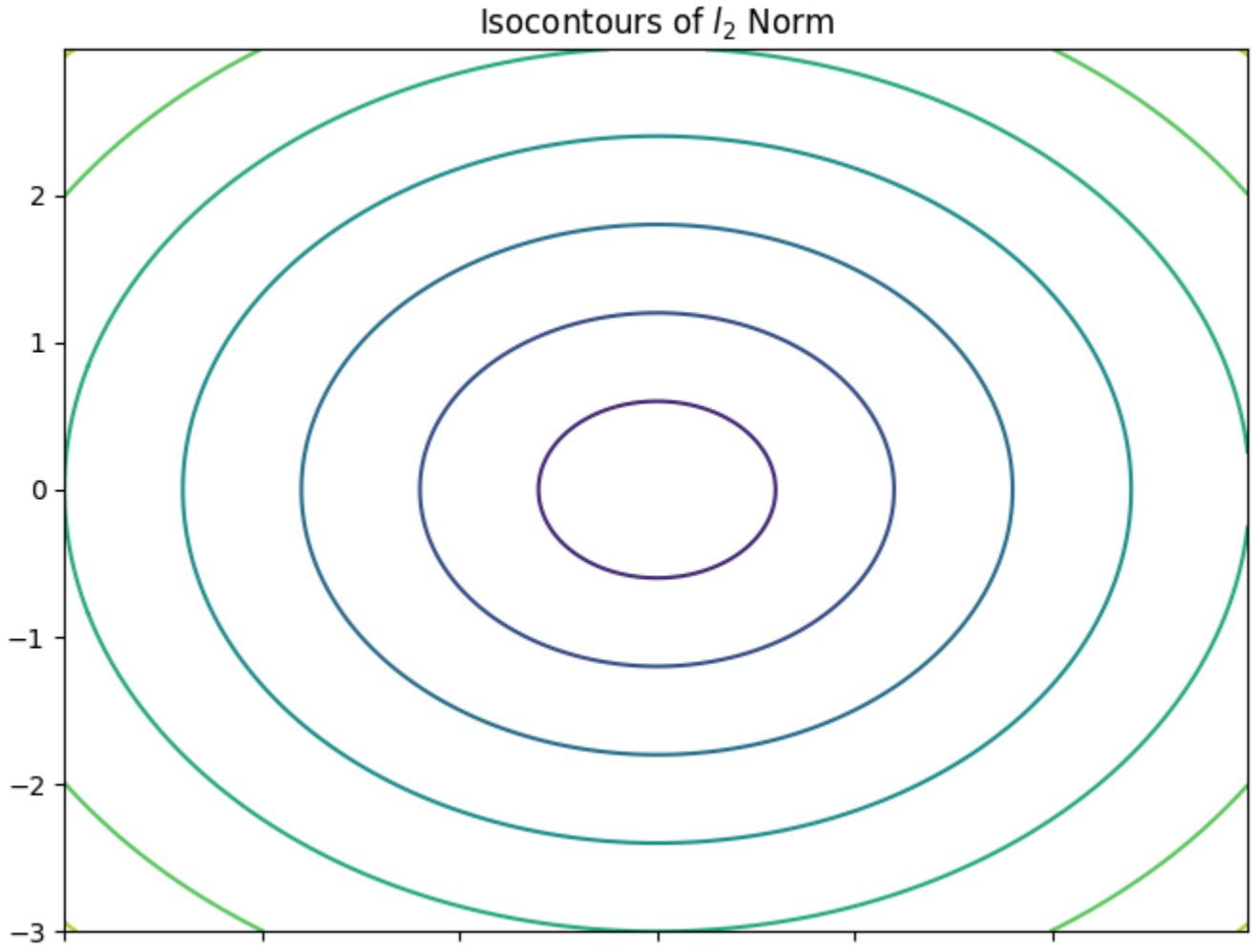
(b) ℓ1

```
## l-1 norm
w1, w2 = np.mgrid[-3:3:.01, -3:3:.01]
norm_w = compute_norm(w1,w2,1)

plt.figure(figsize=(8, 6))
plt.title("Isocontours of $l_{1}$ Norm")
plt.contour(w1, w2, norm_w)
plt.show()
```



(b) ℓ2

```
## l-2 norm
w1, w2 = np.mgrid[-3:3:.01, -3:3:.01]
norm_w = compute_norm(w1,w2,2)

plt.figure(figsize=(8, 6))
plt.title("Isocontours of $l_{2}$ Norm")
plt.contour(w1, w2, norm_w)
plt.show()
```



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.

(Q52)

$$J_1(w) = \|Xw - y\|^2 + \lambda\|w\|_1 \qquad ; \lambda > 0$$

$$= (Xw - y)^T (Xw - y) + \lambda \sum_{i=1}^{d} |w_i|$$

$$= w^T X^T X w - 2\, y^T X w + y^T y + \lambda \sum_{i=1}^{d} |w_i| \qquad (X^T X = nI)$$

$$= n\|w\|^2 - 2\sum_{j=1}^{d} (y \cdot x_{*j}) w_j + \|y\|^2 + \lambda \sum_{j=1}^{d} |w_j|$$

$$J_1(w) = \|y\|^2 + \sum_{i=1}^{n} \underbrace{n \cdot w_i^2 - 2\,(y \cdot x_{*i}) w_i + \lambda |w_i|}_{f(x_{*i},\, w_i)}$$

OR in general:

$$f(\vec{z}, a) = n \cdot a^2 - 2(y \cdot \vec{z}) \cdot a + \lambda |a|$$

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 $w_i^* < 0$.

Q5.3

$J_1(w) = \|y\|^2 + \sum_{i=1}^{n} f(x_{*i}, w_i)$; $f(x_{*i}, w_i) = n \cdot w_i^2 - 2(y \cdot x_{*i}) \cdot w_i + \lambda |w_i|$

• optimize $J_1(w)$ w.r.t w is equivalent to optimize $f(\vec{x}_i, w_i)$ w.r.t w_i

let $k_i := y \cdot x_{*i}$

$f(w_i) = n w_i^2 - 2 k_i \cdot w_i + \lambda |w_i|$

$= \mathbb{I}\{w_i > 0\}(n w_i^2 - 2k_i w_i + \lambda w_i) + \mathbb{I}\{w_i < 0\}(n w_i^2 - 2k_i w_i - \lambda w_i)$

$$\frac{df(w_i)}{dw_i} = \mathbb{I}\{w_i > 0\}(2n w_i - 2k_i + \lambda) + \mathbb{I}\{w_i < 0\}(2n w_i - 2k_i - \lambda)$$

• $w_i^* > 0$: $\frac{df(w_i)}{dw_i} = 0$

$$\Leftrightarrow \quad w_i^* = \frac{2k_i - \lambda}{2n} = \frac{1}{n}\left(k_i - \frac{\lambda}{2}\right) > 0$$

Then $w_i^* > 0 \Leftrightarrow \left(k_i - \frac{\lambda}{2}\right) > 0 \Leftrightarrow y \cdot x_{*i} - \frac{\lambda}{2} > 0$

• $w_i^* < 0$: $\frac{df(w_i)}{dw_i} = 0$

$$w_i^* = \frac{k_i}{n} + \frac{\lambda}{2n} = \frac{1}{n}\left(k_i + \frac{\lambda}{2}\right) < 0$$

Then $w_i^* < 0 \Leftrightarrow k_i + \frac{\lambda}{2} < 0 \Leftrightarrow y \cdot x_{*i} + \frac{\lambda}{2} < 0$

Therefore $w_i^* = 0 \Leftrightarrow \left(y \cdot x_{*i} - \frac{\lambda}{2} \le 0\right)$ AND $\left(y \cdot x_{*i} + \frac{\lambda}{2} \ge 0\right)$

$$\Leftrightarrow \quad -\frac{\lambda}{2} \le y \cdot x_{*i} \le \frac{\lambda}{2}$$

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 $\mathbf{w}^{\#}$.

Q5.4

$$\nabla_w J_2(w) = 2X^T(Xw - y) + 2\lambda w = 0$$

$$X^T X w^{\#} - X^T y + \lambda w^{\#} = 0$$

$$n w^{\#} - X^T y + \lambda w^{\#} = 0$$

$$(n+\lambda) w^{\#} = X^T y$$

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

$$w_i^{\#} = \frac{y \cdot x_i}{n+\lambda}$$

$$w_i^{\#} = 0 \Leftrightarrow \frac{y \cdot x_i}{n+\lambda} = 0 \Leftrightarrow y \cdot x_i = 0$$

Q5.5

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?

w^* is more likely to be sparse because the condition for $w^* = 0 \Leftrightarrow \frac{\lambda}{2} \leq y \cdot x_i \leq \frac{\lambda}{2}$ is an interval whereas the condition for

$w^\# = 0 \Leftrightarrow y \cdot x_i = 0$ is a 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 3...

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.

```
from google.colab import drive
drive.mount('/content/drive')
```

```
Mounted at /content/drive
```

```
!unzip -u "/content/drive/MyDrive/CS 189/hw4.zip" -d /content
```

```
Archive:  /content/drive/MyDrive/CS 189/hw4.zip
```

```
## Locate this notebook inside data
%cd /content/hw4
```

```
/content/hw4
```

CS 189

Homework 2

Submitted by Lan Dinh

```
# import libraries
import matplotlib.pyplot as plt
import pandas as pd
import numpy as np
import scipy
from sklearn.model_selection import train_test_split
```

3. Wine Classification with Logistic Regression

3.1. Batch Gradient Descent Update

```
data = scipy.io.loadmat('data.mat')
```

```
data.keys()
```

```
    dict_keys(['__header__', '__version__', '__globals__', 'y', 'X', 'description', 'X_test'])
```

```
data['description']
```

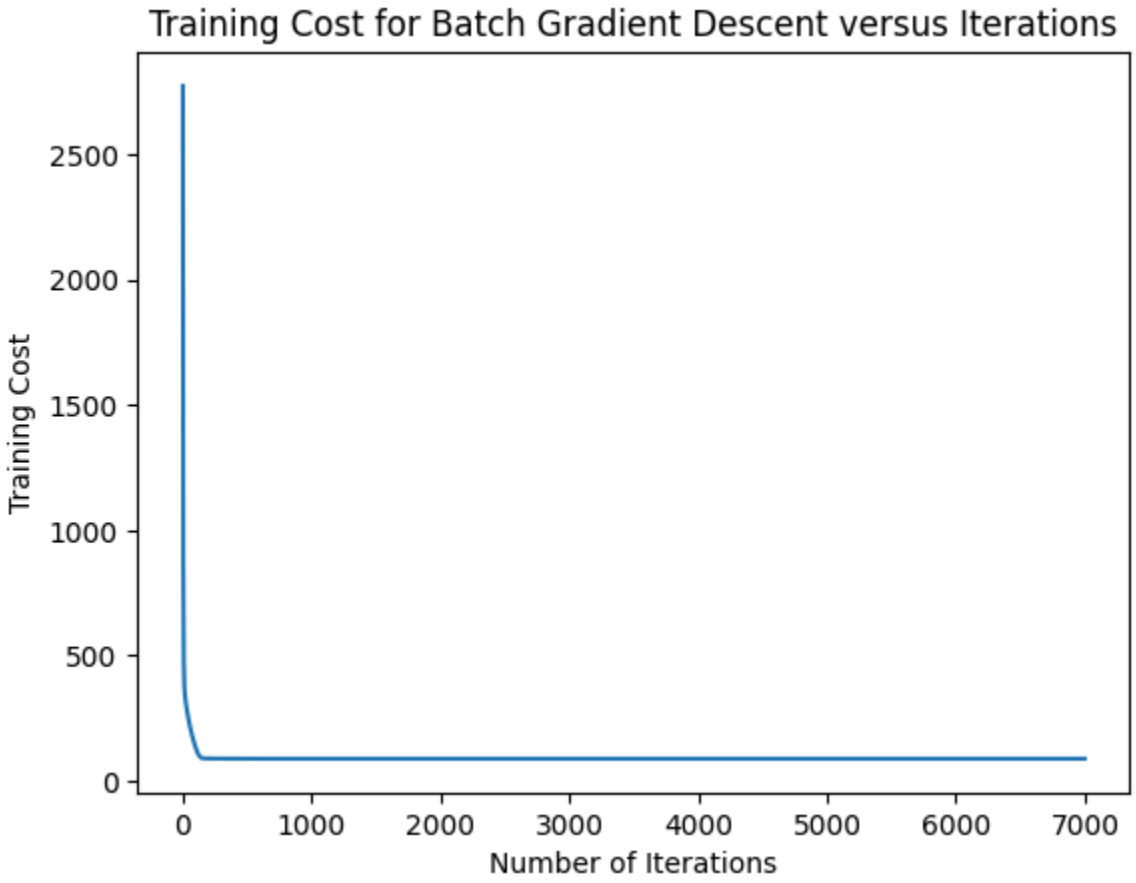
```
    array(['fixed acidity       ', 'volatile acidity    ',
           'citric acid         ', 'residual sugar      ',
           'chlorides           ', 'free sulfur dioxide ',
           'total sulfur dioxide', 'density             ',
           'pH                  ', 'sulphates           ',
           'alcohol             ', 'quality             '], dtype='<U20')
```

```
def normalize():
  return np.mean(X, axis = 0), np.std(X, axis = 0)
#Compute mean and standard deviation based on training data
miu_hat, sigma_hat = normalize()
def pre_process(data, miu, sigma):
  normalized = (data-miu)/sigma
  add_fictitious = np.concatenate((normalized, np.ones((normalized.shape[0],1))),axis=1)
  return add_fictitious
```

```
X = data['X']
y = data['y']
X_test = data['X_test']
```

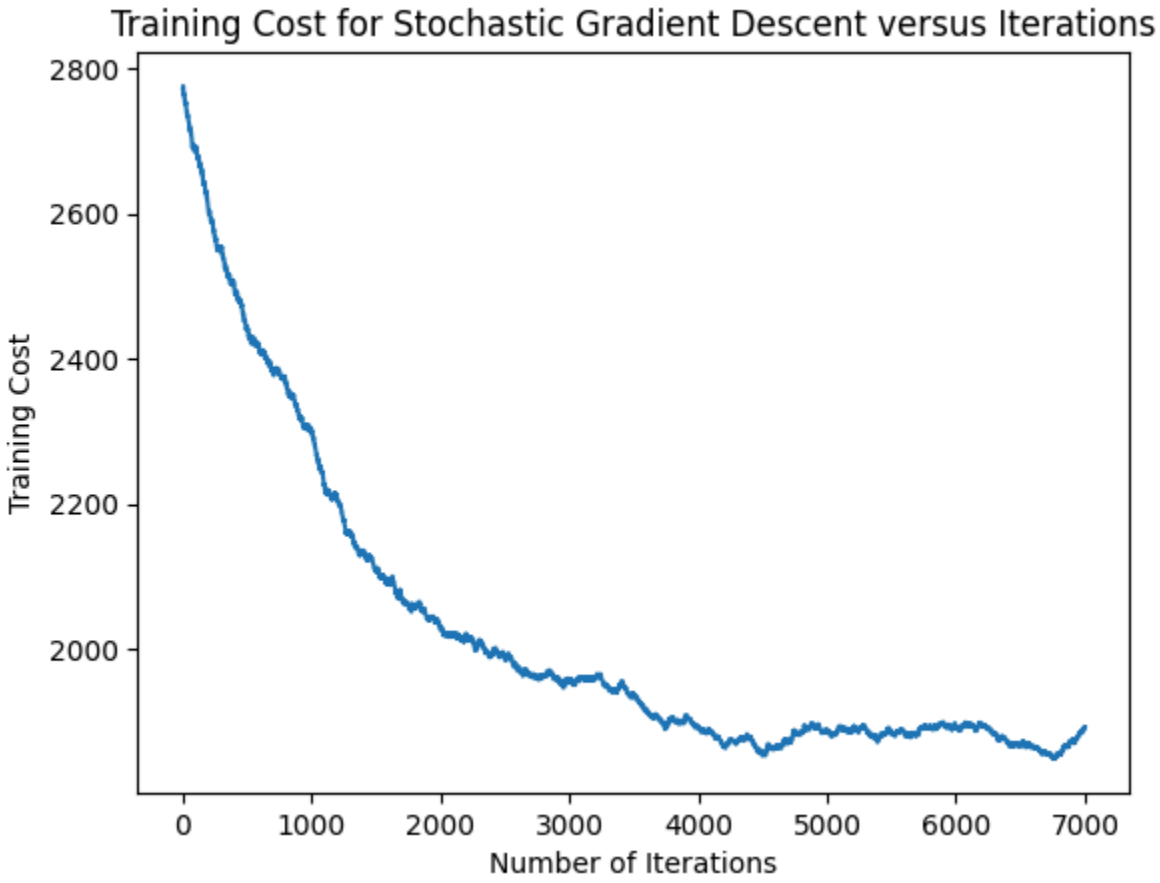
```
X_clean = pre_process(X, miu_hat, sigma_hat)
X_test_clean = pre_process(X_test, miu_hat, sigma_hat)
X_train, X_val, y_train, y_val = train_test_split(X_clean, y, test_size = 0.2, random_state=42)
```

```
def compute_s(X):
  s = scipy.special.expit(X)
  return np.clip(s, 0, 0.99999999)

def compute_cost(X, w, y, lam):
  s = compute_s(X@w)
  return (-1)*np.sum((y*(np.log(s)) + (1-y)*(np.log(1-s)))) + lam*np.linalg.norm(w)


w_t = np.zeros((X_train.shape[1],1))
lam = 0.1
learning_rate = 0.01
cost_arr = []
num_iter = 7000
for i in np.arange(num_iter):

  grad = X_train.T @(compute_s(X_train@w_t)-y_train) + 2*lam*w_t

  cost = compute_cost(X_train, w_t, y_train, lam)
  cost_arr.append(cost)
  #update gradient
  w_t = w_t - learning_rate*grad



plt.title("Training Cost for Batch Gradient Descent versus Iterations")
plt.xlabel("Number of Iterations")
plt.ylabel("Training Cost")
plt.plot(np.arange(num_iter), cost_arr)
plt.show()
```



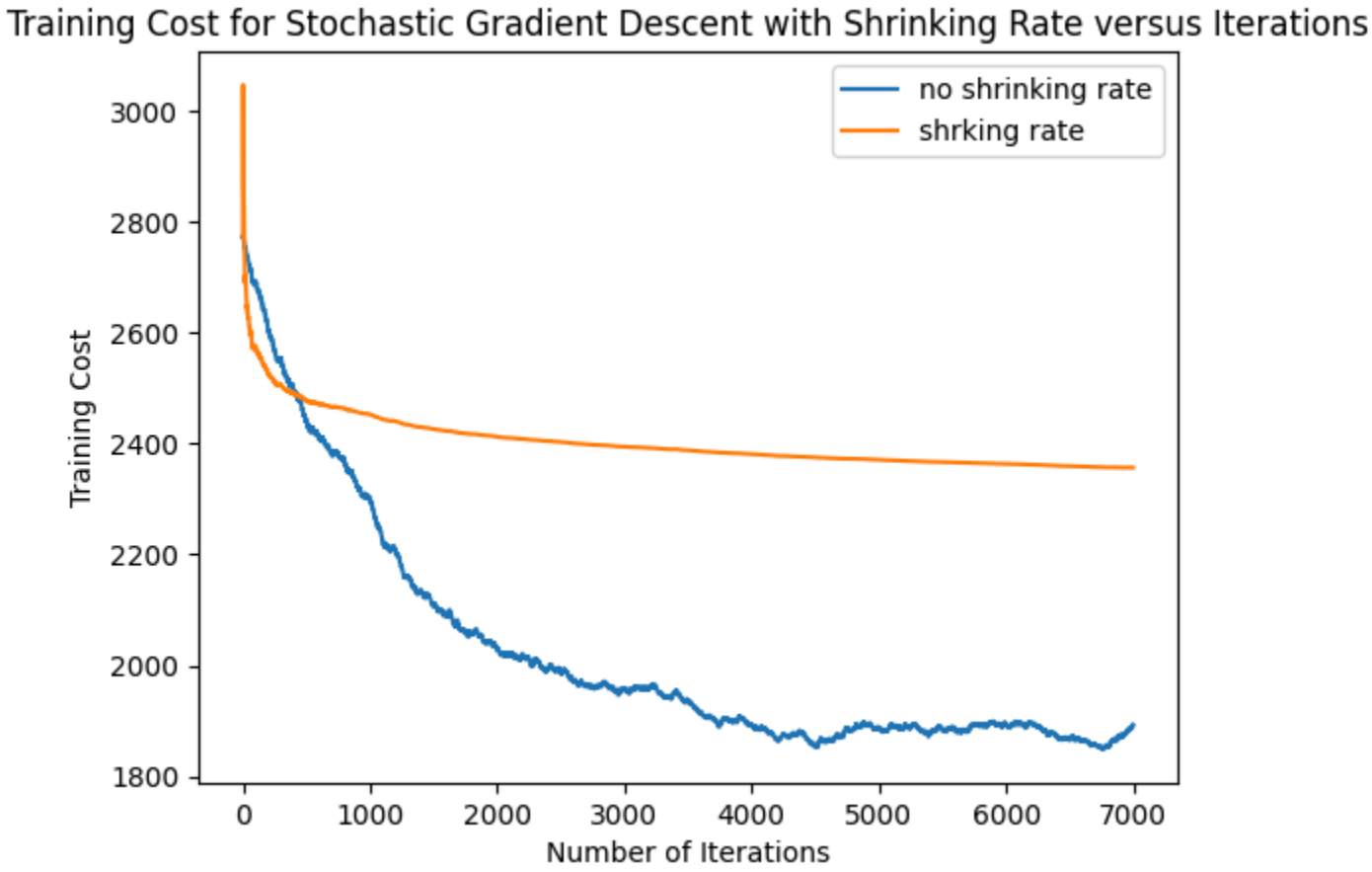
3.2 Stochastic Gradient Descent

```
## SGD
np.random.seed(42)
w_t = np.zeros((X_train.shape[1],1))
lam = 0.1
learning_rate = 0.001
cost_arr_2 = []
train_size = X_train.shape[0]
num_iter=7000
for i in np.arange(num_iter):
  indx = np.random.choice(train_size)
  xi = X_train[indx].reshape(X_train[indx].shape[0],1)
  grad = (compute_s(np.sum(xi*w_t)-y_train[indx]))*xi+ 2*lam*w_t
  cost = compute_cost(X_train, w_t, y_train, lam)

  cost_arr_2.append(cost)
  #update gradient
  w_t = w_t - learning_rate*grad


plt.title("Training Cost for Stochastic Gradient Descent versus Iterations")
plt.xlabel("Number of Iterations")
plt.ylabel("Training Cost")
plt.plot(np.arange(num_iter), cost_arr_2)
plt.show()
```



3.5. Modified Stochastic Gradient Descent

```
## Modified SGD
np.random.seed(42)
w_t = np.zeros((X_train.shape[1],1))
lam = 0.1
learning_rate = 0.1
cost_arr_3 = []
train_size = X_train.shape[0]
num_iter=7000
for i in np.arange(num_iter):
  indx = np.random.choice(train_size)
  xi = X_train[indx].reshape(X_train[indx].shape[0],1)
  grad = (compute_s(np.sum(xi*w_t)-y_train[indx]))*xi+ 2*lam*w_t
  cost = compute_cost(X_train, w_t, y_train, lam)

  cost_arr_3.append(cost)
  #update gradient and learning rate
  alpha = learning_rate/(i+1)
  w_t = w_t - alpha*grad


plt.title("Training Cost for Stochastic Gradient Descent with Shrinking Rate versus Iterations")
plt.xlabel("Number of Iterations")
plt.ylabel("Training Cost")
plt.plot(np.arange(num_iter), cost_arr_2, label='no shrinking rate')
plt.plot(np.arange(num_iter), cost_arr_3, label='shrking rate')
plt.legend()
plt.show()
```



3.6 Kaggle

```
import pandas as pd
import numpy as np

# Usage: results_to_csv(clf.predict(X_test))
def results_to_csv(y_test, name):
    y_test = y_test.astype(int)
    df = pd.DataFrame({'Category': y_test})
    df.index += 1 # Ensures that the index starts at 1
    df.to_csv(f"{name}_submission.csv", index_label='Id')
```

```
arr = np.ones(3)


arr = np.append(arr,1)
arr
```

```
    array([1., 1., 1., 1.])
```

```
columns = np.append(data['description'], 'bias')
columns
```

```
    array(['fixed acidity       ', 'volatile acidity    ',
           'citric acid         ', 'residual sugar      ',
           'chlorides           ', 'free sulfur dioxide ',
           'total sulfur dioxide', 'density             ',
           'pH                  ', 'sulphates           ',
           'alcohol             ', 'quality             ', 'bias'],
          dtype='<U20')
```

```
df = pd.DataFrame(X_clean, columns=columns)
df.columns
```

```
    Index(['fixed acidity       ', 'volatile acidity    ', 'citric acid         ',
           'residual sugar      ', 'chlorides           ', 'free sulfur dioxide ',
           'total sulfur dioxide', 'density             ', 'pH                  ',
           'sulphates           ', 'alcohol             ', 'quality             ',
           'bias'],
          dtype='object')
```

```
w_t = np.zeros((X_train.shape[1],1))
lams = np.arange(0.0001,0.1,0.002)
learning_rate = 0.01
cost_per_lam =[]
cost_arr_4 = []
optimal_w = []
num_iter = 1000
for lam in lams:
  for j in np.arange(num_iter):
    grad = X_train.T @(compute_s(X_train@w_t)-y_train) + 2*lam*w_t
    cost = compute_cost(X_val, w_t, y_val, lam)
    cost_arr_4.append(cost)
    w_t = w_t - learning_rate*grad
  optimal_w.append(w_t)
  cost_per_lam.append(cost_arr_4[-1])
  print("Lambda " + str(lam) + " has cost: "+ str(cost_arr_4[-1]))
```

```
Lambda 0.0001 has cost: 39.52789011225713
Lambda 0.0021 has cost: 39.518121068222555
Lambda 0.0041 has cost: 39.49368329390831
Lambda 0.0061 has cost: 39.47256171536218
Lambda 0.0081 has cost: 39.454668692850646
Lambda 0.0101 has cost: 39.43972883280866
Lambda 0.0121 has cost: 39.427495008164534
Lambda 0.0141 has cost: 39.4177464789176
Lambda 0.0161 has cost: 39.410285372039134
Lambda 0.0181 has cost: 39.404933713668214
Lambda 0.0201 has cost: 39.40153092004182
Lambda 0.022099999999999998 has cost: 39.399931678323185
Lambda 0.0241 has cost: 39.40000413366582
Lambda 0.0261 has cost: 39.40162833685137
Lambda 0.0281 has cost: 39.40469490903498
Lambda 0.0301 has cost: 39.40910388222218
Lambda 0.032100000000000004 has cost: 39.414763699433216
Lambda 0.034100000000000005 has cost: 39.421590338753546
Lambda 0.036100000000000001 has cost: 39.4295065480273
Lambda 0.0381 has cost: 39.438441173045796
Lambda 0.040100000000000004 has cost: 39.44832856586534
Lambda 0.042100000000000005 has cost: 39.45910806238504
Lambda 0.0441 has cost: 39.470723518675356
```

```
Lambda 0.0461 has cost: 39.48312289928413
Lambda 0.048100000000000004 has cost: 39.49625791196719
Lambda 0.050100000000000006 has cost: 39.51008367795721
Lambda 0.05210000000000001 has cost: 39.52455844089037
Lambda 0.0541 has cost: 39.539643303275454
Lambda 0.056100000000000004 has cost: 39.55530199106743
Lambda 0.058100000000000006 has cost: 39.57150063624701
Lambda 0.0601 has cost: 39.588207588224705
Lambda 0.0621 has cost: 39.60539323855075
Lambda 0.0641 has cost: 39.623029863385476
Lambda 0.0661 has cost: 39.64109147984648
Lambda 0.06810000000000001 has cost: 39.65955371560637
Lambda 0.07010000000000001 has cost: 39.678393690085635
Lambda 0.07210000000000001 has cost: 39.69758990603899
Lambda 0.0741 has cost: 39.71712215047567
Lambda 0.0761 has cost: 39.73697140397802
Lambda 0.0781 has cost: 39.7571197575897
Lambda 0.0801 has cost: 39.77755033679889
Lambda 0.0821 has cost: 39.79824723131408
Lambda 0.08410000000000001 has cost: 39.81919542938307
Lambda 0.08610000000000001 has cost: 39.8403807600793
Lambda 0.0881 has cost: 39.861789837841144
Lambda 0.0901 has cost: 39.88341001223025
Lambda 0.0921 has cost: 39.9052293201192
Lambda 0.0941 has cost: 39.92723644331053
Lambda 0.0961 has cost: 39.94942066825311
Lambda 0.0981 has cost: 39.97177184879567
```

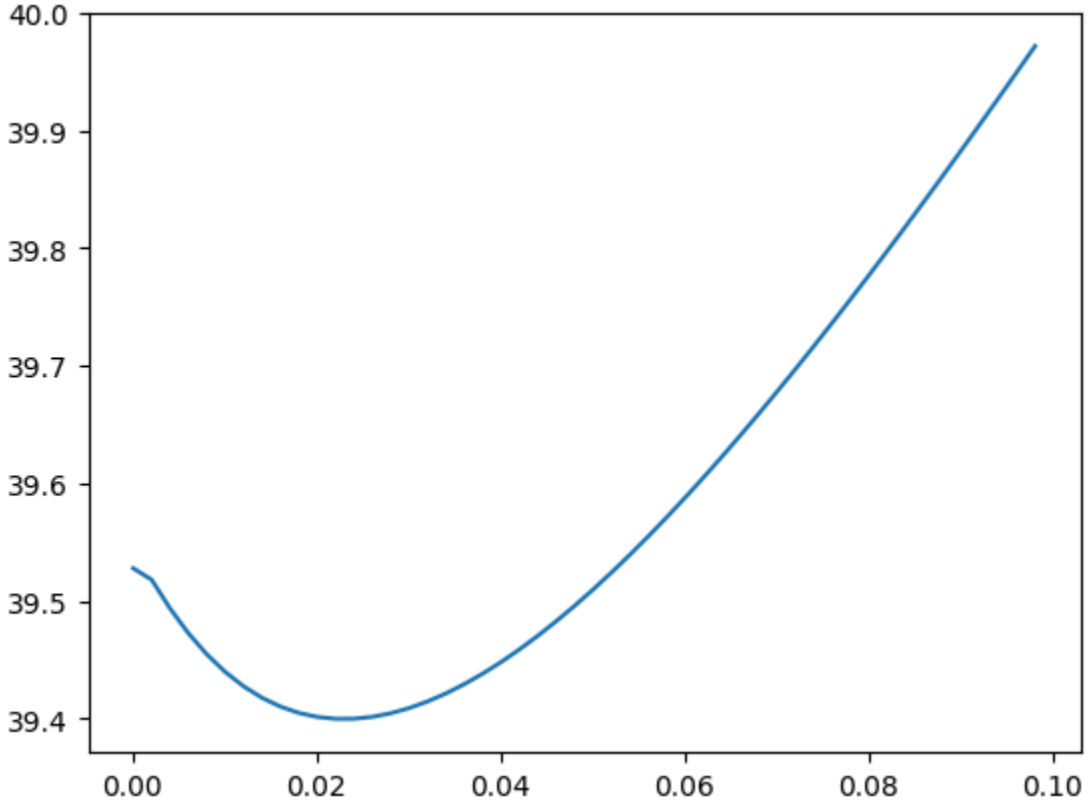
```
map_lams = dict(zip(cost_per_lam, lams))
map_w = dict(zip(cost_per_lam, optimal_w))
optimal_cost = min(cost_per_lam)
best_lam = map_lams[optimal_cost]
best_w = map_w[optimal_cost]
best_lam, best_w
```

```
(0.022099999999999998,
 array([[-1.21223739],
        [ 1.04529254],
        [-0.73096532],
        [-5.66333471],
```

```
[ 0.78871516],
[ 1.05381117],
[-3.15052334],
[ 8.26504463],
[-1.09033517],
[ 0.11836305],
[ 3.61068026],
[ 0.47683573],
[-4.12148806]]))
```

Best lambda: 0.022

```
plt.plot(lams, cost_per_lam)
plt.show()
min(cost_per_lam)
```




```
39.399931678323185
```

```
y_test = compute_s(X_test_clean@best_w)
y_test = y_test.reshape(y_test.shape[0])
y_pred = np.round(y_test)
y_pred[:5], y_test[:5]
```

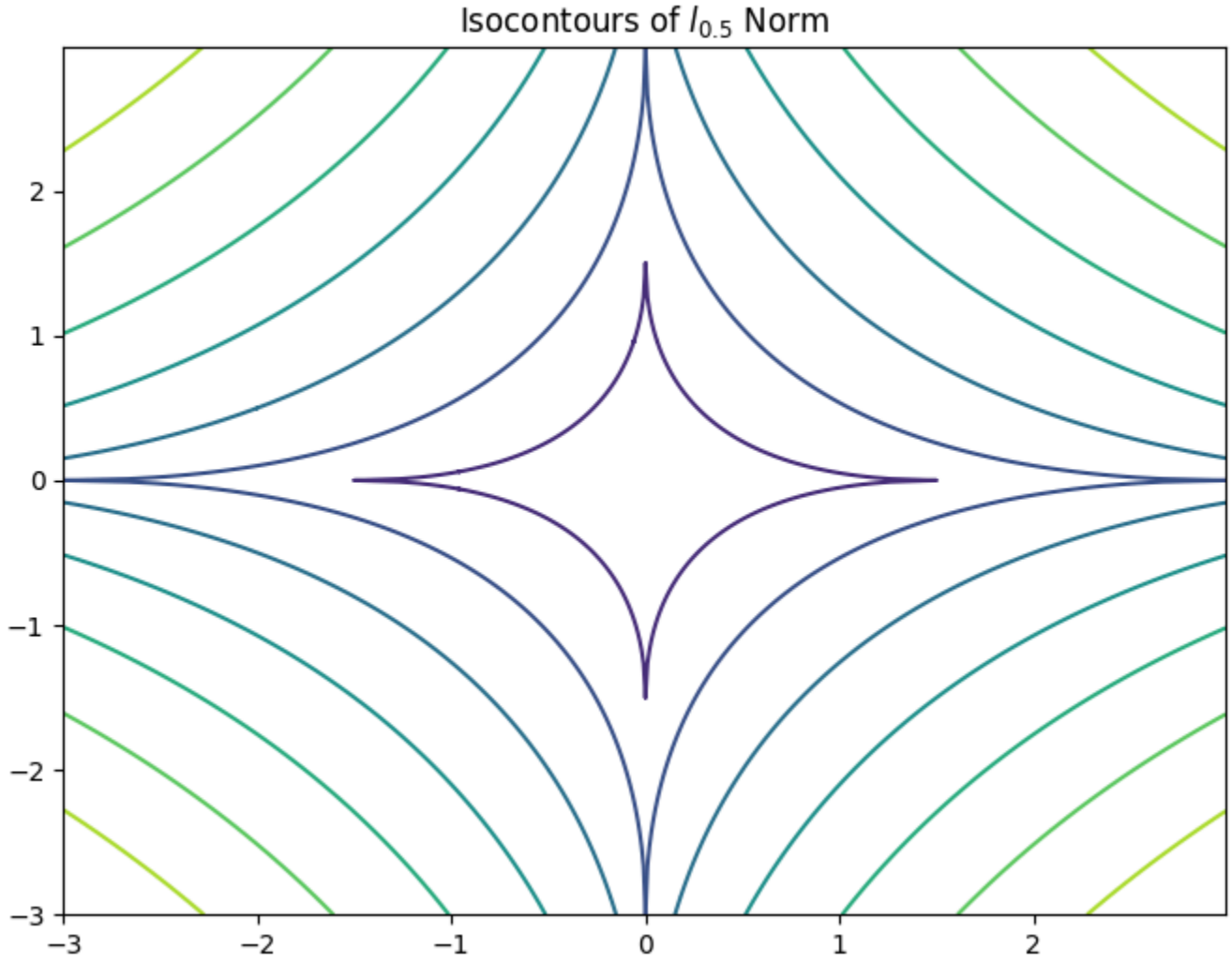
```
(array([0., 0., 0., 1., 0.]),
 array([1.00000000e-11, 8.47351807e-10, 1.00000000e-11, 9.99999990e-01,
        6.96238803e-10]))
```

```
results_to_csv(y_pred, 'wine')
```

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


def compute_norm(w1, w2, p):
  return (np.abs(w1)**p + np.abs(w2)**p)**(1/p)
```

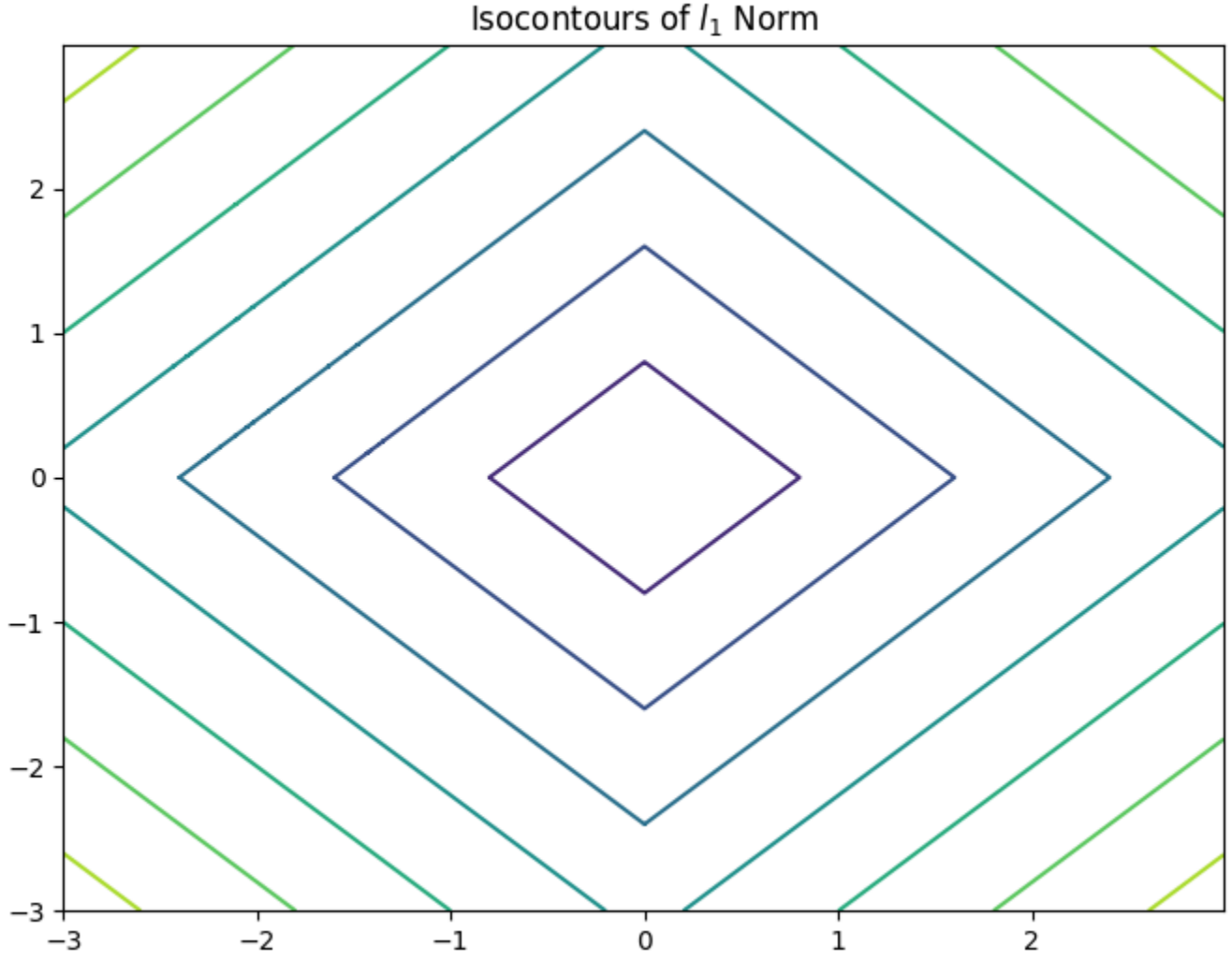
(a) ℓ0.5

```
## l-0.5 norm
w1, w2 = np.mgrid[-3:3:.01, -3:3:.01]
norm_w = compute_norm(w1,w2,0.5)

plt.figure(figsize=(8, 6))
plt.title("Isocontours of $l_{0.5}$ Norm")
plt.contour(w1, w2, norm_w)
plt.show()
```



(b) ℓ1

```
## l-1 norm
w1, w2 = np.mgrid[-3:3:.01, -3:3:.01]
norm_w = compute_norm(w1,w2,1)

plt.figure(figsize=(8, 6))
plt.title("Isocontours of $l_{1}$ Norm")
plt.contour(w1, w2, norm_w)
plt.show()
```



(b) ℓ2

```
## l-2 norm
w1, w2 = np.mgrid[-3:3:.01, -3:3:.01]
norm_w = compute_norm(w1,w2,2)

plt.figure(figsize=(8, 6))
plt.title("Isocontours of $l_{2}$ Norm")
plt.contour(w1, w2, norm_w)
plt.show()
```