Machine Learning & Data Mining
Set 2

Caltech CS/CNS/EE 155
January 12th, 2024

Policies

- Due 5 PM PST, January 19th on Gradescope.
- You are free to collaborate on all of the problems, subject to the collaboration policy stated in the syllabus.
- In this course, we will be using Google Colab for code submissions. You will need a Google account.
- You are allowed to use up to 48 late hours across the entire term. Late hours must be used in units of whole hours. Specify the total number of hours you have used when submitting the assignment.
- **No use of large language models is allowed.** Students are expected to complete homework assignments based on their understanding of the course material.

Submission Instructions

- Submit your report as a single .pdf file to Gradescope (entry code 2P8P28), under ”Set 2 Report”.
- In the report, **include any images generated by your code** along with your answers to the questions.
- Submit your code by **sharing a link in your report** to your Google Colab notebook for each problem (see naming instructions below). Make sure to set sharing permissions to at least ”Anyone with the link can view”. **Links that can not be run by TAs will not be counted as turned in.** Check your links in an incognito window before submitting to be sure.
- For instructions specifically pertaining to the Gradescope submission process, see `https://www.gradescope.com/get_started#student-submission`.

Google Colab Instructions

For each notebook, you need to save a copy to your drive.

1. Open the github preview of the notebook, and click the icon to open the colab preview.
2. On the colab preview, go to File → Save a copy in Drive.
3. Edit your file name to “lastname_firstname_set_problem”, e.g.”yue_yisong_set2_prob1.ipynb”

1 Comparing Different Loss Functions [30 Points]

Relevant materials: lecture 3 & 4

We've discussed three loss functions for linear classification models so far:

- Squared loss: $L_{\text{squared}} = (1 - y\mathbf{w}^T\mathbf{x})^2$
- Hinge loss: $L_{\text{hinge}} = \max(0, 1 - y\mathbf{w}^T\mathbf{x})$
- Log loss: $L_{\text{log}} = \ln(1 + e^{-y\mathbf{w}^T\mathbf{x}})$

where $\mathbf{w} \in \mathbb{R}^n$ is a vector of the model parameters, $y \in \{-1, 1\}$ is the class label for datapoint $\mathbf{x} \in \mathbb{R}^n$, and we're including a bias term in $\mathbf{x}$ and $\mathbf{w}$. The model classifies points according to sign($\mathbf{w}^T\mathbf{x}$).

Performing gradient descent on any of these loss functions will train a model to classify more points correctly, but the choice of loss function has a significant impact on the model that is learned.

Problem A [3 points]: Squared loss is often a terrible choice of loss function to train on for classification problems. Why?

Solution A: *Squared loss is a terrible choice of loss function for classification problems because it augments total loss even when a point is correctly classified. For classification problems, we only care that our data points are correctly classified, and if we wanted to be certain that we had a large margin, we would use Hinge loss. For squared loss, points that are far away from the classification line but correctly classified will be penalized heavily, leading more points to be misclassified.*

Problem B [9 points]: A dataset is included with your problem set: `problem1data1.txt`. The first two columns represent x_1, x_2, and the last column represents the label, $y \in \{-1, +1\}$.

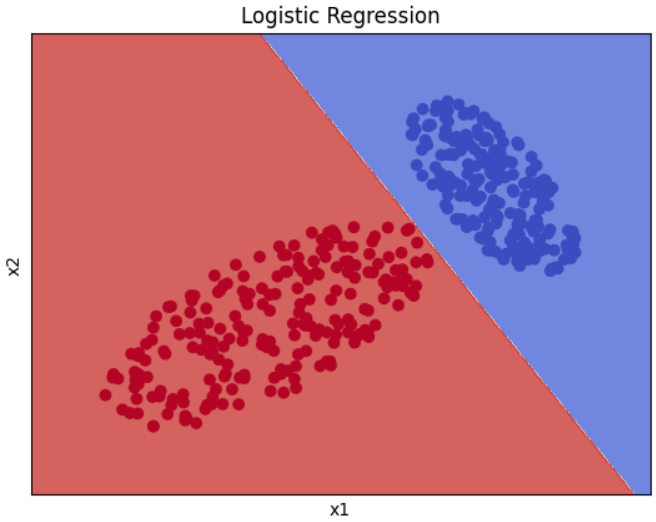
On this dataset, train both a logistic regression model and a ridge regression model to classify the points. (In other words, on each dataset, train one linear classifier using L_{log} as the loss, and another linear classifier using L_{squared} as the loss.) For this problem, you should use the logistic regression and ridge regression implementations provided within scikit-learn (logistic regression documentation) (Ridge regression documentation) instead of your own implementations. Use the default parameters for these classifiers except for setting the regularization parameters so that very little regularization is applied.

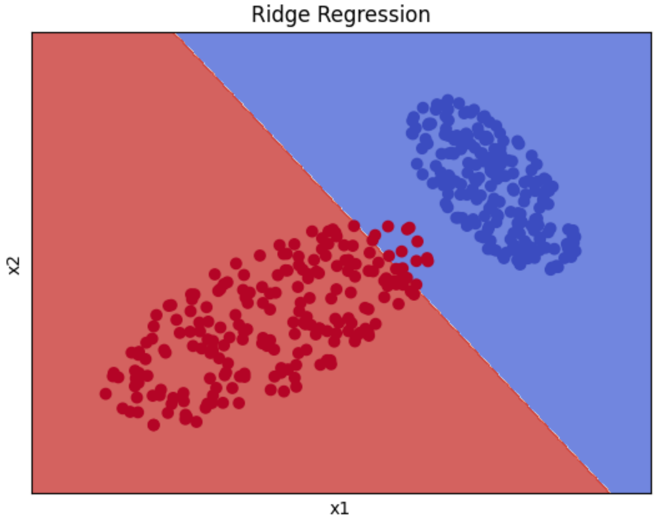
For each loss function/model, plot the data points as a scatter plot and overlay them with the decision boundary defined by the weights of the trained linear classifier. Include both plots in your submission. The template notebook for this problem contains a helper function for producing plots given a trained classifier.

What differences do you see in the decision boundaries learned using the different loss functions? Provide a qualitative explanation for this behavior.

Solution B:

https://colab.research.google.com/drive/1uTulhsD2IGyfmqKb00mJplwpTpeCKoNH?usp=sharing

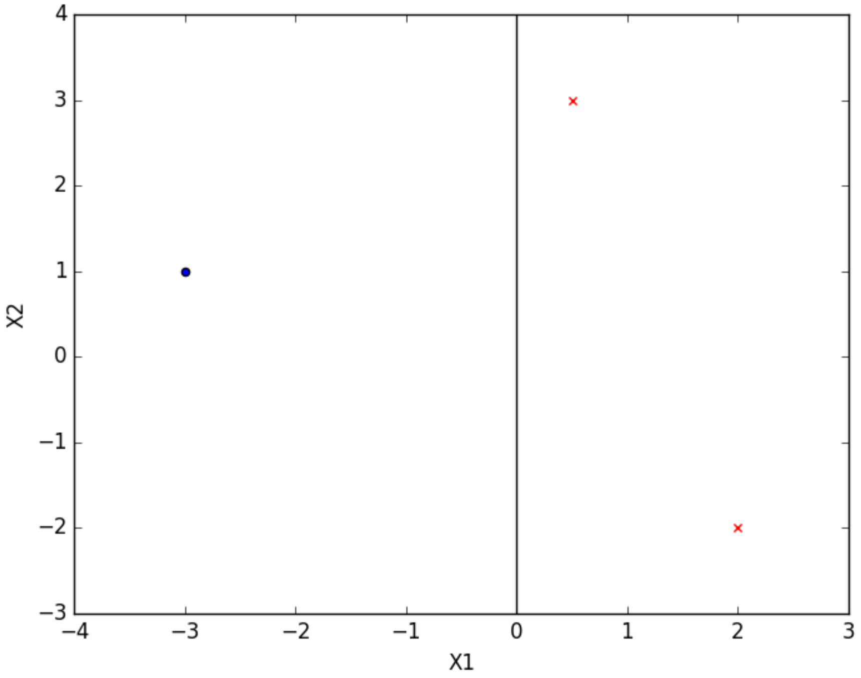




We can see that the decision boundary for Logistic Regression correctly classifies all the points but the Ridge Regression does not correctly classify all points. More specifically, the decision boundary for Ridge Regression is translated left from the Logistic Regression. The reason for this follows our answer to part A. The Ridge Regression is using squared loss, so it penalizes points far away from the decision boundary, which pulls the boundary closer to the left most points in the red cluster.

Problem C [9 points]: Leaving squared loss behind, let's focus on log loss and hinge loss. Consider the set of points $S = \{(\frac{1}{2}, 3), (2, -2), (-3, 1)\}$ in 2D space, shown below, with labels $(1, 1, -1)$ respectively.

Given a linear model with weights $w_0 = 0, w_1 = 1, w_2 = 0$ (where w_0 corresponds to the bias term), compute the gradients $\nabla_w L_{\text{hinge}}$ and $\nabla_w L_{\log}$ of the hinge loss and log loss, and calculate their values for each point in S.



The example dataset and decision boundary described above. Positive instances are represented by red x's, while negative instances appear as blue dots.

Solution C:

$$S = \left\{ \overset{1}{\left(\tfrac{1}{2}, 3\right)}, \overset{1}{(2, -2)}, \overset{-1}{(-3, 1)} \right\} \qquad w = (0, 1, 0)$$

$$L_{hinge} = \max\left\{0, 1 - y w^T x\right\}$$

$$\nabla_w L_{hinge} = \boxed{\begin{cases} 0 & \mid y w^T x \geq 1 \\ -yx & \mid y w^T x < 1 \end{cases}}$$

$$L_{log} = \ln\left(1 + e^{-y w^T x}\right)$$

$$\nabla_w L_{log} = \frac{-yx\, e^{-y w^T x}}{1 + e^{-y w^T x}} = \boxed{\frac{-yx}{e^{y w^T x} + 1}}$$

	x_0, x_1, x_2, y	$\nabla_w L_{hinge}$	$\nabla_w L_{log}$
$y w^T x = 1/2$	$1, \frac{1}{2}, 3, 1$	$(-1, -1/2, -3)$	$(-0.378, -0.189, -1.133)$
$y w^T x = 2$	$1, 2, -2, 1$	0	$(-0.119, -0.238, 0.238)$
$y w^T x = 3$	$1, -3, 1, -1$	0	$(0.047, -0.142, 0.047)$

Problem D [4 points]: Compare the gradients resulting from log loss to those resulting from hinge loss. When (if ever) will these gradients converge to 0? For a linearly separable dataset, is there any way to reduce or altogether eliminate training error without changing the decision boundary?

Solution D: *The gradients from the log loss will converge to 0 as the value of yw^tx increases and dominates yx. In other words, this means that as the points become correctly classified and increase their margin from the decision boundary, the gradients of the log loss converge to 0. Meanwhile the gradients of the hinge loss will converge to 0 when $yw^tx \geq 1$. This essentially means that the points have to be classified correctly and the margin has to be at least 1.*

For a linearly separable dataset, we can reduce (but not fully eliminate) training error for Log Loss by increasing the magnitude of our weight vector while keeping the actual decision boundary the same. We can fully eliminate training errors in Hinge Loss by ensuring that all points are classified correctly and increasing the magnitude of our weight vector while keeping the actual decision boundary the same.

Problem E [5 points]: Based on your answer to the previous question, explain why for an SVM to be a "maximum margin" classifier, its learning objective must not be to minimize just L_{hinge}, but to minimize $L_{\text{hinge}} + \lambda\|w\|^2$ for some $\lambda > 0$.

(You don't need to prove that minimizing $L_{\text{hinge}} + \lambda\|w\|^2$ results in a maximum margin classifier; just show that the additional penalty term addresses the issues of minimizing just L_{hinge}.)

Solution E: *The reasoning for this answer follows the logic set in Solution D. We can minimize Hinge Loss by increasing the magnitude of our weight vector while keeping the decision boundary the same. This is not the optimal way to maximize the margin and classify points correctly. Therefore, we need the additional penalty term to minimize the magnitude of our weight vector such that to minimize the new composite objective function, we have to change the decision boundary and maximize the margin.*

2 Effects of Regularization [40 Points]

Relevant materials: Lecture 3 & 4

For this problem, you are required to implement everything yourself and submit code (i.e. don't use scikit-learn but numpy is fine).

Problem A [4 points]: In order to prevent over-fitting in the least-squares linear regression problem, we add a regularization penalty term. Can adding the penalty term decrease the training (in-sample) error? Will adding a penalty term always decrease the out-of-sample errors? Please justify your answers. Think about the case when there is over-fitting while training the model.

Solution A: *After adding the penalty term, training error can't decrease. The raw learning objective without the added penalty term is prone to overfitting, where the model starts to fit the noise of the training data. This results in a very low training error but higher test errors. The purpose of the penalty term is to constrain model complexity and prevent optimal fitting of the training data.*

However, adding a penalty term will not always decrease the out-of-sample errors. There is usually a sweet spot for the lambda regularization parameter which optimizes out-of-sample errors. When we increase lambda past this value, we start underfitting the data.

Problem B [4 points]: ℓ_1 regularization is sometimes favored over ℓ_2 regularization due to its ability to generate a sparse w (more zero weights). In fact, ℓ_0 regularization (using ℓ_0 norm instead of ℓ_1 or ℓ_2 norm) can generate an even sparser w, which seems favorable in high-dimensional problems. However, it is rarely used. Why?

Solution B: ℓ_0 *regularization is not commonly used because it is not continuous. We prefer continuous functions because we need to differentiate the penalty terms during SGD.*

Implementation of ℓ_2 regularization:

We are going to experiment with regression for the Red Wine Quality Rating data set. The data set is uploaded on the course website, and you can read more about it here: https://archive.ics.uci.edu/ml/datasets/Wine. The data relates 13 different factors (last 13 columns) to wine type (the first column). Each column of data represents a different factor, and they are all continuous features. Note that the original data set has three classes, but one was removed to make this a binary classification problem.

Download the data for training and validation from the assignments data folder. There are two training sets, wine_training1.txt (100 data points) and wine_training2.txt (a proper subset of wine_training1.txt containing only 40 data points), and one test set, wine_validation.txt (30 data points). You will use the wine_validation.txt dataset to evaluate your models.

We will train a ℓ_2-*regularized logistic regression* model on this data. Recall that the unregularized logistic error (a.k.a. log loss) is

$$E = -\sum_{i=1}^{N} \log(p(y_i|\mathbf{x}_i))$$

where $p(y_i = -1|\mathbf{x}_i)$ is

$$\frac{1}{1 + e^{\mathbf{w}^T\mathbf{x}_i}}$$

and $p(y_i = 1|\mathbf{x}_i)$ is

$$\frac{1}{1 + e^{-\mathbf{w}^T\mathbf{x}_i}},$$

where as usual we assume that all $\mathbf{x}_i$ contain a bias term. The ℓ_2-regularized logistic error is

$$\begin{aligned} E &= -\sum_{i=1}^{N} \log(p(y_i|\mathbf{x}_i)) + \lambda \mathbf{w}^T\mathbf{w} \\ &= -\sum_{i=1}^{N} \log\left(\frac{1}{1 + e^{-y_i\mathbf{w}^T\mathbf{x}_i}}\right) + \lambda \mathbf{w}^T\mathbf{w} \\ &= -\sum_{i=1}^{N} \left(\log\left(\frac{1}{1 + e^{-y_i\mathbf{w}^T\mathbf{x}_i}}\right) - \frac{\lambda}{N}\mathbf{w}^T\mathbf{w}\right). \end{aligned}$$

Implement SGD to train a model that minimizes the ℓ_2-regularized logistic error, i.e. train an ℓ_2-regularized logistic regression model. Train the model with 15 different values of λ starting with $\lambda_0 = 0.00001$ and increasing by a factor of 5, i.e.

$$\lambda_0 = 0.00001, \lambda_1 = 0.00005, \lambda_2 = 0.00025, ..., \lambda_{14} = 61,035.15625.$$

Some important notes: Terminate the SGD process after 20,000 epochs, where each epoch performs one SGD iteration for each point in the training dataset. You should shuffle the order of the points before each epoch such that you go through the points in a random order (hint: use `numpy.random.permutation`). Use a learning rate of 5×10^{-4}, and initialize your weights to small random numbers.

You may run into numerical instability issues (overflow or underflow). One way to deal with these issues is by normalizing the input data X. Given the column for the jth feature, $X_{:,j}$, you can normalize it by setting $X_{ij} = \frac{X_{ij} - \overline{X_{:,j}}}{\sigma(X_{:,j})}$ where $\sigma(X_{:,j})$ is the standard deviation of the jth column's entries, and $\overline{X_{:,j}}$ is the mean of the jth column's entries. Normalization may change the optimal choice of λ; the λ range given above corresponds to data that has been normalized in this manner. If you treat the input data differently, simply plot enough choices of λ to see any trends.

Problem C [16 points]: Do the following for both training data sets (wine_training1.txt and wine_training2.txt) and attach your plots in the homework submission (use a log-scale on the horizontal axis):

i. Plot the average training error (E_{in}) versus different λs.

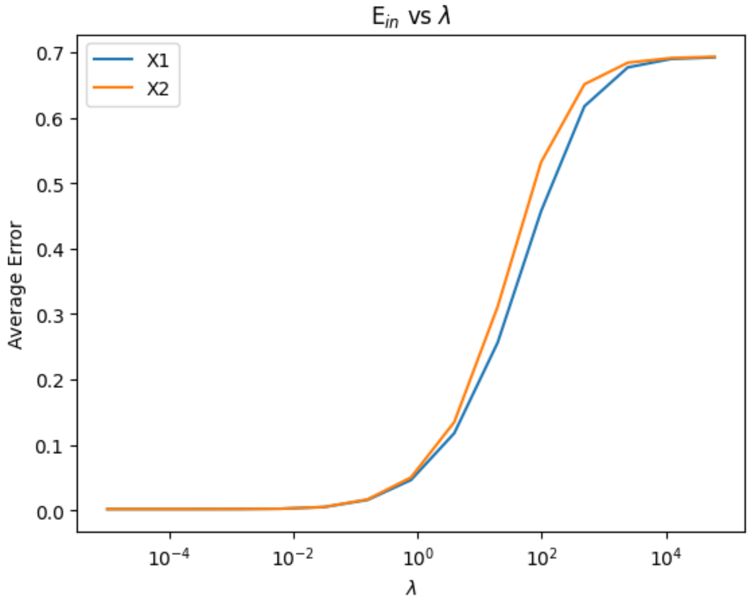
ii. Plot the average test error (E_{out}) versus different λs using wine_validation.txt as the test set.

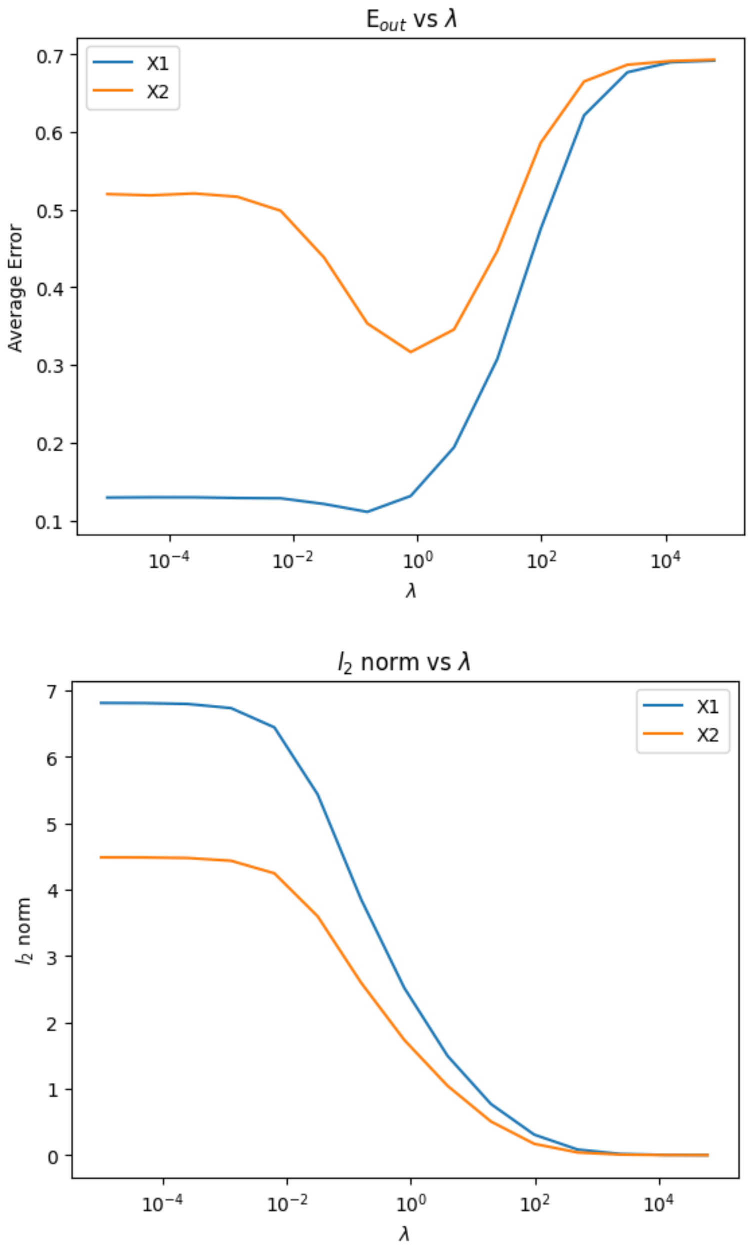
iii. Plot the ℓ_2 norm of $\mathbf{w}$ versus different λs.

You should end up with three plots, with two series (one for wine_training1.txt and one for wine_training2.txt) on each plot. Note that the E_{in} and E_{out} values you plot should not include the regularization penalty — the penalty is only included when performing gradient descent.

Solution C:

https://colab.research.google.com/drive/1aXeeuR_8vc2icna_c8MY55mVJAqjPySe?usp=sharing





Problem D [4 points]: Given that the data in wine_training2.txt is a subset of the data in wine_training1.txt, compare errors (training and test) resulting from training with wine_training1.txt (100 data points) versus wine_training2.txt (40 data points). Briefly explain the differences.

Solution D:

The training errors for data set 1 and 2 are closely mirrored up until λ = 1. When a data set has less points, it is more prone to overfitting. So when the regularization parameter λ is increased, it constricts model complexity and prevents the model from fitting the data as well. This is why we see the increase in λ disproportionately affecting E_{in} for data set 2.

Data set 2 has much larger test errors than data set 1. We know this occurs because data set 2's model is likely overfitting, which means it requires a larger λ regularization parameter to improve its out-of-sample performance again. That is why we see its optimal test performance at λ = 1. On the other hand, data set 1 performs very well with lower λ because it is not overfitting.

Problem E [4 points]: Briefly explain the qualitative behavior (i.e. over-fitting and under-fitting) of the training and test errors with different λs while training with data in wine_training1.txt.

Solution E: *For data set 1, we observe relatively low training and test errors when $\lambda < 1$ because the data set has enough points to avoid overfitting. We see errors increase once $\lambda > 1$ because the model's complexity is being constricted until the model eventually underfits the data.*

Problem F [4 points]: Briefly explain the qualitative behavior of the ℓ_2 norm of **w** with different λs while training with the data in wine_training1.txt.

Solution F: *As λ increases, we increase the penalty attributed to the magnitude of our weight vector, and therefore the ℓ_2 norm decreases.*

Problem G [4 points]: If the model were trained with wine_training2.txt, which λ would you choose to train your final model? Why?

Solution G: *I would choose λ = 1 because it performs the best out-of-sample (has the lowest E_{out}).*

3 Lasso (ℓ_1) vs. Ridge (ℓ_2) Regularization [30 Points]

Relevant materials: Lecture 3

For this problem, you may use the scikit-learn (or other Python package) implementation of Lasso and Ridge regression — you don't have to code it yourself.

The two most commonly-used regularized regression models are Lasso (ℓ_1) regression and Ridge (ℓ_2) regression. Although both enforce "simplicity" in the models they learn, only Lasso regression results in sparse weight vectors. This problem compares the effect of the two methods on the learned model parameters.

Problem A [12 points]: The tab-delimited file problem3data.txt on the course website contains 1000 9-dimensional datapoints. The first 9 columns contain $x_1, \ldots, x_9$, and the last column contains the target value y.

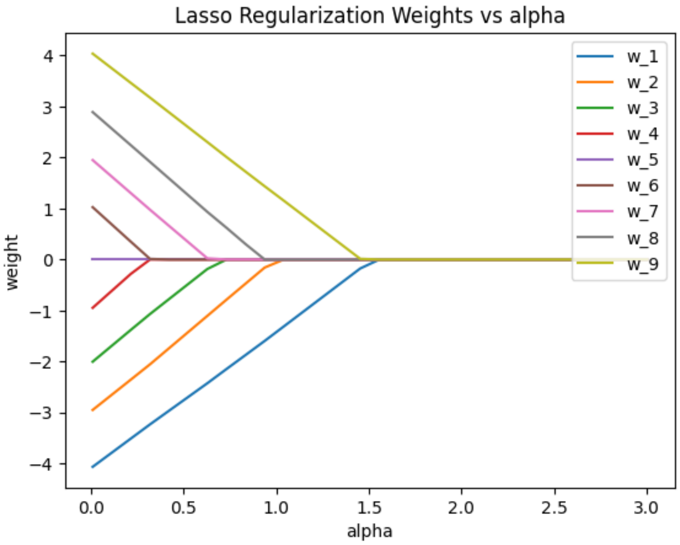
i. Train a linear regression model on the problem3data.txt data with Lasso regularization for regularization strengths α in the vector given by `numpy.linspace(0.01, 3, 30)`. On a single plot, plot each of the model weights $w_1, ..., w_9$ (ignore the bias/intercept) as a function of α.

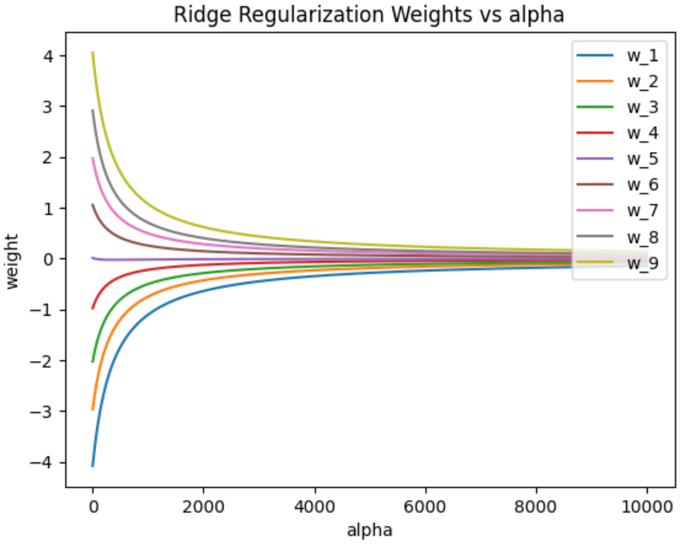
ii. Repeat **i.** with Ridge regression, and this time using regularization strengths $\alpha \in \{1, 2, 3, \ldots, 1e4\}$.

iii. As the regularization parameter increases, what happens to the number of model weights that are exactly zero with Lasso regression? What happens to the number of model weights that are exactly zero with Ridge regression?

Solution A:

https://colab.research.google.com/drive/1ny7TmtYq6MWhGpXhBdKUBl38BpXkC1_C?usp=sharing





As the regularization parameter increases, the number of model weights that are 0 with Lasso regression increases. More specifically, As the regularization parameter increases past 1.5, we see that all weights for Lasso Regularization become 0. However, for Ridge Regression, as the regularization parameter increases, the model weights approach 0 but never quite reach it.

Problem B [9 points]:

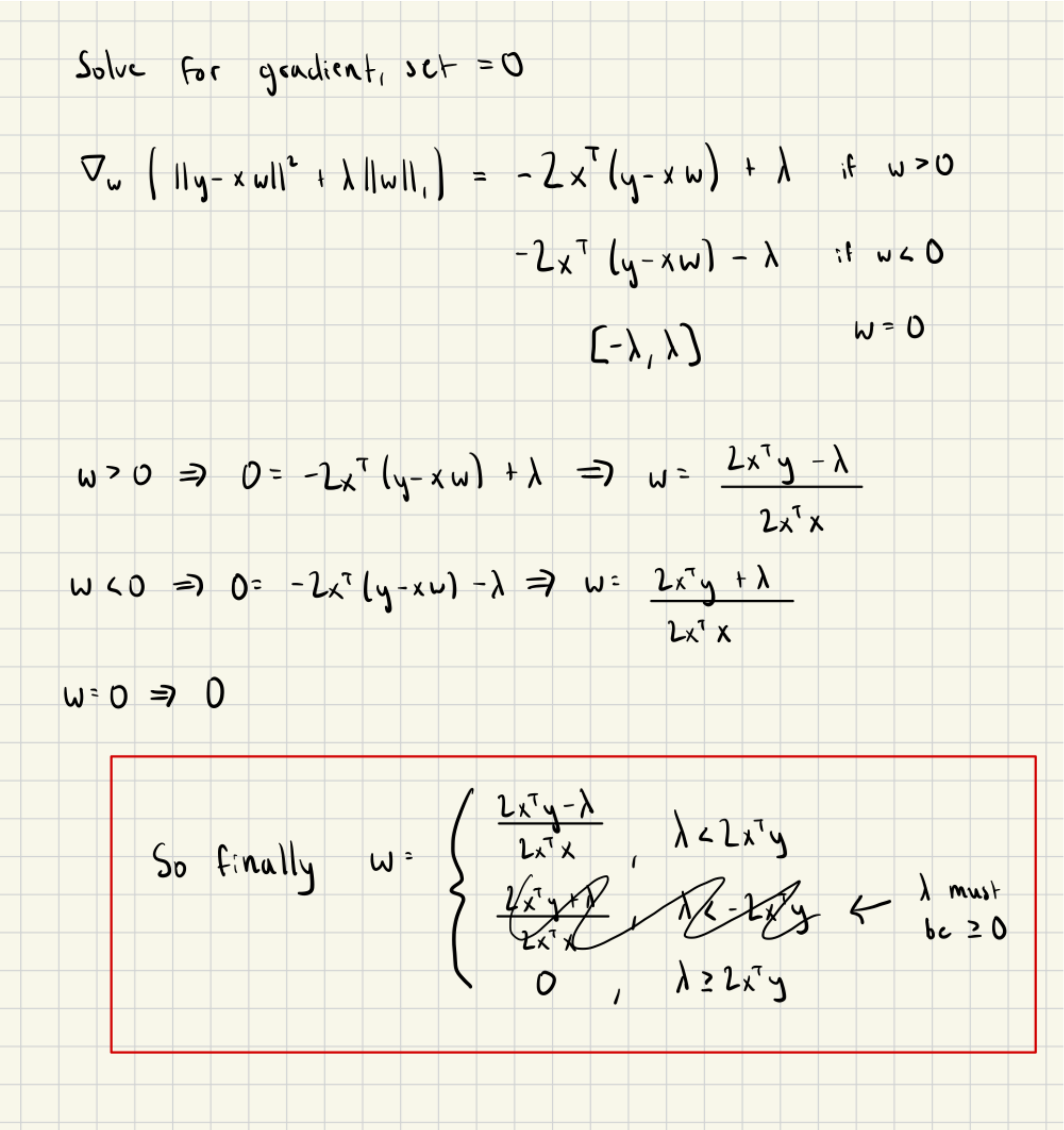
i. In the case of 1-dimensional data, Lasso regression admits a closed-form solution. Given a dataset containing N datapoints, each with $d = 1$ feature, solve for

$$\arg\min_{w} \|\mathbf{y} - \mathbf{x}w\|^2 + \lambda\|w\|_1,$$

where $\mathbf{x} \in \mathbb{R}^N$ is the vector of datapoints and $\mathbf{y} \in \mathbb{R}^N$ is the vector of all output values corresponding to these datapoints. Just consider the case where $d = 1$, $\lambda \geq 0$, and the weight w is a scalar.

This is linear regression with Lasso regularization.

Solution B.i:



Solve for gradient, set = 0

$$\nabla_w \left(\|y - xw\|^2 + \lambda \|w\|_1 \right) = \begin{matrix} -2x^T(y - xw) + \lambda & \text{if } w > 0 \\ -2x^T(y - xw) - \lambda & \text{if } w < 0 \\ [-\lambda, \lambda] & w = 0 \end{matrix}$$

$$w > 0 \Rightarrow 0 = -2x^T(y - xw) + \lambda \Rightarrow w = \frac{2x^Ty - \lambda}{2x^Tx}$$

$$w < 0 \Rightarrow 0 = -2x^T(y - xw) - \lambda \Rightarrow w = \frac{2x^Ty + \lambda}{2x^Tx}$$

$$w = 0 \Rightarrow 0$$

$$\boxed{\text{So finally} \quad w = \begin{cases} \frac{2x^Ty - \lambda}{2x^Tx}, & \lambda < 2x^Ty \\ \cancel{\frac{2x^Ty + \lambda}{2x^Tx}}, & \cancel{\lambda < -2x^Ty} \quad \leftarrow \lambda \text{ must be } \geq 0 \\ 0, & \lambda \geq 2x^Ty \end{cases}}$$

ii. In this question, we continue to consider Lasso regularization in 1-dimension. Now, suppose that $w \neq 0$ when $\lambda = 0$. Does there exist a value for λ such that $w = 0$? If so, what is the smallest such value?

Solution B.ii: *Yes, as shown in the solution for B.i, we know that $\lambda \geq 0$, and that when $\lambda < 2x^Ty$, $w = \frac{2x^Ty - \lambda}{2x^Tx}$, so when $\lambda \geq 2x^Ty$, $w = 0$.*

Problem C [9 points]:

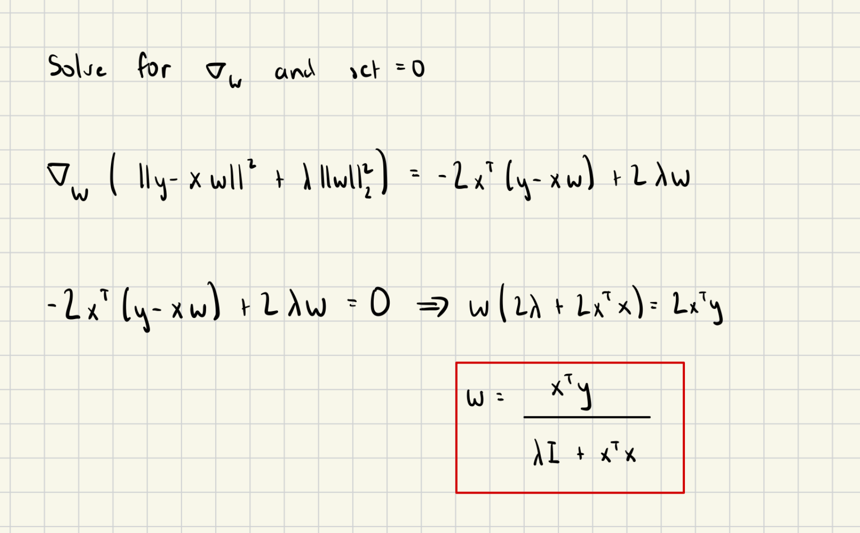
i. Given a dataset containing N datapoints each with d features, solve for

$$\arg\min_{\mathbf{w}} \|\mathbf{y} - \mathbf{X}\mathbf{w}\|^2 + \lambda\|\mathbf{w}\|_2^2$$

where $\mathbf{X} \in \mathbb{R}^{N\times d}$ is the matrix of datapoints and $\mathbf{y} \in \mathbb{R}^N$ is the vector of all output values for these datapoints. Do so for arbitrary d and $\lambda \geq 0$.

This is linear regression with Ridge regularization.

Solution C.i:



Solve for ∇_w and set $= 0$

$$\nabla_w \left(\|y - xw\|^2 + \lambda\|w\|_2^2 \right) = -2x^T(y - xw) + 2\lambda w$$

$$-2x^T(y - xw) + 2\lambda w = 0 \Rightarrow w(2\lambda + 2x^Tx) = 2x^Ty$$

$$\boxed{w = \frac{x^Ty}{\lambda I + x^Tx}}$$

ii. In this question, we consider Ridge regularization in 1-dimension. Suppose that $w \neq 0$ when $\lambda = 0$. Does there exist a value for $\lambda > 0$ such that $w = 0$? If so, what is the smallest such value?

Solution C.ii: *There does not exist a value for λ such that $w = 0$. That would require $X^Ty = 0$, which does not have a dependence on λ.*