Machine Learning & Data Mining
Set 2

Caltech CS/CNS/EE 155
January 19th, 2022

Policies

- Due 9 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.

Submission Instructions

- Submit your report as a single .pdf file to Gradescope (entry code 7426YK), 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 $\text{sign}(\mathbf{w}^T\mathbf{x})$.

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 not a good for classification because the labels are not real-valued but are binary (0 or 1) and squared loss will penalize large values regardless of if it's in the correct class. So, if a data point has output value 15 (or any large number) in our current model, squared loss would nudge the weights to bring the model closer to 1. This is unnecessary because 1 and 15 both have the same sign (+) and would be classified into the same class. So, squared loss is optimizing for unnecessary conditions and it might even not work for linearly separable data.*

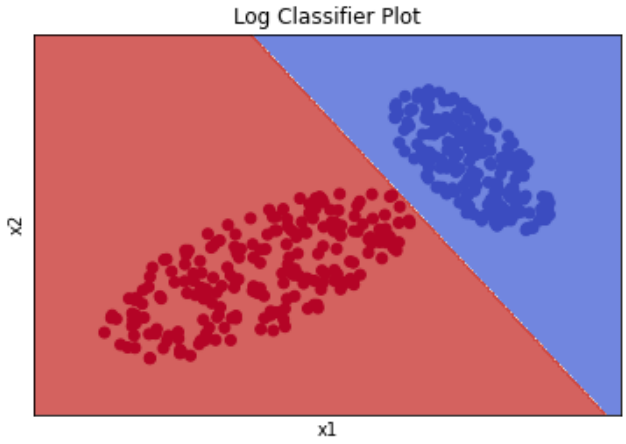
Problem B [9 points]: A dataset is included with your problem set: `problem1data1.txt`. The first two columns represent x_1, x_2, and the last column represents the label, $y \in \{-1, +1\}$.

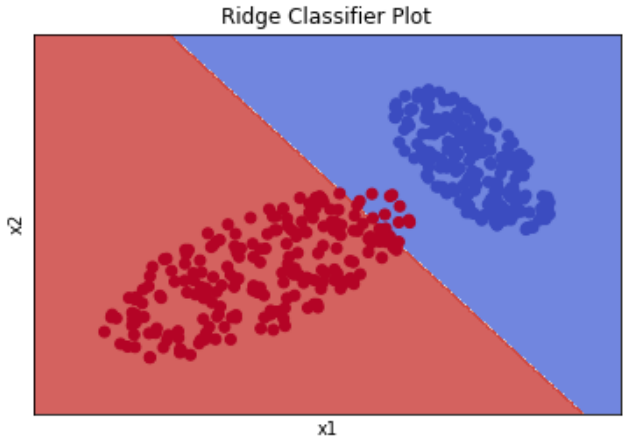
On this dataset, train both a logistic regression model and a ridge regression model to classify the points. (In other words, on each dataset, train one linear classifier using L_{log} as the loss, and another linear classifier using L_{squared} as the loss.) For this problem, you should use the logistic 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/1IUsZBhHqFueJx6JNDYkwKXMLIcITar4v?usp=sharing*





As expected in part A the ridge classifier is not able to separate a linearly separable dataset. This is probably because there are many red points far away from the boundary, this means the squared error is very high for those points and the loss function nudges the classification boundary closer to the red points to bring the value of those red points closer to 1. On the other hand, logistic error only makes nudges for values that are incorrectly classified. So, log loss results in better classification.

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:

$$\nabla L_{hinge} = \begin{cases} 0, & yw^t x \geq 1 \\ -yx, & yw^t x < 1 \end{cases}$$

$$\nabla L_{log} = \frac{-yxe^{-yw^t x}}{1 + e^{-yw^t x}}$$

$$w^t x_1 = (0,1,0)(1,\frac{1}{2},3) = \frac{1}{2}, y = 1$$
$$w^t x_2 = (0,1,0)(1,2,-2) = 2, y = 1$$
$$w^t x_3 = (0,1,0)(1,-3,1) = -3, y = -1$$

For x_1,$\nabla L_{hinge} = (-1,-1/2,-3)$ $\nabla L_{log} = (-0.377541, -0.18877, -1.13262)$
For x_2, $\nabla L_{hinge} = (0,0,0)$ $\nabla L_{log} = (-0.119203, -0.238406, 0.238406)$
For x_3, $\nabla L_{hinge} = (0,0,0)$ $\nabla L_{log} = (0.047426, -0.142278, 0.047426)$

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 hinge loss are only zero when* $yw^t x >= 1$, *i.e. for all correctly classified points the gradient is zero.*

For log loss, the gradients of correctly classified points is small non-zero values. For log loss, as the points go further away from the decision boundary in the correct direction (as $w^t x$ *increases), the gradient approaches 0.*

To reduce the training error without changing the decision boundary, we must scale up the value of w. *Scaling* w *does not change the boundary, but it changes the gradient.*

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: *Minimizing the hinge error* L_{hinge} *could be done by simply scaling up* w *without changing the decision boundary. But, if we add* $\lambda||w||^2$ *into the expression for minimization,* w*'s values are constrained and it cannot be scaled indefinitely up. The classification must happen by changing the decision boundary. That's why, the additional penalty addresses the issue of minimizing the* L_{hinge} *without changing the decision boundary.*

2 Effects of Regularization

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: *The regularization term is intended to decrease the out of sample error at the cost of increasing the in sample error (thereby reducing how well it fits the training data). It could also be viewed as decreasing the variance at increasing the bias. The penalty term will never decrease the in-sample error. Usually, the penalty term will decrease out of sample error, but it could also increase it in the case that the regularization term is too big (the parameter space is too small and there is underfitting).*

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: *The L0 norm is the number of non-zero elements in the matrix. It is non-differentiable and non-convex. So, it's not suitable for optimization. It also might generate ws that are too sparse (underfitting, too many features removed).*

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):

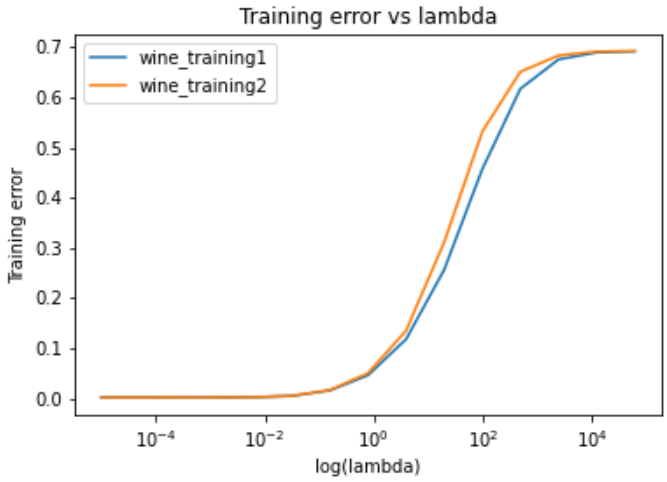
i. Plot the average training error (E_{in}) versus different λs.

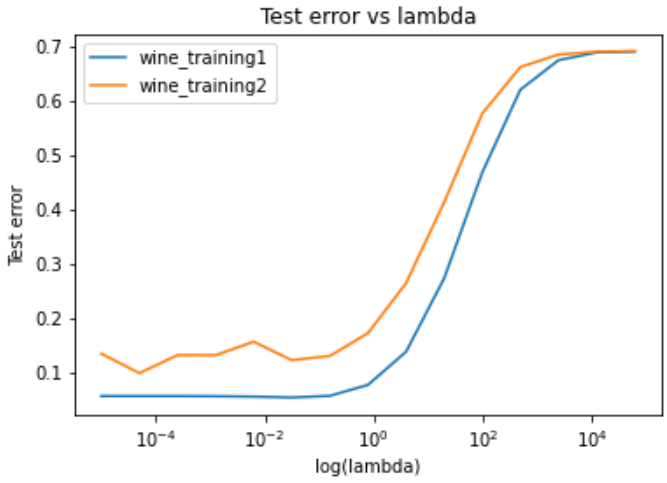
ii. Plot the average test error (E_{out}) versus different λs using wine_validation.txt as the test set.

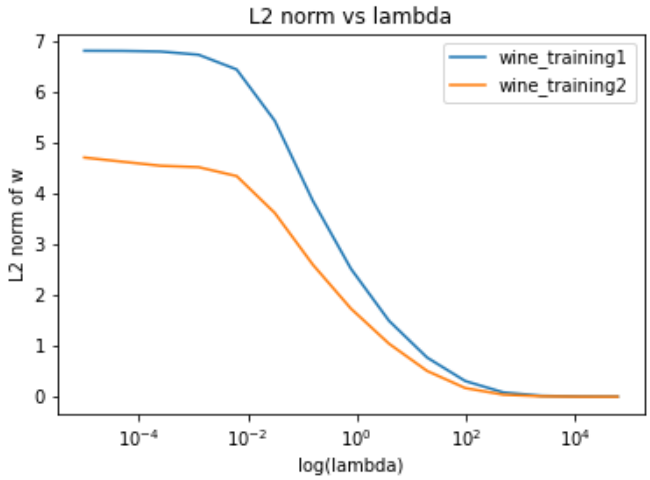
iii. Plot the ℓ_2 norm of $\mathbf{w}$ versus different λs.

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/1AdTSGbgj_sL0jemc0oE72A6XfIvKNilq?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: *Both the average training and test error is generally higher for the smaller training set than the larger.*

The training error for both the smaller and bigger test follow a similar pattern - very low for a small lambda but as lambda increases the training error becomes high. This is because of under fitting because a high lambda will limit the complexity of the model.

The test error for the smaller and bigger dataset are different. The smaller data set initially has a very high out of sample error, but as the lambda increases, the out of sample error decreases because the variance of the model has been controlled, thus controlling overfitting. But, as the lambda increases further, the out of sample error rises steeply again, this is because of underfitting and having a model that is too simple.

With the larger dataset, the out of sample error is very low even with a small lambda. This is because there are enough datapoints for the algorithm to learn a good model with the training set. As lambda increases, underfitting happens again and the model is too simple and with a high out of sample error.

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: *The training and test errors for wine_training1.txt are similar but there are some small differences. As lambda increases, the training error increases. This is because the model isn't able to take on any possible values because lambda constrains the values that the model can take on. This causes underfitting. As lambda increases, there is a sharp increase in training error as underfitting increases.*

The test error on the other hand experiences a small dip as lambda increases. This is because there is less overfitting at this point (this value of lambda). As lambda increases further, overfitting happens again and the test error becomes very high.

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

Solution F: *The L2 norm of w decreases as λ increases. This is because the algorithm is now trying to minimise the L2 norm of w as well as the L_{log}. So, the values that w can take on decrease in magnitude and become more constrained.*

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: *If the model was being trained with wine_training2.txt, I would choose approximately 1 as my lambda because that is where the test error was minimised. So overfitting is kept in check and there is no underfitting either.*

3 Lasso (ℓ_1) vs. Ridge (ℓ_2) Regularization

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.

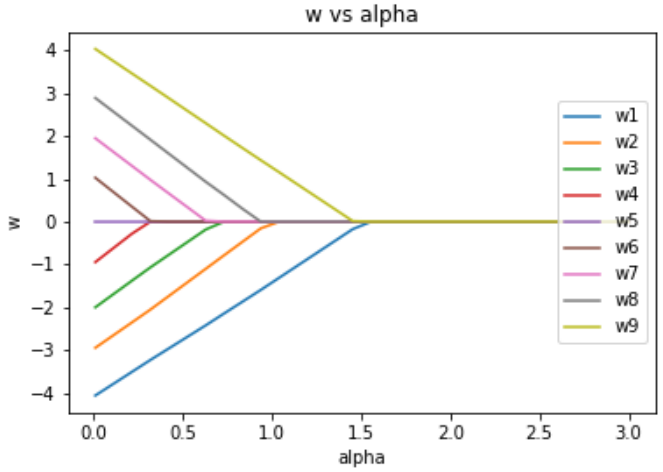
i. Train a linear regression model on the problem3data.txt data with Lasso regularization for regularization strengths α in the vector given by `numpy.linspace(0.01, 3, 30)`. On a single plot, plot each of the model weights $w_1, ..., w_9$ (ignore the bias/intercept) as a function of α.

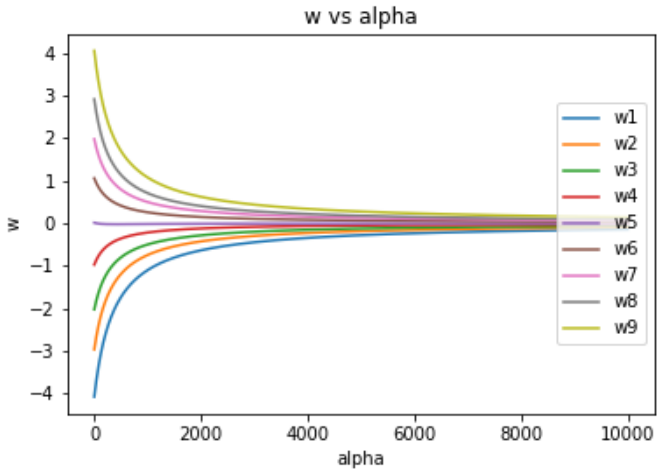
ii. Repeat **i.** with Ridge regression, and this time using regularization strengths $\alpha \in \{1, 2, 3, \ldots, 1e4\}$.

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/1qH6gqBfwfTnaulqTDzZt9kX6HK766pzB?usp=sharing





With lasso regression, for a very small value of α, several elements of w collapse to 0. In ridge regression, even though α increases to very big values, w approaches 0 but doesn't immediately become 0. In Lasso, all w's eventually reach 0.

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: *To find the closed-form solution, we must take the derivative and set it to 0. Taking the derivative of $||w||_1$ is not possible, so we do it piecewise:*

$$\nabla_w \lambda||w|| = \begin{cases} -\lambda, w < 0 \\ \lambda, w > 0 \end{cases}$$

So, the required gradient is:

$$\begin{cases} -2x^t||y - xw|| + \lambda, w > 0 \\ -2x^t||y - xw|| - \lambda, w < 0 \end{cases}$$

Setting that to 0, we find w:

$$w = -(\lambda - 2x^t y)(2x^t x)^{-1}$$

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: *If $\lambda = 2x^t y$, w is 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: *Again, we take the gradient and set it to 0:*

$$2X^t(y - X^t w) + 2\lambda w = 0$$
$$w = X^t y(\lambda I + X^T X)^{-1}$$

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: *No, there is no value of λ for which $w = 0$.*