Policies

- Due 9 PM PST, January 20^{th} 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 K3RPGE), 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: *In classification, we only care that points are on the correct "side" of a dividing hyper-plane, but squared loss penalizes points based on distance to the target symmetrically, so points that are "too far" on the correct side of the hyper-plane will contribute to loss when they shouldn't.*

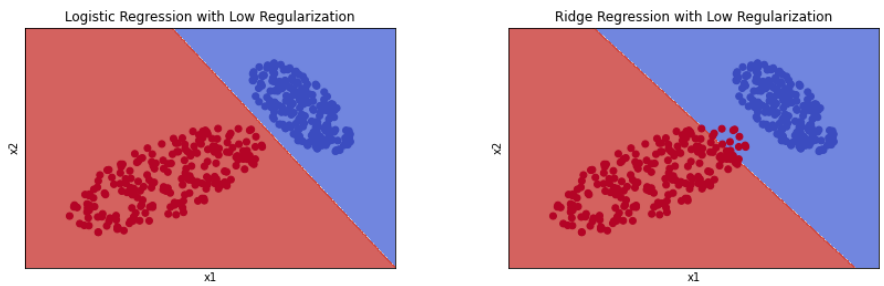
Problem B [9 points]: A dataset is included with your problem set: problem1data1.txt. The first two columns represent x_1, x_2, and the last column represents the label, $y \in \{-1, +1\}$.

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.

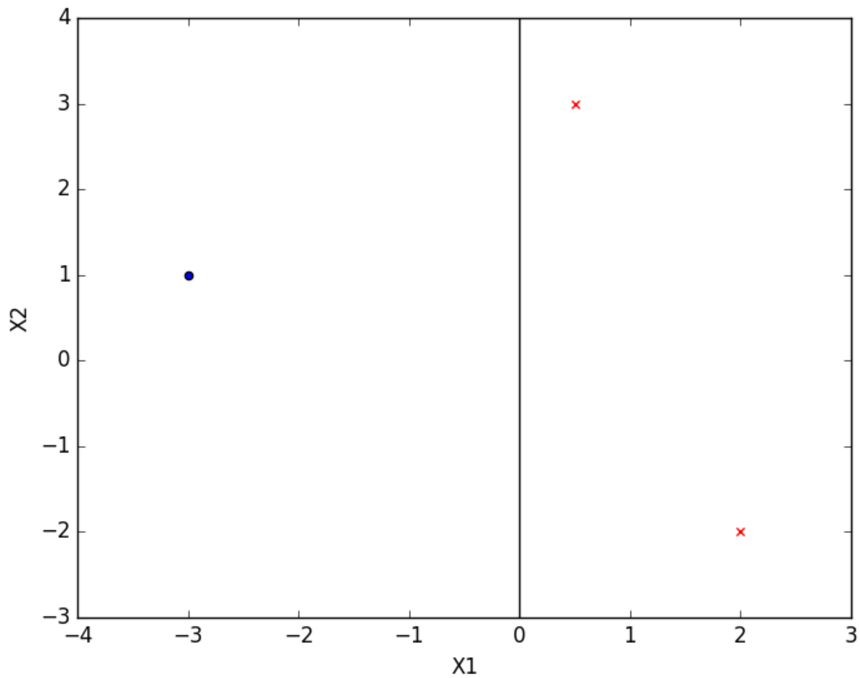
Solution B: *Colab notebook*



The logistic regression separated the data correctly, while the ridge regression's boundary falls within the blob of red points. As explained in 1A, this is because the red points that are far from the boundary contribute to loss which "pulls" the boundary toward them.

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), derive the gradients $\nabla_w L_{\text{hinge}}$ and $\nabla_w L_{\text{log}}$ 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:

$$L_{log} = \ln(1 + e^{-y(w_0 + w_1 x_1 + w_2 x_2)})$$
$$L_{hinge} = \max(0, 1 - y(w_0 + w_1 x_1 + w_2 x_2))$$

$$\partial_{w_0} L_{log} = \frac{-y e^{-y(w_0 + w_1 x_1 + w_2 x_2)}}{1 + e^{-y(w_0 + w_1 x_1 + w_2 x_2)}}$$
$$\partial_{w_1} L_{log} = \frac{-y x_1 e^{-y(w_0 + w_1 x_1 + w_2 x_2)}}{1 + e^{-y(w_0 + w_1 x_1 + w_2 x_2)}}$$
$$\partial_{w_2} L_{log} = \frac{-y x_2 e^{-y(w_0 + w_1 x_1 + w_2 x_2)}}{1 + e^{-y(w_0 + w_1 x_1 + w_2 x_2)}}$$

For the given weights, $e^{-y(w_0+w_1x_1+w_2x_2)} = e^{-yx_1}$.

$$\nabla_w L_{log}(1/2, 3) = \frac{e^{-1/2}}{1+e^{-1/2}} \begin{pmatrix} -1 \\ -1/2 \\ -3 \end{pmatrix}$$

$$\nabla_w L_{log}(2, -2) = \frac{e^{-2}}{1+e^{-2}} \begin{pmatrix} -1 \\ -2 \\ 2 \end{pmatrix}$$

$$\nabla_w L_{log}(-3, 1) = \frac{e^{3}}{1+e^{3}} \begin{pmatrix} 1 \\ -3 \\ 1 \end{pmatrix}$$

If $1 - y(w_0 + w_1x_1 + w_2x_2) < 0$, $\nabla_w L_{hinge} = 0$. *If* $1 - y(w_0 + w_1x_1 + w_2x_2) > 0$,

$$\partial_{w_0} L_{hinge} = -y$$
$$\partial_{w_1} L_{hinge} = -yx_1$$
$$\partial_{w_2} L_{hinge} = -yx_2$$

$$\nabla_w L_{hinge}(1/2, 3) = \begin{pmatrix} -1 \\ -1/2 \\ -3 \end{pmatrix}$$

$$\nabla_w L_{hinge}(2, -2) = \begin{pmatrix} -1 \\ -2 \\ 2 \end{pmatrix}$$

$$\nabla_w L_{hinge}(-3, 1) = \begin{pmatrix} 1 \\ -3 \\ 1 \end{pmatrix}$$

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 gradient from log loss is never zero, but approaches zero as* $y\mathbf{w}^T\mathbf{x}$ *increases (as the loss decreases). The gradient from hinge loss is exactly zero whenever* $y\mathbf{w}^T\mathbf{x} > 1$. *For a linearly separable dataset,* $y\mathbf{w}^T\mathbf{x}$ *can be made arbitrarily large without changing the decision boundary by scaling* $\mathbf{w}$, *so we can make hinge loss zero for any boundary that separates the dataset.*

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 issue in 1D was that we can scale* $\mathbf{w}$ *as large as we want while keeping zero loss. By adding the magnitude of* $\mathbf{w}$ *to the loss function, we penalize making* $\mathbf{w}$ *very large, so we can't "cheat" that way.*

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 with continuous outputs (not classification 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: *Adding regularization cannot decrease in-sample error, because the learning algorithm was minimizing in-sample error before we added it, and now it's minimizing something different.*

Regularization is not guaranteed to reduce out-of-sample error, because it is fundamentally impossible to make guarantees about out of sample error, it just makes overfitting less likely.

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 *is just the number of non-zero weights, which doesn't change with small adjustments to weight values, so it can't be optimized with gradient descent well.*

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 learning objective 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 learning, 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.
- The ℓ_2-*regularized* logistic learning objective is what you aim to minimize during the training. However, when computing the training error and the testing error, you should compute the *unregularized* logistic error.

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.

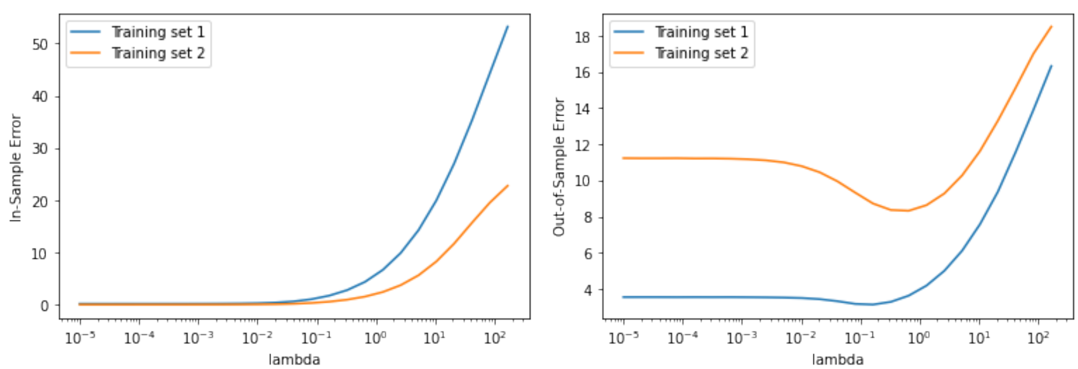
ii. Plot the average test error (E_{out}) versus different λs using wine_validation.txt as the test set.

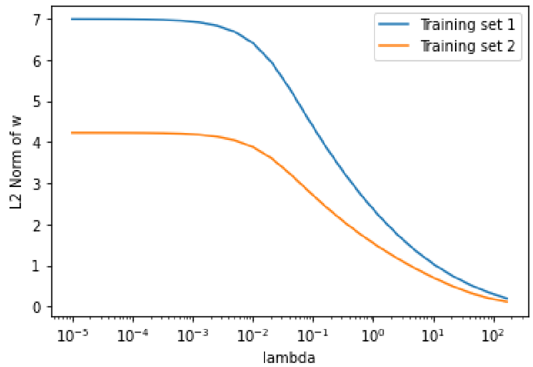
iii. Plot the ℓ_2 norm of $\mathbf{w}$ versus different λs.

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: *Colab notebook*

Note: I wasn't able to observe overfitting with $\eta = 5 \times 10^{-4}$ within a reasonable computation time, so I increased η to 5×10^{-3} and reduced the number of epochs to 2000, which didn't significantly impact the convergence of the model other than speeding it up.





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: *Training set 2 sees slightly lower in-sample error, and much higher out-of-sample error. This is because a smaller training set is easier for gradient descent to fit, giving lower in-sample error, but is a worse repesenation of the actual data, giving higher 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: *With small λ, the models overfit to the training data, which causes a higher out-of-sample error. With very large λ, the models underfit the data, and have high in-sample and out-of-sample error. Some middle values of λ increase in-sample error but reduce out-of-sample error.*

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: *Increasing λ increases the penalty for having large ℓ_2 norm, so the larger λ is, the smaller ℓ_2 ends up being when the regularized loss is minimized.*

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 pick λ near 1, which minimized the validation error as shown in my plots.*

3 Lasso (ℓ_1) vs. Ridge (ℓ_2) Regularization [25 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.

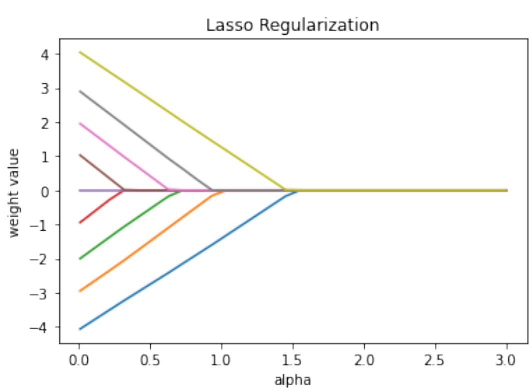
Problem A [11 points]: The tab-delimited file problem3data.txt on the course website contains 1000 9-dimensional datapoints. The first 9 columns contain $x_1, \ldots, x_9$, and the last column contains the target value y.

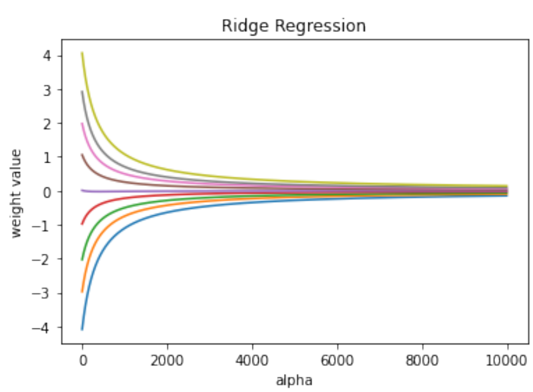
i. Train a linear regression model on the problem3data.txt data with Lasso regularization for regularization strengths α in the vector given by `numpy.linspace(0.01, 3, 30)`. On a single plot, plot each of the model weights $w_1, ..., w_9$ (ignore the bias/intercept) as a function of α.

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: *Colab notebook*





With Lasso regularization, the number of weights that are exaclty zero increases consistently as α increases. With Ridge regression, as far as I can tell none of the weights are ever exactly zero.

Problem B [7 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:

$$\begin{aligned}
0 &= \frac{d}{dw}\left(\lambda|w| + (\mathbf{y} - \mathbf{x}w)^T(\mathbf{y} - \mathbf{x}w)\right) \\
&= \frac{d}{dw}\left(\lambda|w| + \mathbf{y}^T\mathbf{y} - \mathbf{y}^T\mathbf{x}w - \mathbf{x}^T\mathbf{y}w + \mathbf{x}^T\mathbf{x}w^2\right) \\
&= \lambda\,\mathrm{sign}(w) - 2\mathbf{y}^T\mathbf{x} + 2\mathbf{x}^T\mathbf{x}w \\
&= \frac{1}{2}\lambda\,\mathrm{sign}(w) - \mathbf{y}^T\mathbf{x} + \mathbf{x}^T\mathbf{x}w
\end{aligned}$$

If we know the sign of w without regularization

$$w = \frac{\mathbf{y}^T\mathbf{x} - \frac{1}{2}\lambda\,\mathrm{sign}(w)}{\mathbf{x}^T\mathbf{x}}$$

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:

$$\lambda = \frac{2\mathbf{y}^T\mathbf{w}}{\mathbf{x}^T\mathbf{x}}$$

will set w to zero.

Problem C [7 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:

$$\begin{aligned}
0 &= \frac{d}{dw}\left(\lambda \mathbf{w}^T\mathbf{w} + (\mathbf{y}-\mathbf{X}\mathbf{w})^T(\mathbf{y}-\mathbf{X}\mathbf{w})\right) \\
&= \frac{d}{dw}\left(\lambda \mathbf{w}^T\mathbf{w} + \mathbf{y}^T\mathbf{y} - (\mathbf{y}^T\mathbf{X}\mathbf{w})^T - \mathbf{y}(\mathbf{X}\mathbf{w})^T + (\mathbf{X}\mathbf{w})^T\mathbf{X}\mathbf{w}\right) \\
&= \frac{d}{dw}\left(\lambda \mathbf{w}^T\mathbf{w} + \mathbf{y}^T\mathbf{y} - 2\mathbf{w}^T\mathbf{X}^T\mathbf{y} + \mathbf{w}^T\mathbf{X}^T\mathbf{X}\mathbf{w}\right) \\
&= 2\lambda\mathbf{w} - 2\mathbf{X}^T\mathbf{y} + 2\mathbf{X}^T\mathbf{X}\mathbf{w} \\
&= \lambda\mathbf{w} - \mathbf{X}^T\mathbf{y} + \mathbf{X}^T\mathbf{X}\mathbf{w}
\end{aligned}$$

$$\begin{aligned}
(\mathbf{X}^T\mathbf{X} + \lambda\mathbf{I})\mathbf{w} &= \mathbf{X}^T\mathbf{y} \\
\mathbf{w} &= (\mathbf{X}^T\mathbf{X} + \lambda\mathbf{I})^{-1}\mathbf{X}^T\mathbf{y}
\end{aligned}$$

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 is no λ such that $\mathbf{w} = 0$.*

4 Convexity and Lipschitz Continuity [10 Points]

This problem develop the notions of convexity and Lipschitz-continuity. These are widely applicable concepts in machine learning that we will explore further over the next few assignments.

A set C is convex if the line segment between any two points in C lies in C, i.e., if for any $x_1, x_2 \in C$ and any θ with $0 \leq \theta \leq 1$, we have

$$\theta x_1 + (1-\theta)x_2 \in C$$

Problem A [5 points]: Let $C \subseteq \mathbb{R}^n$ be a convex set, with $x_1, \ldots, x_k \in C$, and let $\theta_, \ldots, \theta_k \in \mathbb{R}$ satisfy $\theta_i \geq 0$, and $\theta_1 + \cdots + \theta_k = 1$. Show that $\theta_1 x_1 + \cdots + \theta_k x_k \in C$.
Hint: The definition of convexity is that this holds for $k = 2$; you must show it for arbitrary k.

Solution A: *Assume it holds for k.*

Let $y_k = \theta_1 x_1 + \cdots + \theta_k x_k$, and let $0 \leq \theta_{k+1} \leq 1$. Given $x_{k+1} \in C$, since $y_k \in C$, $(1-\theta_{k+1})y_k + \theta_{k+1}x_{k+1} = (1-\theta_{k+1})\theta_1 x_1 + \cdots + (1-\theta_{k+1})\theta_k x_k + \theta_{k+1}x_{k+1} \in C$.

Therefore the case for k implies the case for $k+1$, so it holds for abitrary $k > 1$ by induction.

Problem B [5 Extra Credit points]: Consider a metric space (X, d_X) and (Y, d_Y), where X and Y are sets and d_X and d_Y denote the metric on X and Y, respectively. Then, a function $f : X \to Y$ is said to be K-Lipschitz continuous if and only if there exists a real-valued constant $K \geq 0$ such that, for all x_1 and x_2 in X, we have:

$$\frac{d_Y(f(x_1), f(x_2))}{d_X(x_1, x_2)} \leq K$$

Consider the metric spaces $(D_i, \|\cdot\|)$ for all $i \in \{1, \ldots, t+1\}$, where $\|\cdot\|$ is the L_2 norm metric. Suppose that the sequence of functions $f_1, \ldots, f_t$ satisfies the property that $f_i : D_i \to D_{i+1}$ is L-Lipschitz continuous for all $i \in \{1, 2, \cdots, t\}$. Then, show that their composition $(f_t \circ f_{t-1} \circ \cdots \circ f_1)$ is L^t-Lipschitz continuous.

Solution B: