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

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: *For a classification problem, the squared loss landscape is non-convex. Thus, gradient descent will not always converge to the correct solution. This loss function is also biased towards the outliers.*

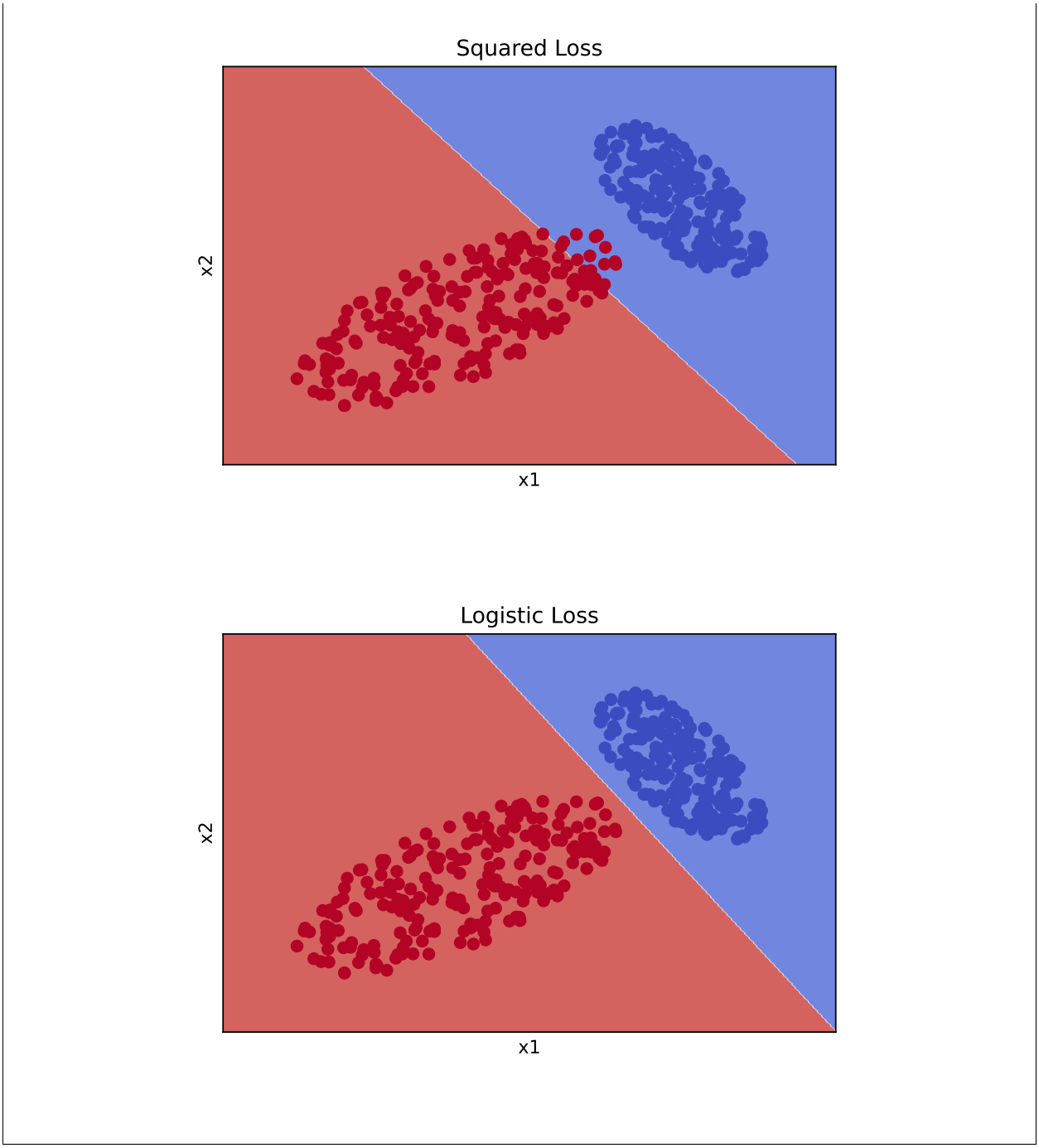
Problem B [9 points]: A dataset is included with your problem set: `problem1data1.txt`. The first two columns represent x_1, x_2, and the last column represents the label, $y \in \{-1, +1\}$.

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.

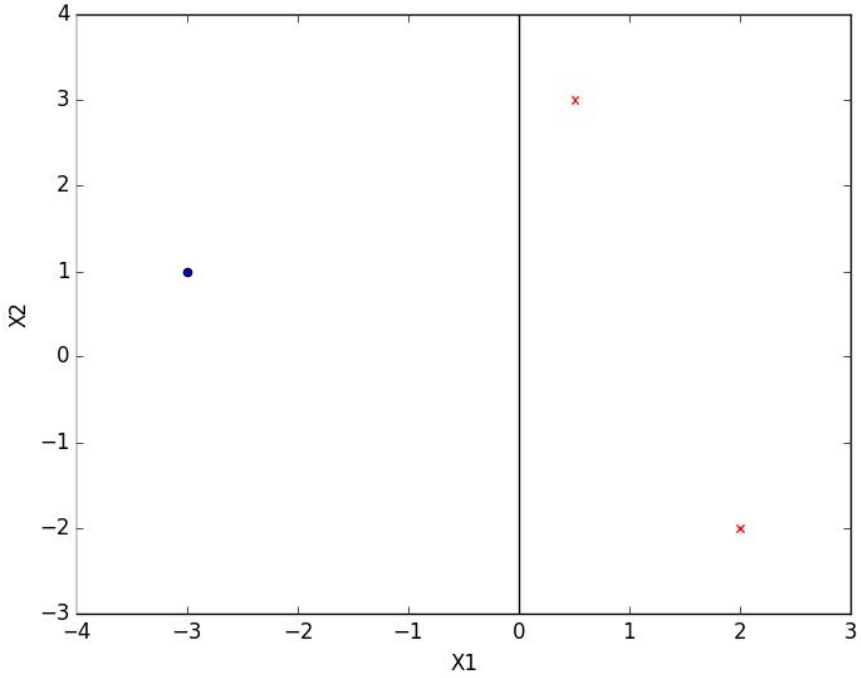
What differences do you see in the decision boundaries learned using the different loss functions? Provide a qualitative explanation for this behavior.

Solution B: *https://colab.research.google.com/drive/1w02zDKYTBq8EoSJIUb3EsAH2dQBc1LAf*



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:

$$\nabla_w L_{log} = \frac{-y\mathbf{x}}{1 + e^{-y\mathbf{w}^T\mathbf{x}}} \tag{1}$$

$$\nabla_w L_{hinge} = \begin{cases} -y\mathbf{x} & y\mathbf{w}^T\mathbf{x} < 1 \\ 0 & y\mathbf{w}^T\mathbf{x} > 1 \end{cases}$$

x pt [1. 0.5 3.] Log Loss Gradient: [-0.37754067 -0.18877033 -1.13262201] Hinge Loss Gradient: [-1. -0.5 -3.] x pt [1 2 -2] Log Loss Gradient: [-0.11920292 -0.23840584 0.23840584] Hinge Loss Gradient: [0. 0. 0.] x pt [1 -3 1] Log Loss Gradient: [0.04742587 -0.14227762 0.04742587] Hinge Loss Gradient: [0. 0. 0.]

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 for the log loss function will never converge to zero. The gradients are sigmoidal, so they will asymptote to zero but never reach zero exactly. On the other hand, the gradients for the hinge loss function become zero as soon as* $y\mathbf{w}^T\mathbf{x} > 1$.

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 hinge loss is not a max margin classifier because the gradients become zero as soon as all of the points are correctly classified, as shown in solution C. Thus, the optimization will terminate early. The regularization term ensures that the gradients will not go to zero prematurely, thus becoming a max margin classifier.*

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 will never decrease the training error because the regularization factor λ and the norm are positive values. The penalty term does not necessarily always decrease the out-of-sample errors. In the case where the regularization term is too large, the weights will all become close to zero, which is most likely not a good fit to the data (high bias).*

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 ℓ_0 norm is non-differentiable, so it cannot be used with gradient descent. Furthermore, it is very punishing, so it would only be useful in high-dimensional problems where extreme sparsity is desired.*

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):

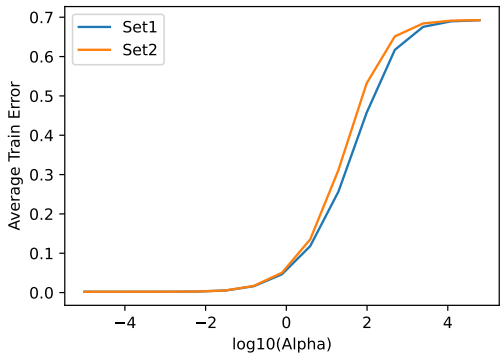
i. Plot the average training error (E_{in}) versus different λs.

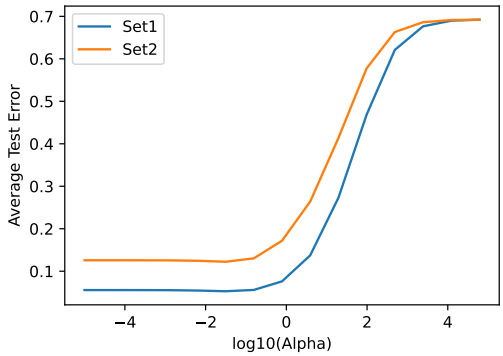
ii. Plot the average test error (E_{out}) versus different λs using wine_validation.txt as the test set.

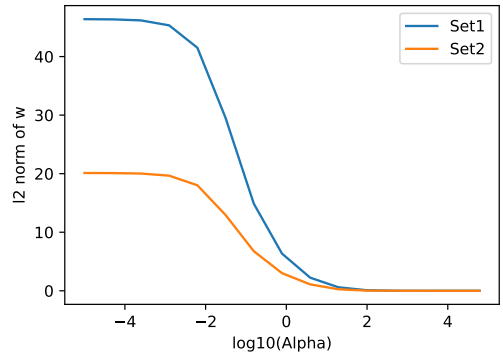
iii. Plot the ℓ_2 norm of $\mathbf{w}$ versus different λs.

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/1WxjpRfV9HYWwGyLJEuzsyYr2Y1ERSxIg#scrollTo=JGkf7kLc3FYW*







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: *While the average training error per sample is similar for the two datasets, the test errors for set 1 are lower, especially for small values of the regularization parameter. This makes sense, because training on a larger dataset will generalize better to the unseen test set.*

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 large $\lambda > 1$, there is evidence of underfitting as the train and test error are both high. The regularization penalty is too high, which prevents the model from learning the data. For small $\lambda < 1e-2$, there is evidence of overfitting, as the test error is higher than that of slightly larger values of λ.*

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 ℓ_2 norm of $\boldsymbol{w}$ decreases as λ is increased. This makes sense because the regularized loss function penalizes large values in $\boldsymbol{w}$, and λ determines the magnitude of that penalty.*

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 $\lambda = 0.0315$, because this corresponds to the lowest value of the average normalized test error (0.1223).*

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.

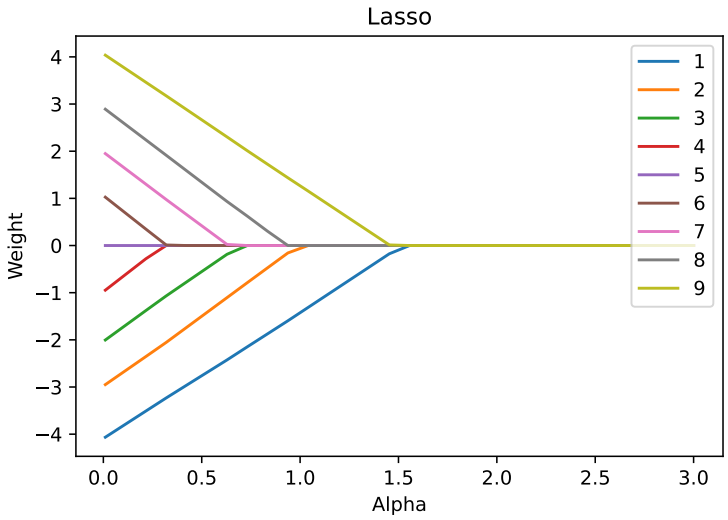
Problem A [12 points]: The tab-delimited file problem3data.txt on the course website contains 1000 9-dimensional datapoints. The first 9 columns contain $x_1, \ldots, x_9$, and the last column contains the target value y.

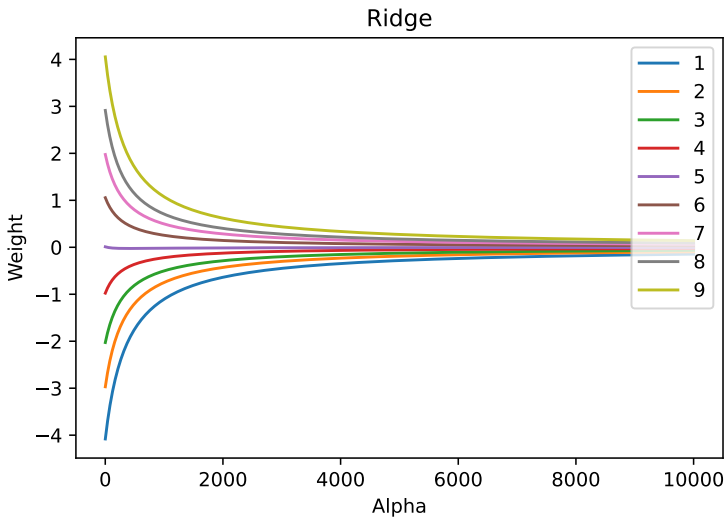
i. Train a linear regression model on the problem3data.txt data with Lasso regularization for regularization strengths α in the vector given by `numpy.linspace(0.01, 3, 30)`. On a single plot, plot each of the model weights $w_1, ..., w_9$ (ignore the bias/intercept) as a function of α.

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/1zArPjMh7TJEn1epv2cTRxZckq3XC2mj8





For the lasso model, as the regularization paremeter increases, more and more of the model weights become exactly zero, until they are all zero. This makes sense since the ℓ_1 norm encourages sparsity in the features. For the ridge model, the weights decrease, but none of them go to zero.

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:

$$\partial_w(\|\mathbf{y} - \mathbf{x}w\|^2 + \lambda\|w\|_1) = -2\mathbf{x}^T\mathbf{y} + 2\mathbf{x}^T\mathbf{x}w + \lambda\frac{w}{|w|} = 0 \tag{2}$$

where $|w|$ is the absolute value of w, so $\frac{w}{|w|}$ is just the sign of w.

$$w = \frac{\mathbf{x}^T\mathbf{y}}{\mathbf{x}^T\mathbf{x}} - \frac{\lambda}{2\mathbf{x}^T\mathbf{x}}\frac{w}{|w|} \tag{3}$$

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: *$w = 0$ when $\lambda \geq 2\mathbf{x}^T\mathbf{y}$.*

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: *To find the minimum, we want to solve:*

$$\frac{\partial}{\partial \mathbf{w}}(\|\mathbf{y} - \mathbf{X}\mathbf{w}\|^2 + \lambda\|\mathbf{w}\|_2^2) = 0 \tag{4}$$

$$\frac{\partial}{\partial \mathbf{w}}[(\mathbf{y} - \mathbf{X}\mathbf{w})^T(\mathbf{y} - \mathbf{X}\mathbf{w}) + \lambda\mathbf{w}^T\mathbf{w}] = 0 \tag{5}$$

$$-2\mathbf{X}^T(\mathbf{y} - \mathbf{X}\mathbf{w}) + 2\lambda\mathbf{w} = 0 \tag{6}$$

$$\mathbf{X}^T\mathbf{y} = \mathbf{X}^T\mathbf{X}\mathbf{w} + \lambda\mathbf{w} \tag{7}$$

$$\mathbf{w} = (\mathbf{X}^T\mathbf{X} + \lambda\mathbf{I})^{-1}\mathbf{X}^T\mathbf{y} \tag{8}$$

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, based on the formula given above, in 1D, if $\lambda > 0$, then both x and y must be zero for w to be zero, which is a trivial solution.*