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 $\rightarrow$ 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 a 2D classification, the squared loss is optimizing to minimize the distance of each point from a line in the 2D plane. As such, there is exactly one set of parameters that minimizes the squared loss function (this is also evident when plotting the loss function, as there is exactly one solution that minimizes the squared loss). However, as we saw in lecture, it is possible that some sets of points may "pull" the decision boundary such that it fails to separate the two sets of points. Conversely, other loss functions have many solutions of parameters (i.e. many possible* $\mathbf{w}$*) that can satisfy the loss function (this is evident by the long right handed tail with loss=0 in the e.g. the perceptron and Hinge losses). Another way of thinking about this: we want our loss function to look as close to the 0/1 loss as possible, and the squared loss does not match it as well as other loss functions (again because the squared loss function lacks an extended right tail where loss=0).*

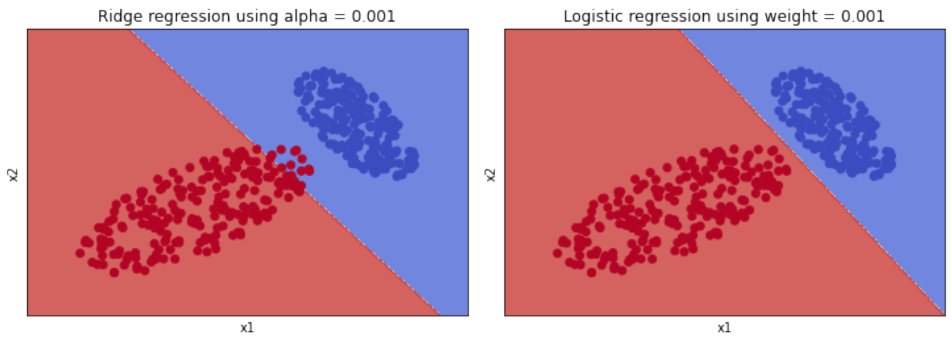
Problem B [9 points]: A dataset is included with your problem set: `problem1data1.txt`. The first two columns represent x_1, x_2, and the last column represents the label, $y \in \{-1, +1\}$.

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.

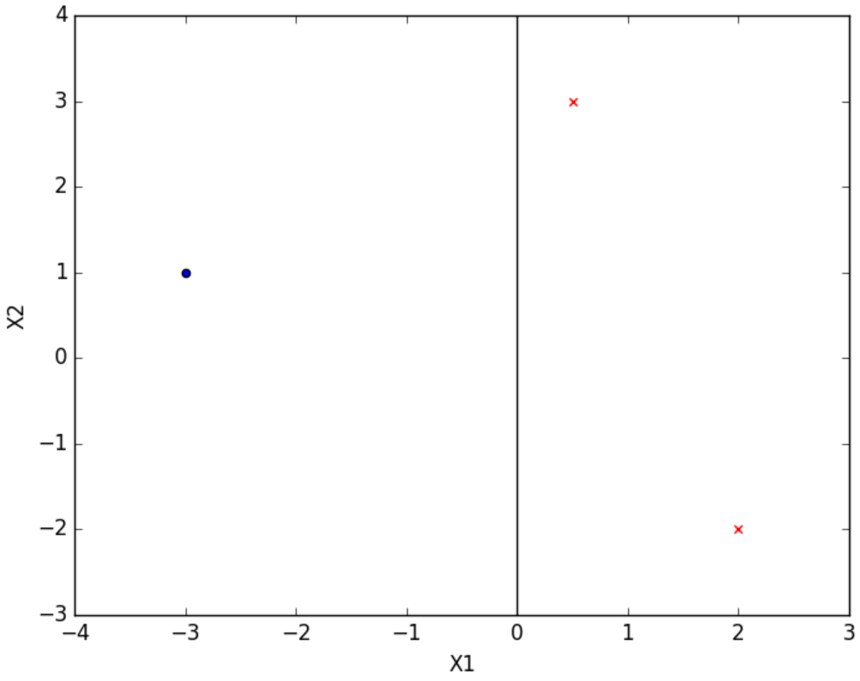
Solution B: *Link to Google Colab noteboke for Problem 1.*



The ridge regression does not separate the data even though the dataset is linearly separable, while the logistic regression does. The ridge regression is minimizing for the distance from the decision boundary, and interestingly the decision boundary does not move very much across different weights of regularization. Logistic regression on the other hand considers that the values of y given $\mathbf{w}$ (the likelihood) and the values of $\mathbf{w}$ (our priors) are both Gaussian distributed, which allows the regularization to impose "slack" in the parameter space that allows us to deviate from the "true solution" of least squares.

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 $\nabla_w L_{\text{log}}$ 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: *To set up the problem, we can define* $\mathbf{X} = \begin{bmatrix} 1 & 0.5 & 3 \\ 1 & 2 & -2 \\ 1 & -3 & 1 \end{bmatrix}$, $\mathbf{y} = \begin{bmatrix} 1 & 1 & -1 \end{bmatrix}$, *and* $\mathbf{w} = \begin{bmatrix} 0 & 1 & 0 \end{bmatrix}$.

Hinge loss:

$$\nabla_w L_{hinge} = \frac{\partial L_{hinge}}{\partial w}$$

$$\nabla_w L_{hinge} = \begin{cases} 0 & \textit{if } y_i \mathbf{w}^T \mathbf{x}_i > 1 \\ \frac{\partial(1 - y_i \mathbf{w}^T \mathbf{x}_i)}{\partial w} & \textit{if } y_i \mathbf{w}^T \mathbf{x}_i < 1 \end{cases}$$

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

We know $y\mathbf{w}^T\mathbf{X} = \begin{bmatrix} 0.5 & 2 & 3 \end{bmatrix}$, *which tells us that the contribution the gradient from the second and last rows of* $\mathbf{X}$ *will be* $\begin{bmatrix} 0 & 0 & 0 \end{bmatrix}$. *As such, for each point* $\mathbf{x}_i$, *where* $\mathbf{x}_i$ *corresponds to the inputs from the ith row of* $\mathbf{X}$, *we* *have $\nabla_w l_i(w)$, the gradient calculated at each point $\mathbf{x}_i$: $\nabla_w l_1 = \begin{bmatrix} -1 & -0.5 & -3 \end{bmatrix}$, $\nabla_w l_2 = \begin{bmatrix} 0 & 0 & 0 \end{bmatrix}$, and $\nabla_w l_3 = \begin{bmatrix} 0 & 0 & 0 \end{bmatrix}$, the sum of which is $\begin{bmatrix} -1 & -0.5 & -3 \end{bmatrix}$.*

Logistic loss:

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

Evaluating the gradient again for each row of $\mathbf{X}$ we find: $\nabla_w l_1 \approx \begin{bmatrix} -0.378 & -0.189 & -1.13 \end{bmatrix}$, $\nabla_w l_2 \approx \begin{bmatrix} -0.119 & -0.238 & 0.238 \end{bmatrix}$, and $\nabla_w l_3 \approx \begin{bmatrix} 0.047 & -0.142 & 0.047 \end{bmatrix}$, the sum of which is $\approx \begin{bmatrix} -0.449 & -0.569 & -0.847 \end{bmatrix}$.

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:
Hinge loss:

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

As the conditions in the piecewise function suggests, the gradient for Hinge loss converges to 0 when $y_i\mathbf{w}^T\mathbf{x} > 1$.

Logistic loss:
Given $\nabla_w L_{log} = \frac{-y\mathbf{x}e^{-y\mathbf{w}^T\mathbf{x}}}{1+e^{-y\mathbf{w}^T\mathbf{x}}}$, the gradient will only ever converge to 0 if $y = 0$ or $\mathbf{x} = 0$.

With the understanding that $\mathbf{w}$ scales such that it does not change the decision boundary, you can imagine a situation where a solution for $\mathbf{w}$ exists where you scale the solution $\mathbf{w}$ that creates a decision boundary, such that you now are over-fitting your data to reduce, or event eliminate, training error. By scaling $\mathbf{w}$, you maintain the decision boundary while still finding a model that better fits the training data. This can be the case of adding a regularization parameter with a small value of λ to the logistic loss case, which will maintain the decision boundary but scale $\mathbf{w}$ such that it over-fits the data, reducing/eliminating training error.

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: *In general the regularization penalty works to increase the value of the loss, such that you limit the solutions the are possible to only the "optimal" solutions that result in a L_{hinge} that maximizes margin (i.e. you are adding stringency to your model to require that the loss is "more minimized" compared to the non-penalty case). Therefore to consider this added penalty that requires stringency, you must minimize the entire loss with penalty. Another way of thinking about this, is with the regularization penalty, you are now more free to scale* $\mathbf{w}$ *to maintain the decision boundary (and find the maximum margin classifier), but with lower risk of over-fitting your data.*

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: *Adding the penalty term alone does not decrease training error (but it is still possible decrease training error when you are including the penalty term), and because the penalty term reduces over-fitting, the penalty term should theoretically decrease the out-of-sample error. The reasoning here is that you are preventing over-fitting such that your model should better represent out-of-sample, real-world data that you introduce when you deploy your model. However, there are situations when the weight of regularization is high that will create an unstable model, and increase both in-sample and out-of-sample error.*

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: *Because the ℓ_0 norm is technically not a norm, there is no closed form solution that you can use to differentiate, and thus it is probably difficult to find the gradient in some cases (i.e. it can be non-convex which makes it difficult to implement).*

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):

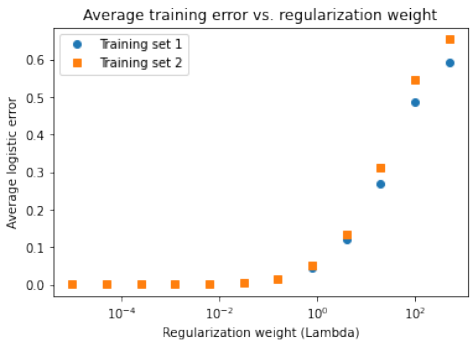
i. Plot the average training error (E_{in}) versus different λs.

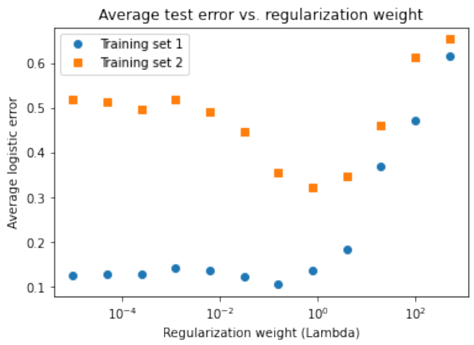
ii. Plot the average test error (E_{out}) versus different λs using wine_validation.txt as the test set.

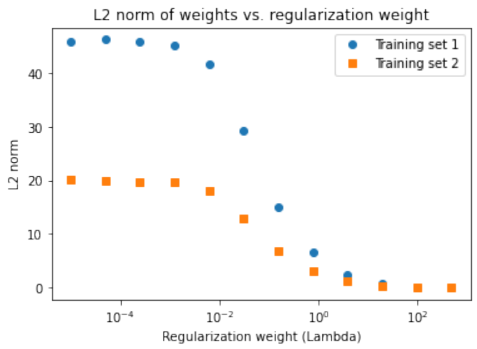
iii. Plot the ℓ_2 norm of $\mathbf{w}$ versus different λs.

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: *Link to Google Colab notebook for Problem 2.*







Note that I observed numerical instability for the largest 3 values of λ, $\lambda_{12} = 2,441.40625$, $\lambda_{13} = 12,207.03125$, *and* $\lambda_{14} = 61,035.15625$.

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 between the two data sets are relatively the same for most values of λ, but the training error from set 2 begins to deviate slightly at λ_7, and the training error from set 2 becomes increasingly greater than the training error from set 1 at $\lambda > \lambda_7$. The test errors from the two data sets vary quite significantly, and the test error for the model trained using set 1 is consistently lower than the test error for the model trained using set 2. The test errors begin to converge at $\lambda > \lambda_8$.*

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: *When λ is small (around $\lambda \leq \lambda_4$), which minimizes the effect of the regularization penalty, you are over-fitting your data. When λ is large (beginning around $\lambda \geq \lambda_8$), which increases the effect of the regularization penalty, you are under-fitting. This is evident by the large training and test errors (and numerical instability) at large values of λ. As such, around $\lambda = \lambda_6$, where the test error is minimized, you see the optimal choice of λ that does not over-fit or under-fit the data.*

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 is more or less constant until $\lambda = \lambda_4$, at which point the ℓ_2 norm decreases until it approaches 0. As such, the ℓ_2 norm is large for small values of λ, and becomes smaller as λ increases.*

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_7 because it has the lowest average test error, while still maintaining a relatively low training error. More notably, the test error at λ_7 is a local minima, which suggests that we are in a regime that is not over-fitting and under-fitting the data when we use λ_7.*

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.

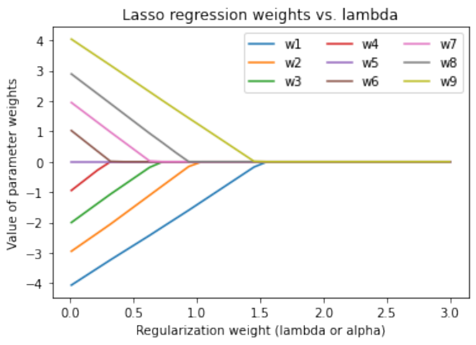
Problem A [12 points]: The tab-delimited file problem3data.txt on the course website contains 1000 9-dimensional datapoints. The first 9 columns contain $x_1, \dots, x_9$, and the last column contains the target value y.

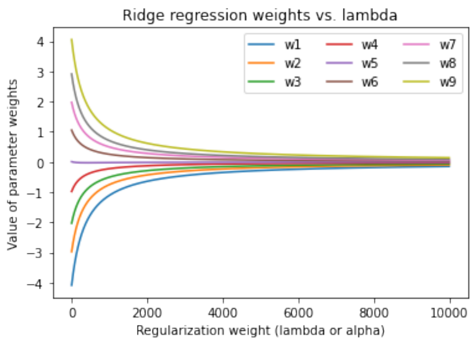
i. Train a linear regression model on the problem3data.txt data with Lasso regularization for regularization strengths α in the vector given by `numpy.linspace(0.01, 3, 30)`. On a single plot, plot each of the model weights $w_1, ..., w_9$ (ignore the bias/intercept) as a function of α.

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

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: *Link to Google Colab notebok for Problem 3.*





With Lasso regression, all the model weights are equal to 0 at large values of α, while with Ridge regression, none of the model weights ever become exactly 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:

$$L_{Lasso} = \sum_{i=1}^{N} (y_i - wx_i)^2 + \lambda |w|$$
$$\frac{\partial L_{Lasso}}{\partial w} = \sum_{i=1}^{N} -2x_i (y_i - wx_i) + \lambda \frac{w}{|w|}$$

Changing notation to vector form:

$$\frac{\partial L_{Lasso}}{\partial w} = -2\mathbf{x}^T (\mathbf{y} - \mathbf{x}w) + \lambda \frac{w}{|w|}$$

Expanding this into a piecewise function:

$$\frac{\partial L_{Lasso}}{\partial w} = \begin{cases} -2\mathbf{x}^T (\mathbf{y} - \mathbf{x}w) + \lambda & \text{if } w > 0 \\ -2\mathbf{x}^T (\mathbf{y} - \mathbf{x}w) - \lambda & \text{if } w < 0 \end{cases}$$

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: *Given that $w \neq 0$ when $\lambda = 0$, we know that the gradient must satisfy $-2\mathbf{x}(\mathbf{y} - \mathbf{x}w) = 0$, which requires both $\mathbf{y}$ and $\mathbf{x}$ to be the same sign. As such, solving for λ in the piecewise function that gives a negative number will give the smallest value of λ such that $w = 0$:*

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

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: *Extending the solution from part B here:*

$$\frac{\partial L_{Ridge}}{\partial \mathbf{w}} = -2\mathbf{X}^T\left(\mathbf{y} - \mathbf{X}\mathbf{w}\right) + \lambda \frac{\partial \mathbf{w}^T\mathbf{w}}{\partial \mathbf{w}}$$
$$\frac{\partial L_{Ridge}}{\partial \mathbf{w}} = -2\mathbf{X}^T\left(\mathbf{y} - \mathbf{X}\mathbf{w}\right) + 2\lambda\mathbf{w}$$

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 does not exist a value for λ such that $w = 0$, as the gradient of the regularization term is dependent on $\mathbf{w}$: $2\lambda\mathbf{w}$. This makes it such that if $w = 0$, then the regularization term is also 0, making the entire term 0.*