Machine Learning & Data Mining
Derek Ing 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?

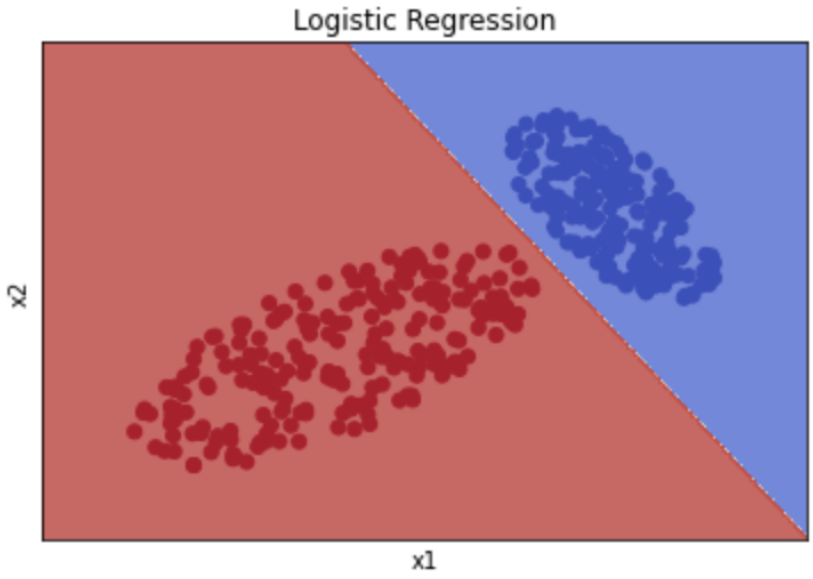
Solution A: *Squared loss is a bad choice of loss function to train on for classification problems because points are classified according to sign$(\mathbf{w}^T\mathbf{x})$, but squared loss will evaluate a point based on its raw score. If a point is classified correctly but has $|\mathbf{w}^T\mathbf{x}| >> 1$, the loss for that point would be large, which does not reflect correct classification.*

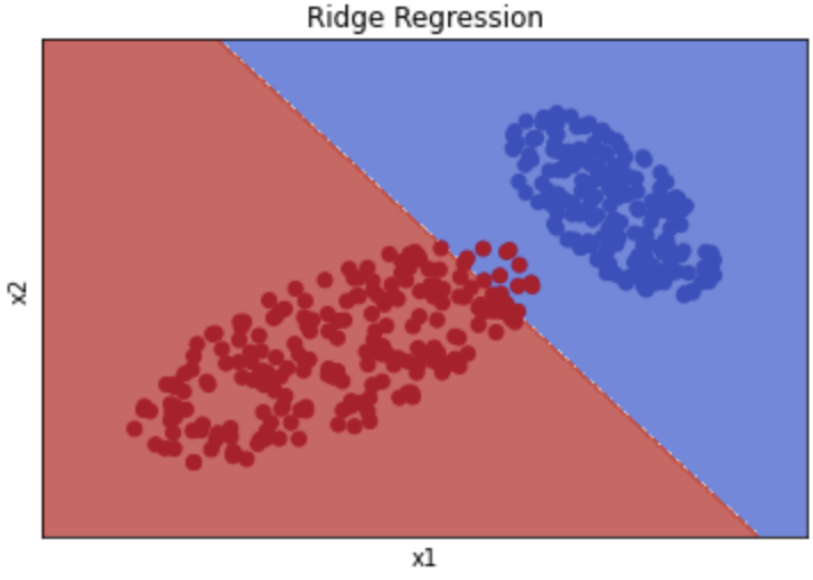
Problem B [9 points]: A dataset is included with your problem set: `problem1data1.txt`. The first two columns represent x_1, x_2, and the last column represents the label, $y \in \{-1, +1\}$.

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:
For the logistic regression model, the decision boundary correctly classifies all the points. However, the ridge regression model has a decision boundary that is shifted to the left of the logistic regression's decision boundary. The ridge regression model's decision boundary also incorrectly classifies some points. This is because $L_{squared}$ penalizes points that are far away from the decision boundary, even if they are classified correctly. Thus, the ridge regression decision boundary is shifted to minimize the points that are far from the boundary.

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.

./plots/SimpleDatasetWithDecisionBoundary.png

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:

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

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

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

x_1	x_2	y	$\nabla_w L_{hinge}$	$\nabla_w L_{log}$
1/2	*3*	*1*	*(-1,-0.5,-3)*	*(-0.378, -0.189, -1.133)*
2	*-2*	*1*	*(0,0,0)*	*(-0.119, -0.238, 0.238)*
-3	*1*	*-1*	*(0,0,0)*	*(0.047, -0.142, 0.047)*

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

Solution D: *The gradient of the log loss will converge to 0 as $y\mathbf{w}^T\mathbf{x}$ approaches infinity. The gradient of hinge loss will converge to 0 if all points are classified correctly and $y\mathbf{w}^T\mathbf{x} > 1$ for all $\mathbf{x}$. For a linearly separable dataset, we can reduce or eliminate training error if all points are correctly classified and we can scale up $\mathbf{w}$, all while preserving the decision boundary. For L_{hinge}, we want $y\mathbf{w}^T\mathbf{x} > 1$ for all $\mathbf{x}$ to reduce/eliminate training error. For L_{log}, the gradient will converge to 0 as $y\mathbf{w}^T\mathbf{x}$ increases.*

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: *Based on problem D, minimizing L_{hinge} can be done by scaling $\mathbf{w}$ without changing the decision boundary, which determines the margin. The penalty term $\lambda\|w\|^2$ constrains $\|w\|^2$ such that we cannot scale up*

$\mathbf{w}$ *to whatever we want. Thus, with the* $\lambda\|w\|^2$ *term, an SVM will minimize* $L_{hinge} + \lambda\|w\|^2$ *by changing the decision boundary and maximizing the margin.*

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 a regularization penalty term will not decrease the training error. The regularization term constrains the model complexity, which can prevent the model from optimally fitting the training data. Adding a penalty term will not always decrease the out-of-sample errors. With regularization, we trade lower variance for higher bias. Thus, if the model is overfitted, we can add regularization to decrease variance and prevent overfitting. However, if the regularization penalty is too large, the model could be underfitted, which may not decrease the 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: ℓ_0 *is rarely used because it is not continuous. This makes it hard to optimize the regularized error because we need to compute derivatives/gradients to minimize the error.*

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.

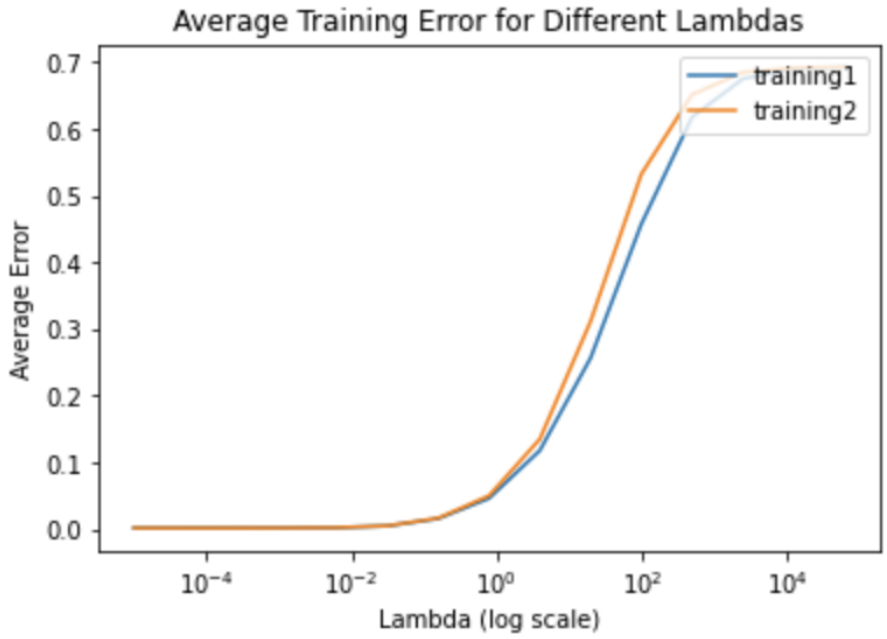
Problem C [16 points]: Do the following for both training data sets (wine_training1.txt and wine_training2.txt) and attach your plots in the homework submission (use a log-scale on the horizontal axis):

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

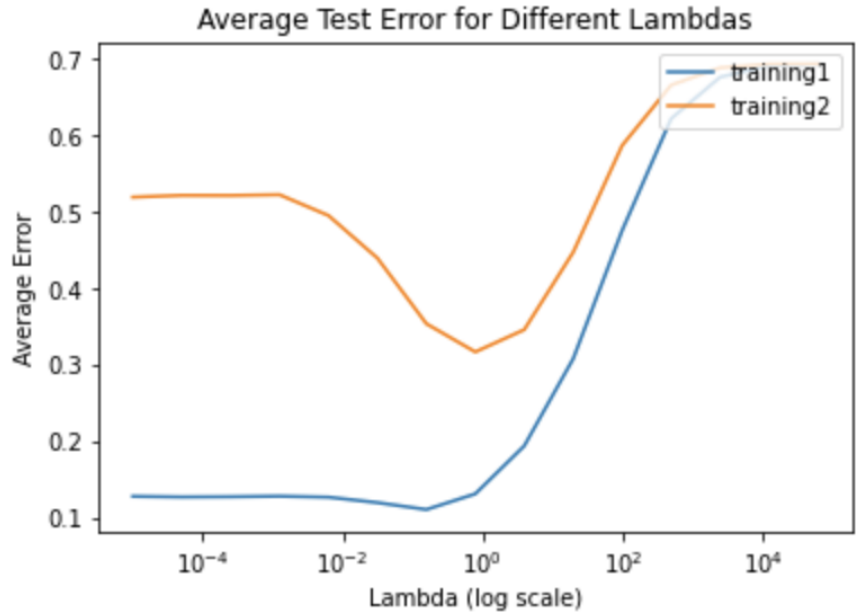
ii. Plot the average test error (E_{out}) versus different λs using wine_validation.txt as the test set.

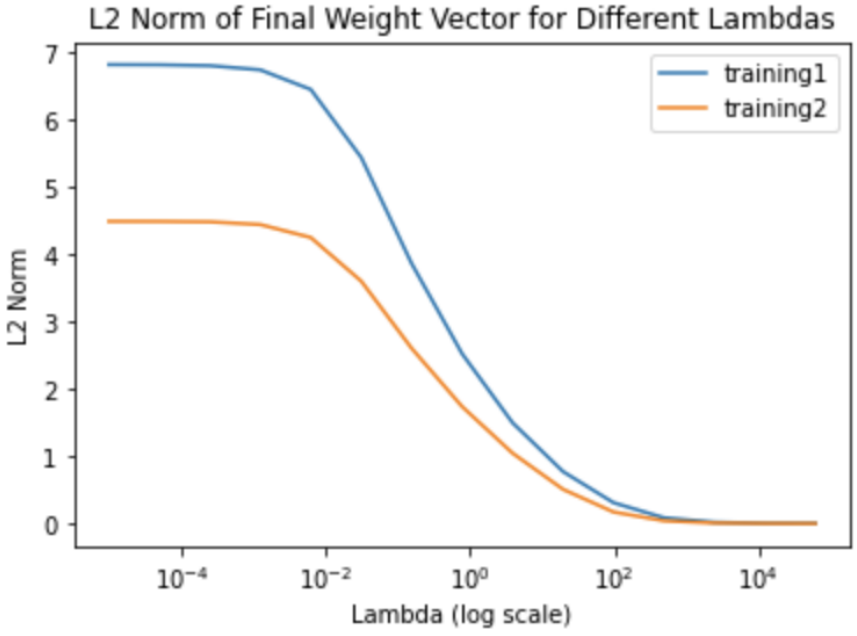
iii. Plot the ℓ_2 norm of $\mathbf{w}$ versus different λs.

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:





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 of wine_training1 and wine_training2 are about the same up until $\lambda = 1$. As λ increases more, wine_training2 increases faster and has a slightly greater training error than wine_training1. With greater values of λ, we will have less complex models. Thus, the model may not be able to fit the training data well. As a result, points with large error will affect the average training error of wine_training2 more than wine_training1 because wine_training2 has less data points.*

For all values of λ, wine_training1 has a smaller test error than wine_training2. This is because wine_training1 contains more data points than wine_training2. More training data decreases variance and can prevent overfitting. We can see there is overfitting on wine_training2 for $\lambda \leq 1$ as the test error is much greater than the training error, which is small. For $\lambda > 1$, the test errors start to increase fast for both training sets, but wine_training1 still has lower test error, which could be because there is too much regularization and the models are underfitting.

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 are small and do not change much for $\lambda < 1$, which means that the model is not overfitting for these values of λ. However, as λ increases, both training and test error increase rapidly. With more regularization, the complexity of the model is restricted, which could be causing the model to underfit.*

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 increases. With greater λ, the regularization penalty term is more significant in the regularized error. Thus, the ℓ_2 norm is constrained more to low values.*

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 $\lambda = 0.78125$ to train the final model. We see that test error is the smallest for wine_training2 at $\lambda = 0.78125$ and the test error is still relatively small. Since both errors are small, the model is not overfitting for $\lambda = 0.78125$. Regularizing with $\lambda = 0.78125$ would train the model to perform best on out-of-sample data.*

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.

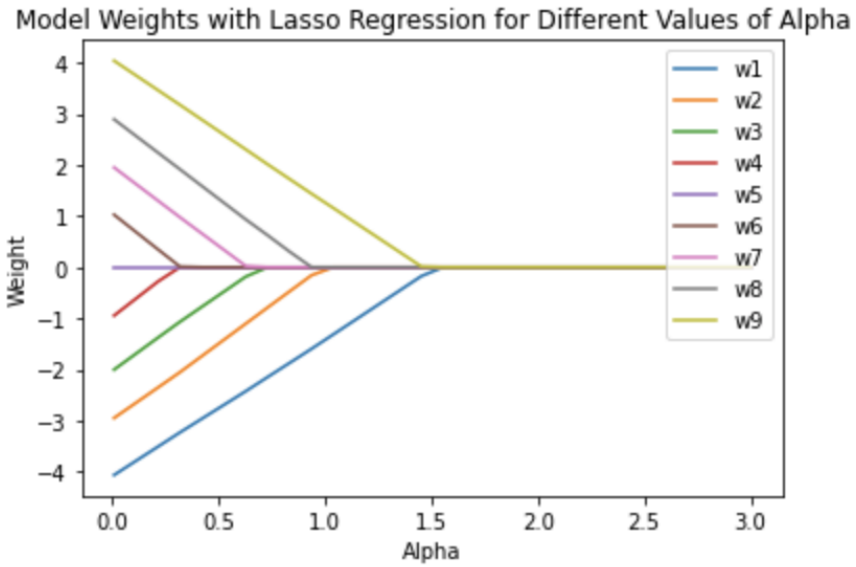
The two most commonly-used regularized regression models are Lasso (ℓ_1) regression and Ridge (ℓ_2) regression. Although both enforce "simplicity" in the models they learn, only Lasso regression results in sparse weight vectors. This problem compares the effect of the two methods on the learned model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

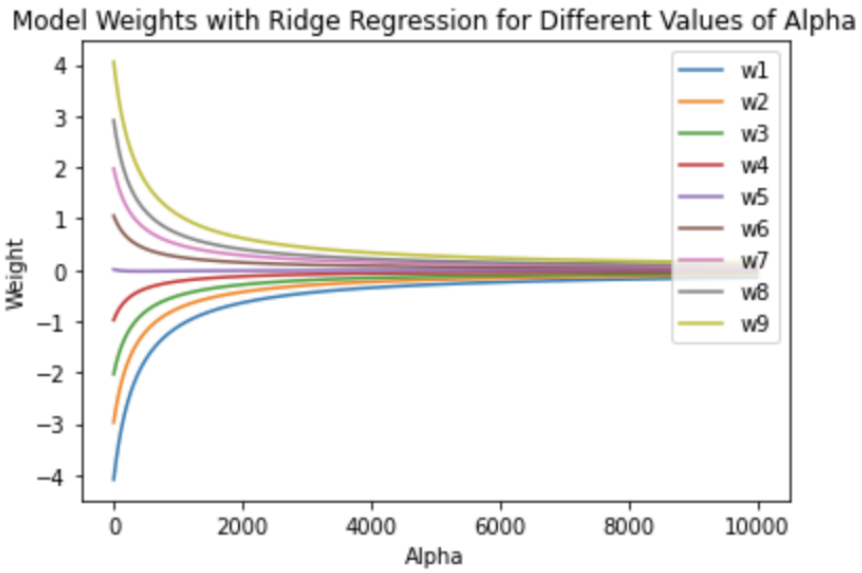
i. Train a linear regression model on the problem3data.txt data with Lasso regularization for regularization strengths α in the vector given by `numpy.linspace(0.01, 3, 30)`. On a single plot, plot each of the model weights $w_1, ..., w_9$ (ignore the bias/intercept) as a function of α.

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:



As the regularization parameter increases, the number of model weights that are exactly zero increase with Lasso Regression. For Ridge Regression, the model weights approach zero as the regularization parameter increases, but none of the model weights are exactly 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:

$$\nabla_w(\|\mathbf{y} - \mathbf{x}w\|^2 + \lambda\|\mathbf{w}\|_1) = -2\mathbf{x}^T(\mathbf{y} - \mathbf{x}w) + \begin{cases} \lambda & w > 0 \\ -\lambda & w < 0 \\ [-\lambda, \lambda] & w = 0 \end{cases}$$

$$w > 0 : w = \frac{2\mathbf{x}^T\mathbf{y} - \lambda}{2\mathbf{x}^T\mathbf{x}}$$
$$w < 0 : w = \frac{2\mathbf{x}^T\mathbf{y} + \lambda}{2\mathbf{x}^T\mathbf{x}}$$

$$w = \begin{cases} \frac{2\mathbf{x}^T\mathbf{x} - \lambda}{2\mathbf{x}^T\mathbf{x}} & 2\mathbf{x}^T\mathbf{y} > \lambda \\ \frac{2\mathbf{x}^T\mathbf{y} + \lambda}{2\mathbf{x}^T\mathbf{x}} & 2\mathbf{x}^T\mathbf{y} < -\lambda \\ 0 & -\lambda \leq 2\mathbf{x}^T\mathbf{y} \leq \lambda \end{cases}$$

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

Solution B.ii: *Yes, there exists a value for λ such that $w = 0$. The smallest such value is $\lambda = |2\mathbf{x}^T\mathbf{y}|$.*

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:

$$\begin{aligned} \nabla_w(\|\mathbf{y} - \mathbf{X}\mathbf{w}\|^2 + \lambda\|\mathbf{w}\|_2^2) &= -2\mathbf{X}^T(\mathbf{y} - \mathbf{X}\mathbf{w}) + 2\lambda\mathbf{w} \\ -2\mathbf{X}^T(\mathbf{y} - \mathbf{X}\mathbf{w}) + 2\lambda\mathbf{w} &= 0 \\ \mathbf{X}^T\mathbf{X}\mathbf{w} - \mathbf{X}^T\mathbf{y} + \lambda\mathbf{w} &= 0 \\ (\mathbf{X}^T\mathbf{X} + \lambda\mathbf{I})\mathbf{w} &= \mathbf{X}^T\mathbf{y} \\ \mathbf{w} &= (\mathbf{X}^T\mathbf{X} + \lambda\mathbf{I})^{-1}(\mathbf{X}^T\mathbf{y}) \end{aligned}$$

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: *No, there is no $\lambda > 0$ such that $w = 0$. For the 1-dimension case, we get the closed form solution as $w = \frac{\mathbf{x}^T\mathbf{y}}{\mathbf{x}^T\mathbf{x}+\lambda}$. Thus, $w = 0$ if $\mathbf{x}^T\mathbf{y} = 0$, which does not depend on lambda. As λ increases, w can asymptotically approach 0 but will never be exactly 0.*

Code links:
Problem 1
Problem 2
Problem 3