Machine Learning & Data Mining
Set 2

Caltech CS/CNS/EE 155
January 6^{th}, 2023
Manuel Rodriguez

Policies

- Due 9 PM PST, January 20th 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”

Machine Learning & Data Mining
Set 2

Caltech CS/CNS/EE 155
January 6^{th}, 2023
Manuel Rodriguez

1 Comparing Different Loss Functions [30 Points]

Relevant materials: lecture 3 & 4

We've discussed three loss functions for linear classification models so far:

- Squared loss: $L_{\text{squared}} = (1 - y\mathbf{w}^T\mathbf{x})^2$
- Hinge loss: $L_{\text{hinge}} = \max(0, 1 - y\mathbf{w}^T\mathbf{x})$
- Log loss: $L_{\text{log}} = \ln(1 + e^{-y\mathbf{w}^T\mathbf{x}})$

where $\mathbf{w} \in \mathbb{R}^n$ is a vector of the model parameters, $y \in \{-1, 1\}$ is the class label for datapoint $\mathbf{x} \in \mathbb{R}^n$, and we're including a bias term in $\mathbf{x}$ and $\mathbf{w}$. The model classifies points according to $\text{sign}(\mathbf{w}^T\mathbf{x})$.

Performing gradient descent on any of these loss functions will train a model to classify more points correctly, but the choice of loss function has a significant impact on the model that is learned.

Problem A [3 points]: Squared loss is often a terrible choice of loss function to train on for classification problems. Why?

Solution A: *When we are training for classification, we only care about the test points being on the correct side of the hypothesis classification. However, Squared loss is biased to try to minimize distance of all points so if there is one point that is very deep into one classification, the Squared Loss will skew the weights such that that distance will be minimized, thereby needlessly changing the weights in a way that prioritizes just one point rather than the general trend of all the points.*

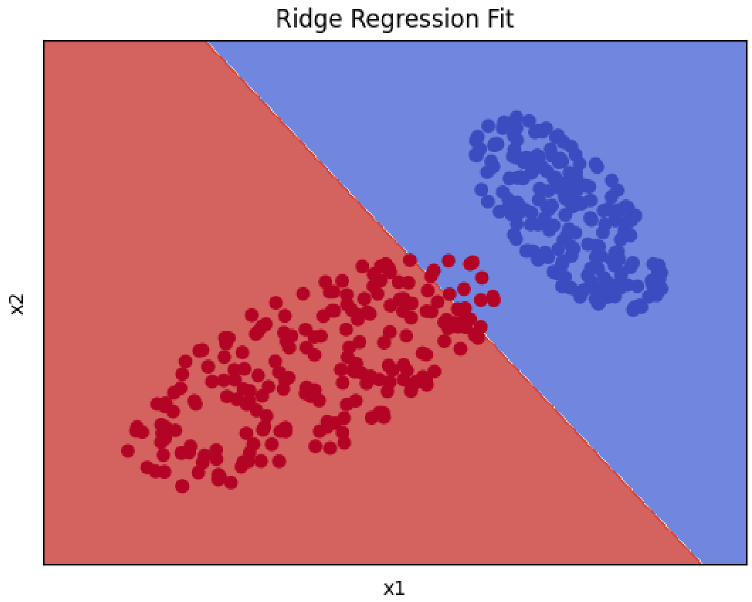
Problem B [9 points]: A dataset is included with your problem set: problem1data1.txt. The first two columns represent x_1, x_2, and the last column represents the label, $y \in \{-1, +1\}$.

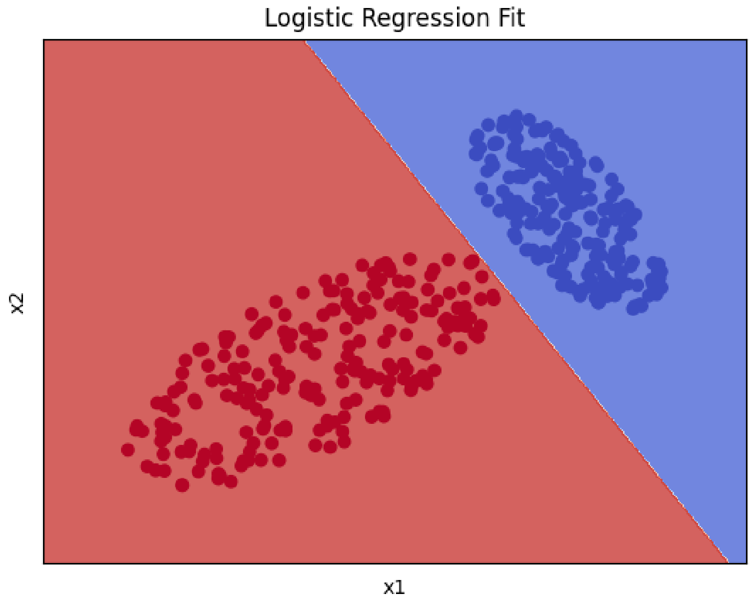
On this dataset, train both a logistic regression model and a ridge regression model to classify the points. (In other words, on each dataset, train one linear classifier using L_{log} as the loss, and another linear classifier using L_{squared} as the loss.) For this problem, you should use the logistic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

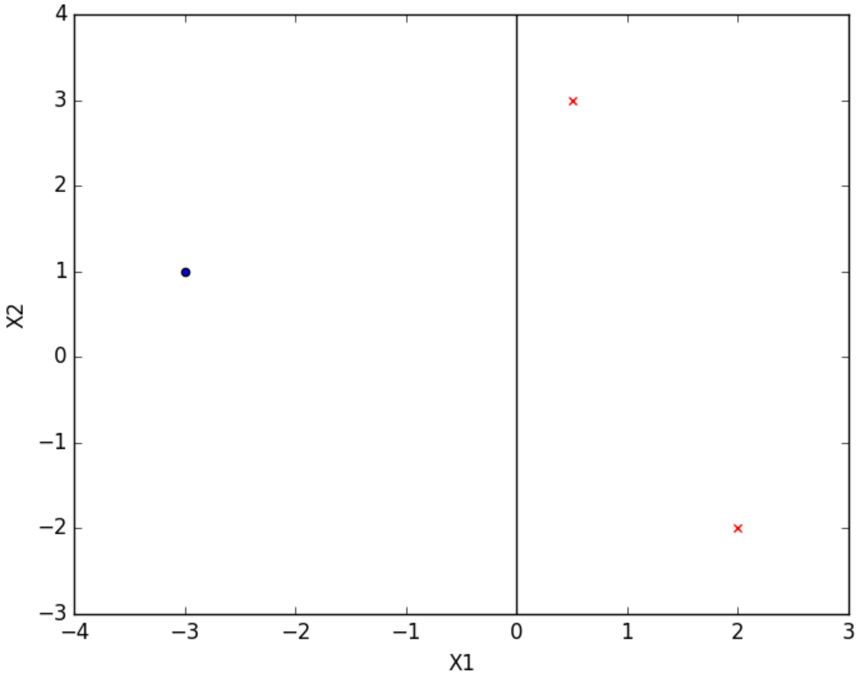




The Ridge Regression Model is positioned closer to the center of the dataset because it is prioritizing attempting to minimize the average squared distance of the dataset to the line and thus misclassifies some points in exchange.

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:

For x where $L_{hinge}(\mathbf{x}) > 0 : \nabla_w L_{hinge}(\mathbf{x}) = -y\mathbf{x}$. For all other x, $\nabla_w L_{hinge}(\mathbf{x}) = \mathbf{0}$

$L_{hinge}(\mathbf{x} = (1, \frac{1}{2}, 3)) = 1 - 1(0 \cdot 1 + 1 \cdot \frac{1}{2} + 0 \cdot 3) = 0.5 > 0 \implies \nabla_w L_{hinge}(\mathbf{x} = (1, \frac{1}{2}, 3)) = (-1, -\frac{1}{2}, -3)$
$L_{hinge}(\mathbf{x} = (1, 2, -2)) = 1 - 1(0 \cdot 1 + 1 \cdot 2 + 0 \cdot -2) = -1 < 0 \implies \nabla_w L_{hinge}(\mathbf{x} = (1, -2, 2)) = \mathbf{0}$
$L_{hinge}(\mathbf{x} = (1, -3, 1)) = 1 + 1(0 \cdot 1 + 1 \cdot -3 + 0 \cdot 1) = -2 < 0 \implies \nabla_w L_{hinge}(\mathbf{x} = (1, -3, 1)) = \mathbf{0}$

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

$\nabla_w L_{Log}(\mathbf{x} = (1, \frac{1}{2}, 3)) = -\frac{1}{e^{\frac{1}{2}}+1} \cdot (1, \frac{1}{2}, 3) \approx (-0.3775, -0.1887, -1.1326)$
$\nabla_w L_{Log}(\mathbf{x} = (1, 2, -2)) = -\frac{1}{e^2+1} \cdot (1, 2, -2) \approx (-0.1192, -0.2384, 0.2384)$
$\nabla_w L_{Log}(\mathbf{x} = (1, -3, 1)) = \frac{1}{e^{-3}+1} \cdot (1, -3, 1) \approx (0.9525, -2.8577, 0.9525)$

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

Solution D: *The gradients from the hinge loss are more abrupt than the log loss. Specifically for the point* $\mathbf{x} = (1, -1/2, 3)$ *we can see that the log loss gradient and hinge loss gradients are in approximately the same direction but the log loss is a smaller step. Both function gradients are* $\mathbf{0}$ *when* $\mathbf{x} = \mathbf{0}$*. The log loss gradient converges to* $\mathbf{0}$ *when* $y\mathbf{x} \to \mathbf{0}$ *or* $e^{y\mathbf{w}^T\mathbf{x}} \to \infty$*. Meanwhile, the hinge loss gradient is zero when* $L_{hinge}(\mathbf{x}) = \mathbf{0}$*. The only way to reduce the training error without changing the training boundary is by either changing your loss function or by choosing another sample of points which are labeled correctly by this boundary.*

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: *Firstly, note that all that hinge loss is all or nothing and thus is a dataset is correctly classified for multiple weights* $\mathbf{w}$ *then it will just pick one at random. From lecture, we showed that the margin of an SVM can be interpreted as* $\frac{b}{\|\mathbf{w}\|}$*, and thus if we try to minimize* $\|\mathbf{w}\|$*, we increase our margin.*

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 in the penalty term cannot reduce the training error because it produces hypotheses which are a subset of the total possible hypotheses that the unrestricted version of the model can produce; therefore, the restricted model can only do as well as the best hypothesis (the one chosen by the unrestricted model to reduce the training 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_1 and ℓ_0 regularization, though they may be more favorable in theory, are much more difficult to compute than the squared magnitude method used by ℓ_2 regularization. Taking the derivative of an absolute value for ℓ_1 can create piecewise or nonsmooth curves and ℓ_0 is even more complex since it just values a binary event of whether the weight is used or not.*

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):

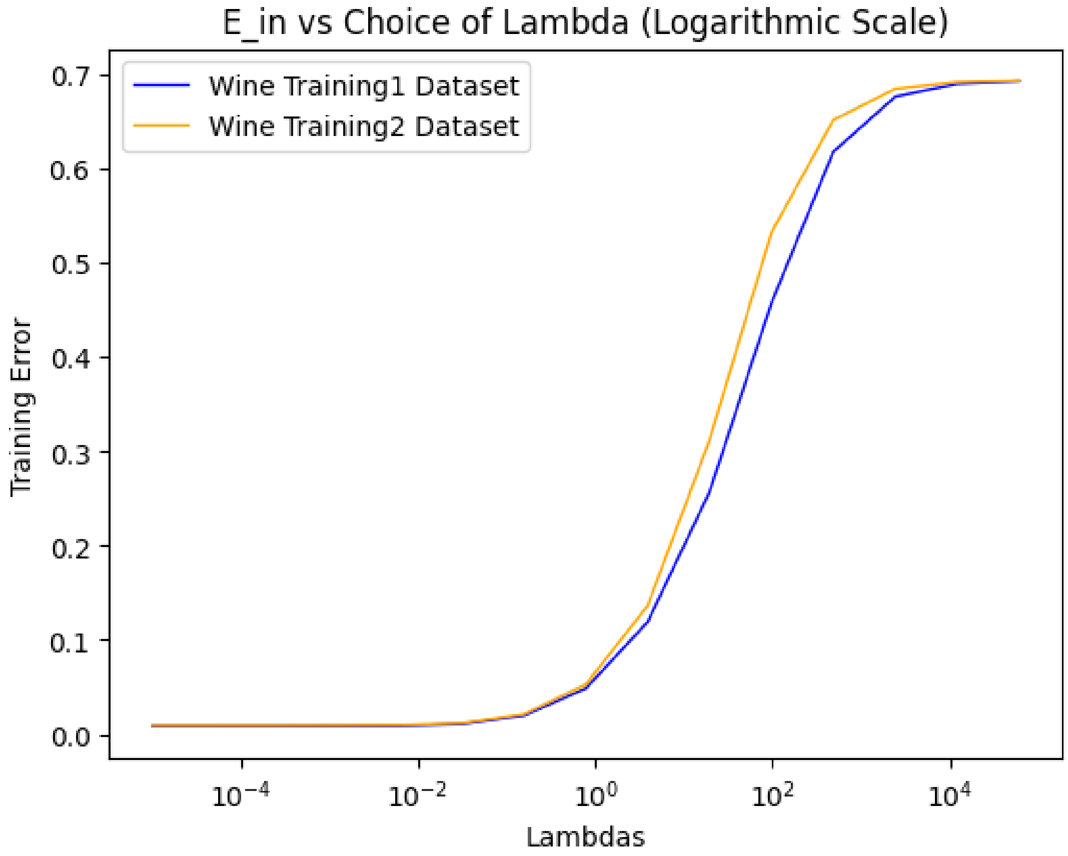
i. Plot the average training error (E_{in}) versus different λs.

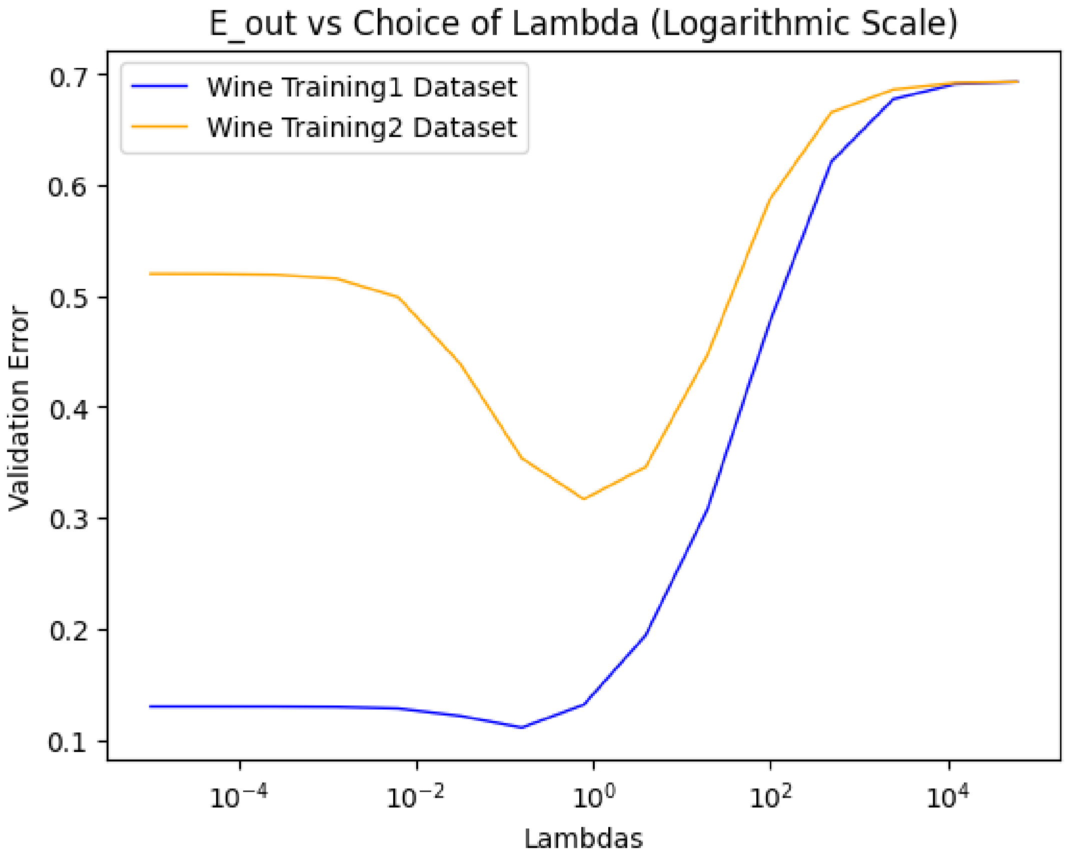
ii. Plot the average test error (E_{out}) versus different λs using wine_validation.txt as the test set.

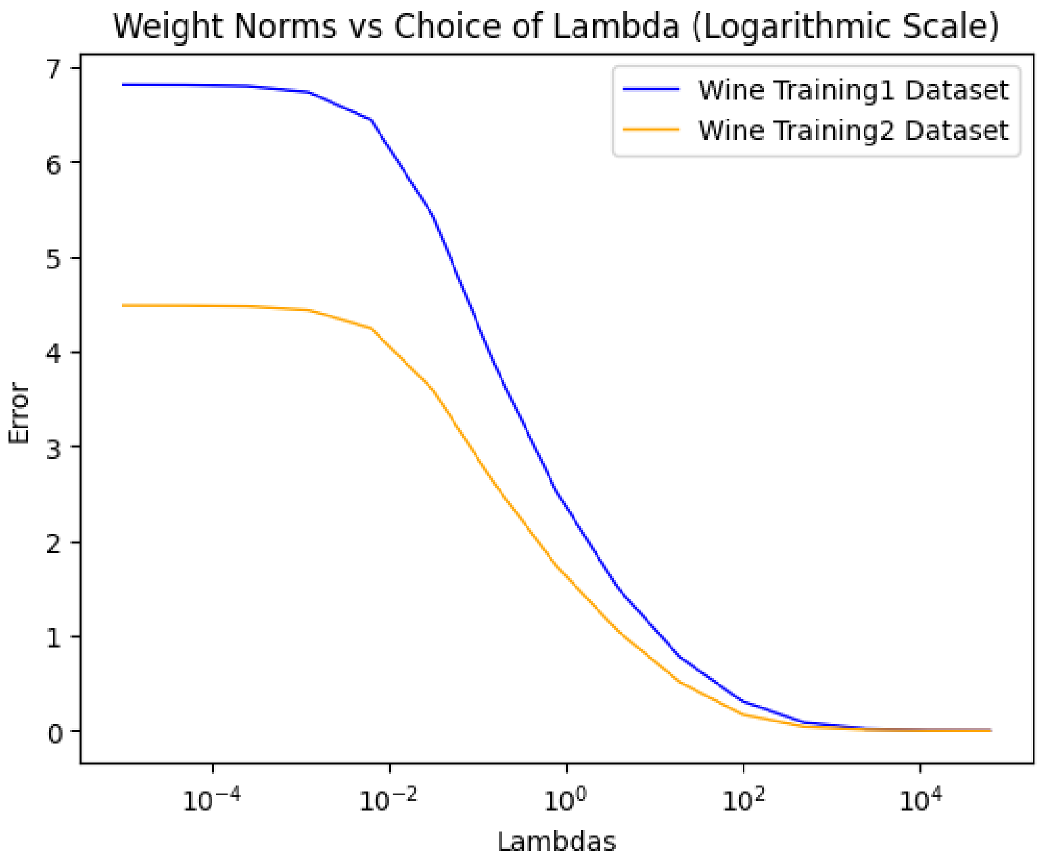
iii. Plot the ℓ_2 norm of $\mathbf{w}$ versus different λs.

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:




E_out vs Choice of Lambda (Logarithmic Scale)
Wine Training1 Dataset
Wine Training2 Dataset
Validation Error
0.7
0.6
0.5
0.4
0.3
0.2
0.1
10^{-4}
10^{-2}
10^{0}
10^{2}
10^{4}
Lambdas



Problem D [4 points]: Given that the data in wine_training2.txt is a subset of the data in wine_training1.txt, compare errors (training and test) resulting from training with wine_training1.txt (100 data points) versus wine_training2.txt (40 data points). Briefly explain the differences.

Solution D: *Both the wine_training2.txt and wine_training1.txt have similar training errors which makes sense because wine_training2.txt is a representative sample of wine_training1.txt. In terms of their out of sample performance, they still both follow similar trends, but the model trained on the wine_training1.txt performs better and more smoothly since it has more data points to influence its changes and better fit the unknown target function.*

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 lambda, there is a training error near zero but there is a relatively high validation error; this is an example of overfitting since the model is not being restricted enough by λ. To contrast this, with a large lambda, there is a high training error, but the validation error is still high; this is an example of underfitting because λ is so high that the model can barely change and does little learning.*

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: *As λ increases, the penalty for large $\mathbf{w}$ grows and causes the model to favor small changes in $\mathbf{w}$ leading to $\mathbf{w}$ approaching $\mathbf{0}$ while the model's ability to learn diminishes.*

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 the λ around $10^0 = 1$ because this λ yields the lowest validation error which is a good approximation for the model's out of sample error, assuming that wine_validation.txt is a representative sample of the target population.*

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.

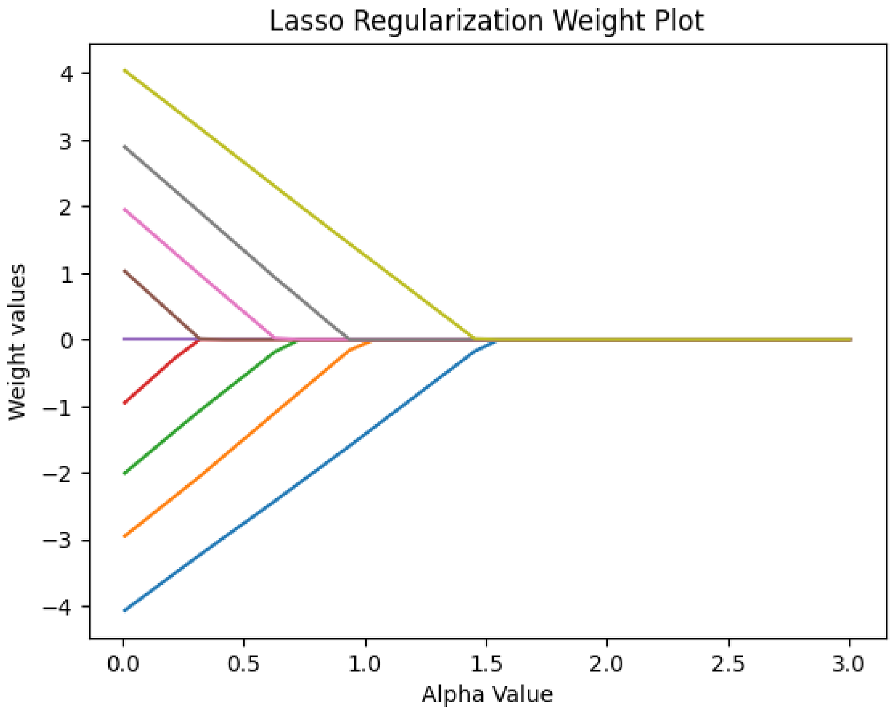
Problem A [11 points]: The tab-delimited file problem3data.txt on the course website contains 1000 9-dimensional datapoints. The first 9 columns contain $x_1, \ldots, x_9$, and the last column contains the target value y.

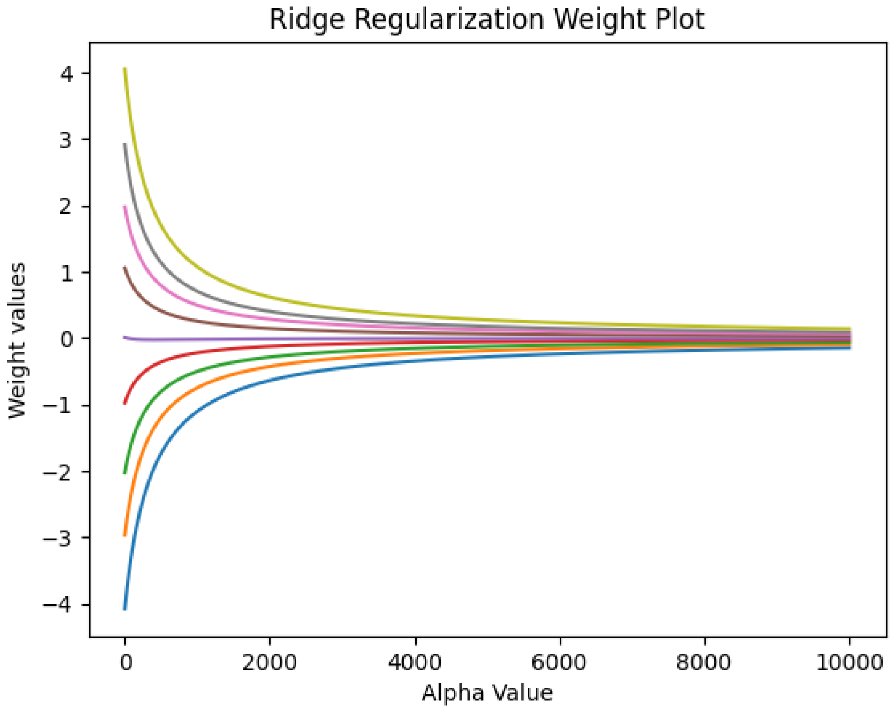
i. Train a linear regression model on the problem3data.txt data with Lasso regularization for regularization strengths α in the vector given by `numpy.linspace(0.01, 3, 30)`. On a single plot, plot each of the model weights $w_1, ..., w_9$ (ignore the bias/intercept) as a function of α.

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:


Lasso Regularization Weight Plot
Weight values
4
3
2
1
0
−1
−2
−3
−4
0.0
0.5
1.0
1.5
2.0
2.5
3.0
Alpha Value


Ridge Regularization Weight Plot
Weight values
4
3
2
1
0
−1
−2
−3
−4
0
2000
4000
6000
8000
10000
Alpha Value

With Lasso Regression, as the regularization parameter increases, the number of weights which are exactly zero increases while with Ridge Regression, the number stays the same but the weights' norms become closer to zero while not being 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:
We will rewrite this as $F(w) \arg\min_{w} \ (\mathbf{y}^T\mathbf{y} - 2w \cdot \mathbf{y}^T\mathbf{x} + w^2\mathbf{x}^T\mathbf{x}) + \lambda|w|$

This is minimized by $0 = \nabla_w F(w) = -2y^T x + 2w \cdot x^T x + \lambda \cdot \partial_w |w|$
$\implies \ w = \frac{2y^T x + \lambda \cdot \partial_w |w|}{2 \cdot x^T x}$

where:

$$\partial_w |w| = \left\{ \begin{array}{ll} -1, & \textit{if } w < 0 \\ 1, & \textit{if } w > 0 \\ \left[-1, 1\right], & \textit{if } w = 0 \end{array} \right\} \tag{1}$$

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

Solution B.ii: *When* $\lambda = \pm\, 2y^T x$*, this implies that* $w = 0$*. The smallest value is the negative value of lambda.*

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:
$E(\mathbf{w}) = \|\mathbf{y} - \mathbf{X}\mathbf{w}\|^2 + \lambda\|\mathbf{w}\|^2$
$0 = \nabla E(\mathbf{w}) = -2\mathbf{X}^T(\mathbf{y} - \mathbf{X}\mathbf{w}) + 2\lambda\mathbf{w}$
$\implies -\mathbf{x}^T\mathbf{y} + \mathbf{X}^T\mathbf{X}\mathbf{w} + 2\lambda\mathbf{w} = 0$
$\implies (\mathbf{X}^T\mathbf{X} + 2\lambda\mathbf{I})^{-1}\mathbf{X}^T\mathbf{y} = \mathbf{w}$

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: *The only way to get $\mathbf{w}$ to be zero, would be to make $(\mathbf{X}^T\mathbf{X} + 2\lambda\mathbf{I})$ be a zero matrix, but this can never be the case because a null matrix is not invertible. Therefore, there is no λ that would result in $\lambda\mathbf{w} = \mathbf{0}$*

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:
We want to show that $\theta_1 x_1 + \cdots + \theta_{k+1} x_{k+1} \in C$ for C, x_i, θ_i as specified in the problem statement. Assume that for some $k \geq 2$, we know that $\theta_1 x_1 + \cdots + \theta_k x_k \in C$. Let this element be called $y_i \in C$. Substituting in, we equivalently want to show that $y_i + \theta_{k+1} x_{k+1} \in C$. Since these are just $2 \leq k$ points in C, we know that this does in fact generate a point in C.

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: