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]

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.

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

Solution : *For classification problems, we only care about classifying the points correctly and don't care about how far the points are from the classification line. If we use squared loss, far away points from the line will be penalized heavily and may cause the classification line to be shifted to misclassify points it otherwise would have correctly classified.*

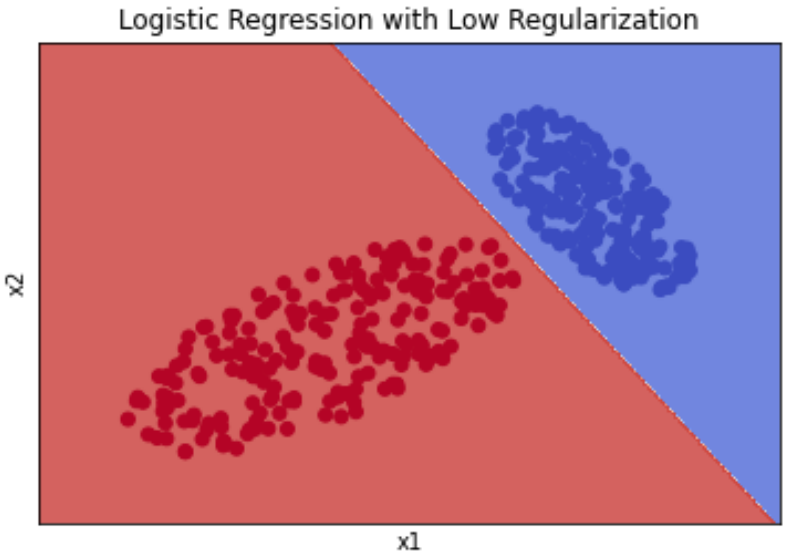
[9] A dataset is included with your problem set: `problem1data1.txt`. The first two columns represent x_1, x_2, and the last column represents the label, $y \in \{-1, +1\}$.

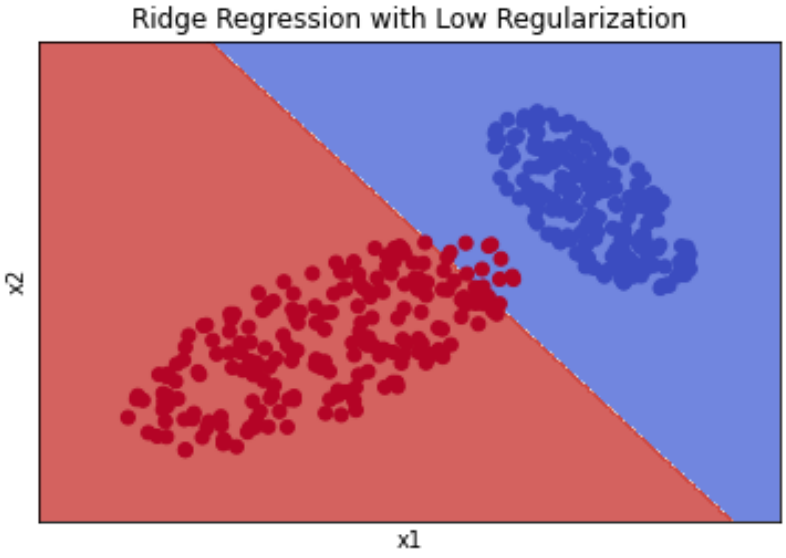
On this dataset, train both a logistic regression model and a ridge regression model to classify the points. (In other words, on each dataset, train one linear classifier using $L_{\log}$ as the loss, and another linear classifier using L_{squared} as the loss.) For this problem, you should use the logistic 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 :
https://colab.research.google.com/drive/1DBaVG1F1CpGF2UCPFlJwiuSAXS9HEyxy?usp=sharing



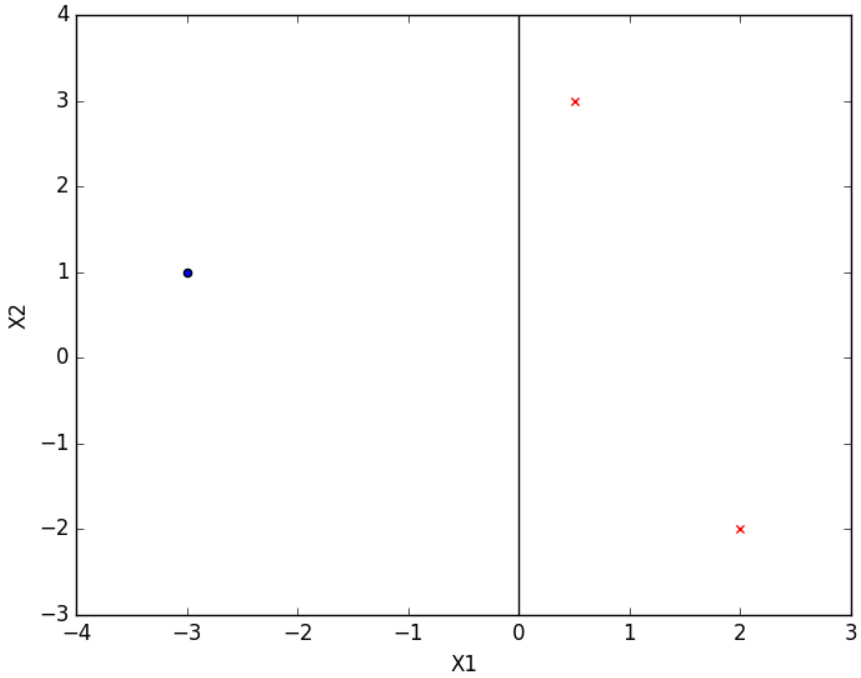


The logistic model has the decision boundary between the two groups, while the ridge model has the decision boundary part way into the red dataset. This is because ridge uses the linear least squares loss function while logistic uses log, so the red points that are far away from the decision boundary have a large effect in the ridge.

[9] 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 :

$$L_{hinge} = \max(0, 1 - y\mathbf{w}^T\mathbf{x})$$

$$\nabla_w L_{reg} = \begin{cases} -y\mathbf{x}, & \textit{if } y\mathbf{w}^T\mathbf{x} < 1 \\ 0, & \textit{otherwise} \end{cases}$$

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

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

$S_1 = (\frac{1}{2}, 3), S_2 = (2, -2), S_3 = (-3, 1), y_1 = 1, y_2 = 1, y_3 = -1, w = [0, 1, 0]$

$y\mathbf{w}^T\mathbf{x} = \frac{1}{2}, 2, 3$ for the three points.
For the three points, we get hinge loss gradients of: $[-1,\ -0.5,\ -3],\ 0,\ 0$
For the three points, we get log loss gradients of: $[-0.37754067,\ -0.18877033,\ -1.13262201]$, $[-0.11920292,\ -0.23840584,\ 0.23840584]$, $[0.04742587,\ -0.14227762,\ 0.04742587]$

[4] 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 : *The gradients that result from log loss are all non-zero, while some of the gradients resulting from hinge loss are zero. The log loss gradients never reach 0 (converge to 0 in the limit as they get infinitely far away), but we can reduce error by increasing the entire weight vector while keeping the decision boundary the same. The hinge loss gradients can be zero when the points are outside the margin, and we can reduce and eliminate error without changing the decision boundary by increasing the entire weight vector so that margin increases without changing the boundary itself.*

[5] 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 : *We need an additional penalty term so that we care about the size of the margin and not just the margin violations. If we just minimize the hinge, we can just increase the weights while keeping the same boundary, in effect reducing the margin, until there are no violations, so we need to introduce a penalty so that we try to maximize 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). [4] 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 : *Adding the penalty term cannot decrease the training error because normal regression will optimally fit the data and the regularization is used to prevent over-fitting. Adding the penalty term can decrease the out-of-sample error if there is over-fitting, but can still increase error if it causes the model to start under-fitting.*

[4] 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 : *The ℓ_0 norm cannot be used directly because it is not continuous.*

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.

[16] 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):

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

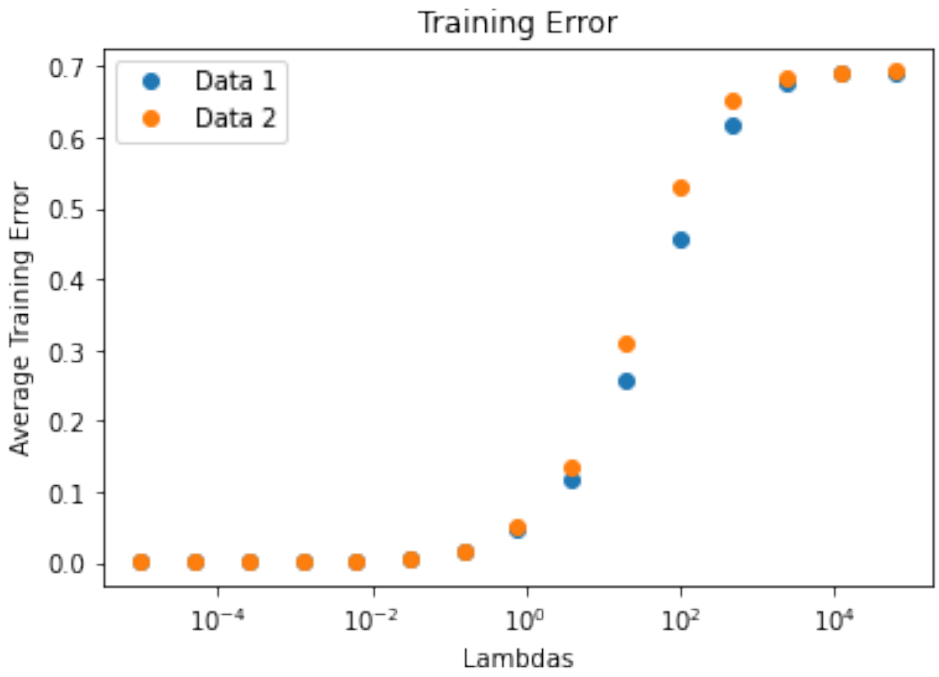
Plot the average test error (E_{out}) versus different λs using wine_validation.txt as the test set.

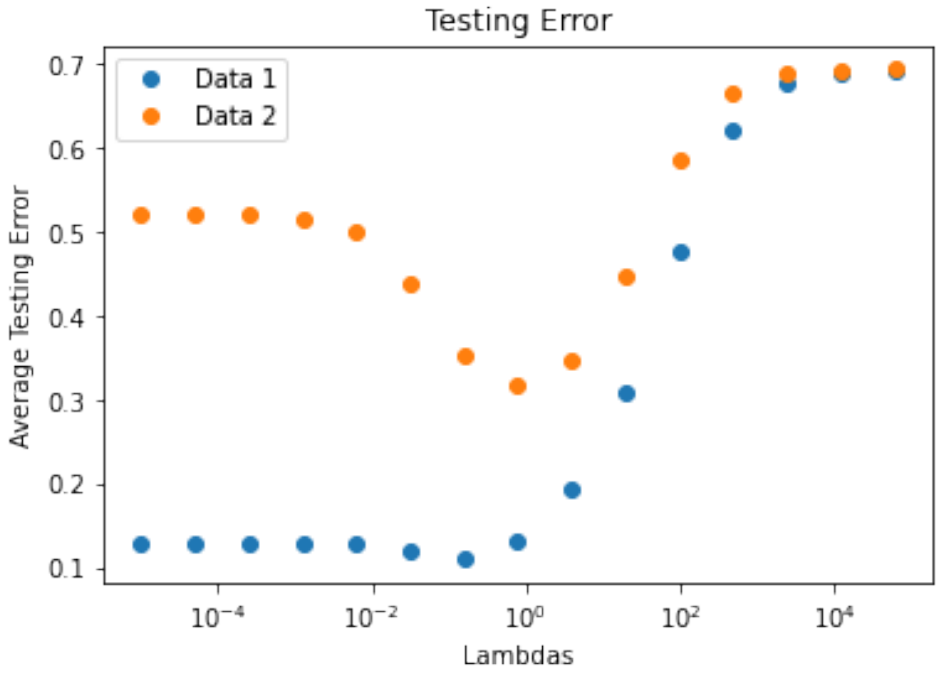
Plot the ℓ_2 norm of $\mathbf{w}$ versus different λs.

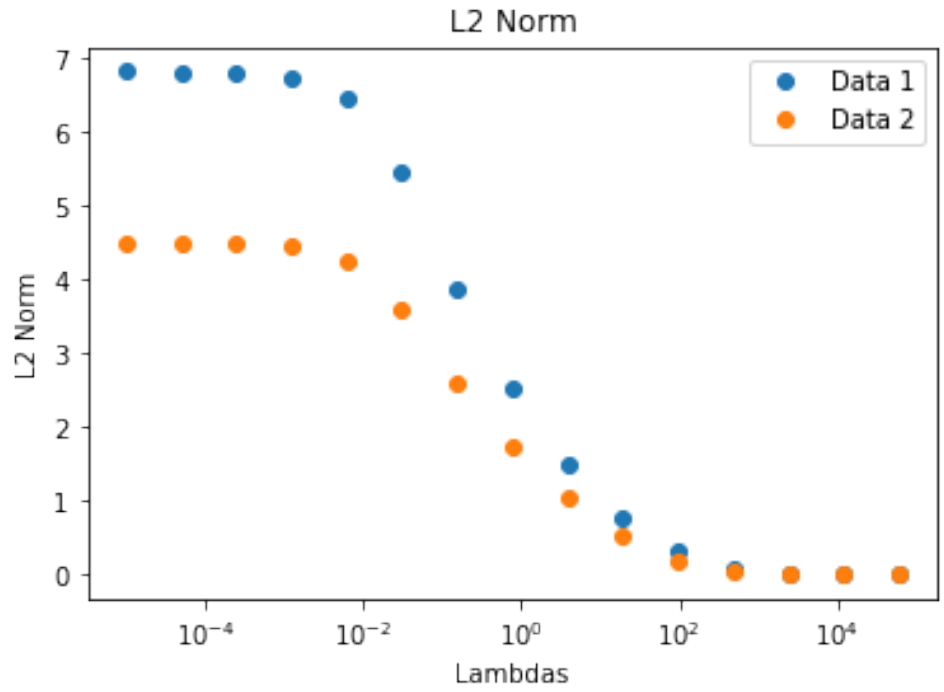
You should end up with three plots, with two series (one for wine_training1.txt and one for wine_training2.txt) on each plot. Note that the E_{in} and E_{out} values you plot should not include the regularization penalty — the penalty is only included when performing gradient descent.

Solution :

https://colab.research.google.com/drive/14Qy_23yp0XvyenOXP9Z46-7EN_XUZsOe?usp=sharing


Training Error
Data 1
Data 2
Average Training Error
0.0
0.1
0.2
0.3
0.4
0.5
0.6
0.7
10^{-4}
10^{-2}
10^{0}
10^{2}
10^{4}
Lambdas


Testing Error
Data 1
Data 2
Average Testing Error
0.1
0.2
0.3
0.4
0.5
0.6
0.7
10^{-4}
10^{-2}
10^{0}
10^{2}
10^{4}
Lambdas



[4] 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 : *Since dataset 2 is a subset of dataset 1 and contains less data, it is more vulnerable to over-fitting. This has little to no affect on the training error and we see nearly identical training error graphs for the two datasets. However, for the testing error, we can see from the testing error graph that the larger dataset is able to perform very well will very low regularization whereas the smaller dataset performs better with some regularization than it does with very low regularization.*

[4] 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 : *For the larger dataset, we can see from both the training error and testing error that large λs cause the model to under-fit the data. From the testing error graph, we can see that the model for this dataset doesn't over-fit the data very much as very low λs perform just barely worse than with some regularization (very small dip).*

[4] 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 : *From the norm graph, we can see that larger the value of λ gets, the smaller the ℓ_2 norm gets. We expect since this is the purpose of adding this regularization term.*

[4] If the model were trained with wine_training2.txt, which λ would you choose to train your final model? Why?

Solution : *If the model were trained with wine_training2.txt, I would choose λ of 1 to train your final model, because this seems to limit over-fitting and under-fitting. It has the smallest testing error, so would likely do the best on unseen 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.

[12] 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.

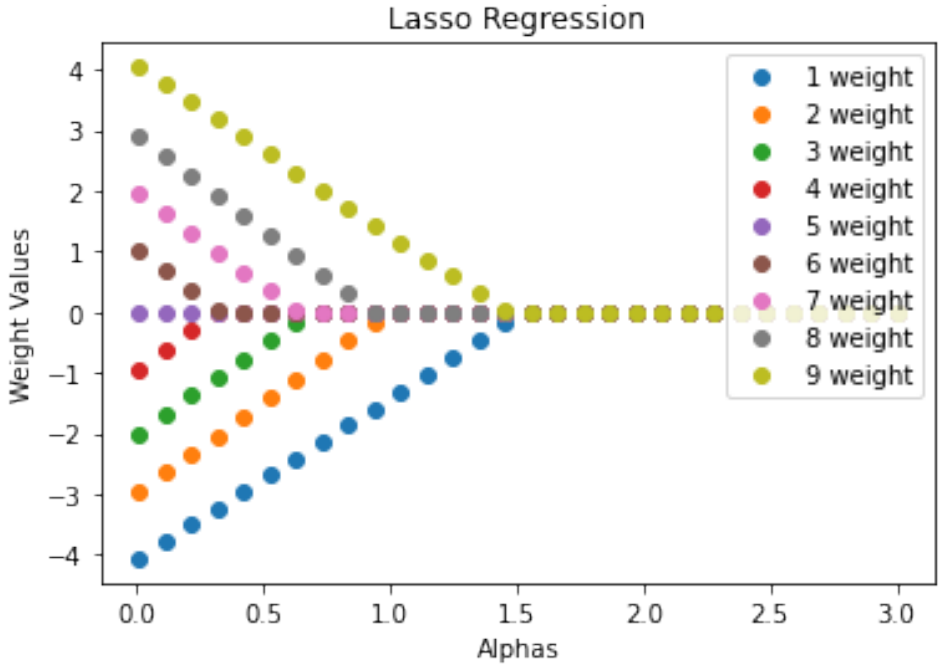
Train a linear regression model on the problem3data.txt data with Lasso regularization for regularization strengths α in the vector given by `numpy.linspace(0.01, 3, 30)`. On a single plot, plot each of the model weights $w_1, ..., w_9$ (ignore the bias/intercept) as a function of α.

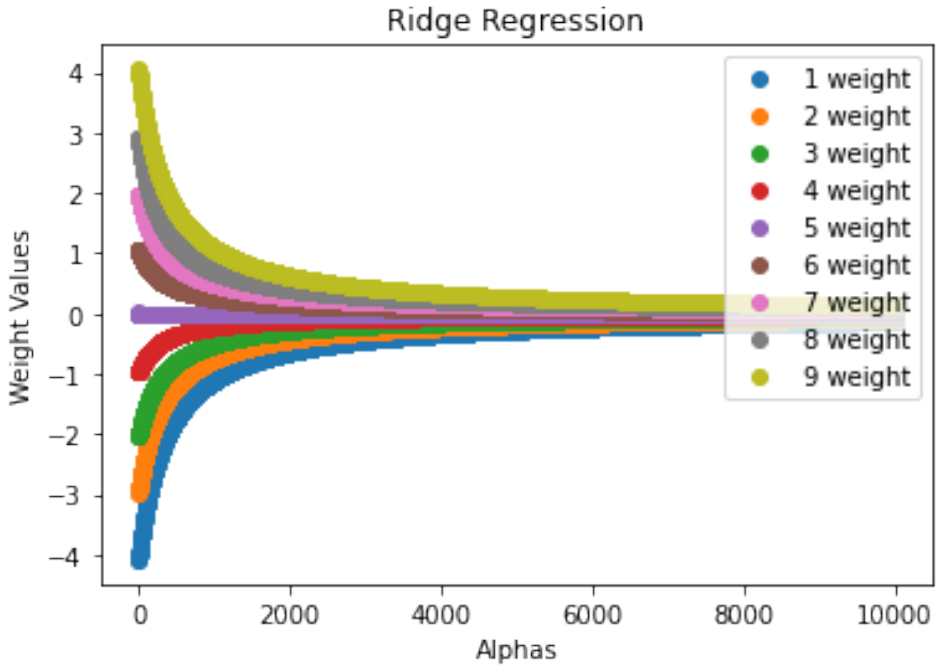
Repeat **i.** with Ridge regression, and this time using regularization strengths $\alpha \in \{1, 2, 3, \ldots, 1e4\}$.

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 : *As the regularization parameter increases, the number of model weights that are exactly zero in Lasso regression increases until all the weights are zero. For Ridge regression, there are no model weights that are exactly zero for any alpha, but as alpha increases all the weights approach zero.*

https://colab.research.google.com/drive/1n-dk1RyyVDl9V693P14yh5RuZA1WJm5Z?usp=sharing


Lasso Regression
Weight Values
Alphas
1 weight
2 weight
3 weight
4 weight
5 weight
6 weight
7 weight
8 weight
9 weight


Ridge Regression
Weight Values
Alphas
1 weight
2 weight
3 weight
4 weight
5 weight
6 weight
7 weight
8 weight
9 weight

[9]

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 .:

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

Since $\lambda \geq 0$, we can see that when $\lambda < 2\mathbf{x}^T\mathbf{y}$, we get $w = \frac{2\mathbf{x}^T\mathbf{y} - \lambda}{2\mathbf{x}^T\mathbf{x}}$ and we get $w = 0$ otherwise.

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 .: *We know when $\lambda < 2\mathbf{x}^T\mathbf{y}$, we get $w = \frac{2\mathbf{x}^T\mathbf{y} - \lambda}{2\mathbf{x}^T\mathbf{x}}$ and we get $w = 0$ otherwise, so $w = 0$ when $\lambda \geq 2\mathbf{x}^T\mathbf{y}$ and the smallest values of λ is $\lambda = 2\mathbf{x}^T\mathbf{y}$.*

[9] 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 .:

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

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 .: *Since $w \neq 0$ when $\lambda = 0$, we can see that there does not exist a value for $\lambda = 0$ such that $w = 0$. $\mathbf{X}^T\mathbf{y}$ is non-zero so therefore we can never get $w = 0$.*