Machine Learning & Data Mining
Set 2

Caltech CS/CNS/EE 155
January 19th, 2022

Dallas Taylor

Google Colab Links

- https://colab.research.google.com/drive/1ingLVg77tC6pZJcC-K52BInEOQhW83zX?usp=sharing
- https://colab.research.google.com/drive/17k3JMwgOhzwmKJNEUFD5zwXegaLugnP?usp=sharing
- https://colab.research.google.com/drive/13QGimAjZVdSjmf9zPKNFRnZcljuKy3Dj?usp=sharing

1 Comparing Different Loss Functions [30 Points]

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.

Question 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 often a poor choice due to the fact that it is not very accurate, and can often fail at classifying data (even if it is linearly separable). This is due to the fact that data points that are classified correctly but are far from the boundary decision can cause undesired effects in weight updating.*

Question 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.

Dallas Taylor

Solution B:

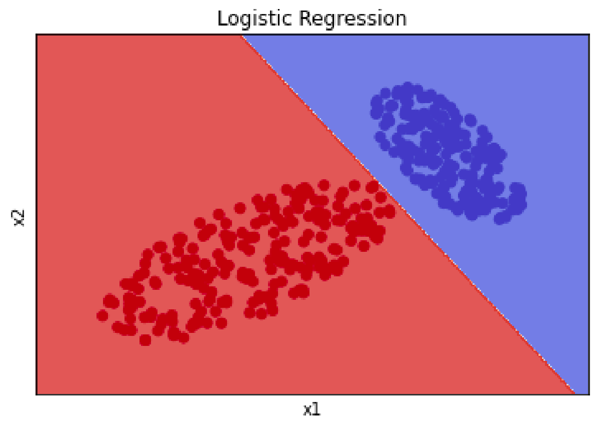


Figure 1: Logistic Regression

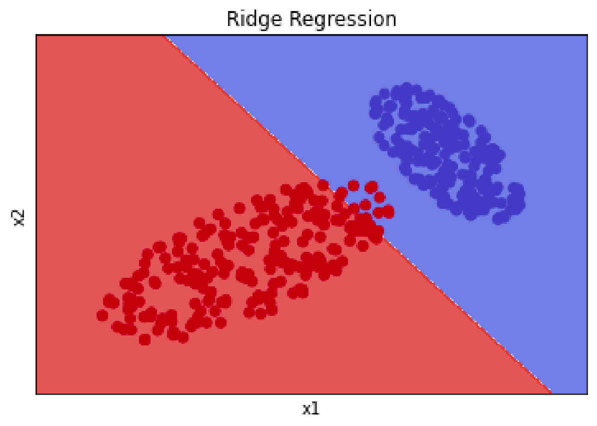


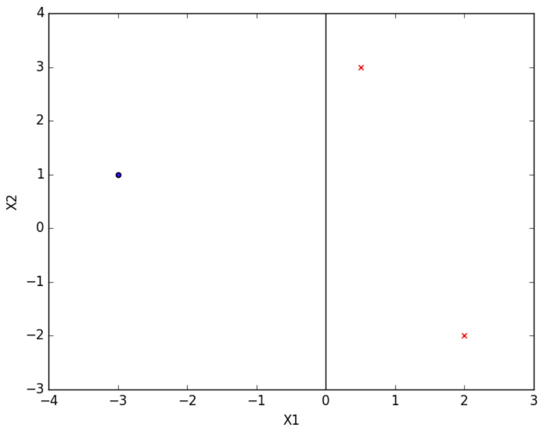
Figure 2: Ridge Regression

There is a clear offset between the two decision boundaries from these two different loss behaviors. The ridge regression final decision boundary is shifted left from the logistic regression's, resulting in some misclassified points, as predicted in our answer to A.

Colab Link : `https://colab.research.google.com/drive/1ingLVg77tC6pZJcC-K52BInEOQhW83zX?usp=sharing`

Question 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:

$w = 0, 1, 0.$

$$\begin{aligned}
L_{log} &= \ln(1 + e^{-y\mathbf{w}^T\mathbf{x}}) \\
\therefore \nabla_w L_{log} &= \frac{-y_i x_i e^{-y_i \mathbf{w}^T x_i}}{1 + e^{-y_i \mathbf{w}^T x_i}} \\
&= \frac{-y_i x_i e^{-y_i \mathbf{w}^T x_i}}{1 + e^{-y_i \mathbf{w}^T x_i}} \cdot \frac{e^{y_i \mathbf{w}^T x_i}}{e^{y \mathbf{w}^T x_i}} \\
&= \frac{-y_i x_i}{e^{y_i \mathbf{w}^T x_i} + 1} \\
----& \quad ----
\end{aligned}$$

$$((x_{1_1}, x_{1_2}), y_1) = ((1, \frac{1}{2}, 3), +1) \quad : \quad \nabla_w L_{log} = \frac{-1(1, \frac{1}{2}, 3)}{e^{(0,1,0)\times(1,\frac{1}{2},3)} + 1} = (-0.378, -0.189, -1.133)$$

$$((x_{2_1}, x_{2_2}), y_2) = ((1, 2, -2), +1) \quad : \quad \nabla_w L_{log} = \frac{-1(1, 2, -2)}{e^{(0,1,0)\times(1,2,-2)} + 1} = (-0.119, -0.238, 0.238)$$

$$((x_{3_1}, x_{3_2}), y_3) = ((1, -3, 1), -1) \quad : \quad \nabla_w L_{log} = \frac{1(1, -3, 1)}{e^{-1\cdot(0,1,0)\times(1,-3,1)} + 1} = (0.047, -0.142, 0.047)$$

$$\begin{aligned}
L_{hinge} &= \max(0, 1 - y\mathbf{w}^T\mathbf{x}) \\
\therefore \nabla_w L_{hinge} &= \begin{cases} 0 & y\mathbf{w}^T\mathbf{x} > 1 \\ -yx & 1 \geq y\mathbf{w}^T\mathbf{x} \end{cases} \\
----\quad &\quad ---- \\
((x_{1_1}, x_{1_2}), y_1) = ((1, \frac{1}{2}, 3), +1) \quad &: \quad \nabla_w L_{hinge} = (-1, -\frac{1}{2}, -3) \;\; (0,1,0) \times (1, \frac{1}{2}, 3) < 1 \\
((x_{2_1}, x_{2_2}), y_2) = ((1, 2, -2), +1) \quad &: \quad \nabla_w L_{hinge} = (0,0,0) \;\; (0,1,0) \times (1, 2, -2) > 1 \\
((x_{3_1}, x_{3_2}), y_3) = ((1, -3, 1), -1) \quad &: \quad \nabla_w L_{hinge} = (0,0,0) \;\; -1 \cdot (0,1,0) \times (1, -3, 1) > 1
\end{aligned}$$

Question 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: *For log loss, the gradient will converge to 0 when $e^{y_i \mathbf{w}^T x_i}$ converges to infinity, and thus whenever $y_i = +1$ and $x_i \to \infty$ or whenever $y_i = -1$ and $x_i \to -\infty$. For hinge loss, the gradient converges to 0 whenever $y\mathbf{w}^T\mathbf{x} > 1$, which is equivalent for our given w when $y_i = +1$ and $x_i > 1$ or when $y_i = -1$ and $x_i < 1$.*

For both cases, we can reduce training error by scaling up our weight vector w. For log loss, we are unable to fully eliminate all training error, however we are able to fully eliminate all training error for hinge loss if we have that all $y_i \mathbf{w}^T x_i \leq 1$

Question 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: *It is important to include this penalty term as it keeps our gradients from converging by simply scaling our weight vector. The additional penalty term counteracts this activity, allowing for an actual "maximum margin" classifier.*

Dallas Taylor

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).

Question 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 does not decrease the training error since both sets of training data (regularized and unregularized) are specifically trained so as to reduce training error. Adding a penalty term also does not always decrease out-of-sample errors. When there is over-fitting, adding the regularization term helps decrease model complexity and thus decrease out-of-sample error. However, when there is under-fitting, the regularization term does not always decrease out-of-sample error.*

Question 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 *regularization is rarely used because it is not continuous, and thus very challenging to optimize.*

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.

Question 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):

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 C:

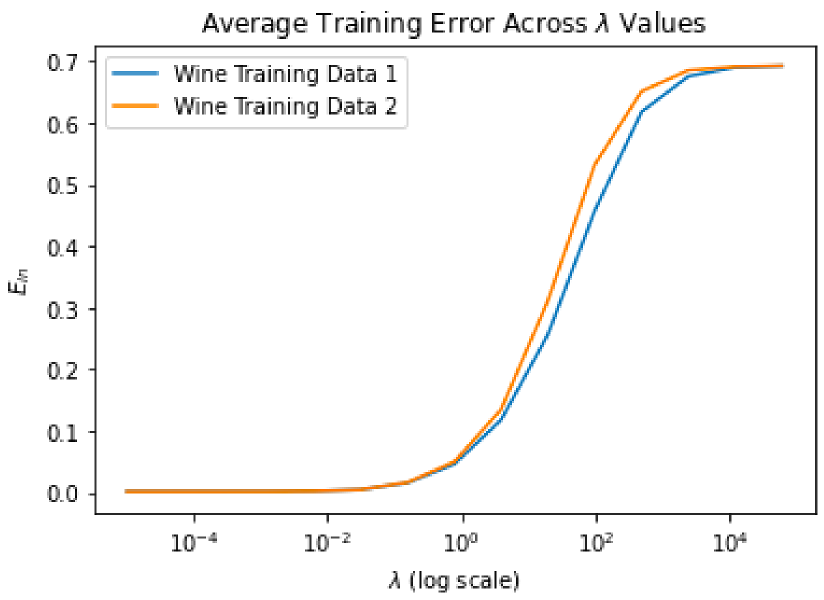


Figure 3: Evolution of Training Error as λ increases.

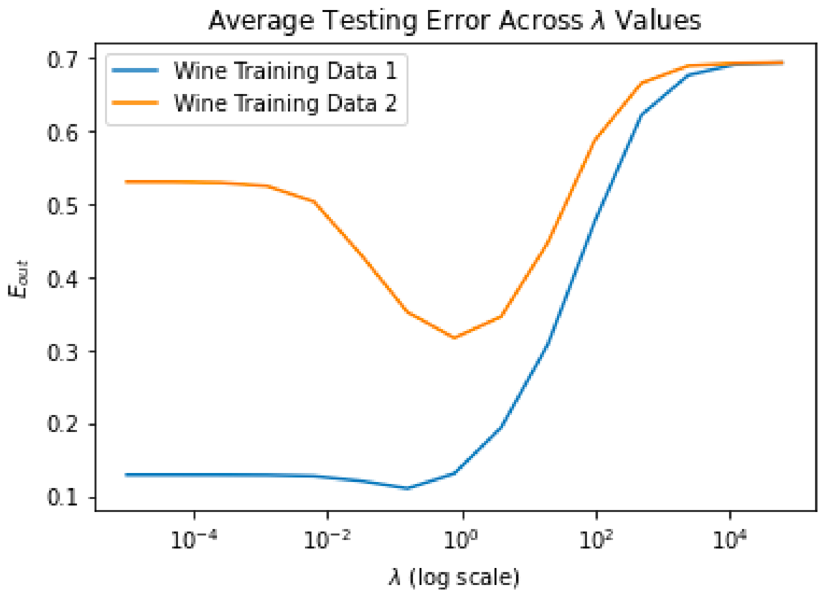


Figure 4: Evolution of Testing Error as λ increases.

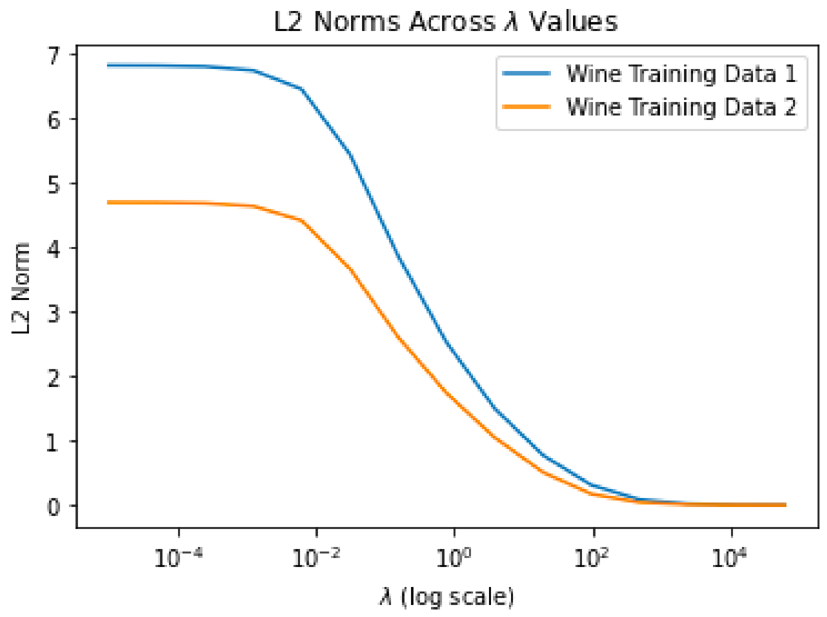


Figure 5: Evolution of ℓ_2 norm of $\mathbf{w}$ as λ increases.

Colab Link : `https://colab.research.google.com/drive/17k3JMwgOhzwmKJNEUFD5zwXegaLugnP?usp=sharing`

Question 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 for both data sets follow the same trajectories and are extremely similar. However, the testing error is much higher for Wine Training 2, indicative of overfitting the data, until the errors become similar for high values of λ, allowing for further generalization of our model.*

Question 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: *Both the training and testing error for both Wine Training 1 are minimal and desirable when $\lambda < 10^0 = 1$. The testing error is slightly higher than the training error for these values of λ, indicating slight over-fitting. Once $\lambda > 1$, we begin to see much higher values of testing and training error, indicating under-fitting.*

Question 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 $\mathbf{w}$ decreases as λ increases, due to the natural minimization of our weight.*

Question 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 $\lambda = 10^0 = 1$ when using Wine Training 2 because it provides a low training error and the lowest testing error across all λ values used. This λ value thus results in less overfitting than lower values and less underfitting than higher values, as desired.*

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.

Question 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:

Dallas Taylor

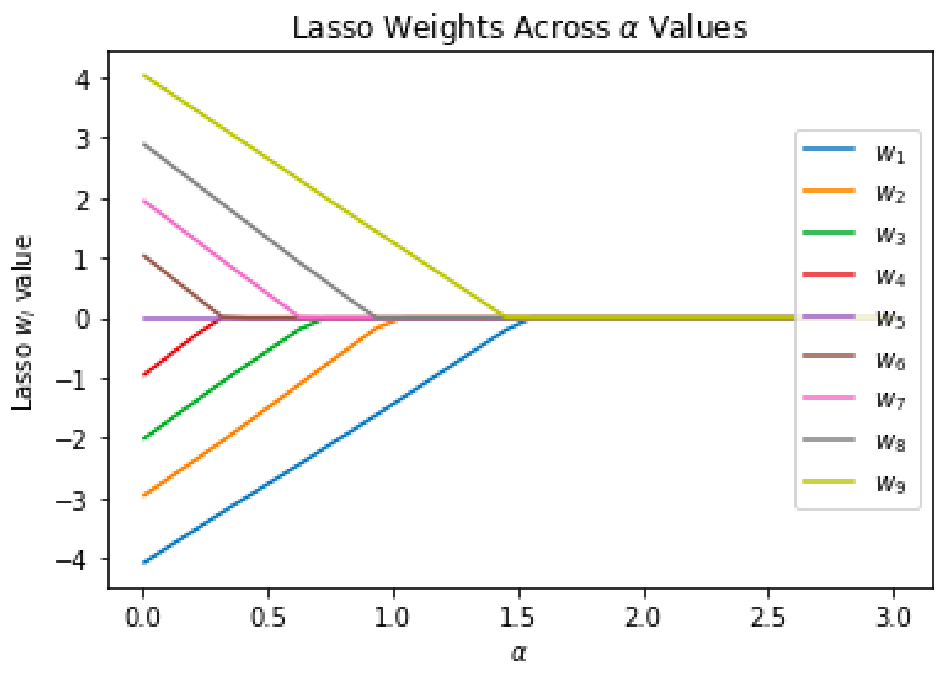


Figure 6: Lasso weights across regularization strengths α.

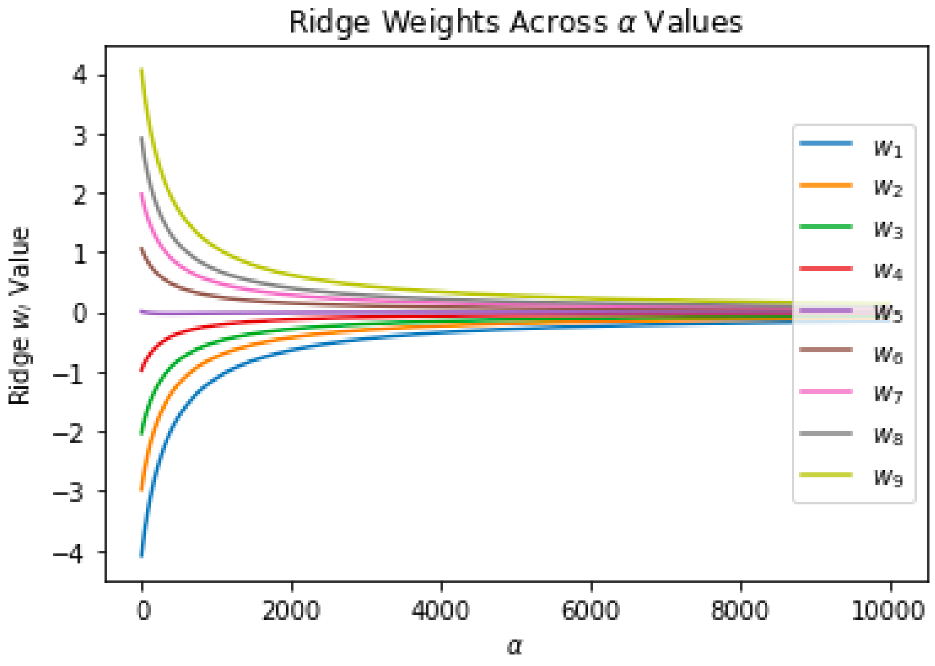


Figure 7: Ridge weights across regularization strengths α.

As α increases, the number of model weights from Lasso regression linearly decrease their magnitude from 0, until eventually, at $\alpha = 1.5$, we have all model weights being 0.

Colab Link : https://colab.research.google.com/drive/13QGimAjZVdSjmf9zPKNFRnZcljuKy3Dj?usp=sharing

Question 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.

Dallas Taylor

Solution B.i:

$$\begin{aligned}
\frac{d}{dw} &= \frac{d}{dw}\left(\|\mathbf{y}-\mathbf{x}w\|^2+\lambda\|w\|_1\right)\\
&= 2\cdot(-\mathbf{x}^T)\cdot(\mathbf{y}-\mathbf{x}w)+\lambda\\
&= -2\mathbf{x}^T\mathbf{y}+2\mathbf{x}w+\lambda\\
\therefore \mathit{min}\ :\ 0 &= -2\mathbf{x}^T\mathbf{y}+2\mathbf{x}w+\lambda\\
\therefore w &= \frac{2\mathbf{x}^T\mathbf{y}-\lambda}{2\mathbf{x}}
\end{aligned}$$

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:

$$\begin{aligned}
w &= \frac{2\mathbf{x}^T\mathbf{y}-\lambda}{2\mathbf{x}}\\
w &= 0\\
\therefore 0 &= \frac{2\mathbf{x}^T\mathbf{y}-\lambda}{2\mathbf{x}}\\
\therefore \lambda &= \|2\mathbf{x}^T\mathbf{y}\|
\end{aligned}$$

Thus, we can see that there exists a value for λ such that $w = 0$. Further, we can see that the smallest such value is $\|2\mathbf{x}^T\mathbf{y}\|$.

Question 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:

$$\begin{aligned}
\frac{d}{d\mathbf{w}} &= \frac{d}{d\mathbf{w}}\left(\|\mathbf{y}-\mathbf{X}\mathbf{w}\|^2+\lambda\|\mathbf{w}\|_2^2\right)\\
&= 2\cdot(-\mathbf{X}^T)\cdot(\mathbf{y}-\mathbf{X}\mathbf{w})+2\cdot\lambda\mathbf{w}\\
&= -2\mathbf{X}^T\mathbf{y}+2\mathbf{X}^T\mathbf{X}\mathbf{w}+2\lambda\mathbf{w}\\
\therefore \mathit{min}\ :\ 0 &= -2\mathbf{X}^T\mathbf{y}+2\mathbf{X}^T\mathbf{X}\mathbf{w}+2\lambda\mathbf{w}\\
\therefore 2\mathbf{X}^T\mathbf{y} &= \mathbf{w}(2\mathbf{X}^T\mathbf{X}+2\lambda\mathbf{I})\\
\therefore \mathbf{w} &= \frac{\mathbf{X}^T\mathbf{y}}{\mathbf{X}^T\mathbf{X}+\lambda\mathbf{I}}
\end{aligned}$$

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:

$$\begin{aligned}
\mathbf{w} &= \frac{\mathbf{X}^T\mathbf{y}}{\mathbf{X}^T\mathbf{X}+\lambda\mathbf{I}}\\
\mathbf{w} &= 0\\
\therefore 0 &= \frac{\mathbf{X}^T\mathbf{y}}{\mathbf{X}^T\mathbf{X}+\lambda\mathbf{I}}
\end{aligned}$$

Since we have our λ in the denominator, there does not exist a value for $\lambda > 0$ such that $w = 0$ (when $\mathbf{X}, \mathbf{y} \neq 0$).