Machine Learning & Data Mining
Set 2

Caltech CS/CNS/EE 155
January 12th, 2018

Policies

- Due 9 PM, January 19th, via Moodle.
- You are free to collaborate on all of the problems, subject to the collaboration policy stated in the syllabus.
- You should submit all code used in the homework. We ask that you use Python 3 and sklearn version 0.19 for your code, and that you comment your code such that the TAs can follow along and run it without any issues.

Submission Instructions

Please submit your assignment as a .zip archive with filename `LastnameFirstname.zip` (replacing `Lastname` with your last name and `Firstname` with your first name), containing a PDF of your assignment writeup **in the main directory** with filename `LastnameFirstname_Set2.pdf` and your code files **in a directory named `LastnameFirstname`**. Failure to do so will result in a **2 point deduction**. Submit your code as Jupyter notebook .ipynb files or .py files, and **include any images generated by your code along with your answers in the solution .pdf file.**

1 Comparing Different Loss Functions [30 Points]

Relevant materials: lecture 3

We've discussed three loss functions for linear 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: *Points far away from the decision boundary of a linear model incur a very high penalty if squared loss is used, even if they're classified correctly (i.e. they're on the correct side of the decision boundary).*

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, $y \in \{-1, +1\}$.

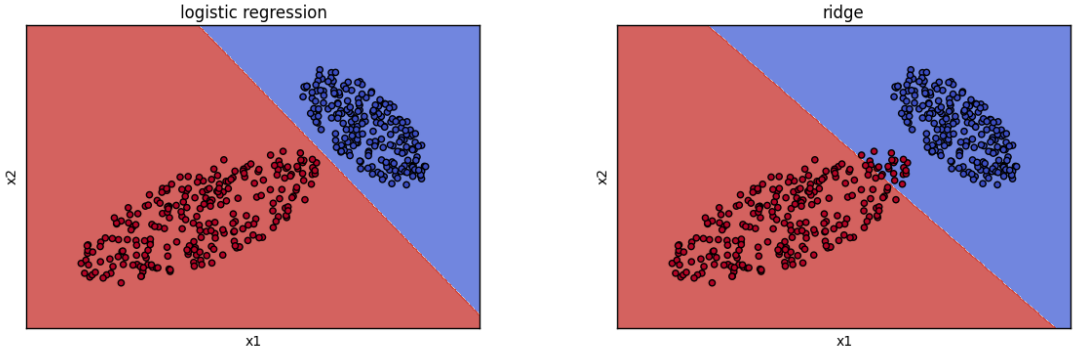
On this dataset, train both a logistic regression model and a ridge regression model to classify the points. (In other words, on each dataset, train one linear classifier using L_{log} as the loss, and another linear classifier using L_{squared} as the loss.) For this problem, you should use the logistic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

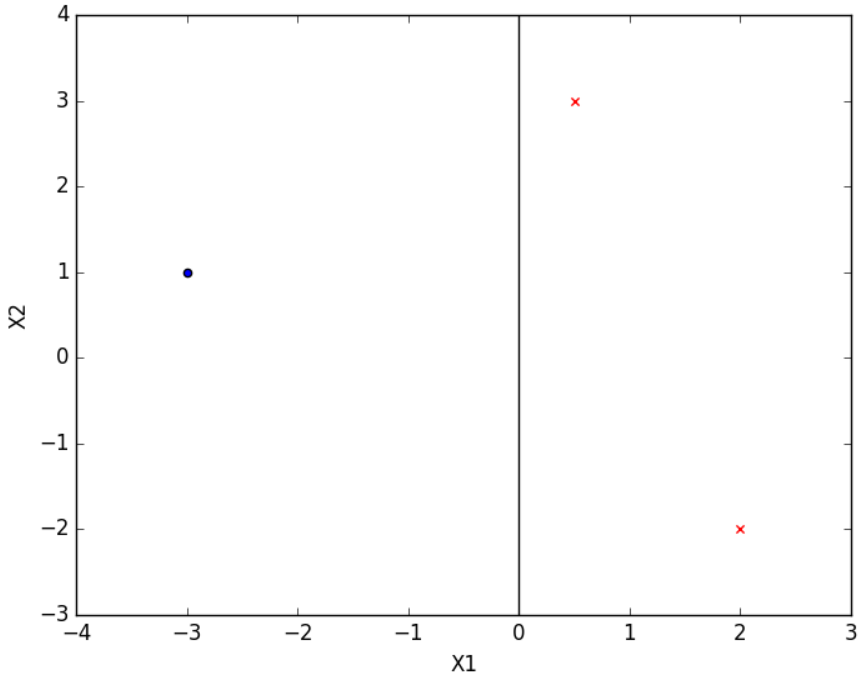
See attached Python code.



We see that logistic regression correctly classifies all points, whereas ridge regression does not. This is because squared loss is highly sensitive to outliers; in general, this illustrates that regression objectives can perform poorly for classification problems.

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 $\nabla_w L_{\log}$ 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: *The gradient for log loss on a single training point is*

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

and for hinge loss is

$$\nabla_w L_{hinge}(w, x, y) = \left\{ \begin{array}{ll} 0 & \textit{for } y(w^T x) < 1 \\ -yx & \textit{otherwise} \end{array} \right\}$$

giving:

```
Point: (0.5, 3)
Hinge loss gradient: (-1.0, -0.5, -3.0)
Log loss gradient: (-0.37754067 -0.18877033 -1.13262201)

Point: (2, -2)
Hinge loss gradient: 0
```

```
Log loss gradient: (-0.11920292 -0.23840584  0.23840584)

Point: (-3, 1)
Hinge loss gradient: 0
Log loss gradient: ( 0.04742587 -0.14227762 0.04742587)
```

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:

We can see that the gradient of hinge loss is 0 for some correctly-classified points (if $yw^Tx > 1$), while the gradient of log-loss is nonzero for such points. In fact, the gradient for log-loss is always nonzero for nonzero x. Thus, if we fit a linear model by minimizing log-loss via gradient descent, w will continue to shift in the direction of correctly-classified points, whereas when using hinge loss, w converges once all training points are correctly-classified with $yw^Tx > 1$.

For a linearly separable dataset, consider any hyperplane w that correctly classifies every point. This implies $y_i w^T x_i = \epsilon_i > 0$ for each training point x_i with label y_i. Let $\epsilon = \min \epsilon_i$. We can achieve a hinge loss of 0 simply by scaling w, e.g setting $w \leftarrow \frac{w}{\epsilon}$, so that $y_i w^T x_i \geq 1$ for all training points.

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:

To see why SVM's loss function must include a penalty term $\lambda\|w\|^2$, note that scaling a separating hyperplane w does not geometrically alter the hyperplane; the set of x for which $w^Tx = 0$ is unchanged. Thus, any hyperplane that separates the training set can be scaled to achieve 0 hinge loss. This is where the notion of margin maximization comes in - for a linearly separable dataset, the geometric margin (distance from our hyperplane to the closest training point) is given by $\frac{\epsilon}{|w|}$. This margin is maximized by minimizing w, which is why the penalty term $\lambda\|w\|^2$ is included when training an SVM.

2 Effects of Regularization

Relevant materials: Lecture 4

For this problem, you are required to implement everything yourself and submit code.

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: *No, it will not decrease the training error; adding a regularization term will only increase the training error since the training error has already been minimized by the unregularized loss function. No, regularization will not always decrease the validation error. It depends on whether there is over-fitting while training the model. If there is over-fitting, adding regularization will decrease the training error for small λ, while for large λ, the model can easily go from over-fitting to under-fitting, which might increase the validation 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 regularization is not used since it is not a continuous function, thus numerical methods of finding the minimum that require a gradient (such as gradient descent) cannot be used.*

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 testing. 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_testing.txt (30 data points). You will use the wine_testing.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 $5e-4$, 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):

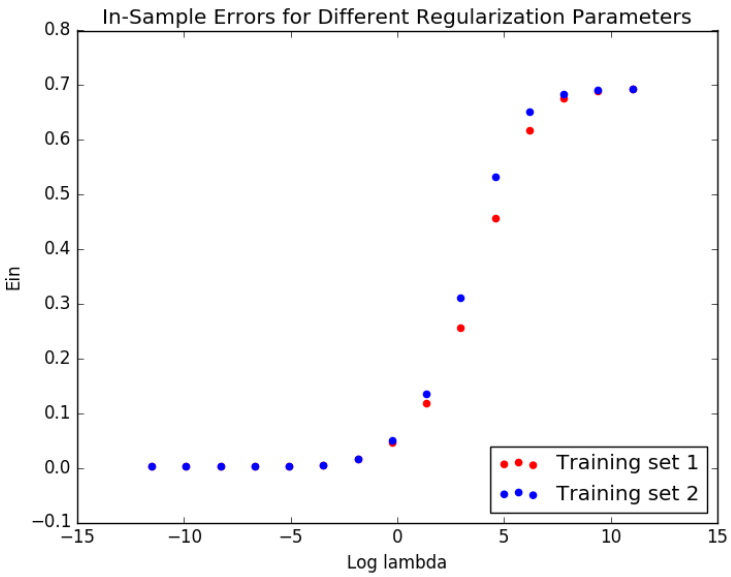
i. Plot the average training error (E_{in}) versus different λs.

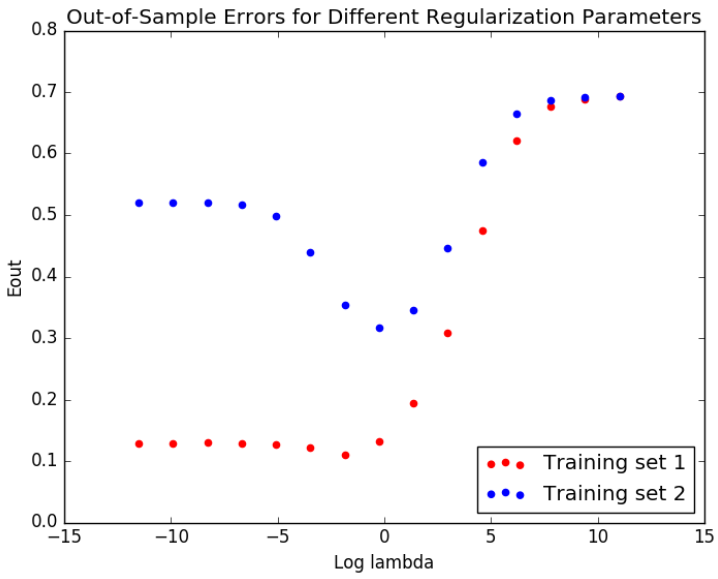
ii. Plot the average test error (E_{out}) versus different λs using wine_testing.txt as the test set.

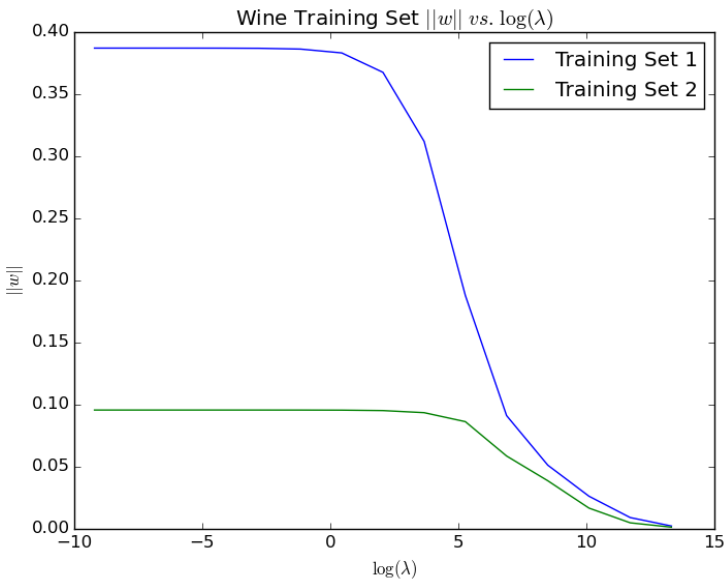
iii. Plot the ℓ_2 norm of $\mathbf{w}$ versus different λs.

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: *Using python, we obtain the following plots (please see the solution code):*







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: *For small values of λ, around 0.001 or lower, the training error in training set 1 is higher than for training set 2. This is because with fewer data points, it is easier to "fit" all the data points perfectly. As λ increases, training error for both training set 1 and training set 2 increases. The validation error in both training sets experience a small "dip" as we increase λ, before increasing rapidly for large λ. This could be because over-fitting exists for both training sets at low λ. But we can see that the validation error for training set 2 is much higher than the validation error for training set 1 with small λ. This again implies that in the smaller training set, over-fitting is more severe.*

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: *As we have answered in question A, training error will always increase as we add regularization term. So there is no surprise here. As far as the test error, it first decreases implying that the regularization is "fighting" against the over-fitting. But after $\lambda = 0.15625$ ($\log(\lambda) \approx -1.86$) for training set 1, and $\lambda = 0.78125$ ($\log(\lambda) \approx -0.25$) for training set 2, the validation error starts to increase, implying under-fitting.*

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 norm of w decreases. This can be explained in terms of complexity – as regularization increases, the model becomes simpler.*

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: *Our goal is to minimize the out of sample error so we can achieve the best generalization. So we will choose the $\lambda = 0.78125$ where the validation error is minimized.*

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.

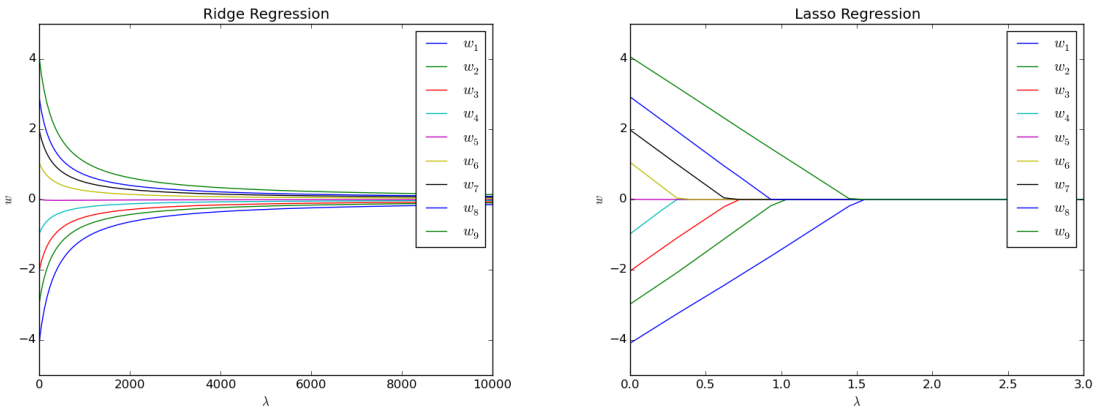
i. Train a linear regression model on the problem3data.txt data with Lasso regularization for regularization strengths α in the vector given by `numpy.linspace(0.01, 3, 30)`. On a single plot, plot each of the model weights $w_1, ..., w_9$ (ignore the bias/intercept) as a function of α.

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:

See attached Python:



At $\alpha = 0$ (no regularization), both Lasso regression and Ridge regression obtain the weights that minimize the

training error (actually with a small error due to the added noise). For Lasso regression, as α increases, the magnitudes of the weights decrease approximately linearly until they reach exactly zero. The weights reach zero roughly in order of magnitude, with the smallest weights reaching zero first. In contrast, for Ridge regression, as α increases, the weights decrease asymptotically towards zero, but the weights never completely reach zero. Because of this behavior, only Lasso regression is able to obtain sparse estimates.

Problem B [18 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 features, where $d = 1$, 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:

$$\begin{aligned}
\hat{w} &= \arg\min_w \|\mathbf{y} - \mathbf{x}w\|^2 + \lambda\|w\|_1 \\
&= \arg\min_w \left(\mathbf{y}^T\mathbf{y} - \mathbf{y}^T\mathbf{x}w - w\mathbf{x}^T\mathbf{y} + w\mathbf{x}^T\mathbf{x}w + \lambda\|w\|_1\right) \\
&= \arg\min_w \left(\mathbf{y}^T\mathbf{y} - 2\mathbf{y}^T\mathbf{x}w + w\mathbf{x}^T\mathbf{x}w + \lambda|w|\right)
\end{aligned}$$

To minimize this, we take the derivative in terms of w and set it to zero, which results in:

$$\begin{aligned}
-2\mathbf{y}^T\mathbf{x} + 2\mathbf{x}^T\mathbf{x}w + \lambda\,\nabla_w|w| &= 0 \\
\mathbf{x}^T\mathbf{x}w &= \mathbf{y}^T\mathbf{x} - \frac{1}{2}\lambda\,\nabla_w|w| \\
w &= \left(\mathbf{x}^T\mathbf{x}\right)^{-1}\left(\mathbf{y}^T\mathbf{x} - \frac{1}{2}\lambda\nabla_w|w|\right) \qquad (1)
\end{aligned}$$

where:

$$\nabla_w|w| = \begin{cases} [-1,+1] & \textit{if } w = 0 \\ \text{sign}(w) & \textit{otherwise} \end{cases}.$$

Note that there is a continuous range of (sub-)gradients (or slopes in this one-dimensional case) at the origin, since $|w|$ is not differentiable there. The optimal solution is $w = 0$ if any subgradient within the range $[-1, +1]$ satisfies the optimality condition of (1). In other words, we take $w = 0$ when any subgradient from $[-1, 1]$ causes

(1) *to go to 0. Then,*

$$w = \begin{cases} \left(\mathbf{x}^T\mathbf{x}\right)^{-1}\left(\mathbf{y}^T\mathbf{x} - \frac{1}{2}\lambda\,\text{sign}(w)\right) & \textit{if } |\mathbf{y}^T\mathbf{x}| > \frac{1}{2}\lambda \\ 0 & \textit{otherwise} \end{cases}$$

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

Solution B.ii: $w = 0$ *when* $\lambda \geq 2|\mathbf{y}^T\mathbf{x}|$ *and so the smallest value for which* $w = 0$ *is* $2|\mathbf{y}^T\mathbf{x}|$.

iii. 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 B.iii:

$$\begin{aligned}
\hat{\mathbf{w}} &= \arg\min_{\mathbf{w}} \|\mathbf{y} - \mathbf{X}\mathbf{w}\|^2 + \lambda\|\mathbf{w}\|_2^2 \\
&= \arg\min_{\mathbf{w}} \left(\mathbf{y}^T\mathbf{y} - \mathbf{y}^T\mathbf{X}\mathbf{w} - \mathbf{w}^T\mathbf{X}^T\mathbf{y} + \mathbf{w}^T\mathbf{X}^T\mathbf{X}\mathbf{w} + \lambda\mathbf{w}^T\mathbf{w}\right) \\
&= \arg\min_{\mathbf{w}} \left(\mathbf{y}^T\mathbf{y} - 2\mathbf{y}^T\mathbf{X}\mathbf{w} + \mathbf{w}^T\mathbf{X}^T\mathbf{X}\mathbf{w} + \lambda\mathbf{w}^T\mathbf{w}\right)
\end{aligned}$$

By differentiating in terms of $\mathbf{w}$ *and setting the result to* $\mathbf{0}$,

$$-2\mathbf{X}^T\mathbf{y} + 2\left(\mathbf{X}^T\mathbf{X} + \lambda\mathbf{I}\right)\mathbf{w} = 0 \tag{2}$$

$$\left(\mathbf{X}^T\mathbf{X} + \lambda\mathbf{I}\right)\mathbf{w} = \mathbf{X}^T\mathbf{y} \tag{3}$$

$$\mathbf{w} = \left(\mathbf{X}^T\mathbf{X} + \lambda\mathbf{I}\right)^{-1}\mathbf{X}^T\mathbf{y} \tag{4}$$

iv. In this question, we consider Ridge regularization in 1-dimension. Suppose that when $\lambda = 0$, $w \neq 0$. Does there exist a value for $\lambda > 0$ such that $w = 0$? If so, what is the smallest such value?

Solution B.iv:

There is no finite choice of λ that can force w to zero. Note, for $\lambda = 0$, $w = \left(\mathbf{X}^T\mathbf{X} + \lambda\mathbf{I}\right)^{-1}\mathbf{X}^T\mathbf{y}$ is non-zero. This requires that $\mathbf{X}^T\mathbf{y}$ is nonzero.

First, we show $(\mathbf{X}^T\mathbf{X} + \lambda\mathbf{I})$ is invertible; this actually holds for any value of d. Recall

$$\det(\mathbf{X}^T\mathbf{X} + \lambda\mathbf{I}) = \det(\mathbf{X}^T\mathbf{X} - (-\lambda\mathbf{I})) = p_{\mathbf{X}^T\mathbf{X}}(-\lambda),$$

the characteristic polynomial of $\mathbf{X}^T\mathbf{X}$, whose roots are the eigenvalues of $\mathbf{X}^T\mathbf{X}$. Note, $\mathbf{X}^T\mathbf{X}$ is positive semi-definite, and has non-negative eigenvalues. Therefore $p_{\mathbf{X}^T\mathbf{X}}(-\lambda) \neq 0 \; \forall \lambda > 0$.

Then, by the Invertible Matrix Theorem, $\mathbf{X}^T\mathbf{X} + \lambda\mathbf{I}$ is invertible, and also null $\{\mathbf{X}^T\mathbf{X} + \lambda\mathbf{I}\} = \{\mathbf{0}\}$.

In particular, in one dimensional ridge regression, $\mathbf{X}^T\mathbf{X} + \lambda\mathbf{I}$ is invertible, and thus a non-zero scalar.

Therefore, $w = \left(\mathbf{X}^T\mathbf{X} + \lambda\mathbf{I}\right)^{-1}\mathbf{X}^T\mathbf{y}$ is non-zero.