Machine Learning & Data Mining
Set 2

Caltech CS/CNS/EE 155
January 12th, 2024

Policies

- Due 5 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.
- You are allowed to use up to 48 late hours across the entire term. Late hours must be used in units of whole hours. Specify the total number of hours you have used when submitting the assignment.
- **No use of large language models is allowed.** Students are expected to complete homework assignments based on their understanding of the course material.

Submission Instructions

- Submit your report as a single .pdf file to Gradescope (entry code 2P8P28), under "Set 2 Report".
- In the report, **include any images generated by your code** along with your answers to the questions.
- Submit your code by **sharing a link in your report** to your Google Colab notebook for each problem (see naming instructions below). Make sure to set sharing permissions to at least "Anyone with the link can view". **Links that can not be run by TAs will not be counted as turned in.** Check your links in an incognito window before submitting to be sure.
- For instructions specifically pertaining to the Gradescope submission process, see https://www.gradescope.com/get_started#student-submission.

Google Colab Instructions

For each notebook, you need to save a copy to your drive.

1. Open the github preview of the notebook, and click the icon to open the colab preview.
2. On the colab preview, go to File → Save a copy in Drive.
3. Edit your file name to "lastname_firstname_set_problem", e.g."yue_yisong_set2_prob1.ipynb"

1 Comparing Different Loss Functions [30 Points]

Relevant materials: lecture 3 & 4

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

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

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

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

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

Solution A:

Squared loss function is directly related to the distance of a point from the classification boundary, thus creating an unneeded weight/penalty on points farther from the boundary. Additionally, the squaring of the loss function $(1-yw^Tx)$ will lead the loss function to build up losses even on points where y and w^Tx have the same sign since the sign of each loss term will always be made positive. For these reasons, the squared loss function would throw off the true error estimates of classification by excessively penalizing points farther from the decision boundary and points that are potentially correctly classified, and thus are not a good choice of loss function for classification problems.

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

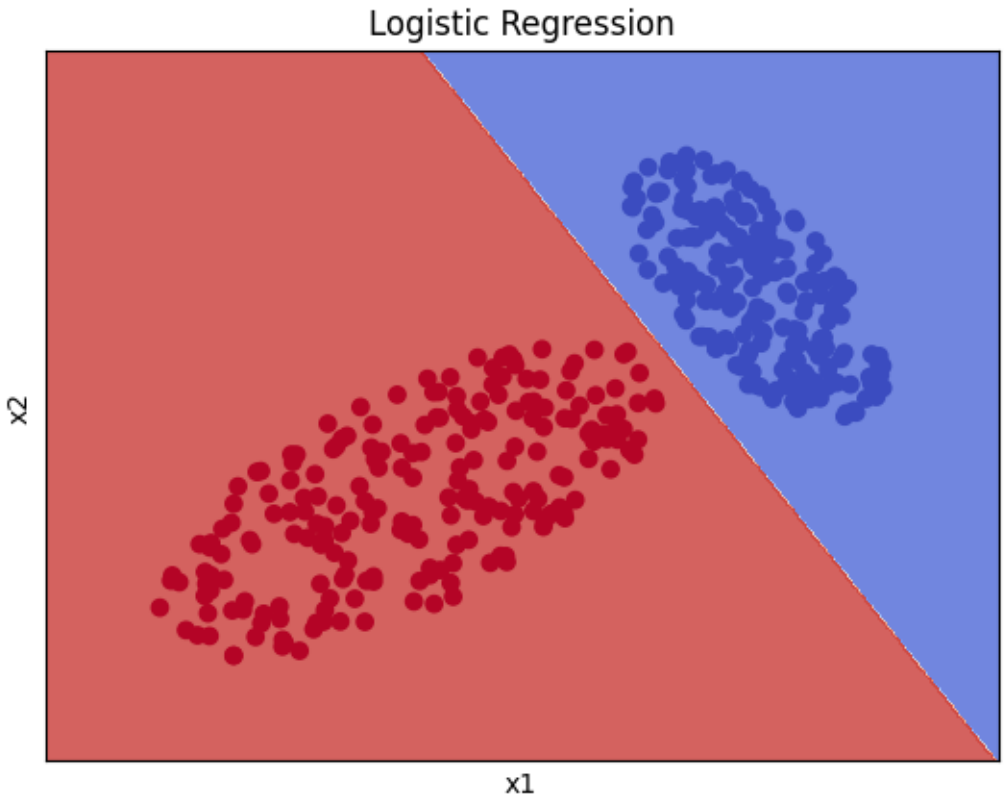
On this dataset, train both a logistic regression model and a ridge regression model to classify the points. (In other words, on each dataset, train one linear classifier using $L_{\log}$ as the loss, and another linear classifier using L_{squared} as the loss.) For this problem, you should use the logistic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

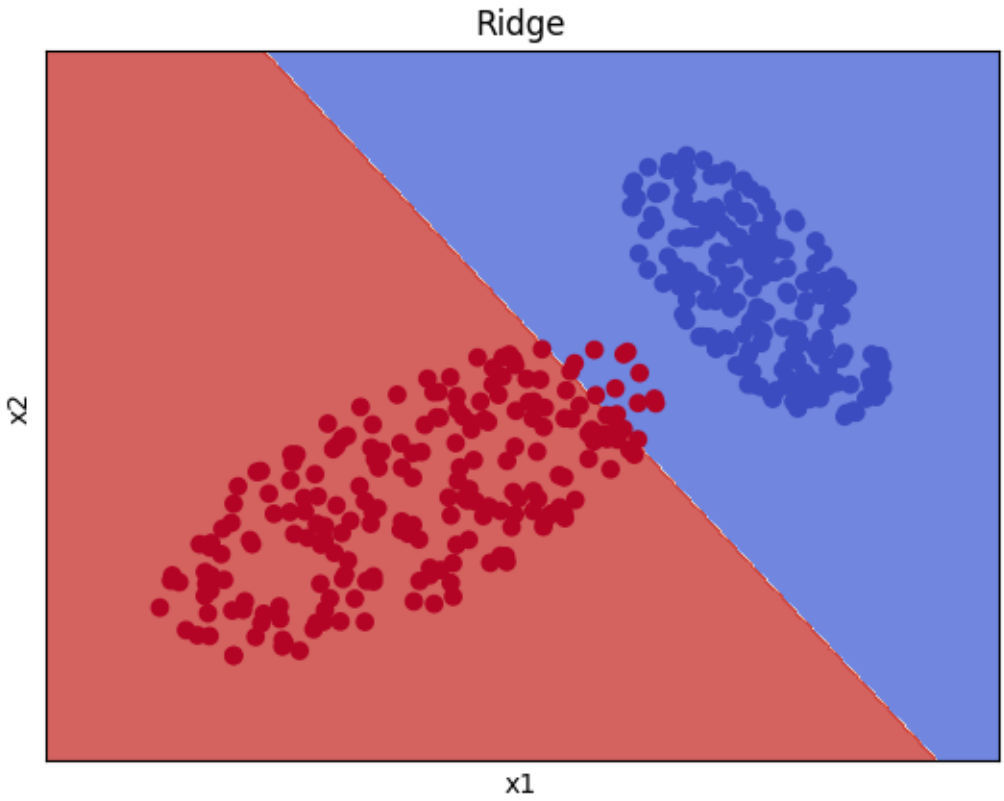
For each loss function/model, plot the data points as a scatter plot and overlay them with the decision boundary defined by the weights of the trained linear classifier. Include both plots in your submission. The template notebook for this problem contains a helper function for producing plots given 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:

Code for solution

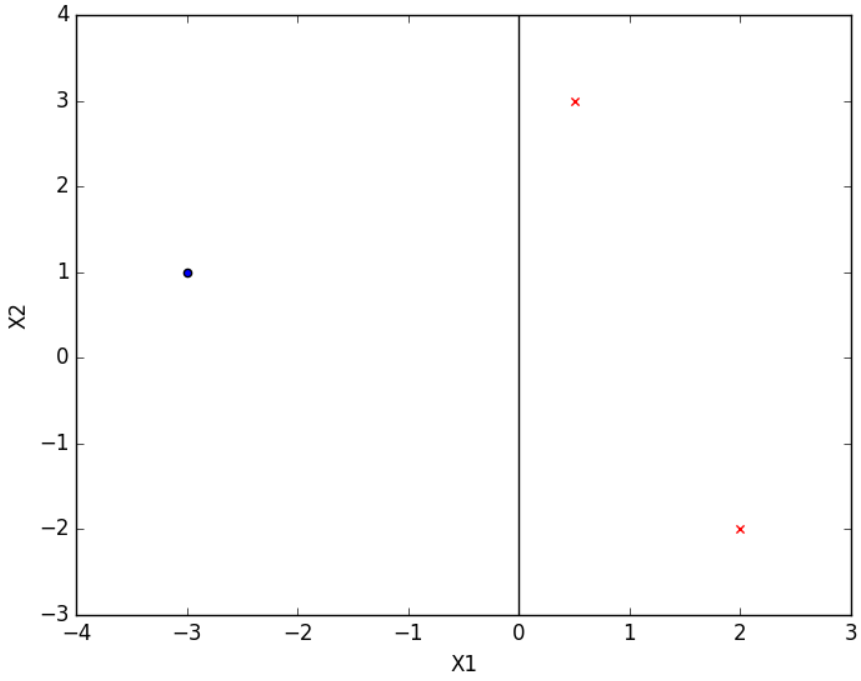




We see that the logistic regression with low regularization is able to perfectly classify and create a decision boundary that very clearly separates both data clusters, while the ridge regression does not classify perfectly on the training data and forms a decision boundary that cuts through one of the classification clusters rather than forming a boundary in the space of the separation of the data. This seems to occur because of the reasoning from part A that one classification of points has a wide spread of data and thus the points in that classification that are farther away from the decision boundary penalize the boundary more and move the boundary towards those points to the degree that the boundary begins to misclassify closer points (this happens because Ridge regression uses a squared loss function for classification not suited well for classification).

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:

Code for solution

$L_{hinge} = \max(0, 1 - y\mathbf{w}^T\mathbf{x}) \rightarrow = 1 - y\mathbf{w}^T\mathbf{x}$ if $y\mathbf{w}^T\mathbf{x} < 1$
$\partial L_{hinge} = \{-y\mathbf{x}$ s.t. $y\mathbf{w}^T\mathbf{x} < 1$, else $0\}$
$L_{log} = \ln 1 + e^{-y\mathbf{w}^T\mathbf{x}} \rightarrow$ chain rule with respect to $1 + e^{-y\mathbf{w}^T\mathbf{x}}$
$\partial L_{log} = \frac{-y\mathbf{x} * e^{-y\mathbf{w}^T\mathbf{x}}}{1+e^{-y\mathbf{w}^T\mathbf{x}}} = \frac{-y\mathbf{x}}{e^{y\mathbf{w}^T\mathbf{x}}+1}$

Hinge loss on S = ($\frac{1}{2}$, 3): (-1, -$\frac{1}{2}$, -3), (2, -2): 0, (-3, 1): 0 (calculated manually second and third points are zero due to $y\mathbf{w}^T\mathbf{x} < 1$)

Log loss on S = ($\frac{1}{2}$, 3): (-0.37754067 -0.18877033 -1.13262201), (2, -2): (-0.11920292 -0.23840584 0.23840584), (-3, 1): (0.04742587 -0.14227762 0.04742587) (computed with code attached to answer)

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 hinge loss gradient can converge to 0 if all the points classified are outside the margin which would yield $y\mathbf{w}^T\mathbf{x} >= 1$ for all points x. On the other hand the gradient for log loss will never approach zero which can be observing the fact that logarithmic functions are asymptotic and do not approach zero (assuming no constant value added alongside the logarithm). For a linearly separable dataset, by analyzing the function we notice that error can be reduced for both hinge and log loss by increasing the magnitude of weight vector w as that would allow the margin to decrease for hinge loss (larger weights mean smaller margin and lesser hinge loss) and the exponential component of the logarithmic loss also decreases with a larger magnitude of w, the change of w magnitude does not change the decision boundary however giving a way to reduce or altogether eliminate training error on linearly separable data without changing the decision 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:

The error with just minimizing L_{hinge} is that simply increasing the magnitude of some given decision boundary (not necessarily the most optimal) can result in the training loss being zero although that is not necessarily the margin maximizing weight vector. This is clear from the fact that increasing magnitude of the weight vector to reduce L_{hinge} consequently also reduces the margin which is not the goal of the SVM classifier. Thus the loss function must also have some component that penalizes increases in the weight vectors magnitude which is $\lambda\|w\|^2$, so the loss function optimizes the loss of L_{hinge} for classification accuracy and $\lambda\|w\|^2$ for the maximizing margin of that accuracy by forcing the magnitude of w to be as small as possible and thus forming a more optimal decision boundary.

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, we add a regularization penalty term. Can adding the penalty term decrease the training (in-sample) error? Will adding a penalty term always decrease the out-of-sample errors? Please justify your answers. Think about the case when there is over-fitting while training the model.

Solution A:

Adding a regularization penalty term cannot decrease training error because without regularization the least-squares regression is the most optimal loss-reducing strategy with respect to solely the training data. The regularization term helps with overfitting and thus would lead the training of the model to be less fit to noise and likely increase training error. Adding a penalty term will not always decrease the out-of-sample error, it depends on the importance/ weighting given to that regularization penalty term. If extremely high penalty is given by the term then the regression will not fit to any data and simply take on the average of the output of the training error and most likely lead to a very high (often higher than without regularization) out-of-sample test error (underfitting). The penalty term decreasing the out-of-sample error thus depends much on the weighting of that term.

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 very often because it is not continuous and thus not differentiable so it is not a good term to add to a loss function since its gradient does not exist.*

Implementation of ℓ_2 regularization:

We are going to experiment with regression for the Red Wine Quality Rating data set. The data set is uploaded on the course website, and you can read more about it here: https://archive.ics.uci.edu/ml/datasets/Wine. The data relates 13 different factors (last 13 columns) to wine type (the first column). Each column of data represents a different factor, and they are all continuous features. Note that the original data set has three classes, but one was removed to make this a binary classification problem.

Download the data for training and validation from the assignments data folder. There are two training sets, wine_training1.txt (100 data points) and wine_training2.txt (a proper subset of wine_training1.txt

containing only 40 data points), and one test set, wine_validation.txt (30 data points). You will use the wine_validation.txt dataset to evaluate your models.

We will train a *ℓ_2-regularized logistic regression* model on this data. Recall that the unregularized logistic error (a.k.a. log loss) is

$$E = -\sum_{i=1}^{N} \log(p(y_i|\mathbf{x}_i))$$

where $p(y_i = -1|\mathbf{x}_i)$ is

$$\frac{1}{1 + e^{\mathbf{w}^T\mathbf{x}_i}}$$

and $p(y_i = 1|\mathbf{x}_i)$ is

$$\frac{1}{1 + e^{-\mathbf{w}^T\mathbf{x}_i}},$$

where as usual we assume that all $\mathbf{x}_i$ contain a bias term. The ℓ_2-regularized logistic error is

$$\begin{aligned} E &= -\sum_{i=1}^{N} \log(p(y_i|\mathbf{x}_i)) + \lambda \mathbf{w}^T\mathbf{w} \\ &= -\sum_{i=1}^{N} \log\left(\frac{1}{1 + e^{-y_i\mathbf{w}^T\mathbf{x}_i}}\right) + \lambda \mathbf{w}^T\mathbf{w} \\ &= -\sum_{i=1}^{N} \left(\log\left(\frac{1}{1 + e^{-y_i\mathbf{w}^T\mathbf{x}_i}}\right) - \frac{\lambda}{N}\mathbf{w}^T\mathbf{w}\right). \end{aligned}$$

Implement SGD to train a model that minimizes the ℓ_2-regularized logistic error, i.e. train an ℓ_2-regularized logistic regression model. Train the model with 15 different values of λ starting with $\lambda_0 = 0.00001$ and increasing by a factor of 5, i.e.

$$\lambda_0 = 0.00001, \lambda_1 = 0.00005, \lambda_2 = 0.00025, ..., \lambda_{14} = 61,035.15625.$$

Some important notes: Terminate the SGD process after 20,000 epochs, where each epoch performs one SGD iteration for each point in the training dataset. You should shuffle the order of the points before each epoch such that you go through the points in a random order (hint: use `numpy.random.permutation`). Use a learning rate of 5×10^{-4}, and initialize your weights to small random numbers.

You may run into numerical instability issues (overflow or underflow). One way to deal with these issues is by normalizing the input data X. Given the column for the jth feature, $X_{:,j}$, you can normalize it by setting $X_{ij} = \frac{X_{ij} - \overline{X_{:,j}}}{\sigma(X_{:,j})}$ where $\sigma(X_{:,j})$ is the standard deviation of the jth column's entries, and $\overline{X_{:,j}}$ is the mean of the jth column's entries. Normalization may change the optimal choice of λ; the λ range given above corresponds to data that has been normalized in this manner. If you treat the input data differently, simply plot enough choices of λ to see any trends.

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

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

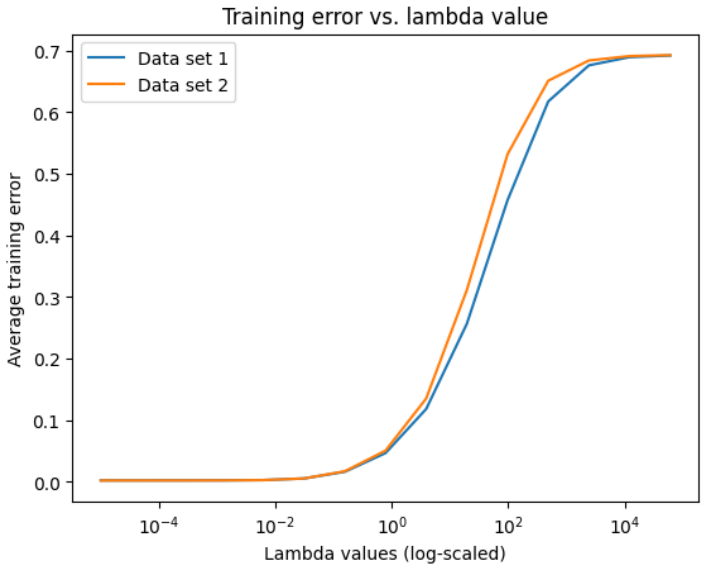
ii. Plot the average test error (E_{out}) versus different λs using wine_validation.txt as the test set.

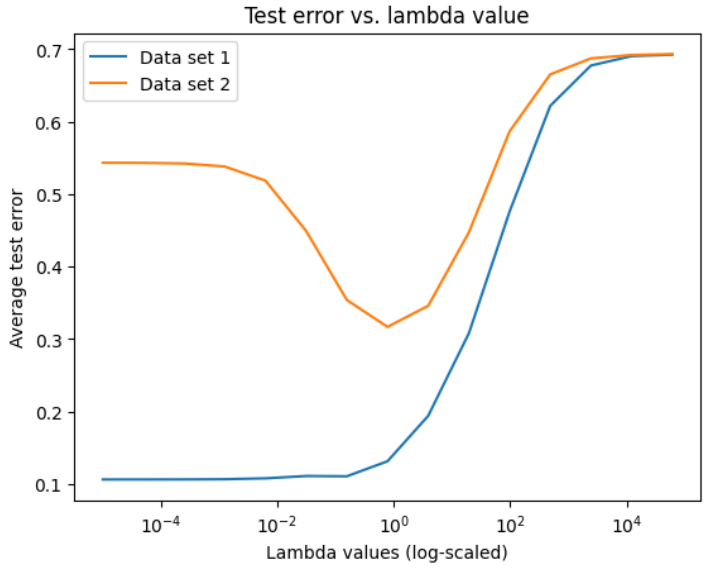
iii. Plot the ℓ_2 norm of $\mathbf{w}$ versus different λs.

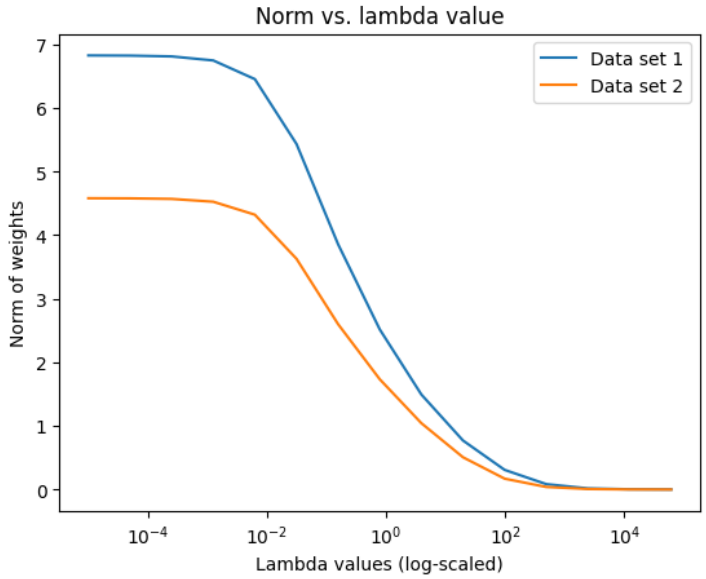
You should end up with three plots, with two series (one for wine_training1.txt and one for wine_training2.txt) on each plot. Note that the E_{in} and E_{out} values you plot should not include the regularization penalty — the penalty is only included when performing gradient descent.

Solution C:

Code for solution







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

Solution D:

The training errors for dataset 1 and 2 are relatively the same which makes sense given that dataset 2 is a subset

of dataset 1. The minor difference is that with higher λ dataset1 produces a slightly smaller training error which makes sense as the increased amount of data points lessen the impact of the overweight regularization term form under fitting as much as dataset2 with less points. For the test set error with training from dataset 1 and 2, the test error as lambda increases for dataset 1 only increases as λ increases whereas for dataset 2 mid-levels of λ produce the lowest test error (dip in the graph as seen in (C)). This makes sense as the many data points in dataset1 already prevent overfitting so regularization will only just increase the error, where as dataset2 being small with only 40 points will have some overfitting without regularization so adding the regularization term with the right amount of weighting will lessen the test error.

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:

Initially for the training of the first dataset with smaller λ the dataset does not produce much overfitting as the small levels of regularization create little difference between the training and test errors. However with higher lambdas the model begins to severely underfit which is seen by the increase in both the test and training error.

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:

Initially the $\ell 2$ norm of $\boldsymbol{w}$ for dataset 1 with smaller lambdas has a very high $\ell 2$ norm which means the regularization term does not have a strong effect as the weights remain large and thus produce smaller margins. However as λ increases, the $\ell 2$ norm converges to 0 which makes sense given that the extreme weighting of the regularization term forces the weight vector $\boldsymbol{w}$ to converge to 0 norm.

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 a λ = 1 as seen by the graph in C, this λ value produces the lowest test error once trained on dataset2 and also the smallest difference between test and training error across all and thus suggests that it generalizes the data best.

3 Lasso (ℓ_1) vs. Ridge (ℓ_2) Regularization [30 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 [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.

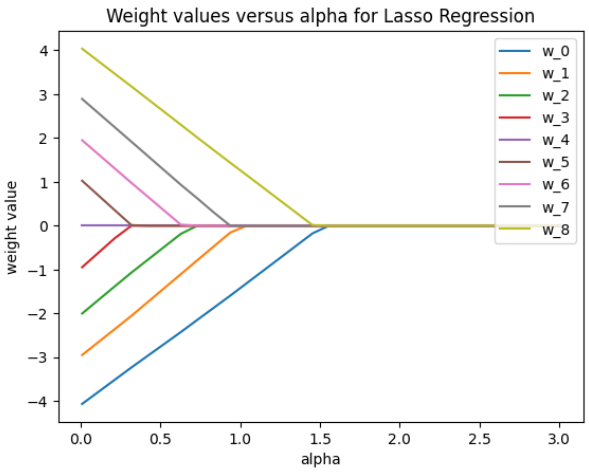
i. Train a linear regression model on the problem3data.txt data with Lasso regularization for regularization strengths α in the vector given by `numpy.linspace(0.01, 3, 30)`. On a single plot, plot each of the model weights $w_1, ..., w_9$ (ignore the bias/intercept) as a function of α.

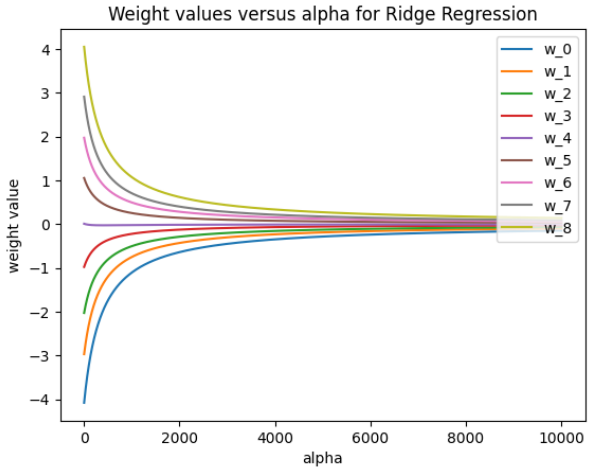
ii. Repeat **i.** with Ridge regression, and this time using regularization strengths $\alpha \in \{1, 2, 3, \ldots, 1e4\}$.

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:

Code for solution





As the regularization parameter increases the number of model weights that are exactly zero increases as well for the Lasso regression. For the Ridge regression the convergence of weights to zero appear to be asymptotic so no model weights are exactly zero but they approach very close to being zero.

Problem B [9 points]:

i. In the case of 1-dimensional data, Lasso regression admits a closed-form solution. Given a dataset containing N datapoints, each with $d = 1$ feature, solve for

$$\arg\min_{w} \|\mathbf{y} - \mathbf{x}w\|^2 + \lambda\|w\|_1,$$

where $\mathbf{x} \in \mathbb{R}^N$ is the vector of datapoints and $\mathbf{y} \in \mathbb{R}^N$ is the vector of all output values corresponding to these datapoints. Just consider the case where $d = 1$, $\lambda \geq 0$, and the weight w is a scalar.

This is linear regression with Lasso regularization.

Solution B.i:

$\nabla\|\mathbf{y} - \mathbf{x}w\|^2 + \lambda\|w\|_1 = -2\mathbf{x}(\mathbf{y} - \mathbf{x}w) \pm \lambda$(dependent on value of w)

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

Now solve both components for w when set to zero, restrict on $\lambda < \pm 2\mathbf{x}^T\mathbf{y}$ otherwise $w = 0$

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

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

Solution B.ii:

Yes notice according to the solution from B.i that if $\lambda = \pm 2\mathbf{x}^T\mathbf{y}$ then the computation for w equals zero, so we can conclude that $2\mathbf{x}^T\mathbf{y}$ is the smallest value of λ for which $w = 0$.

Problem C [9 points]:

i. Given a dataset containing N datapoints each with d features, solve for

$$\arg\min_{\mathbf{w}} \|\mathbf{y} - \mathbf{X}\mathbf{w}\|^2 + \lambda\|\mathbf{w}\|_2^2$$

where $\mathbf{X} \in \mathbb{R}^{N \times d}$ is the matrix of datapoints and $\mathbf{y} \in \mathbb{R}^N$ is the vector of all output values for these datapoints. Do so for arbitrary d and $\lambda \geq 0$.

This is linear regression with Ridge regularization.

Solution C.i:

$\nabla\|\mathbf{y} - \mathbf{X}\mathbf{w}\|^2 + \lambda\|\mathbf{w}\|_2^2 = -2\mathbf{x}^\mathbf{T}(\mathbf{y} - \mathbf{x}w) + 2\lambda w$

$\rightarrow 0 = -2\mathbf{x}^\mathbf{T}\mathbf{y} + 2\mathbf{x}^T\mathbf{x}w + 2\lambda w = -\mathbf{x}^T\mathbf{y} + \mathbf{x}^T\mathbf{x}w + \lambda w$

$w = (\mathbf{x}^T\mathbf{x} + \lambda\mathbf{I_d})^{-1}(\mathbf{x}^\mathbf{T}\mathbf{y})$

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:

Notice that the first term for w is $(\mathbf{x}^T\mathbf{x} + \lambda\mathbf{I_d})^{-1}$, so no matter the value of λ the term is finite and non-zero and the second term has no λ term at all and thus as long as $\mathbf{X}^T y \neq 0$ (which should be true as long as X is not empty) no lambda value produce a w exactly zero.