Policies

• Due 9 PM PST, January 20th on Gradescope.

• You are free to collaborate on all of the problems, subject to the collaboration policy stated in the syllabus.

• In this course, we will be using Google Colab for code submissions. You will need a Google account.

Submission Instructions

• Submit your report as a single .pdf file to Gradescope (entry code K3RPGE), under ”Set 2 Report”. • In the report, **include any images generated by your code** along with your answers to the questions.

• Submit your code by **sharing a link in your report** to your Google Colab notebook for each problem (see naming instructions below). Make sure to set sharing permissions to at least ”Anyone with the link can view”. **Links that can not be run by TAs will not be counted as turned in.** Check your links in an incognito window before submitting to be sure.

• For instructions specifically pertaining to the Gradescope submission process, see https://www.gradescope.com/get_started#student-submission.

Google Colab Instructions

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

1. Open the github preview of the notebook, and click the icon to open the colab preview. 2. On the colab preview, go to File → Save a copy in Drive.

3. Edit your file name to “lastname_firstname_originaltitle”, e.g.”yue_yisong_3_notebook_part1.ipynb”

Caltech CS/CNS/EE 155
Jayden Nyamiaka

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 x and 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 cor rectly, 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 is a poor loss function for classification problems because of how it penalizes correct predictions. This can be analyzed given some correct classification of some point x with label y; let's say our model predicts a large positive value $f(\mathbf{x}) = \mathbf{w}^T\mathbf{x} >> 0$, and our actual label y is 1. This means our model did a good job of predicting the point since both our prediction and label are of the same sign; however, squared loss would output $L_{\text{squared}} = (1 - y\mathbf{w}^T\mathbf{x})^2 >> 0$. Since our prediction was far from 1, our loss was relatively large when our model actually did a good job regarding classification. This happens because of how the squared loss function expects the distribution to be. The squared loss function sees any distance between the prediction and the label as bad when in reality, we usually only care about the distance for incorrect classifications. Hinge loss for example outputs no loss on correct classification regardless of the distance between the label and the prediction, which is why it does a way better job for classification.

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

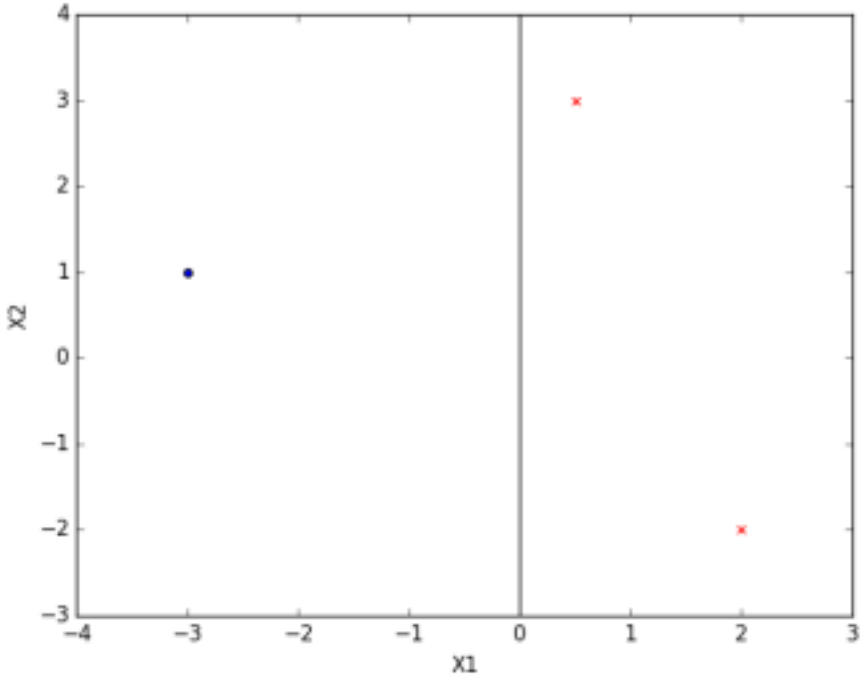
On this dataset, train both a logistic regression model and a ridge regression model to classify the points. (In other words, on each dataset, train one linear classifier using $L_{\log}$ as the loss, and another linear classi fier using L_{squared} as the loss.) For this problem, you should use the 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:
https://colab.research.google.com/drive/1wvko6yrDYlcPQOpDsHncoRduF7QuPOdh?usp=sharing
Generated graphs below.
Log regression correctly classifies all points while ridge regression misclassifies a few points. This occurs because log regression uses $L_{log} = \ln(1 + e^{-yw_T x})$ as its loss function while ridge regression uses $L_{squared} = (1 - yw^T x)$. As I described in Solution A, $L_{squared}$ loss penalizes its model even for correctly classified points that are far from the correct label. So, even if a point is classified correctly by the model, if it's far from the decision boundary, it will drag the decision boundary/classification to that direction. So because the red points have points that are very far from the decision boundary, they drag the decision boundary towards those far red points, misclassifying some red points close to the decision boundary in the process. This is why $L_{squared}$ does a poor job as a loss function for classification due to its penalizing even in the case of correct predictions. On the other hand for log regression, since it uses L_{log} as its loss function it doesn't output much error for correctly classified points and instead focuses its weight vector on misclassified points, which is why it is actually able to correctly classify all the points.

Problem C [9 points]: Leaving squared loss behind, let's focus on log loss and hinge loss. Consider the set of points $S = \{(\frac{1}{2}, 3), (2, -2), (-3, 1)\}$ in 2D space, shown below, with labels $(1, 1, -1)$ respectively. Given a linear model with weights $w_0 = 0$, $w_1 = 1$, $w_2 = 0$ (where w_0 corresponds to the bias term), derive the gradients $\nabla_w L_{hinge}$ 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:

$L_{hinge} = \max(0, 1 - yw^T x) \rightarrow \nabla_w L_{hinge} = \begin{cases} 0 & yw^T x \geq 1 \\ -yx & yw^T x < 1 \end{cases}$

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

$w = [0, 1, 0]$

$yw^T x = y[0 * b, 1 * x_1, 0 * x_2] = yx_1$

Points

$(S_1, y_1) = (\frac{1}{2}, 3), 1 \rightarrow yw^T x = \frac{1}{2}$

$(S_2, y_2) = (2, -2), 1 \rightarrow yw^T x = 2$

$(S_3, y_3) = (-3, 1), -1 \rightarrow yw^T x = 3$

Hinge Loss

(S_1, y_1): $yw^T x = \frac{1}{2} < 1$ so $\nabla_w L_{log} = -yx = -1(1, \frac{1}{2}, 3) = (-1, -\frac{1}{2}, -3)$

(S_2, y_2): $yw^T x = 2 \geq 1$ so $\nabla_w L_{log} = (0, 0, 0)$

(S_3, y_3): $yw^T x = 3 \geq 1$ so $\nabla_w L_{log} = (0, 0, 0)$

Log Loss

*Used Calculator for e computations

(S_1, y_1): $\nabla_w L_{log} = \frac{-yx}{1 + e^{yw^T x}} = \frac{-[1, \frac{1}{2}, 3]}{1 + e^{\frac{1}{2}}} \approx (-0.38, -0.19, -1.13)$

(S_2, y_2): $\nabla_w L_{log} = \frac{-yx}{1 + e^{yw^T x}} = \frac{-[1, 2, -2]}{1 + e^{2}} \approx (-0.12, -0.24, 0.24)$

(S_3, y_3): $\nabla_w L_{log} = \frac{-yx}{1 + e^{yw^T x}} = \frac{1 \cdot [1, -3, 1]}{1 + e^{3}} \approx (0.05, -0.14, 0.05)$

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: Here, we have 3 data points and 2 dimensions (since the bias term doesn't count as a dimension), so from the conclusion from the last problem set, we know it's possible to perfectly classify all 3 points given that none of the points are coplanar. For hinge loss, the gradients of two of the 3 points are already zero vectors with only one nonzero loss. We can easily have the loss for hinge loss be 0 if all points have $yw^T x >= 1$. So, if we increase the weight vector without changing the decision boundary, we can achieve this. On the other hand, for log loss, none of the errors are nonzero although the error is relatively low. This is because the error for log loss can never really be 0. Instead the error can asymptotically converge to 0 with larger and larger values of $yw^T x$. We can similarly reduce the error for log loss by increasing the weight vector so that $yw^T x$ increases.

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 $L_{hinge} + \lambda \, // \, w \, //^2$ for some $\lambda > 0$.

(You don't need to prove that minimizing $L_{hinge} + \lambda \, // \, w \, //^2$ results in a maximum margin classifier; just show that the additional penalty term addresses the issues of minimizing just L_{hinge}.)

Solution E: From the last example, we have seen that we can just increase the weight vector without changing the decision boundary in order to minimize L_{hinge} which is why the $\lambda \, // \, w \, //^2$ term also must be minimized. By imposing that the $\lambda \, // \, w \, //^2$ term also must be minimized, it forces the decision boundary to also change in order to reach the learning objective since $\lambda \, // \, w \, //^2$ restricts the weight vector to be low by the factor lambda. This is why we must be careful in choosing lambda.

2 Effects of Regularization [40 Points]

Relevant materials: Lecture 3 & 4

For this problem, you are required to implement everything yourself and submit code (i.e. don't use scikit learn but numpy is fine).

Problem A [4 points]: In order to prevent over-fitting in the least-squares linear regression problem with continuous outputs (not classification problem), we add a regularization penalty term. Can adding the penalty term decrease the training (in-sample) error? Will adding a penalty term always decrease the out-of-sample errors? Please justify your answers. Think about the case when there is over-fitting while training the model.

Solution A: Adding a regularization term will never decrease the training in-sample error. Adding the regularization term has the effect of making the model less complex so that it doesn't overfit the data. Essentially, it makes it so the model is less likely to encode the idiosyncrasies of the training data by making the model slightly worse in modeling the training data. This makes it so that the model is more likely to generalize (have less error) on out-of-sample data, but since it does this by making the model slightly worse in encoding the training data, it will never decrease the in-sample training error.

Adding a penalty term is likely to decrease the out-of-sample errors but it's not always[\ guaranteed. The penalty terms helps by making the model more simple and able to generalize assuming there are some patterns present in the training data that aren't present in the true distribution. However, if the distribution of the training data is so perfect and well sampled that it matches the true distribution very closely and there are almost none of those noise patterns, then there is no need for the penalty term, and the model will actually do best without it. Imagine some linear relationship that we're trying to model with a linear model class such that all our data is exactly correct and representative of the true distribution. In this case, there is no thing such as overfitting and the penalty will only make our model stray away from the perfect linear relationship and thus make our model worse. So, while in most cases and almost all real world cases, a penalty term will decrease the out-of-sample errors, it will not always.

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: I imagine that there are many reasons not to use ℓ_0 regularization, but the main two I can think of are that it doesn't care much about the actual values of the weight vector and that it makes it hard to optimize the learning objective. Since ℓ_0 regularization simply counts the number of non-zero entries as its regularization penalty, it's not tailored to the actual values of w and doesn't increase with the magnitude of w. I imagine this loss of information can make the model less accurate. Even worse is the fact that it's discontinuous and not differentiable. In order to find our optimal model, we often use gradients to update the values of our weight vector. Since the ℓ_0 norm can't be differentiated, you'd have to either not include its effect in the update or find another way to update your model. Both of these seem like a significant inconvenience, which is probably why ℓ_0 regularization is rarely 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 validation from the assignments data folder. There are two train ing 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.

See GitHub

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

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

Some important notes:

- Terminate the SGD process after 20,000 epochs, where each epoch performs one SGD iteration for each point in the training dataset.
- You should shuffle the order of the points before each epoch such that you go through the points in a random order (hint: use numpy.random.permutation).
- Use a learning rate of 5×10^{-4}, and initialize your weights to small random numbers.
- The *ℓ_2-regularized* logistic learning objective is what you aim to minimize during the training. How ever, when computing the training error and the testing error, you should compute the *unregularized* logistic error.

You may run into numerical instability issues (overflow or underflow). One way to deal with these issues is by normalizing the input data X. Given the column for the jth feature, $X_{:j}$, you can normalize it by setting $X_{ij} = \frac{X_{ij} - X_{:j}}{\sigma(X_{:j})}$ where $\sigma(X_{:j})$ is the standard deviation of the jth column's entries, and $X_{:j}$ 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 train ing2.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 w versus different λs.

You should end up with three plots, with two series (one for wine training1.txt and one for wine train ing2.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:
https://colab.research.google.com/drive/1GtcubBoxaTGMTf9OOhhkYWCu-xTgQwqu?usp=sharing

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 average training error for the 2 training sets are very close to each other for all values of λ, only moving apart slightly in the region around $\lambda = 100$. The average training error for both training sets was almost 0 up until $\lambda = 1$, and then went up to round 0.7 with larger values of λ. However, the validation error shows distinct differences between the models where the testing error is very low for training set 1 (almost 0) and significantly higher for training set 2 (around 0.5) up until around $\lambda = 1$. The testing error for training set 2 however dips and then goes back up around $\lambda = 1$. Both training sets show ideal λ values around $\lambda = 1$ as that's where their testing error seems to be lowest. The fact that the models have low training error with low values for λ show that the model had more than enough complexity to model the data. We can conclude the regularization parameter λ is needed for this model class as the testing error decreases for both models when $\lambda = 1$. For both training and testing data, we see a high error after $\lambda = 1$; this is intuitively obvious because as $\lambda = 1$, the penalty parameter forces the weights of the model to get smaller and smaller, amking the model simpler to the point where it poorly models the data. We can also attribute the difference in testing error for the 2 models to the fact that training set 1 has more data. Since training set 1 has more data, it's a better representation of the true distribution and the model doesn't overfit as much like it does with training set 2's low amount of data points. This makes training model 1 better able to predict unseen data since it had more data to go off of, explaining the differences in the data.

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: I mentioned this a little in the last question, but now I will focus on specifically overfitting and underfitting. We can see that after $\lambda = 1$, both models underfit the data since bth the training and testing errors are very high. This makes sense because the larger values of λ make it such that the weights of the models are very small during optimization, so the models become oversimplified and poorly predict the data as a result. For model 1, it has very low training and testing error before $\lambda = 1$, showing that its modeling the data very well without much overfitting. Not only does it yield low training error but also low testing error showing that its model's complexity is ideal and not encoding the idiosyncrasies of the training data. Although the difference in testing error is very slight, Model 1 has the best validation error right before $\lambda = 1$, showing that this point is where we have the best balance between model complexity and fitting. Model 2 on the other hand overfits its training data significantly before $\lambda = 1$. We can see this because its training error is low being very close to model 1's, but its testing error is way higher. This shows that model 2 is too complex for the amount of training data we have which causes our model to encode the idiosyncrasies of the training data. Around $\lambda = 0.01$, the testing error for model 2 decreases until it reaches its lowest point at $\lambda = 1$, showing that this combination of parameters is able to restrict the model complexity, so it doesn't overfit the training data as much. This shows how crucial the regularization parameter is as the testing error is almost cut in half comparing this $\lambda = 1$ to the small values of λ. This difference between models occurs because model 1 has more data to train with, so it doesn't overfit like model 2 does. Underfitting at high values of λ, however, is unavoidable no matter how much data we have because high values of λ make our weights converge to 0.

Problem F [4 points]: Briefly explain the qualitative behavior of the ℓ_2 norm of **w** with different λs while training with the data in wine training1.txt.

Solution F: With larger values of the penalty parameter λ, the ℓ_2 norm of **w** decreases until almost 0. This can be expected because the goal is to minimize the learning objective, so with larger values of λ, the value of w must decrease to compensate.

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 $\lambda = 1$ because the data has shown that for this normalized data, $\lambda = 1$ makes for the lowest testing error. With low amounts of data (which is the problem wine_training2.txt has) overfitting is a huge issue because models usually end up encoding unwanted patterns and idiosyncrasies present in the training data that aren't reflected in the true distribution. At $\lambda = 1$ however, we are able to limit overfitting despite having low amounts of data as we can see the testing error decrease despite not changing our mathematical function from logistic regression. Adding λ gives us another parameter to tune model complexity, and here, we can see the effect.

3 Lasso (ℓ_1) vs. Ridge (ℓ_2) Regularization [25 Points]

Relevant materials: Lecture 3

For this problem, you may use the scikit-learn (or other Python package) implementation of Lasso and Ridge regression — you don't have to code it yourself.

The two most commonly-used regularized regression models are Lasso (ℓ_1) regression and Ridge (ℓ_2) regression. Although both enforce "simplicity" in the models they learn, only Lasso regression results in sparse weight vectors. This problem compares the effect of the two methods on the learned model parameters.

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

i. Train a linear regression model on the problem3data.txt data with Lasso regularization for regularization strengths α in the vector given by numpy.linspace(0.01, 3, 30). On a single plot, plot each of the model weights $w_1, \ldots, w_9$ (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:
https://colab.research.google.com/drive/1e7Q_4ISd0QsIdgfz9bu-5nMn2rdLGSVv?usp=sharing
Generated graphs below.
As the regularization parameter increases for Lasso regression, the number of weights that equal exactly zero increases until all of the weights equal exactly 0. For ridge regression, however, all of the weights asymptotically approach 0 getting super close, but none of them ever actually equal 0. This has to do with how their loss functions and learning objectives affect their training of the model.

Problem B [7 points]:

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

See GitHub

where $x \in R^N$ is the vector of datapoints and $y \in R^N$ 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:

$$\underset{w}{\text{argmin}} \; \|\vec{y} - \vec{x}w\|^2 + \lambda \|w\|_1$$

$$\underset{w}{\text{argmin}} \; (\vec{y} - \vec{x}w)^2 + \lambda |w|$$

$$\frac{d}{dw}(\vec{y} - w\vec{x})^2 + \lambda |w| = 0$$

$$-2\vec{x}^T(\vec{y} - \vec{x}w) + \lambda \, \text{sign}(w) = 0$$

$$-2\vec{x}^T\vec{y} + 2w\vec{x}^T\vec{x} + \lambda \, \text{sign}(w) = 0$$

$$-2\vec{x}^T\vec{y} + \lambda \, \text{sign}(w) = -2w\vec{x}^T\vec{x}$$

$$\boxed{w = \frac{2\vec{x}^T\vec{y} - \lambda}{2\vec{x}^T\vec{x}}}$$

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: The smallest value such that w = 0 is $\lambda = 2x^T y$.

Problem C [7 points]:

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

See GitHub

where $X \in R^{N \times d}$ is the matrix of datapoints and $y \in R^N$ 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:

$$\underset{w}{\text{argmin}} \; \|\vec{y} - X\vec{w}\|^2 + \lambda \|\vec{w}\|_2^2$$

$$\underset{w}{\text{argmin}} \; (\vec{y} - X\vec{w})^2 + \lambda \vec{w}^T \vec{w}$$

$$\frac{d}{dw}(\vec{y} - X\vec{w})^2 + \lambda \vec{w}^T \vec{w} = 0$$

$$-2X^T(\vec{y} - X\vec{w}) + 2\lambda\vec{w} = 0$$

$$-X^T\vec{y} + X^T X\vec{w} + \lambda\vec{w} = 0$$

$$X^T\vec{y} = X^T X\vec{w} + \lambda\vec{w}$$

$$w = \frac{X^T\vec{y}}{X^T X + \lambda I}$$

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: There is no value for λ such that w = 0. This makes sense considering that the weights asymptotically approached 0 in our plot for ridge regression, but never equaled 0.

4 Convexity and Lipschitz Continuity [10 Points]

This problem develop the notions of convexity and Lipschitz-continuity. These are widely applicable concepts in machine learning that we will explore further over the next few assignments.

A set C is convex if the line segment between any two points in C lies in C, i.e., if for any $x_1, x_2 \in C$ and any θ with $0 \leq \theta \leq 1$, we have

$$\theta x_1 + (1 - \theta)x_2 \in C$$

Problem A [5 points]: Let $C \subseteq \mathrm{R}^n$ be a convex set, with $x_1, \ldots, x_k \in C$, and let $\theta, \ldots, \theta_k \in \mathrm{R}$ satisfy $\theta_i \geq 0$, and $\theta_1 + \cdots + \theta_k = 1$. Show that $\theta_1 x_1 + \cdots + \theta_k x_k \in C$.
Hint: The definition of convexity is that this holds for $k = 2$; you must show it for arbitrary k.

Solution A: *Answer on the next page*.

We can prove our theorem using Induction.

Let our inductive hypothesis statement $A(n)$ for $n \geq 2$ be that given our convex set $C \subseteq \mathbb{R}^n$ with $x_1, \ldots, x_n \in C$, and $\theta_1, \ldots, \theta_n \in \mathbb{R}$ that satisfy $\theta_i > 0$ and $\theta_1 + \ldots + \theta_n = 1$, $\theta_1 x_1 + \ldots + \theta_n x_n \in C$.

First, let's prove our base case $A(n=2)$:

By definition of convexity, we know $\theta_1 x_1 + (1-\theta_1) x_2 \in C$.

We start with $\theta_1 + \theta_2 = 1$ by definition,

which we can rewrite as $\theta_2 = 1 - \theta_1$.

Thus, we can substitute $1-\theta_1$ for θ_2 to yield:

$\theta_1 x_1 + \theta_2 x_2 \in C$.

Thus, we've proven our base case $A(n=2)$.

Now, we'll do the inductive step, proving $A(k+1)$ using $A(k)$, $A(k-1)$, $A(k \ldots)$.

To prove $A(k+1)$, we must prove $\theta_1 x_1 + \ldots + \theta_k x_k + \theta_{k+1} x_{k+1} \in C$.

To do that let's consider $\theta_1 x_1 + \ldots + \theta_k x_k + \theta_{k+1} x_{k+1}$ and isolate, WLOG, some $\theta_u, x_u, \theta_v, x_v$ from the $A(k)$ case with $1 \leq u, v \leq k$, $u \neq v$. Thus, we have

$$\theta_1 x_1 + \ldots + \theta_u x_u + \ldots + \theta_v x_v + \ldots + \theta_k x_k + \theta_{k+1} x_{k+1}$$

Note that because $A(k)$ is true and u, v are from the $A(k)$ case, we know

$$\theta_u x_u + \theta_v x_v \in C.$$

Now, let us rearrange terms.

$$\theta_u x_u + \theta_v x_v + \theta_1 x_1 + \ldots + \theta_k x_k + \theta_{k+1} x_{k+1}$$

Now, we divide θ_i for all $i \neq u, v$ by $(1-\theta_u-\theta_v)$ then multiply back by $(1-\theta_u-\theta_v)$.

$$\theta_u x_u + \theta_v x_v + (1-\theta_u-\theta_v)\left(\frac{\theta_1}{1-\theta_u-\theta_v} x_1 + \ldots + \frac{\theta_k}{1-\theta_u-\theta_v} x_k\right)$$

Note that this is still equivalent to our original $\theta_1 x_1 + \ldots + \theta_k x_k + \theta_{k+1} x_{k+1}$.

By definition of $A(k+1)$, $\theta_1 + \ldots + \theta_u + \ldots + \theta_v + \ldots + \theta_{k+1} = 1$ so $\sum_{i=1, i\neq u,v}^{k+1} \theta_i = 1 - \theta_u - \theta_v$.

Considering this, observe the sum of the $\frac{\theta_i}{1-\theta_u-\theta_v}$ terms where $i \neq u, v$.

$$\sum_{i=1, i\neq u,v}^{k+1} \frac{\theta_i}{1-\theta_u-\theta_v} = \frac{1}{1-\theta_u-\theta_v} \sum_{i=1, i\neq u,v}^{k+1} \theta_i = \frac{1-\theta_u-\theta_v}{1-\theta_u-\theta_v} = 1.$$

Since the sum of all $\frac{\theta_i}{1-\theta_u-\theta_v}$ terms equals 1, we know

$\frac{\theta_1}{1-\theta_u-\theta_v} x_1 + \ldots + \frac{\theta_k}{1-\theta_u-\theta_v} x_k \in C$ due to $A(k), A(k-1)$ being true,

and thus $(1-\theta_u-\theta_v)\left(\frac{\theta_1}{1-\theta_u-\theta_v} x_1 + \ldots + \frac{\theta_k}{1-\theta_u-\theta_v} x_k\right) \in C$.

Now, since we know both

$\theta_u x_u + \theta_v x_v \in C$ and $(1-\theta_u-\theta_v)\left(\frac{\theta_1}{1-\theta_u-\theta_v} x_1 + \ldots + \frac{\theta_k}{1-\theta_u-\theta_v} x_k\right) \in C$,

we have shown that $\theta_1 x_1 + \ldots + \theta_k x_k + \theta_{k+1} x_{k+1} \in C$, proving $A(k+1)$.

Therefore, we've proven our theorem by Induction. □

Problem B [5 Extra Credit points]: Consider a metric space (X, d_X) and (Y, d_Y), where X and Y are sets and d_X and d_Y denote the metric on X and Y, respectively. Then, a function $f : X \rightarrow Y$ is said to be K-Lipschitz continuous if and only if there exists a real-valued constant $K \geq 0$ such that, for all x_1 and x_2 in X, we have:

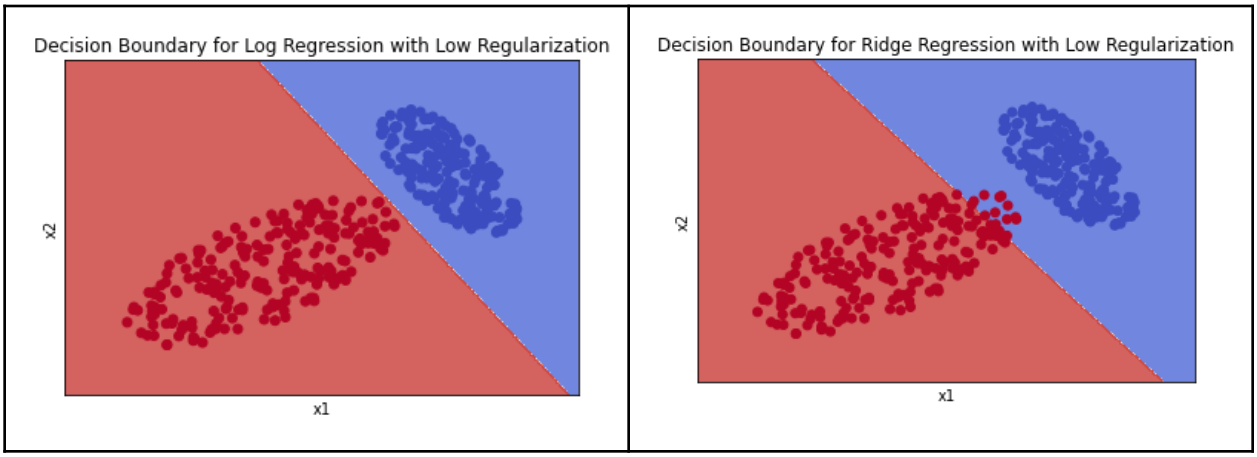
$$\frac{d_Y(f(x_1), f(x_2))}{d_X(x_1, x_2)} \leq K$$

Consider the metric spaces $(D_i, \| \cdot \|)$ for all $i \in \{1, \ldots, t+1\}$, where $\| \cdot \|$ is the L_2 norm metric. Suppose that the sequence of functions $f_1, \ldots, f_t$ satisfies the property that $f_i: D_i \rightarrow D_{i+1}$ is L-Lipschitz continuous for all $i \in \{1, 2, \cdots, t\}$. Then, show that their composition $(f_t \circ f_{t-1} \circ \cdots \circ f_1)$ is L^t-Lipschitz continuous.

Solution B: *I don't know; sorry.*

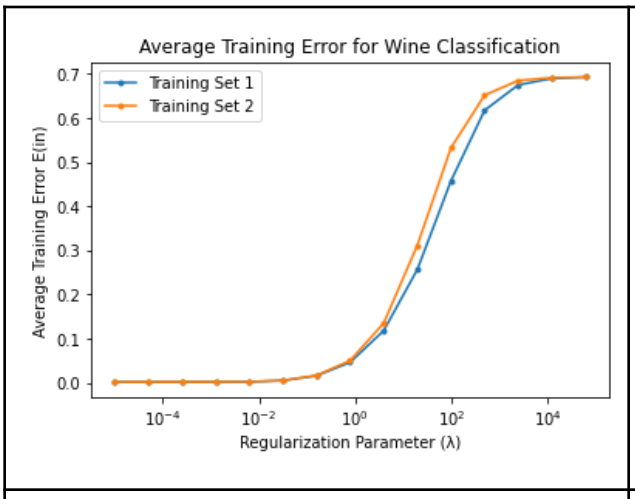
Generated Images

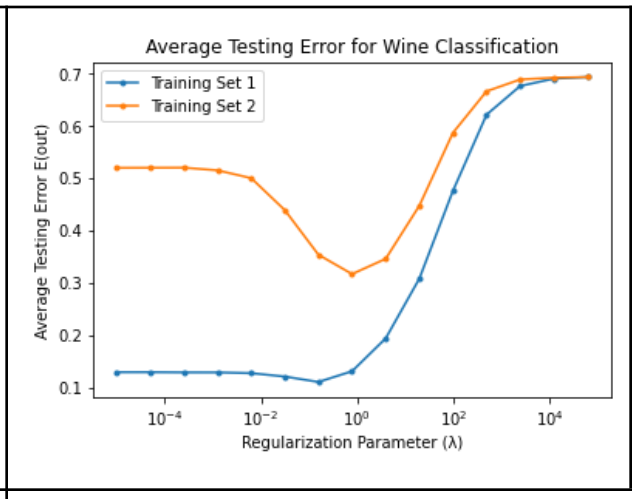
Part 1 B

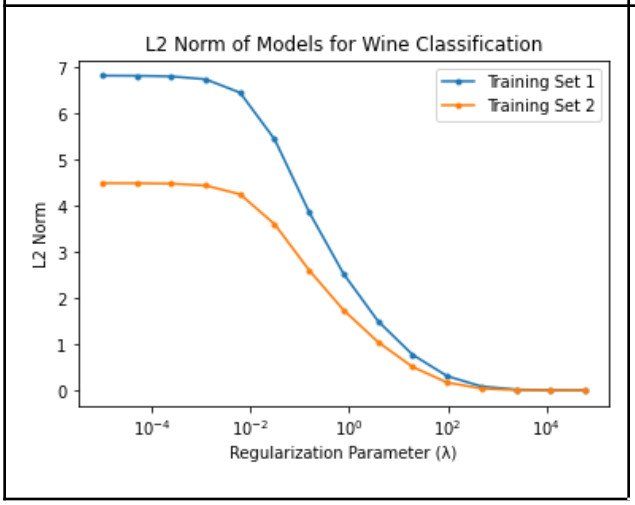


Caltech CS/CNS/EE 155
Jayden Nyamiaka

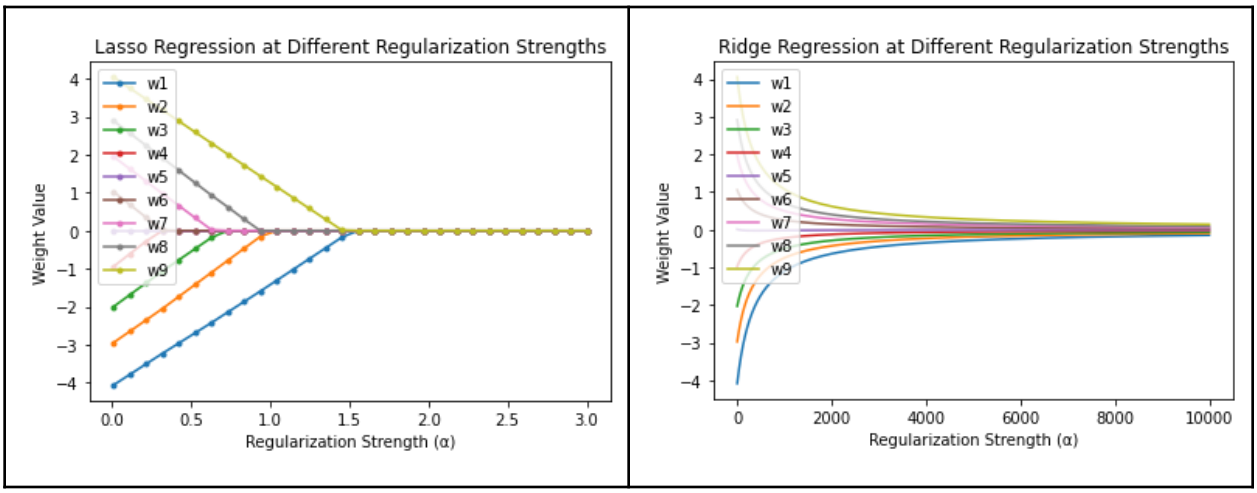
Part 2 C







Part 3 A


Lasso Regression at Different Regularization Strengths
Weight Value
Regularization Strength (α)
w1
w2
w3
w4
w5
w6
w7
w8
w9
Ridge Regression at Different Regularization Strengths
Weight Value
Regularization Strength (α)