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

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: *Classifications problems compare the model output with discrete values corresponding to the different classes. When squared loss is computed on discrete classification values, it can penalize certain classes based on the numbering system. Squared loss is better for analyzing real valued functions, since there is a need to measure how inaccurate the model is. If the problem is classification, the model needs to classify correctly, which least-squares does a poor job of measuring how accurate the model is.*

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 Link*

https://colab.research.google.com/drive/1VjRsAW8tTSVNla07FEH3yLSqAfo9t536?usp=sharing

Figure 1: Result of ridge regression

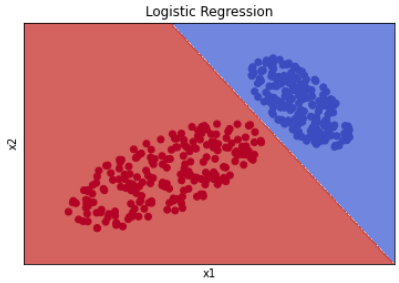
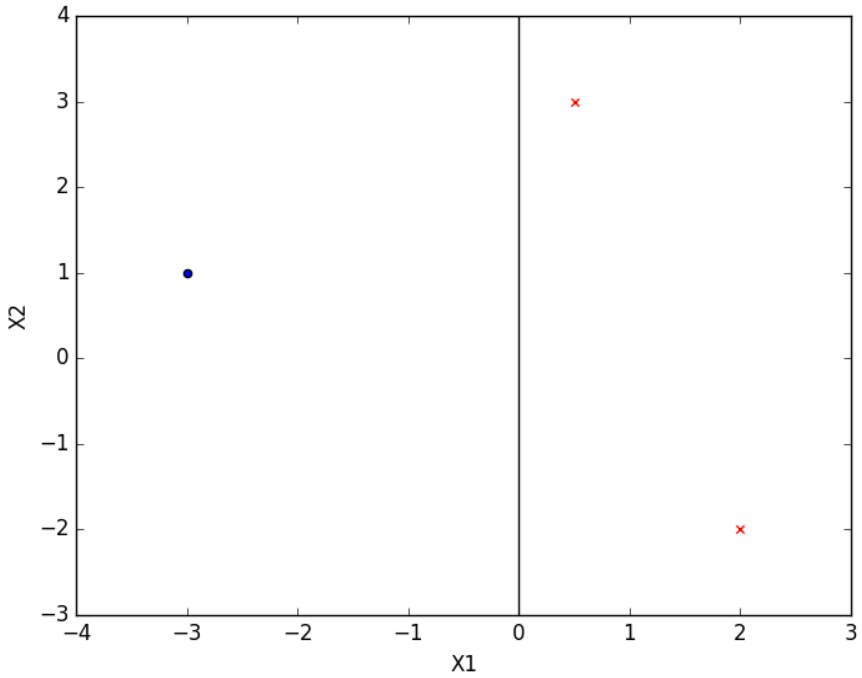


Figure 2: Result of logistic regression

The ridge regression model splits the cluster of red points while the logistic regression boundary separates the two clusters. This is because ridge regression uses least-squares loss, so the furthest cluster of red points "drag" the weight vector out, since the model needs to minimize the squared distance to those points. Logistic regression, on the other hand, has a loss based on classification errors, so it is more likely to separate the clusters of points based on 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: *For hinge loss,*

$$\nabla_w L = \begin{cases} -yx & y(w \cdot x) < 1 \\ 0 & y(w \cdot x) > 1 \end{cases}$$

Calculating for the 3 points,

$$x = \begin{bmatrix} 1 \\ \frac{1}{2} \\ 3 \end{bmatrix}, y = 1 \Rightarrow \nabla_w L(x, y) = \begin{bmatrix} -1 \\ -\frac{1}{2} \\ -3 \end{bmatrix}$$

$$x = \begin{bmatrix} 1 \\ 2 \\ -2 \end{bmatrix}, y = 1 \Rightarrow \nabla_w L(x, y) = 0$$

$$x = \begin{bmatrix} 1 \\ -3 \\ 2 \end{bmatrix}, y = -1 \Rightarrow \nabla_w L(x, y) = 0$$

For logistic loss,

$$\nabla_w L(x, y) = -\frac{yxe^{-yw \cdot x}}{1 + e^{-yw \cdot x}}$$

Calculating for the 3 points,

$$x = \begin{bmatrix} 1 \\ \frac{1}{2} \\ 3 \end{bmatrix}, y = 1 \Rightarrow \nabla_w L(x, y) = -\frac{e^{-\frac{1}{2}}}{1 + e^{-\frac{1}{2}}} \begin{bmatrix} 1 \\ \frac{1}{2} \\ 3 \end{bmatrix} = \begin{bmatrix} -0.3775 \\ -0.1888 \\ -1.1326 \end{bmatrix}$$

$$x = \begin{bmatrix} 1 \\ 2 \\ -2 \end{bmatrix}, y = 1 \Rightarrow \nabla_w L(x, y) = -\frac{e^{-2}}{1 + e^{-2}} \begin{bmatrix} 1 \\ 2 \\ -2 \end{bmatrix} = \begin{bmatrix} -0.1192 \\ -0.2384 \\ 0.2384 \end{bmatrix}$$

$$x = \begin{bmatrix} 1 \\ -3 \\ 1 \end{bmatrix}, y = -1 \Rightarrow \nabla_w L(x, y) = \frac{e^{-3}}{1 + e^{-3}} \begin{bmatrix} 1 \\ -3 \\ 1 \end{bmatrix} = \begin{bmatrix} 0.0474 \\ -0.1423 \\ 0.0474 \end{bmatrix}$$

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: *For log loss, the gradients become smaller, but never go exactly to 0. For hinge loss, the gradient converges to 0 if the correctly classified point is at least 1 away from the decision boundary. For a linearly separable dataset, it is not possible to eliminate training error for logistic loss, since it will always be non-zero by definition. For hinge loss, the training error cannot be eliminated if non-zero, but it is possible for the hinge loss to have no training error.*

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 gradient of hinge loss for correctly classified points at least 1 away from the decision boundary becomes 0. Thus, when all points are classified correctly with a margin of 1, the model stops converging. However, this margin is not the maximum margin. The additional term causes the weight vector to shift until the margin is maximized, not just with a value of 1.*

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).

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, adding the term cannot decrease the training error. The model will seek to minimize the training error, but the regularization term and the new optimization objective using it could prevent a minimum training error for a smaller regularization term. At best, the model will match the unregularized training error if the regularization term has a minimum that coincides with the loss function, but it cannot do better.*

Adding a penalty term will not always decrease the out-of-sample error either. Imagine that we add an extremely large regularization term. Then, the model will minimize the regularization term and ignore the training data. This could result in a model that severly underfits the data and thus has a high out-of-sample 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: *The L^0 norm takes on discrete values for the number of non-zero entries in w. Thus it is not differentiable and impossible to compute the gradients for. If we are trying to use a gradient-based optimization algorithm such as SGD, we cannot tell how to update the weights to minimize the L^0 norm. Additionally, the L^0 norm is not convex, so it cannot be optimized.*

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 link*

https://colab.research.google.com/drive/19-iOw1Acna3jRDQlfqKoDdRcoWjpk4l0?usp=sharing

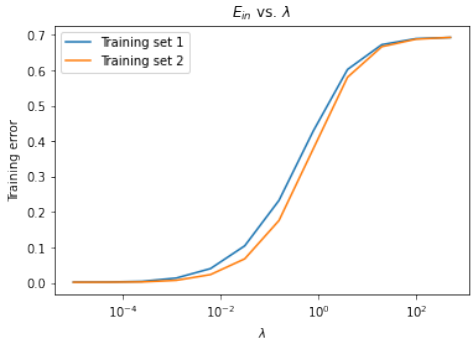


Figure 3: In-sample error plotted for different regularization values

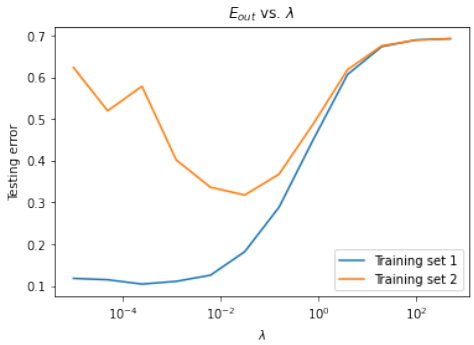


Figure 4: Out-of-sample error plotted for different regularization values

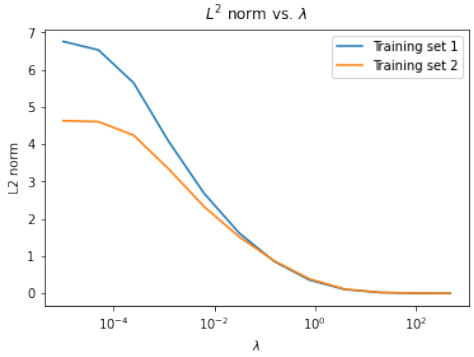


Figure 5: L2 norm plotted for different regularization values

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: *Training on both datasets produces similar training error. However, the test error for training on the smaller dataset was much higher. As the regularization term increased, the two converged in error on the test set. This is because the model overfit on the smaller dataset for low regularization, but as regularization got larger, both models were restricted to underfitting, and the error increased out-of-sample.*

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: *While regularization was low, both training and test error were generally low, so no overfitting or underfitting occured. However, as the regularization term increased, the training and test error both increased, which is an indication of underfitting.*

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 the regularization term increased, the L^2 norm decreased. This is because more of the calculated loss is proportionally the value of the norm, so the gradient will choose to minimize the norm to minimize the overall loss more than the logistic loss.*

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: *The value for lambda that minimized overfitting on the smaller dataset and produced the lowest out-of-sample error was around $\lambda = 10^{-1}$, so the regularization value close to that value would be the one I would choose.*

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: *Code link*

https://colab.research.google.com/drive/1MFb8LmIevdUfWzm4alZgalx_ds-Kh64_?usp=sharing

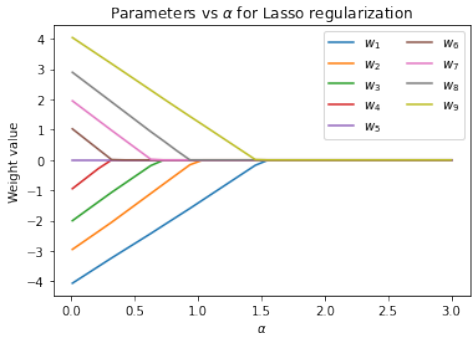


Figure 6: Weight parameters plotted for different regularization values, lasso regularization

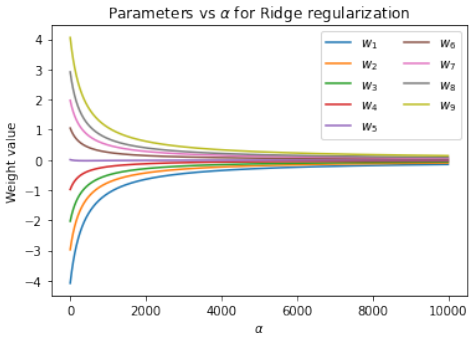


Figure 7: Weight parameters plotted for different regularization values, ridge regularization

As the regularization strength increases, the number of weights that go to 0 increases for Lasso regression. For Ridge regression, none of the weights go exactly to 0, but they become much smaller values.

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: *In one dimension,*

$$L(x, y, w) = ||y - xw||^2 + \lambda||w||_1 = (y - xw)^2 + \lambda w$$

To find the closed form solution,

$$\begin{aligned} 0 &= \frac{d}{dw} L(x, y, w) \\ &= \frac{d}{dw}(x^2w^2 - 2ywx + y^2 + \lambda w \\ &= 2x^2w - 2yx + \lambda \\ w &= \frac{2yx - \lambda}{2x^2} \end{aligned}$$

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: *The smallest such value is $\lambda = 2yx$, so the numerator of the closed-form solution becomes 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: *Again, we take the derivative and set to 0.*

$$\begin{aligned} 0 &= 2X^T(y - Xw) + 2\lambda w \\ \lambda w &= X^T y - X^T X w \\ (\lambda I + X^T X) w &= X^T y \\ w &= (\lambda I + X^T X)^{-1} X^T y \end{aligned}$$

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: *There is no such value, since the matrix $(\lambda I + X^T X)$ cannot become 0 if it is not 0 when $\lambda = 0$.*