Machine Learning & Data Mining
Set 2

Caltech CS/CNS/EE 155
January 17th, 2025

Policies

- Due 11:59 PM PST, January 24th 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.
- Students are expected to complete homework assignments based on their understanding of the course material. Student can use LLMs as a resource (e.g., helping with debugging, or grammar checking), but the assignments (including code) should be principally authored by the student.

Submission Instructions

- Submit your report as a single .pdf file to Gradescope (entry code DKB4KW), under "Problem Set 2".
- 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 is a poor choice for classification problems because it is sensitive to outliers, does not enforce a large margin, and treats classification as a regression problem, leading to slower convergence and suboptimal decision boundaries.*

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 Logistic regression creates a sharper decision boundary because it is designed for classification, maximizing class separation. In contrast, ridge regression treats classification as a regression problem, leading to a less defined boundary and poorer separation of classes*

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.

figs/SimpleDatasetWithDecisionBoundary.png

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: ***Gradient:***

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

Hinge Loss:

$$\nabla_w L_{hinge}(w, x, y) = \begin{cases} 0, & \textit{for } y(w^T x) > 1 \\ -yx, & \textit{otherwise} \end{cases} \tag{2}$$

- ***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: *Hinge loss gradients become zero once all points satisfy the margin $yw^T x > 1$, while log loss gradients never fully reach zero but become arbitrarily small as confidence increases. This means log loss continuously updates even correctly classified points, whereas hinge loss stops updating once the margin is met. To reduce training error without changing the decision boundary, one can scale up w to increase confidence or apply regularization to control unnecessary weight updates.*

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: *For an SVM to achieve a maximum margin, its objective must not only minimize L_{hinge}, but also include a regularization term $\lambda\|w\|^2$ to control the model complexity. Minimizing just L_{hinge} can lead to multiple solutions with different magnitudes of w, and without regularization, the model may not generalize well. The term $\lambda\|w\|^2$ encourages a smaller norm of w, which directly leads to a larger margin between classes, making SVM a maximum margin classifier.*

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 in least-squares linear regression typically increases training error since it restricts the model's flexibility. However, it often reduces out-of-sample error by preventing over-fitting, leading to better generalization. While regularization usually improves generalization, excessively large penalty values can underfit the data, increasing both training and test errors.*

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: *While ℓ_0 regularization can generate an even sparser w than ℓ_1 regularization, it is rarely used because it leads to a combinatorial optimization problem, which is NP-hard and computationally infeasible for high-dimensional datasets.*

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*

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 in training set 2. This occurs because a smaller dataset makes it easier to perfectly fit all data points. As λ increases, the training error for both training sets also increases. The validation error in both sets exhibits a slight decrease before rising sharply for larger values of λ. This behavior suggests that overfitting is present at low λ values. Furthermore, the validation error for training set 2 is significantly higher than that of training set 1 when λ is small, indicating that overfitting is more pronounced in the smaller dataset.*

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: *The training error consistently increases as the regularization term is introduced, which is expected. As for the test error, it initially decreases, indicating that regularization counteracts overfitting.*

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 $\mathbf{w}$ decreases. This can be understood in terms of model complexity—higher regularization forces the model to be simpler by reducing the magnitude of the weight vector.*

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* $\lambda = 1$ *as it minimizes validation error while balancing model complexity and generalization.*

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 As the regularization parameter λ increases, the number of model weights that are exactly zero increases in Lasso regression, as it promotes sparsity by setting some coefficients to zero. In contrast, Ridge regression shrinks the weights towards zero but does not make them exactly 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:

$$\hat{w} = \arg\min_w \|y - xw\|^2 + \lambda\|w\|_1$$

$$= \arg\min_w \left(y^T y - y^T xw - wx^T y + wx^T xw + \lambda|w|\right)$$

$$= \arg\min_w \left(y^T y - 2y^T xw + wx^T xw + \lambda|w|\right)$$

$$\partial f = -2x^T y + 2x^T xw + \lambda\nabla_w|w|$$

$$= \begin{cases} -2x^T y - \lambda, -2x^T y + \lambda & \textit{if } w = 0 \\ -2x^T y + 2x^T xw + \lambda\,\text{sign}(w) & \textit{otherwise} \end{cases}$$

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

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*

$$\lambda_{\min} = 2|y^T x|$$

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:

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

$$= \arg\min_{\mathbf{w}} \left(\mathbf{y}^T\mathbf{y} - \mathbf{y}^T\mathbf{X}\mathbf{w} - \mathbf{w}^T\mathbf{X}^T\mathbf{y} + \mathbf{w}^T\mathbf{X}^T\mathbf{X}\mathbf{w} + \lambda\mathbf{w}^T\mathbf{w}\right)$$

$$= \arg\min_{\mathbf{w}} \left(\mathbf{y}^T\mathbf{y} - 2\mathbf{y}^T\mathbf{X}\mathbf{w} + \mathbf{w}^T\mathbf{X}^T\mathbf{X}\mathbf{w} + \lambda\mathbf{w}^T\mathbf{w}\right)$$

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

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

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

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:

$$\hat{w} = \arg\min_{w} \|y - xw\|^2 + \lambda w^2$$

$$\frac{d}{dw}\left(\|y - xw\|^2 + \lambda w^2\right) = 0$$

$$-2x^T(y - xw) + 2\lambda w = 0$$

$$2x^Ty - 2x^Txw + 2\lambda w = 0$$

$$w(2x^Tx + 2\lambda) = 2x^Ty$$

$$w = \frac{2x^Ty}{2x^Tx + 2\lambda} = \frac{x^Ty}{x^Tx + \lambda}$$

$$\frac{x^Ty}{x^Tx + \lambda} = 0$$

$$x^T y = 0$$