

Policies

- Due 9 PM, January 23th, via Moodle.
- You are free to collaborate on all of the problems, subject to the collaboration policy stated in the syllabus.
- You should submit all code used in the homework. We ask that you use Python 3 and sklearn version 0.19 for your code, and that you comment your code such that the TAs can follow along and run it without any issues.

Submission Instructions

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

1 Comparing Different Loss Functions [30 Points]

Relevant materials: lecture 3

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

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

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

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

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

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

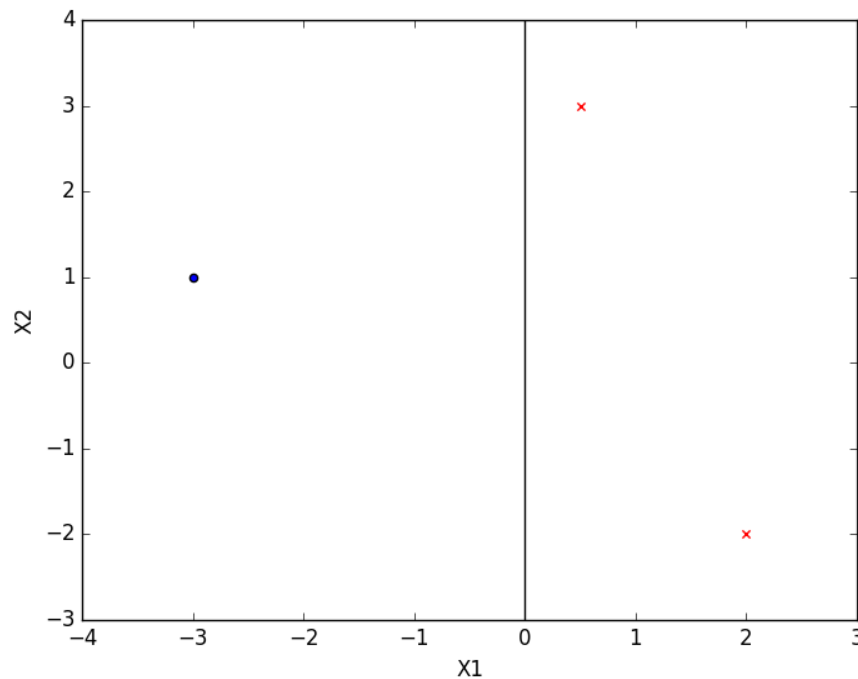
On this dataset, train both a logistic regression model and a ridge regression model to classify the points. (In other words, on each dataset, train one linear classifier using L_{log} as the loss, and another linear classifier using L_{squared} as the loss.) For this problem, you should use the logistic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

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 $\nabla_w L_{\text{log}}$ 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.

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?

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} .)

2 Effects of Regularization

Relevant materials: Lecture 4

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

Problem A [4 points]: In order to prevent over-fitting in the least-squares linear regression problem, we add a regularization penalty term. Can adding the penalty term decrease the training (in-sample) error? Will adding a penalty term always decrease the out-of-sample errors? Please justify your answers. Think about the case when there is over-fitting while training the model.

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?

Implementation of ℓ_2 regularization:

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

Download the data for training and testing. There are two training sets, `wine_training1.txt` (100 data points) and `wine_training2.txt` (a proper subset of `wine_training1.txt` containing only 40 data points), and one test set, `wine_testing.txt` (30 data points). You will use the `wine_testing.txt` dataset to evaluate your models.

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

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

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

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

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

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

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

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

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

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

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

You may run into numerical instability issues (overflow or underflow). One way to deal with these issues is by normalizing the input data X . Given the column for the j th feature, $X_{:,j}$, you can normalize it by setting $X_{ij} = \frac{X_{ij} - \bar{X}_{:,j}}{\sigma(X_{:,j})}$ where $\sigma(X_{:,j})$ is the standard deviation of the j th column's entries, and $\bar{X}_{:,j}$ is the mean of the j th 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 λ s.
- ii. Plot the average test error (E_{out}) versus different λ s using `wine_testing.txt` as the test set.
- iii. Plot the ℓ_2 norm of \mathbf{w} versus different λ s.

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.

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.

Problem E [4 points]: Briefly explain the qualitative behavior (i.e. over-fitting and under-fitting) of the

training and test errors with different λ s while training with data in wine_training1.txt.

Problem F [4 points]: Briefly explain the qualitative behavior of the ℓ_2 norm of \mathbf{w} with different λ s while training with the data in wine_training1.txt.

Problem G [4 points]: If the model were trained with wine_training2.txt, which λ would you choose to train your final model? Why?

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, \dots, 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, \dots, w_9 (ignore the bias/intercept) as a function of α .
- ii. Repeat i. with Ridge regression, and this time using regularization strengths $\alpha \in \{1, 2, 3, \dots, 1e4\}$.
- 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?

Problem B [18 points]:

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

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

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

This is linear regression with Lasso regularization.

- ii. In this question, we continue to consider Lasso regularization in 1-dimension. Now, suppose that when $\lambda = 0$, $w \neq 0$. Does there exist a value for λ such that $w = 0$? If so, what is the smallest such value?
- iii. Given a dataset containing N datapoints each with d features, solve for

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

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

This is linear regression with Ridge regularization.

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