Homework #4

RELEASE DATE: 11/01/2023

RED CORRECTION: 11/06/2023 05:30

BLUE CORRECTION: 11/11/2023 15:00

DUE DATE: 11/15/2023, BEFORE 13:00 on GRADESCOPE

QUESTIONS ARE WELCOMED ON DISCORD (INFORMALLY) OR NTU COOL (FORMALLY).

You will use Gradescope to upload your scanned/printed solutions. For problems marked with (*), please follow the guidelines on the course website and upload your source code to Gradescope as well. Any programming language/platform is allowed.

Any form of cheating, lying, or plagiarism will not be tolerated. Students can get zero scores and/or fail the class and/or be kicked out of school and/or receive other punishments for those kinds of misconducts.

Discussions on course materials and homework solutions are encouraged. But you should write the final solutions alone and understand them fully. Books, notes, and Internet resources can be consulted, but not copied from.

Since everyone needs to write the final solutions alone, there is absolutely no need to lend your homework solutions and/or source codes to your classmates at any time. In order to maximize the level of fairness in this class, lending and borrowing homework solutions are both regarded as dishonest behaviors and will be punished according to the honesty policy.

You should write your solutions in English or Chinese with the common math notations introduced in class or in the problems. We do not accept solutions written in any other languages.

This homework set comes with 240 points and 20 bonus points. In general, every homework set would come with a full credit of 240 points, with some possible bonus points.

Beyond Binary Linear Classification

1. (20 points) If some algorithm always takes a total CPU time of aN^3 for training a binary classifier on a size-N binary classification data set. Consider a size-N K-class classification data set where each class is of size N/K. What is the total CPU time needed for training a K-class classifier via one-versus-one decomposition on the data set (ignoring the minor time needed for re-labeling the data set for the sub-problems)? List your derivation steps.

(Note: This result tells you that one-versus-one may actually be computationally "cheap" because each sub-problem has fewer data.)

2. (20 points) Consider the following matrix, which is called the Vandermonde matrix.

$$V = \begin{bmatrix} 1 & x_1 & x_1^2 & \ldots & x_1^{N-1} \\ 1 & x_2 & x_2^2 & \ldots & x_2^{N-1} \\ \vdots & \vdots & \vdots & \ldots & \vdots \\ 1 & x_N & x_N^2 & \ldots & x_N^{N-1} \end{bmatrix}$$

An N by N Vandermonde matrix has a determinant of

$$\det(V) = \prod_{1 \le n < m \le N} (x_m - x_n)$$

and is thus invertible if all $\{x_n\}_{n=1}^N$ are different.

Consider some one-dimensional data $\{(x_n, y_n)\}_{n=1}^N$ where $x_n \in \mathbb{R}$ and $y_n \in \mathbb{R}$. Assume that all $\{x_n\}_{n=1}^N$ are different. Obtain a hypothesis $g(x) = \tilde{\mathbf{w}}^T \mathbf{\Phi}_Q(x)$ by applying a Q-th order polynomial transform $\mathbf{z}_n = \mathbf{\Phi}_Q(x_n)$, and running linear regression on $\{(\mathbf{z}_n, y_n)\}_{n=1}^N$ to get some $\tilde{\mathbf{w}}$. Use the property of the Vandermonde matrix above to prove that there exists some Q such that $E_{\text{in}}(g) = 0$ when E_{in} is measured by the squared error.

(Note: This result tells you that if you transform the data with enough "power" within the polynomials, you can always fit the training data perfectly.)

3. (20 points) Assume that a transformer (no, not chat-Generative-Pretrained-Transformer!) peeks some one-dimensional examples and decides the following transform $\mathbf{\Phi}$ "intelligently" from the data of size N. The transform maps $x \in \mathbb{R}$ to $\mathbf{z} = (z_1, z_2, \ldots, z_N) \in \mathbb{R}^N$, where

$$(\mathbf{\Phi}(x))_n = z_n = [\![x = x_n]\!].$$

Assume that each training and testing example is generated i.i.d. from a joint distribution $p(x, y)$ where x is sampled uniformly from $[-1, 1]$ and $y = x + \epsilon$, where ϵ is independently sampled from a Gaussian distribution with mean 0 and variance 1. For simplicity, you can assume that all x_n are different in the training data set. Consider a learning algorithm that performs linear regression after the feature transform (for simplicity, please exclude $z_0 = 1$) to get a $g(x) = \tilde{\mathbf{w}}^T \mathbf{\Phi}(x)$. Consider the squared error. What is $E_{\text{in}}(g)$? What is $E_{\text{out}}(g)$? List your derivation steps.

(Note: This result tells you that "snooping" your data too much can be a bad idea.)

Combatting Overfitting

4. (20 points) On page 20 of Lecture 13, we discussed about adding "virtual examples" (hints) to help combat overfitting. One way of generating virtual examples is to add a small noise to the input vector $\mathbf{x} \in \mathbb{R}^{d+1}$ (including the 0-th component x_0) For each $(\mathbf{x}_1, y_1), (\mathbf{x}_2, y_2), \ldots, (\mathbf{x}_N, y_N)$ in our training data set, assume that we generate virtual examples $(\tilde{\mathbf{x}}_1, y_1), (\tilde{\mathbf{x}}_2, y_2), \ldots, (\tilde{\mathbf{x}}_N, y_N)$ where $\tilde{\mathbf{x}}_n$ is simply $\mathbf{x}_n + \boldsymbol{\epsilon}$ and each component of the noise vector $\boldsymbol{\epsilon} \in \mathbb{R}^{d+1}$ is generated i.i.d. from a uniform distribution within $[-\delta, \delta]$. The vector $\boldsymbol{\epsilon}$ is a random vector that varies for each virtual example.

Recall that when training the linear regression model, we need to calculate $\mathrm{X}^T\mathrm{X}$ first. Define the hinted input matrix

$$\mathrm{X}_h = \begin{bmatrix} | & \ldots & | & | & \ldots & | \\ \mathbf{x}_1 & \ldots & \mathbf{x}_N & \tilde{\mathbf{x}}_1 & \ldots & \tilde{\mathbf{x}}_N \\ | & \ldots & | & | & \ldots & | \end{bmatrix}^T.$$

What is the expected value $\mathbb{E}(\mathrm{X}_h^T\mathrm{X}_h)$ as a function of X and δ, where the expectation is taken over the (uniform)-noise generating process above? Prove your result.

(Note: This result may ring a bell on how such virtual examples can act like regularizers.)

5. (20 points) Consider the augmented error

$$E_{\text{aug}}(\mathbf{w}) = E_{\text{in}}(\mathbf{w}) + \frac{\lambda}{N}\mathbf{w}^T\mathbf{w}$$

with some $\lambda > 0$. When minimizing E_{aug} with the fixed-learning rate gradient descent algorithm with a learning rate $\eta > 0$, the update rule is

$$\mathbf{w}_{t+1} \leftarrow \alpha(\mathbf{w}_t - \beta\nabla E_{\text{in}}(\mathbf{w}_t)).$$

What are α and β? Prove your result.

(Note: You should get some $\alpha < 1$, which means that the weight vector is decayed (decreased). This is why L2 regularizer is often also called the weight-decay regularizer.)

6. (20 points) Consider a one-dimensional data set $\{(x_n, y_n)\}_{n=1}^N$ where each $x_n \in \mathbb{R}$ and $y_n \in \mathbb{R}$. Then, solve the following one-variable regularized linear regression problem:

$$\min_{w \in \mathbb{R}} \frac{1}{N} \sum_{n=1}^{N} (w \cdot x_n - y_n)^2 + \frac{\lambda}{N} w^2.$$

If the optimal solution to the problem above is w^*, it can be shown that w^* is also the optimal solution of

$$\min_{w \in \mathbb{R}} \frac{1}{N} \sum_{n=1}^{N} (w \cdot x_n - y_n)^2 \text{ subject to } w^2 \leq C$$

with $C = (w^*)^2$. This allows us to express the relationship between C in the constrained optimization problem and λ in the augmented optimization problem for any $\lambda > 0$. In particular,

$$\lambda = \frac{\alpha}{\sqrt{C}} + \beta$$

What are α and β? Prove your result.

(Note: This should allow you to see how λ decreases [when $\lambda > 0$] as C increases [until some upper bound].)

7. (20 points) Scaling can affect regularization. Consider a data set $\{(\mathbf{x}_n, y_n)\}_{n=1}^N$. Define $\Phi(\mathbf{x}) = \mathrm{V}\mathbf{x}$ where V is a diagonal matrix with the i-th diagonal component storing a *positive* value to scale the i-th feature. Now, conduct L1-regularized linear regression with the transformed data $\{(\Phi(\mathbf{x}_n), y_n)\}_{n=1}^N$.

$$\min_{\tilde{\mathbf{w}} \in \mathbb{R}^{d+1}} \frac{1}{N} \sum_{n=1}^{N} (\tilde{\mathbf{w}}^T \Phi(\mathbf{x}_n) - y_n)^2 + \frac{\lambda}{N} \|\tilde{\mathbf{w}}\|_1$$

The problem is equivalent to the following regularized linear regression problem on the original data with a different regularizer.

$$\min_{\mathbf{w} \in \mathbb{R}^{d+1}} \frac{1}{N} \sum_{n=1}^{N} (\mathbf{w}^T \mathbf{x}_n - y_n)^2 + \frac{\lambda}{N} \Omega(\mathbf{w})$$

What is $\Omega(\mathbf{w})$? How do the optimal $\tilde{\mathbf{w}}$ and the optimal $\mathbf{w}$ correspond to each other? Prove your result.

(Note: The result shows you how scaling the data effectively changes the regularizer.)

8. (20 points) Consider a binary classification algorithm $\mathcal{A}_{\text{minority}}$, which returns a constant classifier that always predicts the minority class (i.e., the class with fewer instances in the data set that it sees). As you can imagine, the returned classifier is the worst-E_{in} one among all constant classifiers. Consider the 0/1 error. For a binary classification data set with N positive examples and N negative examples, what is $E_{\text{loocv}}(\mathcal{A}_{\text{minority}})$? Prove your result.

(Note: This result may tell you that in some special situations, leave-one-out cross-validation is not always trustworthy.)

Learning Principles

9. In Lecture 16, we talked about the probability to fit data perfectly when the labels are random. For instance, page 6 of Lecture 16 shows that the probability of fitting the data perfectly with decision stumps is $(2N)/2^N$. Consider five points in $\mathbb{R}^2$ as input vectors $\mathbf{x}_1 = (+1, +1)$, $\mathbf{x}_2 = (+1, -1)$, $\mathbf{x}_3 = (-1, +1)$, $\mathbf{x}_4 = (-1, -1)$, $\mathbf{x}_5 = (2, 0)$, and a 2D perceptron model that minimizes $E_{\text{in}}(\mathbf{w})$ to the lowest possible value. One way to measure the power of the model is to consider five random labels y_1, y_2, y_3, y_4, y_5, each in ± 1 and generated by i.i.d. fair coin flips, and then compute

$$\mathbb{E}_{y_1,y_2,y_3,y_4,y_5}\left(\min_{\mathbf{w}\in\mathbb{R}^{2+1}} E_{\text{in}}(\mathbf{w})\right)$$

in terms of the 0/1 error. For a perfect fitting, $\min_{\mathbf{w}} E_{\text{in}}(\mathbf{w})$ will be 0; for a less perfect fitting (when the data is not linearly separable), $\min_{\mathbf{w}} E_{\text{in}}(\mathbf{w})$ will be some non-zero value. The expectation above averages over all 32 possible combinations of y_1, y_2, y_3, y_4, y_5. What is the value of the expectation? Prove your result.

(*Note: It can be shown that* 1 *minus twice the expected value above is the same as the so-called empirical Rademacher complexity of 2D perceptrons. Rademacher complexity, similar to the VC dimension, is another tool to measure the complexity of a hypothesis set. If a hypothesis set shatters some data points, zero E_{in} can always be achieved and thus Rademacher complexity is* 1*; if a hypothesis set cannot shatter some data points, Rademacher complexity provides a soft measure of how "perfect" the hypothesis set is.*)

Experiments with Regularized Logistic Regression

Consider L2-regularized logistic regression with third-order polynomial transformation.

$$\mathbf{w}_\lambda = \text{argmin}_{\mathbf{w}} \frac{\lambda}{N}\|\mathbf{w}\|^2 + \frac{1}{N}\sum_{n=1}^{N} \ln(1 + \exp(-y_n \mathbf{w}^T \mathbf{\Phi}_3(\mathbf{x}_n))),$$

Here $\mathbf{\Phi}_3$ is the third-order polynomial transformation introduced on page 2 of Lecture 12 (with $Q = 3$), defined as

$$\mathbf{\Phi}_3(\mathbf{x}) = (1, x_1, x_2, \ldots, x_d, x_1^2, x_1x_2, \ldots, x_1x_d, x_2^2, x_2x_3, \ldots, x_2x_d, \ldots, x_d^2, x_1^3, \ldots, x_d^3)$$

Given that $d = 6$ in the following data sets, your $\Phi_3(\mathbf{x})$ should be of 84 dimensions (including the constant dimension).

Next, we will take the following file as our training data set $\mathcal{D}$:

`http://www.csie.ntu.edu.tw/~htlin/course/ml23fall/hw4/hw4_train.dat`

~~and the following file as our test data set for evaluating E_{out}:~~

~~`http://www.csie.ntu.edu.tw/~htlin/course/ml23fall/hw4/hw4_test.dat`~~

We call the algorithm for solving the problem above as $\mathcal{A}_\lambda$. The problem guides you to use LIBLINEAR (`https://www.csie.ntu.edu.tw/~cjlin/liblinear/`), a machine learning package developed in our university, to solve this problem. In addition to using the default options, what you need to do when running LIBLINEAR are

- set option `-s 0`, which corresponds to solving regularized logistic regression
- set option `-c C`, with a parameter value of `C` calculated from the λ that you want to use; read `README` of the software package to figure out how `C` and your λ should relate to each other
- set option `-e 0.000001`, which corresponds to getting a solution that is really really close to the optimal solution

LIBLINEAR can be called from the command line or from major programming languages like python. If you run LIBLINEAR in the command line, please upload the scripts that include the commands; if you run LIBLINEAR from any programming language, please include your code.

We will consider the data set as a *binary classification problem* and take the "regression for classification" approach with regularized logistic regression (see page 6 of Lecture 10). So please evaluate all errors below with the 0/1 error.

10. (20 points, *) Select the best λ^* as

$$\underset{\log_{10}\lambda \in \{-6,-4,-2,0,2\}}{\text{argmin}} E_{\text{in}}(\mathbf{w}_\lambda).$$

Break the tie, if any, by selecting the largest λ. What is $\log_{10}(\lambda^*)$?

11. (20 points, *) Now randomly split the given training examples in $\mathcal{D}$ to two sets: 120 examples as $\mathcal{D}_{\text{train}}$ and 80 as $\mathcal{D}_{\text{val}}$. Run $\mathcal{A}_\lambda$ on *only* $\mathcal{D}_{\text{train}}$ to get $\mathbf{w}_\lambda^-$ (the weight vector within the g^- returned), and validate $\mathbf{w}_\lambda^-$ with $\mathcal{D}_{\text{val}}$ to get $E_{\text{val}}(\mathbf{w}_\lambda^-)$. Select the best λ^* as

$$\underset{\log_{10}\lambda \in \{-6,-4,-2,0,2\}}{\text{argmin}} E_{\text{val}}(\mathbf{w}_\lambda^-).$$

Break the tie, if any, by selecting the largest λ. Repeat the experiment above for 128 times, each with a different random split. Plot a histogram on the distribution of $\log_{10}(\lambda^*)$ selected from the 128 experiments.

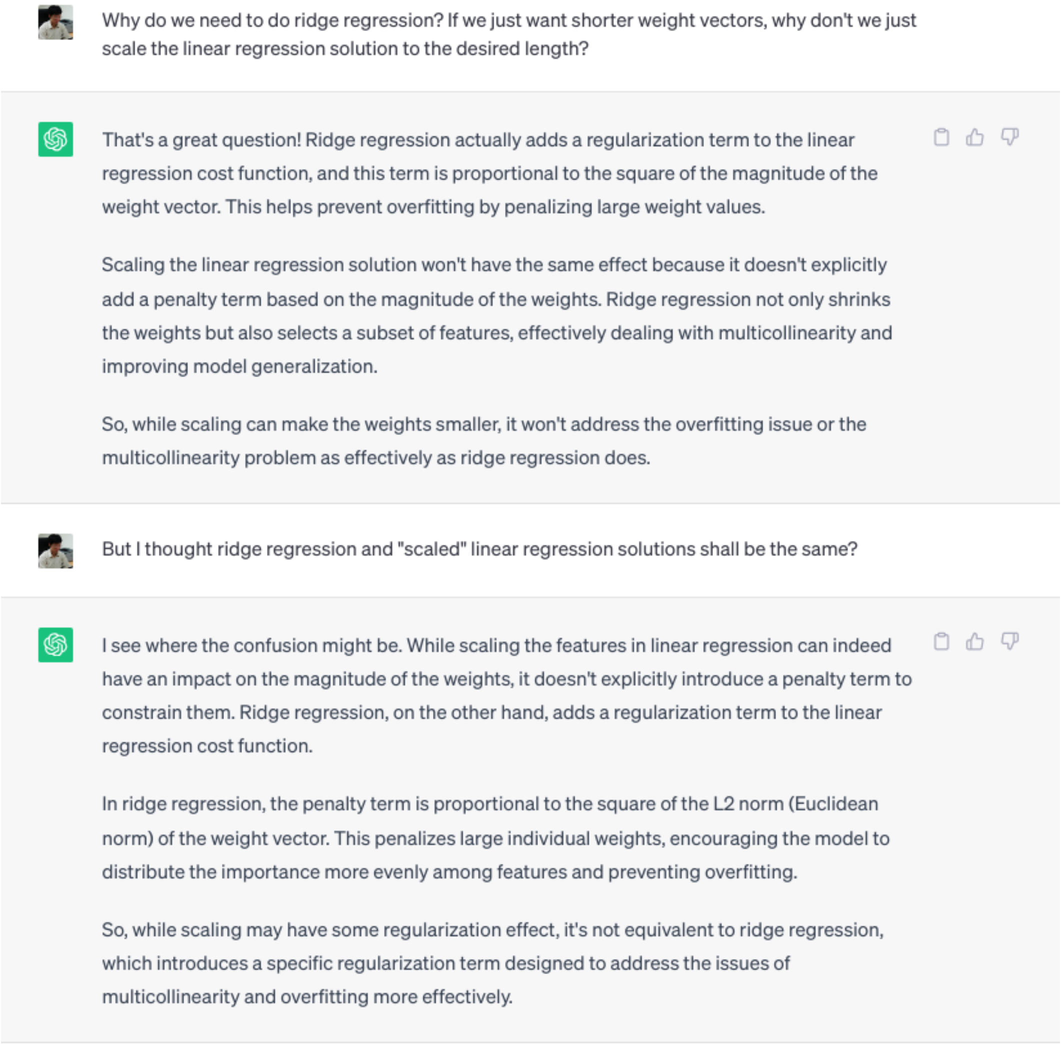
12. (20 points, *) Now randomly split the given training examples in $\mathcal{D}$ to five folds, 40 being fold 1, another 40 being fold 2, and so on. Select the best λ^* as

$$\underset{\log_{10}\lambda \in \{-6,-4,-2,0,2\}}{\text{argmin}} E_{\text{CV}}(\mathcal{A}_\lambda).$$

Break the tie, if any, by selecting the largest λ. Repeat the experiment above for 128 times, each with a different random split. Plot a histogram on the distribution of $\log_{10}(\lambda^*)$ selected from the 128 experiments. Compare your result with the $\log_{10}(\lambda^*)$ selected for the two problems above. Describe your findings.

Bonus: Scale or Regularize?

13. (Bonus 20 points) Dr. Regularize recently learned regularization and thought that its basic goal is to restrict the length of the weight vector $\mathbf{w}$ to be less than $\sqrt{C}$. Ze then designed a "new" regularization algorithm—simply performing linear regression first to get some $\mathbf{w}_{\text{LIN}}$, and then get $\mathbf{w}_C = \frac{\mathbf{w}_{\text{LIN}}}{\|\mathbf{w}_{\text{LIN}}\|} \cdot \sqrt{C}$. Then, $\mathbf{w}_C$ would be of length $\sqrt{C}$ only. Ze then asks chatGPT whether this is equivalent to the C-constrained regularization (and hence equivalent to λ-penalized L2 regularization) that ze learned in class, and got the following answer.



Why do we need to do ridge regression? If we just want shorter weight vectors, why don't we just scale the linear regression solution to the desired length?

That's a great question! Ridge regression actually adds a regularization term to the linear regression cost function, and this term is proportional to the square of the magnitude of the weight vector. This helps prevent overfitting by penalizing large weight values.

Scaling the linear regression solution won't have the same effect because it doesn't explicitly add a penalty term based on the magnitude of the weights. Ridge regression not only shrinks the weights but also selects a subset of features, effectively dealing with multicollinearity and improving model generalization.

So, while scaling can make the weights smaller, it won't address the overfitting issue or the multicollinearity problem as effectively as ridge regression does.

But I thought ridge regression and "scaled" linear regression solutions shall be the same?

I see where the confusion might be. While scaling the features in linear regression can indeed have an impact on the magnitude of the weights, it doesn't explicitly introduce a penalty term to constrain them. Ridge regression, on the other hand, adds a regularization term to the linear regression cost function.

In ridge regression, the penalty term is proportional to the square of the L2 norm (Euclidean norm) of the weight vector. This penalizes large individual weights, encouraging the model to distribute the importance more evenly among features and preventing overfitting.

So, while scaling may have some regularization effect, it's not equivalent to ridge regression, which introduces a specific regularization term designed to address the issues of multicollinearity and overfitting more effectively.

After reading the answer from chatGPT, ze still does not understand why scaling after linear regression is different from the C-constrained linear regression. Please help ze understand chatGPT's answer by proving that if $\mathbf{w}_{\text{LIN}} \neq \mathbf{0}$ and $\|\mathbf{w}_{\text{LIN}}\|^2 > C$, then $\mathbf{w}_C$ solves the C-constrained linear regression problem

$$\min_{\mathbf{w}} E_{\text{in}}(\mathbf{w}) \text{ subject to } \mathbf{w}^T\mathbf{w} \leq C.$$

if and only if $\mathrm{X}^T\mathrm{X} = \alpha\mathrm{I}$, meaning that there is *no* dependence between the features.

(Hint: The "no dependence" condition tells you that regularized regression (i.e. ridge regression) handles the dependence cases better, which corresponds to the "multicollinearity" issue mentioned by chatGPT.)