EECS 189 Introduction to Machine Learning
Fall 2020 HW7

This homework is due **Wednesday, October 21 at 11:59 p.m.**

1 Getting Started

Read through this page carefully. You may typeset your homework in latex or submit neatly handwritten/scanned solutions. Please start each question on a new page. Deliverables:

1. Submit a PDF of your writeup, **with an appendix for your code**, to the appropriate assignment on Gradescope. If there are graphs, include those graphs in the correct sections. Do not simply reference your appendix.
2. If there is code, submit all code needed to reproduce your results.
3. If there is a test set, submit your test set evaluation results.

After you've submitted your homework, watch out for the self-grade form.

(a) Who else did you work with on this homework? In case of course events, just describe the group. How did you work on this homework? Any comments about the homework?

(b) Please copy the following statement and sign next to it. We just want to make it *extra* clear so that no one inadvertently cheats.

I certify that all solutions are entirely in my words and that I have not looked at another student's solutions nor have I looked at any online solutions to any of these problems. I have credited all external sources in this write up.

The first part of this homework is just a redo of the midterm. In a sense, this is a short part of the assignment since everything is intended to be completable in essentially 2 hours and you've already spent 2 hours on it. However, we want to encourage you to take a deep breath and redo the problems or at least check your existing work carefully to make sure that you actually understand what is going on. Talk to your group. Use Piazza to ask questions. Get everything right before having to look at the solutions.

2 Regularization from the Augmentation Perspective (10 points)

Assume $\mathbf{w}$ is a d-dimensional Gaussian random vector $\mathbf{w} \sim \mathcal{N}(\mathbf{0}, \mathbf{\Sigma})$ and $\mathbf{\Sigma}$ is symmetric positive-definite. Our model for how the $\{y_i\}$ training data is generated is

$$y = \mathbf{w}^\top \mathbf{x} + Z, \quad Z \sim \mathcal{N}(0, 1), \tag{1}$$

where the noise variables Z are independent of $\mathbf{w}$ and iid across training samples. Notice that all the training $\{y_i\}$ and the parameters $\mathbf{w}$ are jointly normal/Gaussian random variables conditioned on the training inputs $\{\mathbf{x}_i\}$. Let us define the standard data matrix and measurement vector:

$$\mathbf{X} = \begin{bmatrix} \mathbf{x}_1^T \\ \mathbf{x}_2^T \\ \vdots \\ \mathbf{x}_n^T \end{bmatrix}, \quad \mathbf{y} = \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_n \end{bmatrix}.$$

In this model, the MAP estimate of $\mathbf{w}$ is given by the Tikhonov regularization counterpart of ridge regression:

$$\widehat{\mathbf{w}} = (\mathbf{X}^\top \mathbf{X} + \mathbf{\Sigma}^{-1})^{-1} \mathbf{X}^\top \mathbf{y}, \tag{2}$$

In this question, we explore Tikhonov regularization from the data augmentation perspective.

Define the matrix $\mathbf{\Gamma}$ as a $d \times d$ matrix that satisfies $\mathbf{\Gamma}^\top \mathbf{\Gamma} = \mathbf{\Sigma}^{-1}$. Consider the following augmented design matrix (data) $\hat{\mathbf{X}}$ and augmented measurement vector $\hat{\mathbf{y}}$:

$$\hat{\mathbf{X}} = \begin{bmatrix} \mathbf{X} \\ \mathbf{\Gamma} \end{bmatrix} \in \mathbb{R}^{(n+d) \times d}, \quad \text{and} \quad \hat{\mathbf{y}} = \begin{bmatrix} \mathbf{y} \\ \mathbf{0}_d \end{bmatrix} \in \mathbb{R}^{n+d},$$

where $\mathbf{0}_d$ is the zero vector in $\mathbb{R}^d$. **Show that the ordinary least squares problem**

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

has the same solution as (2).

(HINT: Feel free to just use the formula you know for the OLS solution. You don't have to rederive that. This problem is not intended to be hard or time consuming.)

3 What is going on? (12 points)

This question relies on your understanding of the behavior of ridge regression and kernel ridge regression as the hyperparameters change. Part (a) uses a piecewise linear function which you have already seen in homework. Parts (b) and (c) show the results of (kernel) ridge regression using heatmaps which you should already be familiar with from homework. For parts (b) and (c) we used the circles dataset which you have already seen, with an 80% train/ 20% test split illustrated in Figure 1 below.

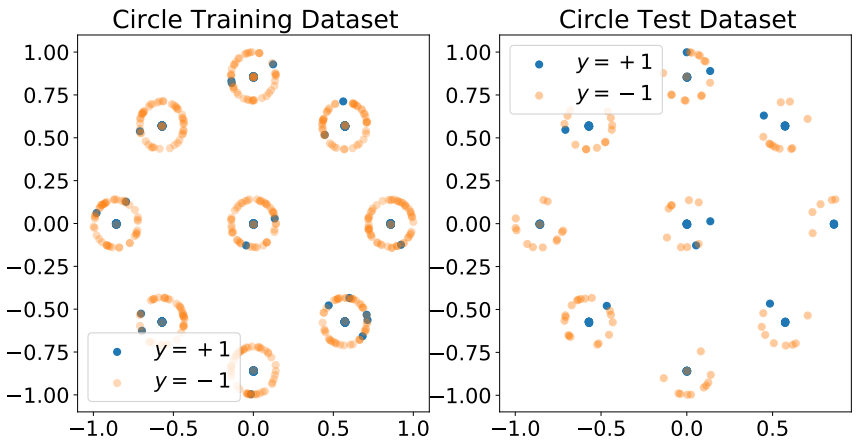


Figure 1: Training and test datasets, with partial transparency to show overlapping datapoints. Notice that there are "mislabeled" points in this dataset.

(a) (6 pts) In this part we have performed kernel ridge regression with the RBF kernel

$$k(x, x') = \exp\left(-\gamma \|x - x'\|^2\right)$$

on a piecewise linear 1D function which you should recognize from homework. Both the regularization parameter λ and the smoothing parameter γ have been varied in the plots on the next page. Your goal in this problem is to identify the qualitative level of the γ and λ parameters given the visible plots of what was learned.

Determine whether the regression was over- or under-regularized and whether the smoothing factor was too wide (small γ) or too narrow (large γ), and write the letter of each subfigure from Figure 2 in the appropriate location on your answer sheet following Figure 3. We have provided the location of (b) to get you started. Not all the marked spots will be used and each spot will correspond to at most one letter.

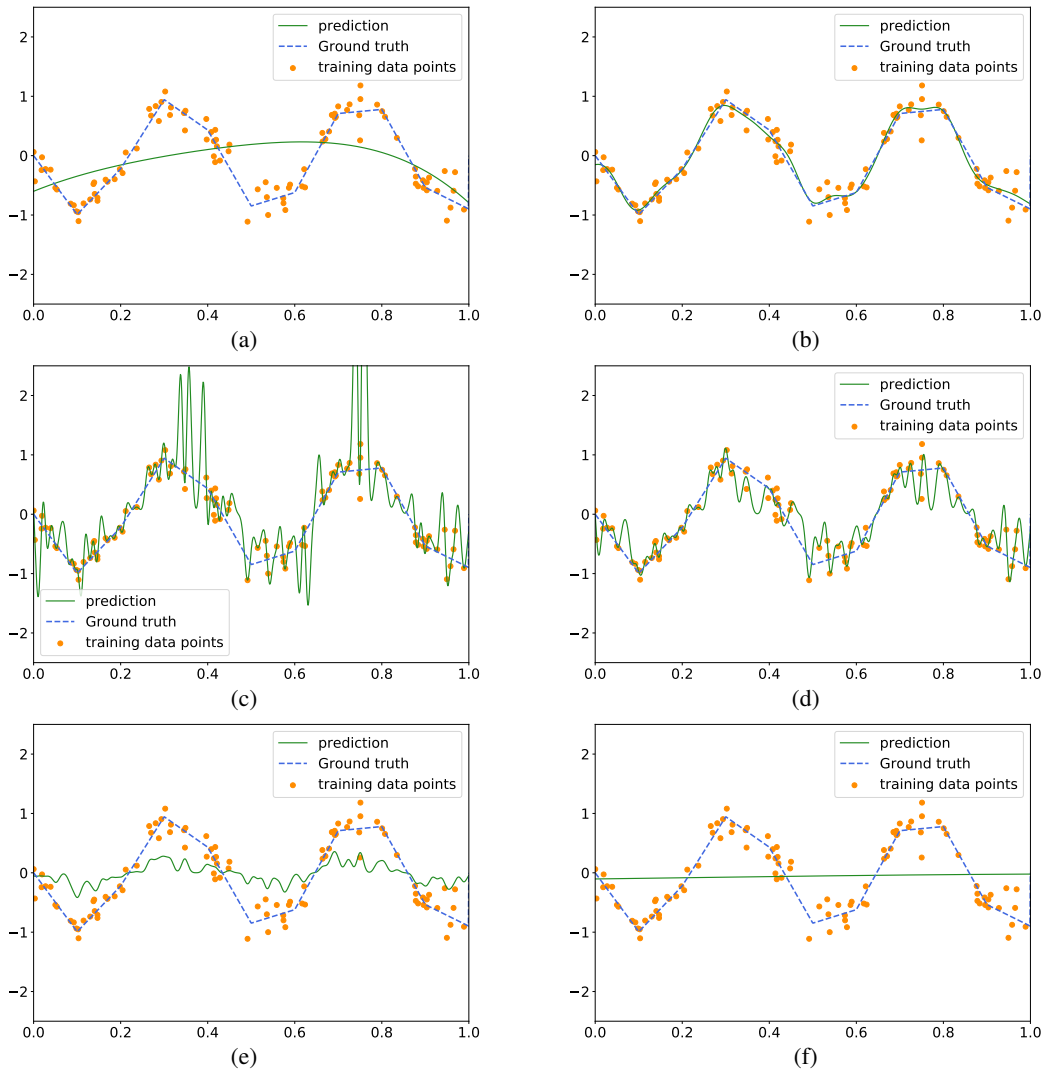


Figure 2: Match these subfigures to the (γ, λ) hyper-parameter space.

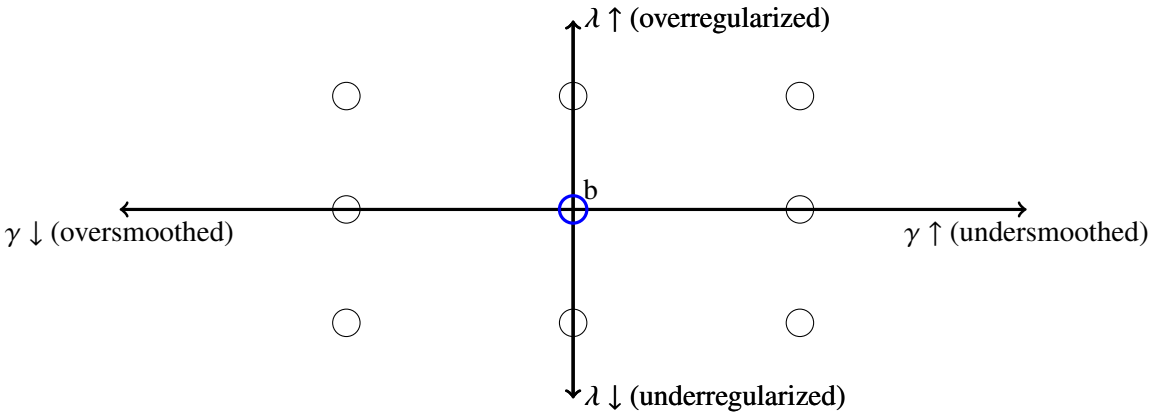
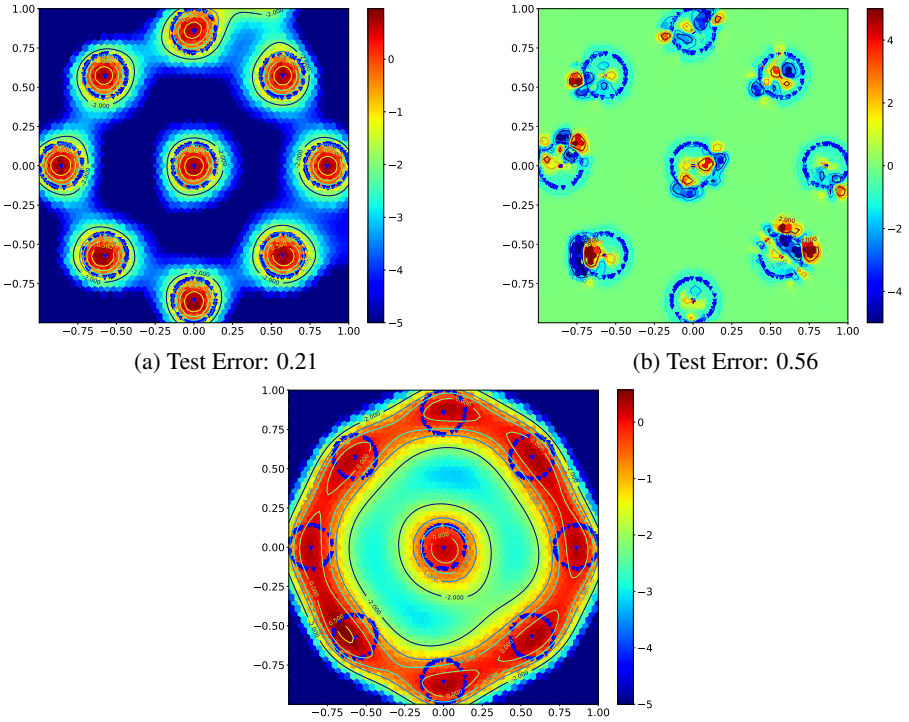


Figure 3: **Write subfigure letters from Figure 2 in the appropriate location on your answer sheet.**

(b) (3 pts) In this part we have performed kernel ridge regression with the RBF kernel $\exp\left(-\frac{\|x-x'\|^2}{2\sigma^2}\right)$ and $\lambda = 0.001$. **Match each subfigure (a), (b), and (c) of Figure 4 with the appropriate location (1), (2), or (3) on the test error curve of Figure 5.** For example, write: $(a) \rightarrow (1)$; $(b) \rightarrow (2)$; $(c) \rightarrow (3)$.



(a) Test Error: 0.21

(b) Test Error: 0.56

(c) Test Error: 0.58

Figure 4: Heatmaps corresponding to the functions learned by kernel ridge regression. Match these subfigures to the test error curve below. Don't forget to use the test-error information.

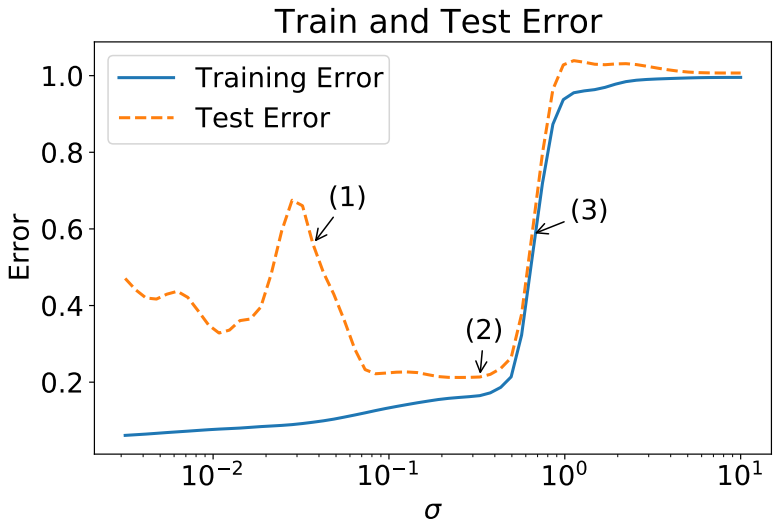
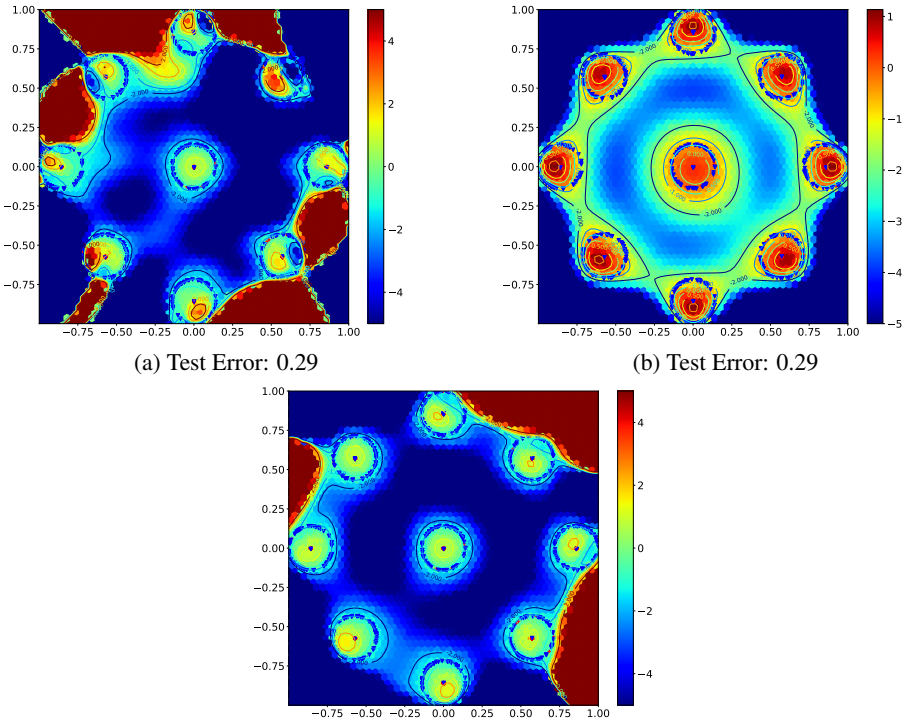


Figure 5: Train and test errors for varying RBF kernel parameter σ.

(c) (3 pts) In this part we have performed ridge regression on the circles dataset with polynomial features of increasing degree D. After examining Figure 6, **match each subfigure (a), (b), and (c) with the appropriate location (1), (2), or (3) on the train/test error plot in Figure 7.** For example, write: $(a) \rightarrow (1); (b) \rightarrow (2); (c) \rightarrow (3)$.



(a) Test Error: 0.29

(b) Test Error: 0.29

(c) Test Error: 0.22

Figure 6: Heatmaps corresponding to the functions learned by ridge regression. Match these subfigures to the test error curve below. Don't forget to use the test-error information.

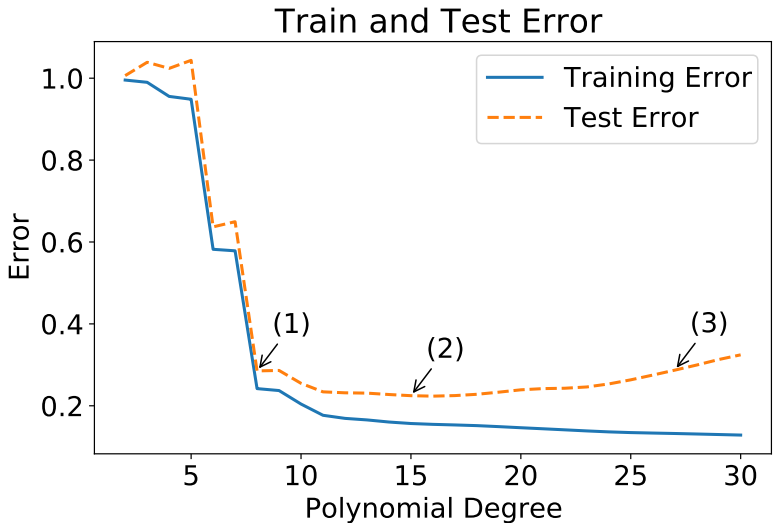


Figure 7: Train and test errors for increasing polynomial degrees.

4 Validation can sometimes fail (40 points)

This problem is a simplified setting where we investigate the probability that validation ends up picking a more complicated false model instead of the simpler true model. Consider the setting of learning a one-dimensional function from noisy samples.

In this problem, suppose the true underlying function is the constant zero function. We have access to noisy training data consisting of n_t data points that we arrange into a pair of n_t-dimensional vectors $(\mathbf{x}_t, \mathbf{y}_t)$. Because the underlying function is the zero function, we have:

$$\mathbf{y}_t = \boldsymbol{\epsilon}_t,$$

where $\boldsymbol{\epsilon}_t \sim \mathcal{N}(0, \sigma^2 \mathbf{I}_{n_t})$. Further we have access to similarly noisy validation data that has been arranged into a pair of n_v-dimensional vectors $(\mathbf{x}_v, \mathbf{y}_v)$ where

$$\mathbf{y}_v = \boldsymbol{\epsilon}_v,$$

and $\boldsymbol{\epsilon}_v \sim \mathcal{N}(0, \sigma^2 \mathbf{I}_{n_v})$. We assume $\boldsymbol{\epsilon}_v$ is independent of $\boldsymbol{\epsilon}_t$.

For ease of computation, assume

$$\|\mathbf{x}_t\|_2^2 = n_t, \qquad \|\mathbf{x}_v\|_2^2 = n_v.$$

We use the training data to learn models and then use the validation data to choose the better performing one.

In this simplified case, the 0-th model is just the constant zero. There are no parameters to learn for this model.

The 1-st model is a simple linear function $f(x) = wx$ where we need to learn the single parameter w from the training data.

For this problem, the various parts are largely independent of each other even though they collectively tell one story. Since this is an exam, don't get bogged down in any one part and lose easy points elsewhere.

(a) (4 pts) Suppose that we just decided to perform OLS on the *validation data* and computed

$$\widehat{w}_v = \arg\min_{w \in \mathbb{R}} \left\| \mathbf{y}_v - \mathbf{x}_v w \right\|_2^2.$$

Just use the formula for OLS to show that

$$\widehat{w}_v = \frac{1}{n_v} \mathbf{x}_v^\top \boldsymbol{\epsilon}_v.$$

and give the mean and variance of the Gaussian random variable $\widehat{w}_v$.

Here, we are viewing this hypothetical $\widehat{w}_v$ as a random variable because the noise in the validation data is random.

(b) (4 pts) By doing a calculation analogous to the previous part, you can understand the actually learned parameter $\widehat{w}_t$ obtained by performing OLS on the training data. This gives

$$\widehat{w}_t = \frac{1}{n_t}\mathbf{x}_t^\top \boldsymbol{\epsilon}_t.$$

and this has a Gaussian distribution $\widehat{w}_t \sim \mathcal{N}(0, \frac{\sigma^2}{n_t})$. Here, the randomness in $\widehat{w}_t$ comes from the random noise in the training data.

You now have the marginal distributions for $\widehat{w}_t$ and $\widehat{w}_v$. **Find the joint distribution of $\widehat{w}_t$ and $\widehat{w}_v$**, i.e. find the distribution of

$$\mathbf{w} = \begin{bmatrix} \widehat{w}_t \\ \widehat{w}_v \end{bmatrix}.$$

(c) (12 pts) Suppose we have the following two choices for the predictor:
Choice 0: $\widehat{f}(x) = 0$.
Choice 1: $\widehat{f}(x) = x\widehat{w}_t$.
We use the validation data to determine which choice to use by computing the validation errors:

$$\mathcal{E}_0^{\text{val}} = \left\|\mathbf{y}_v\right\|_2^2,$$
$$\mathcal{E}_1^{\text{val}} = \left\|\mathbf{y}_v - \mathbf{x}_v\widehat{w}_t\right\|_2^2.$$

If $\mathcal{E}_0^{\text{val}} \leq \mathcal{E}_1^{\text{val}}$, we correctly choose the zero predictor.
If $\mathcal{E}_0^{\text{val}} > \mathcal{E}_1^{\text{val}}$, we choose the linear model (Choice 1).

Since the true function is actually zero, the probability that validation fails is given by:

$$P(\text{Validation fails}) = P(\mathcal{E}_1^{\text{val}} < \mathcal{E}_0^{\text{val}}).$$

Show that:

$$P(\mathcal{E}_1^{\mathbf{val}} < \mathcal{E}_0^{\mathbf{val}}) = P\left((\widehat{w}_v - \widehat{w}_t)^2 < \widehat{w}_v^2\right).$$

(Hint 1: Start expanding $\mathcal{E}_1^{val} - \mathcal{E}_0^{val}$.
Hint 2: At some point, you'll need to expand out squares $\|\mathbf{v}\|_2^2 = \mathbf{v}^T\mathbf{v}$.
Hint 3: At an opportune point, use the trick

$$\mathbf{y}_v = \mathbf{y}_v - \mathbf{x}_v\widehat{w}_v + \mathbf{x}_v\widehat{w}_v.$$

Hint 4: Remember that the orthogonality principle in ordinary least squares tells us that the residual vector $\mathbf{y}_v - \mathbf{x}_v\widehat{w}_v$ is orthogonal to the vector $\mathbf{x}_v$.)

(d) (4 pts) Simple algebra takes the previous result $P(\mathcal{E}_1^{\text{val}} < \mathcal{E}_0^{\text{val}}) = P\left((\widehat{w}_v - \widehat{w}_t)^2 < \widehat{w}_v^2\right)$ and simplifies it to

$$P(\mathcal{E}_1^{\text{val}} < \mathcal{E}_0^{\text{val}}) = P\left(\widehat{w}_t(\widehat{w}_t - 2\widehat{w}_v) < 0\right).$$

This lets us illustrate the realizations of $\widehat{w}_t, \widehat{w}_v$ for which validation fails as the shaded region in Fig. 8:

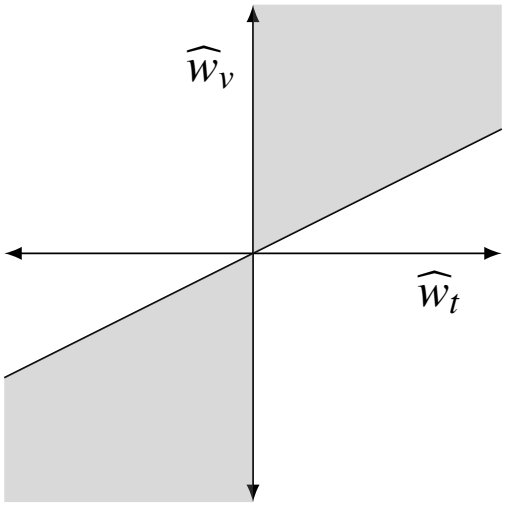


Figure 8: Illustrating the region where validation makes the wrong choice.

Now suppose we had some particular value of $\widehat{w}_t > 0$. **Draw a one-dimensional region corresponding to values of $\widehat{w}_v$ for which validation fails.**

Your drawing should have a 1d real line with both zero and $\widehat{w}_t$ marked, and a shaded region that corresponds to those values of $\widehat{w}_v$ for which validation would make the wrong choice. Be sure to label the boundary of the region.

(e) (12 pts) Use the illustration in Fig. 8 and the fact that $P(\mathcal{E}_1^{\text{val}} < \mathcal{E}_0^{\text{val}}) = P\left(\widehat{w}_t(\widehat{w}_t - 2\widehat{w}_v) < 0\right)$ to **show that the probability validation fails is given by**

$$P(\mathcal{E}_1^{\textbf{val}} < \mathcal{E}_0^{\textbf{val}}) = \frac{1}{\pi}\cos^{-1}\left(\frac{1}{\sqrt{1 + \frac{4n_t}{n_v}}}\right). \tag{3}$$

(Hint: Recall from Homework 6, Question 4 that if $\mathbf{X} = \begin{bmatrix} X_1 \\ X_2 \end{bmatrix} \sim \mathcal{N}(0, \Sigma)$ *where* Σ *is a positive definite*

$$\Sigma = \begin{bmatrix} a & c \\ c & b \end{bmatrix},$$

then

$$P(X_1 > 0, X_2 > 0) = \frac{1}{2\pi} \cos^{-1}\left(\frac{-c}{\sqrt{ab}}\right).$$

and ask yourself what the relevant 2d Gaussian random variables are here that you need to think about to understand the probability of the event we want to understand.)

(f) (4 pts) Use what you proved in the previous part, $P(\mathcal{E}_1^{\text{val}} < \mathcal{E}_0^{\text{val}}) = \frac{1}{\pi}\cos^{-1}\left(\frac{1}{\sqrt{1+\frac{4n_t}{n_v}}}\right)$, to

evaluate $P(\mathcal{E}_1^{\text{val}} < \mathcal{E}_0^{\text{val}})$ when $n_t = 50, n_v \to \infty$ as well as
evaluate $P(\mathcal{E}_1^{\text{val}} < \mathcal{E}_0^{\text{val}})$ when $n_t \to \infty, n_v = 50$.

Feel free to use the facts that $\cos^{-1}(1) = 0$ and $\cos^{-1}(0) = \frac{\pi}{2}$.

5 Kernel Controversy (42 points)

Your friend rushes up to you with an excited tone. They say that they've got a more flexible way of understanding Kernel Ridge Regression:

"At the end of the day, we represent the learned function using the following representation:

$$\widehat{f}(\mathbf{x}) = \sum_{i=1}^{n} \beta[i] k(\mathbf{x}_i, \mathbf{x}) \tag{4}$$

where $\mathbf{x}_i$ are the training inputs. If we take that as a given, all we are doing is giving ourselves a new set of n features. I can just think about the i-th feature as being $\phi_i(\mathbf{x}) = k(\mathbf{x}_i, \mathbf{x})$. Instead of doing something special like kernel ridge regression, we can just use our $\{\phi_i(\cdot)\}$ features and do ordinary ridge regression to get our weights $\boldsymbol{\beta}$. If we think about things this way, we don't even have to worry about things like positive-semi-definiteness and anything like that. It just works."

This problem is about exploring your friend's idea. You'll do traditional kernel ridge regression first. Throughout this problem you will use the linear kernel only.

(a) (10 pts) Suppose we have training pairs $\{(\mathbf{x}_i, y_i)\}$ and arrange the input data into the standard data matrix $\mathbf{X} = \begin{bmatrix} \mathbf{x}_1^T \\ \mathbf{x}_2^T \\ \vdots \\ \mathbf{x}_n^T \end{bmatrix}$ and the training outputs into a vector $\mathbf{y} = \begin{bmatrix} y_1 \\ \vdots \\ y_n \end{bmatrix}$.

Consider the "linear kernel" $k(\mathbf{x}, \mathbf{z}) = \mathbf{x}^T\mathbf{z}$. For kernel ridge regression, we have the standard way of learning weights

$$\widehat{\boldsymbol{\beta}} = (\mathbf{K} + \lambda \mathbf{I})^{-1}\mathbf{y} \tag{5}$$

where the symmetric Kernel Gram matrix $\mathbf{K}$ is defined by

$$\mathbf{K} = \begin{bmatrix} k(\mathbf{x}_1, \mathbf{x}_1) & k(\mathbf{x}_1, \mathbf{x}_2) & \cdots & k(\mathbf{x}_1, \mathbf{x}_n) \\ k(\mathbf{x}_2, \mathbf{x}_1) & k(\mathbf{x}_2, \mathbf{x}_2) & \cdots & k(\mathbf{x}_2, \mathbf{x}_n) \\ \vdots & \ddots & \ddots & \vdots \\ k(\mathbf{x}_n, \mathbf{x}_1) & k(\mathbf{x}_n, \mathbf{x}_2) & \cdots & k(\mathbf{x}_n, \mathbf{x}_n) \end{bmatrix}. \tag{6}$$

If we use kernel ridge regression with the linear kernel, **give an expression for what the final learned function $\widehat{f}(\mathbf{x})$ would be if we followed (5) to learn weights for use in (4).**

Please give your expression in the matrix form $\widehat{y}_{\mathbf{test}} = \mathbf{x}_{\mathbf{test}}^T\big(\ \cdot\ \big)\mathbf{y}$.

In other words, tell us what goes inside those parentheses. You can use $\mathbf{X}$, $\mathbf{I}$, λ, and mathematical operations.

(HINT: What is the relationship between $\mathbf{X}$ *and* $\mathbf{K}$*?)*

(b) (10 pts) Now we will follow the approach given by your friend. For each potential input $\mathbf{x}$, you can compute an $n-$dimensional feature vector $\boldsymbol{\phi}(\mathbf{x}) = \begin{bmatrix} \phi_1(\mathbf{x}) \\ \phi_2(\mathbf{x}) \\ \vdots \\ \phi_n(\mathbf{x}) \end{bmatrix} = \begin{bmatrix} k(\mathbf{x}_1, \mathbf{x}) \\ k(\mathbf{x}_2, \mathbf{x}) \\ \vdots \\ k(\mathbf{x}_n, \mathbf{x}) \end{bmatrix}$ and you can define a new $n \times n$ dimensional feature matrix from your training data in the usual way:

$$\boldsymbol{\Phi} = \begin{bmatrix} \boldsymbol{\phi}(\mathbf{x}_1)^T \\ \boldsymbol{\phi}(\mathbf{x}_2)^T \\ \vdots \\ \boldsymbol{\phi}(\mathbf{x}_n)^T \end{bmatrix}. \tag{7}$$

Using the "linear kernel" $k(\mathbf{x}, \mathbf{z}) = \mathbf{x}^T\mathbf{z}$ to construct our features above, do standard ridge regression using this feature data matrix and the training outputs $\mathbf{y}$ to learn feature weights $\boldsymbol{\beta} = (\boldsymbol{\Phi}^T\boldsymbol{\Phi} + \lambda\mathbf{I})^{-1}\boldsymbol{\Phi}^T\mathbf{y}$ for our new learned function $\widehat{f_{\text{new}}}(\mathbf{x}) = \sum_{i=1}^{n} \beta[i]\phi_i(\mathbf{x}) = \sum_{i=1}^{n} \beta[i]k(\mathbf{x}_i, \mathbf{x})$. You can use this function to make predictions for a test point $\widehat{y}_{\text{test, new}} = \widehat{f_{\text{new}}}(\mathbf{x}_{\text{test}})$.

Give an expression in matrix form involving $\mathbf{X}, \lambda, \mathbf{I}$ for: $\widehat{y}_{\mathbf{test,\ new}} = \mathbf{x}_{\mathbf{test}}^T\big(\ \cdot\ \big)\mathbf{y}$.

In other words, tell us what goes inside those parentheses.

(HINT: What is the relationship between the $\mathbf{K}$ *matrix from the previous part and the* $\boldsymbol{\Phi}$ *matrix here?)*

(c) (18 pts) Suppose $\mathbf{X} = \mathbf{U}\Sigma\mathbf{V}^T$ where we are using the full-sized SVD and so $\mathbf{U}$ is $n \times n$ and the matrix $\mathbf{V} = m \times m$ where m is the size of the input vectors $\mathbf{x}$.

We know in this case that the ordinary-least-squares prediction based on the input data would just be

$$\sum_{i=1}^{\min(n,m)} (\mathbf{u}_i^T\mathbf{y})\Big(\frac{1}{\sigma_i}\Big)\mathbf{v}_i^T\mathbf{x}_{test} \tag{8}$$

where the matrix $\mathbf{U} = [\mathbf{u}_1, \mathbf{u}_2, \cdots, \mathbf{u}_n]$ and $\mathbf{V} = [\mathbf{v}_1, \mathbf{v}_2, \cdots, \mathbf{v}_m]$ and σ_i are the singular values of the matrix $\mathbf{X}$.

Give counterpart expressions for what would be between the central parentheses (where OLS has $\frac{1}{\sigma_i}$) in (8) for three approaches for learning:

- **What you did in part (a): kernel ridge regression with a linear kernel.**
- **What you did in part (b): your friend's approach to ridge regression with kernel features for the linear kernel.**
- **What you would have gotten had you done $k-$PCA+OLS to learn the pattern.**

Your answers should only have a dependence on the singular values and the chosen λ for the two approaches above (*this can help you check your work for the earlier parts*), and the singular values and k for the case of using PCA for dimensionality reduction followed by OLS.

(d) (4 pts) **Comment on whether the two approaches to ridge regression with kernels are the exactly the same, similar, or in what sense they are different in what they end up doing.**

(HINT: What is the overall behavior as you sweep through possible values for the hyperparameter λ? Although this part doesn't ask you to comment on the relationship to k−PCA+OLS, you might find it helpful to remember what happens there as you sweep through k.)

6 Regularization with one-hot coupons (18 points)

The first p features in our input data are regular input features. The last d features are a one-hot encoding of a coupon that can have d different types, each equally likely. For each input x, a d-sided die is independently rolled and an integer ξ is picked uniformly at random from $p+1$ to $p+d$. Then, because we are using one-hot encoding, the ξ-th feature is a 1 while all the other coupon-features are 0. Mathematically:

$$x[j] = \begin{cases} 0, & \text{if } j \geq p+1 \text{ and } j \neq \xi, \\ 1, & \text{if } j = \xi. \end{cases}$$

Depending on whether or not we include coupon features, we get two different training data matrices:

$$\mathbf{X}_{\text{train}} = \begin{bmatrix} x_1[1] & x_1[2] & \cdots & x_1[p] \\ x_2[1] & x_2[2] & \cdots & x_2[p] \\ \vdots & \vdots & \ddots & \vdots \\ x_n[1] & x_n[2] & \cdots & x_n[p] \end{bmatrix}; \quad \mathbf{X}_{\text{train, long}} = \begin{bmatrix} x_1[1] & x_1[2] & \cdots & x_1[p+d] \\ x_2[1] & x_2[2] & \cdots & x_2[p+d] \\ \vdots & \vdots & \ddots & \vdots \\ x_n[1] & x_n[2] & \cdots & x_n[p+d] \end{bmatrix}.$$

Your training data set has n i.i.d. training samples $\{(\mathbf{x}_i, y_i)\}_{i=1}^{n}$, where all $\mathbf{x}_i$ are drawn as described above. You also collect an independent test set with m i.i.d. test samples $\{(\mathbf{x}_i, y_i)\}_{i=n+1}^{n+m}$ from the same distribution.

Two data analysts are planning the following experiment:

- The first data analyst thinks that the coupons are irrelevant, so she will remove the d coupon features. Then she will perform ridge regression with the remaining p-dimensional data with $\lambda = 1$. In other words, she just uses the matrix $\mathbf{X}_{\text{train}}$ above along with the $\mathbf{y}$ to do ridge regression to learn p weights.
- The second data analyst will simply find the minimum norm interpolating solution while including the coupon-features. So, the second data analyst uses the much wider training data matrix $\mathbf{X}_{\text{train, long}}$ together with the $\mathbf{y}$ and learns the $p+d$ weights by computing:

$$\widehat{\mathbf{w}} = \underset{\mathbf{w} \text{ such that } \mathbf{X}_{\text{train, long}}\mathbf{w}=\mathbf{y}}{\arg\min} \|\mathbf{w}\|^2$$

Both analysts use their learned models to compute predictions on the test set and compare results.

With probability $\prod_{i=0}^{m+n-1}(1-\frac{i}{d})$, the $m+n$ coupons you get for the training and test set will all be distinct: in this event, you won't get the same coupon twice across all $m+n$ samples. **Show that the predictions of both data analysts on the entire test data will be exactly the same if all $m+n$ coupons turn out to be distinct.**

(Hint: Draw a picture of what that looks like for the training data and for the test data that we have. Which perspective on ridge regression is likely to be useful here?)

7 Local approximation using neighbors (20 points)

Suppose we are learning a one-dimensional function f. We have access to noisy samples (x_i, y_i) with $x_i, y_i \in \mathbb{R}$, We consider the estimator

$$\widehat{f}(x) = \sum_i y_i h(x - x_i),$$

where $h(x - x_i)$ is the local influence of the i-th training sample on the value of $\widehat{f}$ at x.

(a) (5 pts) Suppose our training samples x_i were equally spaced (Δ apart: so $x_{i+1} = x_i + \Delta$) and ordered (i.e., $x_i < x_j$ for $i < j$), with i going through the integers from $-\infty$ to $+\infty$. If $\widehat{f}$ is a k-nearest neighbor estimator (i.e., the prediction for a test point is the average of the closest k samples from the training data), and $k = 2\ell$ for some strictly positive integer ℓ, **what is the relevant local influence function $h(\cdot)$ that would give this estimator?** Your answer should involve ℓ and Δ in some way.

(HINT: Draw what is going on to help.)

(b) (15 pts) Now, let us consider an explicit function that we want to learn: the absolute value function $f(x) = |x|$. Suppose we have training samples (x_i, y_i) that are separated by $\Delta = 1$ with $x_i = i + \frac{1}{2}$ and $y_i = f(x_i) + \epsilon_i$ *for all integers i.* The training noise is i.i.d. with $\epsilon_i \sim \mathcal{N}(0, \sigma^2)$.

If our test point is just $x = 0$, our goal is to understand the best value of the hyperparameter ℓ so that our mean squared error on this test point would be minimized when using a $k = 2\ell$-nearest neighbor estimator. **What ℓ should we choose as a function of the training noise standard deviation σ?**

(HINT: The only randomness here is in the noise on the training points. To understand the dependence of terms you might want to make sure you understand how the mean-squared error behaves when $\ell = 1$ and $\ell = 2$ before working with a formula. What gets worse as you increase ℓ? What might be getting better?)

Depending on how you you choose to solve this, you might find the formula for the sum of an arithmetic series useful $\sum_{i=1}^{\ell} i = \frac{1}{2}\ell(\ell+1)$.

Feel free to compute a possibly continuous value for ℓ — on the exam, don't worry about what the right way to round it might be.

Now, the new part of the homework begins on material that you haven't done homework on before.

8 ℓ_1-Penalized Linear Regression: LASSO

The ℓ_1-norm is one of the popular penalty functions used to enhance the robustness of regression models. Regression with an ℓ_1-penalty is referred to as the LASSO regression. One of its most important aspects is that it promotes sparsity in the resulting solution. In this problem, we will explore the optimization of the LASSO in a simplified setting.

Assume the training data points are denoted as the rows of a $n \times d$ matrix $\mathbf{X}$ and their corresponding output value as an $n \times 1$ vector $\mathbf{y}$. The generic parameter vector and its optimal value (relative to the LASSO cost function) are represented by $d \times 1$ vectors $\mathbf{w}$ and $\widehat{\mathbf{w}}$, respectively. For the sake of simplicity, we assume columns of data have been standardized to have mean 0 and variance 1, and are also uncorrelated (i.e. $\mathbf{X}^\top \mathbf{X} = n\mathbf{I}$). (*We center the data mean to zero, so that the penalty treats all features similarly. We assume uncorrelated features as a simplified assumption in order to reason about LASSO in this question. In general, this is a major simplification since the power of regularlizers is that they often enable us to make reliable inferences even when we do not have as many samples as we have raw features.*)

For LASSO regression, the optimal parameter vector is given by:

$$\widehat{\mathbf{w}} = \arg\min_{\mathbf{w}} \{J_\lambda(\mathbf{w}) = \frac{1}{2}\|\mathbf{y} - \mathbf{X}\mathbf{w}\|_2^2 + \lambda\|\mathbf{w}\|_1\},$$

where $\lambda > 0$.

(a) **Show that for data with uncorrelated features, one can learn the parameter w_i corresponding to each i-th feature independently from the other features, one at a time, and get a solution which is equivalent to having learned them all jointly as we normally do.**

Hint: To show this, write $J_\lambda(\mathbf{w})$ in the following form for appropriate functions g and f:

$$J_\lambda(\mathbf{w}) = g(\mathbf{y}) + \sum_{i=1}^{d} f(\mathbf{X}_i, \mathbf{y}, w_i, \lambda)$$

where $\mathbf{X}_i$ is the i-th column of $\mathbf{X}$. By having no interaction terms that link the different w_i variables, you know that the joint optimization can be decomposed into individual optimizations.

(b) Assume that $\widehat{w}_i > 0$. **What is the value of $\widehat{w}_i$ in this case?**

(c) Assume that $\widehat{w}_i < 0$. **What is the value of $\widehat{w}_i$ in this case?**

(d) From the previous two parts, **what is the condition for $\widehat{w}_i$ to be zero?**

If you put this part together with the previous parts, you can understand why this is often referred to as "soft thresholding."

(e) Now consider the ridge regression problem. In ridge, the regularization term is replaced by $\lambda\|\mathbf{w}\|_2^2$ where the optimal parameter vector is now given by:

$$\widehat{\mathbf{w}} = \arg\min_{\mathbf{w}} \{J_\lambda(\mathbf{w}) = \frac{1}{2}\|\mathbf{y} - \mathbf{X}\mathbf{w}\|_2^2 + \lambda\|\mathbf{w}\|_2^2\},$$

where $\lambda > 0$.

What is the condition for $\widehat{w}_i = 0$? How does it differ from the condition you obtained in the previous part? Can you see why the ℓ_1 norm promotes sparsity?

(f) Assume that we have a sparse image vectorized in the vector $\mathbf{w}$ (so $\mathbf{w}$ is a sparse vector). We have a Gaussian matrix $n \times d$ matrix $\mathbf{X}$ and an $n \times 1$ noise vector $\mathbf{z}$ where $n > 1$. Our measurements take the form $\mathbf{y} = \mathbf{X}\mathbf{w} + \mathbf{z}$. We want to extract the original image $\mathbf{w}$ given matrix $\mathbf{X}$ knowing that this image is sparse. The fact that $\mathbf{w}$ is sparse suggests using ℓ_1 regularization. Use the provided Jupyter notebook and apply ℓ_1 regularization.

(This might remind you of EE16A. That's intended. OMP is another way to find sparse solutions. OMP is generally faster to run than LASSO and can be understood without as much optimization background. The issue with vanilla OMP is that while you can view when to stop as a hyperparameter, this isn't quite the same as being able to tune λ. It turns out that there are algorithms that are inspired by OMP that can spiritually compute something pretty close to what LASSO would compute as you vary the λ parameter. One such algorithm is called LARS, but it is out of scope for this course.)

Change the hyperparameter λ to extract the best looking image and report it.

9 Contamination of Sparse Linear Models Obtained by Thresholding

In this question, we will analyze the benefits of learning sparse linear models when the true pattern is indeed sparse. In particular, we will analyze two simple procedures (computing the OLS solution and then just keeping the biggest entries or just keeping entries bigger than a threshold) that perform feature selection in linear models, and show quantitatively that feature selection lowers the resulting contamination in a way that reduces the mean-squared prediction error. This is sometimes viewed as a "variance reduction" in the literature.

This should make sense to you at an intuitive level for two reasons. The first should be pretty straightforward: zeroing out a bunch of feature weights when the truth is sparse is going to reduce contamination if it is contaminating weights that are being zeroed out. A more roundabout way is to think about model complexity (using a vague analogy with polynomial degrees, etc...) since enforcing sparsity is equivalent to deliberately constraining model complexity. Simpler models should be easier to learn in a more reliable way as long as we have enough data.

There are, however, some concerns. The most basic one is that you might be worried that the thresholding procedure will end up eliminating the true signal itself! How do we know that the true signal will make the cut? If the true signal doesn't survive, then we could get bad prediction error performance from that fact, notwithstanding any reduction in contamination.

The other issue is more subtle. There is a difference between feature selection before looking at training data and after seeing the training data. If we use fewer features to begin with (and we have enough training data), our intuitions so far imply that we will have lower prediction error because of smaller model complexity — there are simply fewer features that the training noise can put contamination into.

What we learn from working through this problem is that selecting features adaptively (that is, based on the training data itself) does not hurt either, if done properly. In other words, although there is a philosophical difference between doing feature selection before or after using the data, post training feature selection still leads to improved prediction error (via contamination reduction) under certain assumptions.

First, some setup. Data from a sparse linear model is generated using

$$\mathbf{y} = \mathbf{X}\mathbf{w}^* + \mathbf{z},$$

where $\mathbf{y} \in \mathbb{R}^n$ denotes a vector of responses, $\mathbf{X} \in \mathbb{R}^{n \times d}$ is our data matrix, $\mathbf{w}^* \in \mathbb{R}^d$ is an unknown, s-sparse vector of true parameters (with at most s non-zero entries), and $\mathbf{z} \sim N(0, \sigma^2 I_n)$ is an n-dimensional vector of i.i.d. Gaussian noise of variances σ^2. Notice the critical difference from the Bi-Level model you had seen earlier when we were discussing overparameterized models. In that model, we knew which s features were favored and where the true parameters lay. In this model, all that we know is that the true parameter vector is sparse — we don't know which features are favored before seeing the labels.

The solution to first three parts of the problem can be filled out in the provided Jupyter notebook. The point of the first three parts is to make sure that you understand the magnitude of this effect. The rest is designed to help you see why this effect is there. Parts (d)-(j) must have a separate, written solution. All logarithms are to the base e.

(a) Let us first do some numerical exploration. **In the provided Jupyter notebook, you will find code to generate and plot the behavior of the ordinary least squares algorithm.**

(b) In this problem, we will analyze two estimators that explicitly take into account the fact that $\mathbf{w}^*$ is sparse, and consequently attain lower error than the vanilla least-squares estimate.

Let us define two operators. Given a vector $\mathbf{v} \in \mathbb{R}^d$, the operation $\tau_k(\mathbf{v})$ zeroes out all but the top k entries of $\mathbf{v}$ measured in absolute value. The operator $T_\lambda(\mathbf{v})$, on the other hand, zeros out all entries that are less than λ in absolute value.

Recall that the least squares estimate was given by $\widehat{\mathbf{w}}_{\mathsf{LS}} = \mathbf{X}^\dagger \mathbf{y} = \mathbf{X}^\top \mathbf{y}$, where $\mathbf{X}^\dagger$ is the appropriate pseudo-inverse of $\mathbf{X}$. As we did earlier, we'll make the simplifying assumption that $\mathbf{X}$ has orthonormal columns even in the training data. (Notice that by making this assumption, we are also making it clear that the input data matrix $\mathbf{X}$ doesn't give any hints as to which features or linear combinations of features are to be prefered — everything looks equally strong as a feature. This is a way of forcing the $\mathbf{y}$ to speak if they have anything to say.) We now define

$$\widehat{\mathbf{w}}_{\mathsf{top}}(s) = \tau_s(\widehat{\mathbf{w}}_{\mathsf{LS}})$$
$$\widehat{\mathbf{w}}_T(\lambda) = T_\lambda(\widehat{\mathbf{w}}_{\mathsf{LS}}),$$

which are the two sparsity-inducing estimators that we will consider.

Now implement the two estimators described above and **plot their performance as a function of n, d and s.**

Here, we are going to assume that the underlying features are also orthonormal relative to the test distribution so the Euclidean-norm squared of the error in estimating the weights is a correct estimate of the prediction error on the test distribution.

(c) **Now generate data from a non-sparse linear model, and numerically compute the estimators for this data. Explain the behavior you are seeing in these plots in terms of the underlying trade-off between approximation error, survival and contamination.**

(d) In the rest of the problem, we will theoretically analyze the prediction error for the top-k procedure for sparse estimation, and try to explain the curves you saw in the numerical explorations above. We will need to use a handy tool, which is a bound on the maximum of d Gaussian random variables.

Show that given d Gaussians $\{Z_i\}_{i=1}^d$ (not necessarily independent) with mean 0 and variance σ^2, we have

$$\mathbf{P}\left\{\max_{i\in\{1,2,\ldots,d\}} |Z_i| \geq 2\sigma\sqrt{\log d}\right\} \leq \frac{1}{d}.$$

Hint 1: You may use without proof the fact that for a Gaussian random variable $Z \sim N(0,\sigma^2)$ and scalar $t > 0$, we have $\mathbf{P}\{|Z| \geq t\} \leq e^{-\frac{t^2}{2\sigma^2}}$.

Hint 2: For the maximum to be large, one of the Gaussians must be large. Now use the union bound.

(If you wanted to, you could prove tighter concentration for the maximum of iid Gaussian random variables by invoking/proving the weak-law-of-large numbers for the maximum of iid random variables and introducing an appropriate tolerance ϵ, etc. But we don't ask you to do that here to keep the math lighter.)

(e) As in the previous problem, for algebraic convenience, we will restrict attention in this entire problem to the special case where the input data matrix $\mathbf{X}$ has orthonormal columns.

Show that $\widehat{\mathbf{w}}_{\mathsf{top}}(s)$ can be expressed as the top s entries of the vector $\mathbf{w}^* + z'$ in absolute value, where z' is an i.i.d. Gaussian random variable with variance σ^2.

(f) **Argue that the (random) error vector $\mathbf{e} = \widehat{\mathbf{w}}_{\mathsf{top}}(s) - \mathbf{w}^*$ is always (at most) $2s$-sparse.**

(g) Let us now condition on the event $\mathcal{E} = \{\max |z_i'| \leq 2\sigma\sqrt{\log d}\}$. Conditioned on this event, **show that we have $|e_i| \leq 4\sigma\sqrt{\log d}$ for each index i.**

(h) **Conclude that with probability at least $1 - 1/d$, we have**

$$\|\widehat{\mathbf{w}}_{\mathsf{top}}(s) - \mathbf{w}^*\|_2^2 \leq 32\sigma^2 s \log d.$$

We can understand if you feel that this was a bit too slick since we never truly engaged with what the true signal was. Essentially, we finessed this issue by saying that the worst random competing false feature can't be too loud, and so if we lost any parts of the true signal, they weren't that important to begin with.

(i) We can view this from a denoising perspective on the original training data. Use the above part to **show that with probability at least** $1 - 1/d$**, we have**

$$\frac{1}{n}\|\mathbf{X}(\widehat{\mathbf{w}}_{\mathsf{top}}(s) - \mathbf{w}^*)\|_2^2 \leq 32\sigma^2 \frac{s \log d}{n}.$$

(j) Considering the simple least-squares estimator $\widehat{\mathbf{w}}_{\mathsf{LS}} = (\mathbf{X}^\top \mathbf{X})^{-1}\mathbf{X}^\top \mathbf{y}$, we can show that

$$\mathbb{E}\left[\frac{1}{n}\|\mathbf{X}(\widehat{\mathbf{w}}_{\mathsf{LS}} - \mathbf{w}^*)\|_2^2\right] = \sigma^2 \frac{d}{n}, \text{ and} \tag{9}$$

$$\mathbb{E}\left[\|\widehat{\mathbf{w}}_{\mathsf{LS}} - \mathbf{w}^*\|_2^2\right] = \sigma^2 \mathsf{trace}\left[(\mathbf{X}^\top \mathbf{X})^{-1}\right], \tag{10}$$

respectively. Equations (9) and (10) represent the "denoising" squared error and the mean squared error of the parameters of our OLS model, respectively. Recall that in this problem, we have been assuming that the matrix $\mathbf{X}$ has orthonormal columns, and so the bounds become

$$\mathbb{E}\left[\frac{1}{n}\|\mathbf{X}(\widehat{\mathbf{w}}_{\mathsf{LS}} - \mathbf{w}^*)\|_2^2\right] = \sigma^2 \frac{d}{n}, \text{ and} \tag{11}$$

$$\mathbb{E}\left[\|\widehat{\mathbf{w}}_{\mathsf{LS}} - \mathbf{w}^*\|_2^2\right] = \sigma^2 d. \tag{12}$$

Compare these to the previous part, assume that the statement proved with probability $1 - \frac{1}{d}$ can actually be modified with more work to be an analogous statement that holds with high probability, and **conclude that by leveraging the sparsity assumption, we have smaller prediction error as long as** $s \leq \frac{d}{32 \log d}$**.**

In conclusion, roughly speaking the sparse solution has a mean prediction error upper bound of $O(\sigma^2 \frac{s \log d}{n})$ with high probability. On the other hand, the least square solution has an expected mean prediction error of $\sigma^2 \frac{d}{n}$. Thus when $s \log d < c \times d$, the sparse solution has smaller error than the least square solution, where c is some constant.

It turns out that appropriate sparsity seeking estimators can go well beyond the $n = d$ barrier by exploiting core effects similar to what are being discussed here. Fundamentally, truly false features have a hard time by dumb luck competing with true ones because of chance alignment with the training noise. Meanwhile, true features can stand out. This is one of the reasons that OMP works well, but that is not a topic for this problem.

(k) **Now consider the case if we already knew the important** s **features to begin with. What would be the prediction squared error of the sparse OLS estimator that just used the** s **important features? How does this compare to the behavior for the sparse** $\widehat{\mathbf{w}}_{\mathsf{top}}(s)$ **derived above? What is the price of not knowing in advance which are the important features?**

10 Sensors, Objects, and Localization

In this problem, we will be using gradient descent to solve the problem of figuring out where objects are, given noisy distance measurements. (This is roughly how GPS works and students who have taken EE16A have seen a variation on this problem in lecture and lab. The hope is that this problem will provide you with a way to build a more physical intuition about what is going on with both gradient descent as well as the concept of local minima, etc. and their relationship with machine-learning relevant concepts like noise, number of features, etc.)

First, the setup. Let us say there are m sensors and n objects located in a two dimensional plane. The m sensors are located at the points $(a_1, b_1), \ldots, (a_m, b_m)$. The n objects are located at the points $(x_1, y_1), \ldots, (x_n, y_n)$. We have measurements for the distances between the sensors and the objects: D_{ij} is the measured distance from sensor i to object j. The distance measurement has noise in it. Specifically, we model

$$D_{ij} = \|(a_i, b_i) - (x_j, y_j)\|_2 + Z_{ij},$$

where $Z_{ij} \sim N(0, 1)$. The noise is independent across different measurements.

Code has been provided for data generation to aid your explorations.

This is the first part of this problem. We will return to this in a later homework where you will see how neural network approaches stack up on such problems.

(a) Consider the case where $m = 7$ and $n = 1$. That is, there are 7 sensors and 1 object. Suppose that we know the exact location of the 7 sensors but not the 1 object. We have 7 measurements of the distances from each sensor to the object $D_{i1} = d_i$ for $i = 1, \ldots, 7$. Because the underlying measurement noise is modeled as iid Gaussian, the log likelihood function, which we would like to maximize, is

$$L(x_1, y_1) = -\sum_{i=1}^{7} (\sqrt{(a_i - x_1)^2 + (b_i - y_1)^2} - d_i)^2 + C, \tag{13}$$

where C is a constant. **Manually compute the symbolic gradient of the log likelihood function with respect to x_1 and y_1.**

(b) The provided code generates

- $m = 7$ sensor locations (a_i, b_i) sampled from $N(\mathbf{0}, \sigma_s^2 \mathbf{I})$
- $n = 1$ object locations (x_1, y_1) sampled from $N(\boldsymbol{\mu}, \sigma_o^2 \mathbf{I})$
- $mn = 7$ distance measurements $D_{i1} = \|(a_i, b_i) - (x_1, y_1)\| + N(0, 1)$.

for $\boldsymbol{\mu} = [0, 0]^T$, $\sigma_s = 100$ and $\sigma_o = 100$. **Solve for the maximum likelihood estimator of (x_1, y_1) by gradient descent on the *negative* log-likelihood. Report the true and estimated (x_1, y_1) for the given sensor locations. Try two approaches for initializing gradient descent: starting at 0 and starting at a random point. Which of the following step sizes is the most reasonable one, $1, 0.01, 0.001$ or 0.0001?**

(c) (Local Minima of Gradient Descent) In this part, we vary the location of the single object among different positions:

$$(x_1, y_1) \in \{(0, 0), (100, 100), (200, 200), \ldots, (900, 900)\}.$$

For each choice of (x_1, y_1), generate the following data set 10 times, for a total of 100 data sets:

- Generate $m = 7$ sensor locations (a_i, b_i) from $N(\mathbf{0}, \sigma_s^2 \mathbf{I})$ (Use the same σ_s from the previous part.)
- Generate $mn = 7$ distance measurements $D_{i1} = \|(a_i, b_i) - (x_1, y_1)\|_2 + N(0, 1)$.

For each data set, run the gradient descent methods 100 times to find a prediction for (x_1, y_1). We are pretending we do not know (x_1, y_1) and are trying to predict it. For each gradient descent, take 1000 iterations with step-size 0.1 and a random initialization of (x, y) from $N(\mathbf{0}, \sigma^2 \mathbf{I})$, where $\sigma = x_1 + 1$.

These are the results you should report for this problem:

- **Draw the contour plot of the log likelihood function of a particular data set for $(x_1, y_1) = (0, 0)$ and $(x_1, y_1) = (200, 200)$.**
- For each of the ten data sets and each of the ten choices of (x_1, y_1), calculate the number of distinct points that gradient descent converges to. Then, for each of the ten choices of (x_1, y_1), calculate the average of the number of distinct points over the ten data sets. **Plot the average number of local minima against x_1.** For this problem, two local minima are considered identical if their distance is within 0.01.

  Hint: `np.unique` and `np.round` will help.
- For each of the ten data sets and each of the ten choices of (x_1, y_1), calculate the proportion of gradient descents which converge to what you believe to be a global minimum (that is, the minimum point in the set of local minima that you have found). Then, for each of the ten choices of (x_1, y_1), calculate the average of the proportion over the ten data sets. **Plot the average proportion against x_1.**
- For the 7th data set for the object location of (500, 500), **plot the sensor locations, the ground truth object location and the MLE object locations found by 100 times of gradient descent. Do you find any patterns?**

Please be aware that the code might take a while to run.

(d) Repeat the previous part, except explore what happens as you reduce the variance of the measurement noise. **Present the same plots and comment on the differences and why they occur.**

For the sake of saving time, you can experiment with only one smaller noise, such as $\mathcal{N}(0, 0.01^2)$. You might need to change the learning rate to make the gradient descent converge.

(e) Repeat the part (c) again, except explore what happens as you increase the number of sensors. For the sake of saving time, you can experiment with only one number of sensors, such as 20. **Present the same plots and comment on the differences and why they occur.**

11 ReLU Elbow Update under SGD

In this question we will explore the behavior of the ReLU nonlinearity with Stochastic Gradient Descent (SGD) updates. The hope is that this problem should help you build a more intuitive understanding for how SGD works and how it iteratively adjusts the learned function.

We want to model a 1D function $y = f(x)$ using a 1-hidden layer network with ReLU activations and no biases in the linear output layer. Mathematically, our network is

$$\hat{f}(x) = \mathbf{W}^{(2)}\Phi\left(\mathbf{W}^{(1)}x + \mathbf{b}\right)$$

where $x, y \in \mathbb{R}$, $\mathbf{b} \in \mathbb{R}^d$, $\mathbf{W}^{(1)} \in \mathbb{R}^{d\times 1}$, and $\mathbf{W}^{(2)} \in \mathbb{R}^{1\times d}$. We define our loss function to be the squared error,

$$\ell\left(x, y, \mathbf{W}^{(1)}, \mathbf{b}, \mathbf{W}^{(2)}\right) = \frac{1}{2}\left\|\hat{f}(x) - y\right\|_2^2.$$

For the purposes of this problem, we define the gradient of a ReLU at 0 to be 0.

You will visualize the behavior derived here using the notebook from Discussion 7.

(a) Let's start by examining the behavior of a single ReLU with a linear function of x as the input,

$$\phi(x) = \begin{cases} wx + b, & wx + b > 0 \\ 0, & \text{else} \end{cases}.$$

Notice that the slope of $\phi(x)$ is w in the non-zero domain.
We define a loss function $\ell(x, y, \phi) = \frac{1}{2}\|\phi(x) - y\|_2^2$. **Find the following:**

- **(i) The location of the 'elbow' e of the function, where it transitions from 0 to something else.**
- **(ii) The derivative of the loss w.r.t. $\phi(x)$, namely $\frac{d\ell}{d\phi}$**
- **(iii) The partial derivative of the loss w.r.t. w, namely $\frac{\partial \ell}{\partial w}$**
- **(iv) The partial derivative of the loss w.r.t. b, namely $\frac{\partial \ell}{\partial b}$**

(b) Now suppose we have some training point (x, y) such that $\phi(x) - y = 1$. In other words, the prediction $\phi(x)$ is 1 unit above the target y — we are too high and are trying to pull the function downward.

Describe what happpens to the slope and elbow of $\phi(x)$ when we perform gradient descent in the following cases:

- **(i)** $\phi(x) = 0$**.**
- **(ii)** $w > 0$**,** $x > 0$**, and** $\phi(x) > 0$**. It is fine to check the behavior of the elbow numerically in this case.**
- **(iii)** $w > 0$**,** $x < 0$**, and** $\phi(x) > 0$**.**
- **(iv)** $w < 0$**,** $x > 0$**, and** $\phi(x) > 0$**. It is fine to check the behavior of the elbow numerically in this case.**

Additionally, draw and label $\phi(x)$, the elbow, and the qualitative changes to the slope and elbow after a gradient update to w and b. You should label the elbow location and a candidate (x, y) pair. Remember that the update for some parameter vector $\mathbf{p}$ and loss ℓ under SGD is

$$\mathbf{p}' = \mathbf{p} - \lambda \nabla_{\mathbf{p}}(\ell),\ \lambda > 0.$$

(c) Now we return to the full network function $\hat{f}(x)$. **Derive the location e_i of the elbow of the i'th elementwise ReLU activation.**

(d) **Derive the new elbow location e_i' of the i'th elementwise ReLU activation after one stochastic gradient update with learning rate λ.** (*Hint: we found the vector partial derivatives of this network formulation in Discussion 7 using backpropagation.*)

12 Your Own Question

Write your own question, and provide a thorough solution.

Writing your own problems is a very important way to really learn the material. The famous "Bloom's Taxonomy" that lists the levels of learning is: Remember, Understand, Apply, Analyze, Evaluate, and Create. Using what you know to create is the top-level. We rarely ask you any HW questions about the lowest level of straight-up remembering, expecting you to be able to do that yourself. (e.g. make yourself flashcards) But we don't want the same to be true about the highest level.

As a practical matter, having some practice at trying to create problems helps you study for exams much better than simply counting on solving existing practice problems. This is because thinking about how to create an interesting problem forces you to really look at the material from the perspective of those who are going to create the exams.

Besides, this is fun. If you want to make a boring problem, go ahead. That is your prerogative. But it is more fun to really engage with the material, discover something interesting, and then come up with a problem that walks others down a journey that lets them share your discovery. You don't have to achieve this every week. But unless you try every week, it probably won't happen ever.

Contributors:

- Alexander Tsigler
- Anant Sahai
- Ashwin Pananjady
- Chawin Sitawarin
- Inigo Incer
- Josh Sanz
- Raaz Dwivedi
- Rahul Arya
- Yang Gao
- Yaodong Yu