EECS 189 Introduction to Machine Learning
Fall 2020 HW5

This homework is due **Wednesday, October 7 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.

2 Kernel PCA

You have seen how to use PCA to do dimensionality reduction by projecting the data to a subspace that captures most of the variability visible in the observed features. The underlying hope is that these directions of variation are also relevant for prediction the quantities of interest.

Standard PCA works well for data that is roughly Gaussian shaped, but many real-world high dimensional datasets have underlying low-dimensional structure that is not well captured by linear subspaces. However, when we lift the raw data into a higher-dimensional feature space by means of a nonlinear transformation, the underlying low-dimensional structure once again can manifest as an approximate subspace. Linear dimensionality reduction can then proceed. As we have seen

in class so far, kernels are an alternate way to deal with these kinds of nonlinear patterns without having to explicitly deal with the augmented feature space. This problem asks you to discover how to apply the "kernel trick" to PCA.

Let $\mathbf{X} \in \mathbb{R}^{n \times \ell}$ be the data matrix, where n is the number of samples and ℓ is the dimension of the raw data. Namely, the data matrix contains the data points $\mathbf{x}_j \in \mathbb{R}^\ell$ as rows

$$\mathbf{X} = \begin{pmatrix} \mathbf{x}_1^\top \\ \mathbf{x}_2^\top \\ \vdots \\ \mathbf{x}_n^\top \end{pmatrix} \in \mathbb{R}^{n \times \ell}. \tag{1}$$

(a) **Compute $\mathbf{X}\mathbf{X}^\top$ in terms of the singular value decomposition $\mathbf{X} = \mathbf{U}\mathbf{\Sigma}\mathbf{V}^\top$ where $\mathbf{U} \in \mathbb{R}^{n \times n}, \mathbf{\Sigma} \in \mathbb{R}^{n \times \ell}$ and $\mathbf{V} \in \mathbb{R}^{\ell \times \ell}$.** Notice that $\mathbf{X}\mathbf{X}^\top$ is the matrix of pairwise Euclidean inner products for the data points. **How would you get $\mathbf{U}$ if you only had access to $\mathbf{X}\mathbf{X}^\top$?**

(b) Given a new test point $\mathbf{x}_{test} \in \mathbb{R}^\ell$, one central use of PCA is to compute the projection of $\mathbf{x}_{test}$ onto the subspace spanned by the k top singular vectors $\mathbf{v}_1, \mathbf{v}_2, \ldots, \mathbf{v}_k$.

Express the scalar projection $z_j = \mathbf{v}_j^\top \mathbf{x}_{test}$ onto the j-th principal component as a function of the inner products

$$\mathbf{X}\mathbf{x}_{test} = \begin{pmatrix} \langle \mathbf{x}_1, \mathbf{x}_{test} \rangle \\ \vdots \\ \langle \mathbf{x}_n, \mathbf{x}_{test} \rangle \end{pmatrix}. \tag{2}$$

Assume that all diagonal entries of $\mathbf{\Sigma}$ are nonzero and non-increasing, that is $\sigma_1 \geq \sigma_2 \geq \cdots > 0$.

Hint: Express $\mathbf{V}^\top$ in terms of the singular values $\mathbf{\Sigma}$, the left singular vectors $\mathbf{U}$ and the data matrix $\mathbf{X}$. If you want to use the compact form of the SVD, feel free to do so.

(c) How would you define kernelized PCA for a general kernel function $k(\mathbf{x}_i, \mathbf{x}_j)$ (to replace the Euclidean inner product $\langle \mathbf{x}_i, \mathbf{x}_j \rangle$)? For example, the RBF kernel $k(\mathbf{x}_i, \mathbf{x}_j) = \exp\left(-\frac{\|\mathbf{x}_i - \mathbf{x}_j\|^2}{\delta^2}\right)$.

Describe this in terms of a procedure which takes as inputs the training data points $\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_n \in \mathbb{R}^\ell$ and the new test point $\mathbf{x}_{test} \in \mathbb{R}^\ell$, and outputs the analog of the previous part's z_j coordinate in the kernelized PCA setting. You should include how to compute $\mathbf{U}$ from the data, as well as how to compute the analog of $\mathbf{X}\mathbf{x}_{test}$ from the previous part.

Invoking the SVD or computing eigenvalues/eigenvectors is fine in your procedure, as long as it is clear what matrix is having its SVD or eigenvalues/eigenvectors computed. The kernel $k(\cdot, \cdot)$ can be used as a black-box function in your procedure as long as it is clear what arguments it is being given.

3 Total Least Squares

Make sure to submit the code you write in this problem to "HW5 Code" on Gradescope.

In most of the models we have looked at so far, we've accounted for noise in the observed y measurement and adjusted accordingly. However, in the real world it could easily be that our feature matrix X of data is also corrupted or noisy. Total least squares is a way to account for this. Whereas previously we were minimizing the y distance from the data point to our predicted line because we had assumed the features were definitively accurate, now we are minimizing the entire distance from the data point to our predicted line. In this problem we will explore the mathematical intuition for the TLS formula. We will then apply the formula to adjusting the lighting of an image which contains noise in its feature matrix due to inaccurate assumptions we make about the image, such as the image being a perfect sphere.

Let $\mathbf{X}$ and $\mathbf{y}$ be the true measurements. Recall that in the least squares problem, we want to solve for $\mathbf{w}$ in $\min_{\mathbf{w}} \|\mathbf{X}\mathbf{w} - \mathbf{y}\|$. We measure the error as the difference between $\mathbf{X}\mathbf{w}$ and $\mathbf{y}$, which can be viewed as adding an error term $\boldsymbol{\epsilon}_{\mathbf{y}}$ such that the equation $\mathbf{X}\mathbf{w} = \mathbf{y} + \boldsymbol{\epsilon}_{\mathbf{y}}$ has a solution:

$$\min_{\boldsymbol{\epsilon}_{\mathbf{y}}, \mathbf{w}} \|\boldsymbol{\epsilon}_{\mathbf{y}}\|_2, \text{ subject to } \mathbf{X}\mathbf{w} = \mathbf{y} + \boldsymbol{\epsilon}_{\mathbf{y}} \tag{3}$$

Although this optimization formulation allows for errors in the measurements of $\mathbf{y}$, it does not allow for errors in the feature matrix $\mathbf{X}$ that is measured from the data. In this problem, we will explore a method called *total least squares* that allows for both error in the matrix $\mathbf{X}$ and the vector $\mathbf{y}$, represented by $\boldsymbol{\epsilon}_{\mathbf{X}}$ and $\boldsymbol{\epsilon}_{\mathbf{y}}$, respectively. For convenience, we absorb the negative sign into $\boldsymbol{\epsilon}_{\mathbf{y}}$ and $\boldsymbol{\epsilon}_{\mathbf{X}}$ and define true measurements $\mathbf{y}$ and $\mathbf{X}$ like so:

$$\mathbf{y}^{true} = \mathbf{y} + \boldsymbol{\epsilon}_{\mathbf{y}} \tag{4}$$

$$\mathbf{X}^{true} = \mathbf{X} + \boldsymbol{\epsilon}_{\mathbf{X}} \tag{5}$$

Specifically, the total least squares problem is to find the solution for $\mathbf{w}$ in the following minimization problem:

$$\min_{\boldsymbol{\epsilon}_{\mathbf{y}}, \boldsymbol{\epsilon}_{\mathbf{X}}, \mathbf{w}} \|[\boldsymbol{\epsilon}_{\mathbf{X}}, \boldsymbol{\epsilon}_{\mathbf{y}}]\|_F^2, \text{ subject to } (\mathbf{X} + \boldsymbol{\epsilon}_{\mathbf{X}})\mathbf{w} = \mathbf{y} + \boldsymbol{\epsilon}_{\mathbf{y}} \tag{6}$$

where the matrix $[\boldsymbol{\epsilon}_{\mathbf{X}}, \boldsymbol{\epsilon}_{\mathbf{y}}]$ is the concatenation of the columns of $\boldsymbol{\epsilon}_{\mathbf{X}}$ with the column vector $\mathbf{y}$. Notice that the minimization is over $\mathbf{w}$ because it's a free parameter, and it does not necessarily have to be in the objective function. Intuitively, this equation is finding the smallest perturbation to the matrix of data points $\mathbf{X}$ and the outputs $\mathbf{y}$ such that the linear model can be solved exactly. The constraint in the minimization problem can be rewritten as

$$[\mathbf{X} + \boldsymbol{\epsilon}_{\mathbf{X}}, \mathbf{y} + \boldsymbol{\epsilon}_{\mathbf{y}}] \begin{bmatrix} \mathbf{w} \\ -1 \end{bmatrix} = \mathbf{0} \tag{7}$$

(a) Let the matrix $\mathbf{X} \in \mathbb{R}^{n \times d}$ and $\mathbf{y} \in \mathbb{R}^n$ and note that $\epsilon_X \in \mathbb{R}^{n \times d}$ and $\boldsymbol{\epsilon}_{\mathbf{y}} \in \mathbb{R}^n$. Assuming that $n > d$ and $\text{rank}(\mathbf{X} + \boldsymbol{\epsilon}_{\mathbf{X}}) = d$, **explain why rank**$([\mathbf{X} + \boldsymbol{\epsilon}_{\mathbf{X}}, \mathbf{y} + \boldsymbol{\epsilon}_{\mathbf{y}}]) = d$**.**

(b) For the solution $\mathbf{w}$ to be unique, the matrix $[\mathbf{X} + \boldsymbol{\epsilon}_\mathbf{X}, \mathbf{y} + \boldsymbol{\epsilon}_\mathbf{y}]$ must have exactly d linearly independent columns. Since this matrix has d+1 columns in total, it must be rank-deficient by 1. Recall that the Eckart-Young-Mirsky Theorem tells us that the closest lower-rank matrix in the Frobenius norm is obtained by discarding the smallest singular values. Therefore, the matrix $[\mathbf{X} + \boldsymbol{\epsilon}_\mathbf{X}, \mathbf{y} + \boldsymbol{\epsilon}_\mathbf{y}]$ that minimizes

$$\|[\boldsymbol{\epsilon}_\mathbf{X}, \boldsymbol{\epsilon}_\mathbf{y}]\|_F^2 = \|[\mathbf{X}^{true}, \mathbf{y}^{true}] - [\mathbf{X}, \mathbf{y}]\|_F^2$$

is given by

$$[\mathbf{X} + \boldsymbol{\epsilon}_\mathbf{X}, \mathbf{y} + \boldsymbol{\epsilon}_\mathbf{y}] = U \begin{bmatrix} \Sigma_d & \\ & 0 \end{bmatrix} V^\top$$

where $[\mathbf{X}, \mathbf{y}] = U\Sigma V^\top$.

Suppose we express the SVD of $[\mathbf{X}, \mathbf{y}]$ in terms of submatrices and vectors:

$$[\mathbf{X}, \mathbf{y}] = \begin{bmatrix} \mathbf{U}_{xx} & \mathbf{u}_{xy} \\ \mathbf{u}_{yx}^\top & u_{yy} \end{bmatrix} \begin{bmatrix} \Sigma_d & \\ & \sigma_{d+1} \end{bmatrix} \begin{bmatrix} \mathbf{V}_{xx} & \mathbf{v}_{xy} \\ \mathbf{v}_{yx}^\top & v_{yy} \end{bmatrix}^\top$$

$\mathbf{u}_{xy} \in \mathbb{R}^{n-1}$ is the first $(n-1)$ elements of the $(d+1)$-th column of $\mathbf{U}$, $\mathbf{u}_{yx}^\top \in \mathbb{R}^d$ is the first d elements of the n-th row of $\mathbf{U}$, u_{yy} is the n-th element of the $(d+1)$-th column of $\mathbf{U}$, $\mathbf{U}_{xx} \in \mathbb{R}^{(n-1)\times d}$ is the $(n-1) \times d$ top left submatrix of $\mathbf{U}$.

Similarly, $\mathbf{v}_{xy} \in \mathbb{R}^d$ is the first d elements of the $(d+1)$-th column of $\mathbf{V}$, $\mathbf{v}_{yx}^\top \in \mathbb{R}^d$ is the first d elements of the $(d+1)$-th row of $\mathbf{V}$, v_{yy} is the $(d+1)$-th element of the $(d+1)$-th column of $\mathbf{V}$, $\mathbf{V}_{xx} \in \mathbb{R}^{d\times d}$ is the $d \times d$ top left submatrix of $\mathbf{V}$. σ_{d+1} is the $(d+1)$-th eigenvalue of $[\mathbf{X}, \mathbf{y}]$. Σ_d is the diagonal matrix of the d largest singular values of $[\mathbf{X}, \mathbf{y}]$

Using this information show that

$$[\boldsymbol{\epsilon}_\mathbf{X}, \boldsymbol{\epsilon}_\mathbf{y}] = -\begin{bmatrix} \mathbf{u}_{xy} \\ u_{yy} \end{bmatrix} \sigma_{d+1} \begin{bmatrix} \mathbf{v}_{xy} \\ v_{yy} \end{bmatrix}^\top$$

(c) **Using the result from the previous part and the fact that v_{yy} is not 0, find a nonzero solution to $[\mathbf{X} + \boldsymbol{\epsilon}_\mathbf{X}, \mathbf{y} + \boldsymbol{\epsilon}_\mathbf{y}] \begin{bmatrix} \mathbf{w} \\ -1 \end{bmatrix} = \mathbf{0}$ and thus solve for w in Equation (7).**

HINT: Looking at the last column of the product $[\mathbf{X}, \mathbf{y}]\mathbf{V}$ may or may not be useful for this problem, depending on how you solve it.

(d) From the previous part, you can see that $\begin{bmatrix} \mathbf{w} \\ -1 \end{bmatrix}$ is a right-singular vector of $[\mathbf{X}, \mathbf{y}]$. **Show that**

$$(\mathbf{X}^\top\mathbf{X} - \sigma_{d+1}^2 I)\mathbf{w} = \mathbf{X}^\top\mathbf{y} \tag{14}$$

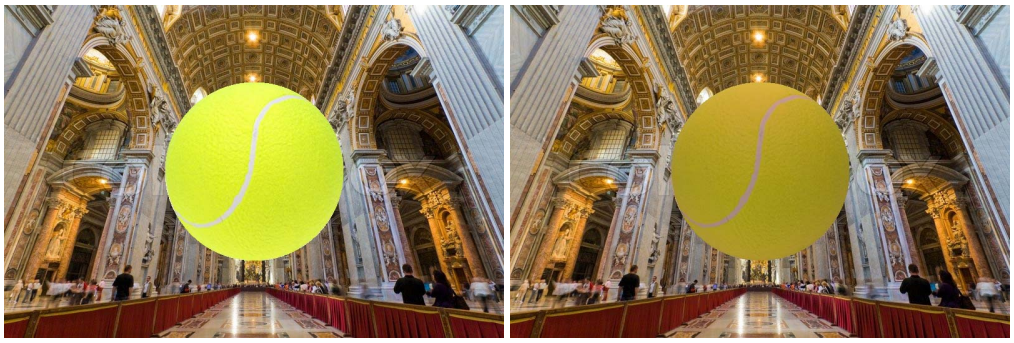

Figure 1: Tennis ball pasted on top of image of St. Peter's Basillica without lighting adjustment (left) and with lighting adjustment (right)

Notice that this result is like ridge regression, but with an appropriately chosen *negative* regularization constant! This can be hypothetically interepreted as coming from the belief that there is some ambient noise at the σ_{d+1}^2 level that is corrupting all measurements.

(e) In this problem, we will use total least squares to approximately learn the lighting in a photograph, which we can then use to paste new objects into the image while still maintaining the realism of the image. You will be estimating the lighting coefficients for the interior of St. Peter's Basillica, and you will then use these coefficients to change the lighting of an image of a tennis ball so that it can be pasted into the image. In Figure 1, we show the result of pasting the tennis ball in the image without adjusting the lighting on the ball. The ball looks too bright for the scene and does not look like it would fit in with other objects in the image.

To convincingly add a tennis ball to an image, we need to need to apply the appropriate lighting from the environment onto the added ball. To start, we will represent environment lighting as a spherical function $\mathbf{f}(\mathbf{n})$ where $\mathbf{n}$ is a 3 dimensional unit vector ($\|\mathbf{n}\|_2 = 1$), and $\mathbf{f}$ outputs a 3 dimensional color vector, one component for red, green, and blue light intensities. Because $\mathbf{f}(\mathbf{n})$ is a spherical function, the input $\mathbf{n}$ must correspond to a point on a sphere. The function $\mathbf{f}(\mathbf{n})$ represents the total incoming light from the direction $\mathbf{n}$ in the scene. The lighting function of a spherical object $\mathbf{f}(\mathbf{n})$ can be approximated by the first 9 spherical harmonic basis functions.

The first 9 unnormalized sperical harmonic basis functions are given by:

$$
\begin{aligned}
L_1 &= 1 \\
L_2 &= y \\
L_3 &= x \\
L_4 &= z \\
L_5 &= xy \\
L_6 &= yz \\
L_7 &= 3z^2 - 1
\end{aligned}
$$

Figure 2: Image of a spherical mirror inside of St. Peter's Basillica

$$L_8 = xz$$
$$L_9 = x^2 - y^2$$

where $\mathbf{n} = [x, y, z]^\top$. The lighting function can then be approximated as

$$\mathbf{f}(\mathbf{n}) \approx \sum_{i=1}^{9} \boldsymbol{\gamma}_i L_i(\mathbf{n})$$

$$\begin{bmatrix} — & \mathbf{f}(\mathbf{n_1}) & — \\ — & \mathbf{f}(\mathbf{n_2}) & — \\ & \vdots & \\ — & \mathbf{f}(\mathbf{n_n}) & — \end{bmatrix}_{n\times 3} = \begin{bmatrix} L_1(\mathbf{n_1}) & L_2(\mathbf{n_1}) & ... & L_9(\mathbf{n_1}) \\ L_1(\mathbf{n_2}) & L_2(\mathbf{n_2}) & ... & L_9(\mathbf{n_2}) \\ & \vdots & & \\ L_1(\mathbf{n_n}) & L_2(\mathbf{n_n}) & ... & L_9(\mathbf{n_n}) \end{bmatrix}_{n\times 9} \begin{bmatrix} — & \boldsymbol{\gamma}_1 & — \\ — & \boldsymbol{\gamma}_2 & — \\ & \vdots & \\ — & \boldsymbol{\gamma}_9 & — \end{bmatrix}_{9\times 3}$$

where $L_i(\mathbf{n})$ is the ith basis function from the list above.

The function of incoming light $\mathbf{f}(\mathbf{n})$ can be measured by photographing a spherical mirror placed in the scene of interest. In this case, we provide you with an image of the sphere as seen in Figure 2. In the code provided, there is a function extractNormals(img) that will extract the training pairs $(\mathbf{n}_i, \mathbf{f}(\mathbf{n}_i))$ from the image. An example using this function is in the code.

Use the spherical harmonic basis functions to create a 9 dimensional feature vector for each sample. Use this to formulate an ordinary least squares problem and solve for the unknown coefficients $\boldsymbol{\gamma}_i$. Report the estimated values for $\boldsymbol{\gamma}_i$ and include a visualization of the approximation using the provided code. We have provided the starter code in the iPython notebook. The code provided will load the images, extracts the training data, relights the tennis ball with incorrect coefficients, and saves the results. Your task is to compute the basis functions and solve the least squares problems to provide the code with the correct coefficients.

We mark the corresponding part to fill in as the "TODO". In the following parts, you will also implement other least squares. Therefore we code in the style of first loading and pre-processing data, and then conduct each least square one by one.

If you run the starter code locally, you will need to Python packages `numpy` and `scipy`. Because the resulting data set is large, we reduce it in the code by taking every 50^{th} entry in the data. This is done for you in the starter code, but you can try using the entire data set or reduce it by a different amount.

(f) When we extract from the data the direction $\mathbf{n}$ to compute $(\mathbf{n}_i, \mathbf{f}(\mathbf{n}_i))$, we make some approximations about how the light is captured on the image. We also assume that the spherical mirror is a perfect sphere, but in reality, there will always be small imperfections. Thus, our measurement for $\mathbf{n}$ contains some error, which makes this an ideal problem to apply total least squares. **Solve this problem with total least squares by allowing perturbations in the matrix of basis functions. Report the estimated values for γ_i and include a visualization of the approximation.** The output image will be visibly wrong, and we'll explore how to fix this problem in the next part. Your implementation may only use the SVD and the matrix inverse functions from the linear algebra library in numpy as well as np.linalg.solve.

(g) In the previous part, you should have noticed that the visualization is drastically different than the one generated using least squares. Recall that in total least squares we are minimizing $\|[\boldsymbol{\epsilon}_{\mathbf{X}}, \boldsymbol{\epsilon}_{\mathbf{y}}]\|_F^2$. Intuitively, to minimize the Frobenius norm of components of both the inputs and outputs, the inputs and outputs should be on the same scale. However, this is not the case here. Color values in an image will typically be in $[0, 255]$, but the original image had a much larger range. We compressed the range to a smaller scale using tone mapping, but the effect of the compression is that relatively bright areas of the image become less bright. As a compromise, we scaled the image colors down to a maximum color value of 384 instead of 255. Thus, the inputs here are all unit vectors, and the outputs are 3 dimensional vectors where each value is in $[0, 384]$. **Propose a value by which to scale the outputs $\mathbf{f}(\mathbf{n}_i)$ such that the values of the inputs and outputs are roughly on the same scale. Solve this scaled total least squares problem, report the computed spherical harmonic coefficients and provide a rendering of the relit sphere.**

4 Linear Regression in Overparameterized Regime

Note: For this problem we 0-index every vector so the first entry of a vector $\mathbf{x} \in \mathbb{R}^n$ is referred to as x_0 and the last entry is x_{n-1}.

In this problem we study linear regression in the overparameterized regime where the number of features d is greater than the number of training points n. The setup for this problem is *highly* stylized and simplified to reduce the problem to the minimal example required to demonstrate the kind of effects from overparameterization that we want you to see and understand. Real problems are unlikely to be, for example, 1-sparse, but as you will see in many cases more complex interactions can be viewed through an appropriate change of basis to create an equivalent simpler problem.

For the sake of simplicity and ease of visualization, assume that we are trying to learn a one-dimensional function $f(x)$ for $x \in [0, 1]$. For mathematical ease, consider a standard complex Fourier featurization:

$$\phi_k^u(x) = e^{j2\pi kx}$$

for $k = \{0, 1, \ldots, d-1\}$. Notice that this featurization is orthonormal relative to a uniform test-distribution on $[0, 1]$. (Here, we have to remember that when dealing with complex vectors, we need to take a conjugate transpose when computing an inner product, and thus when dealing with the relevant complex inner product between two functions relative to this test-distribution, $\langle f, g \rangle = \int_0^1 \bar{g}(t) f(t) dt$.)

Assume that the number of features $d = (M+1)n$ for a positive integer M. Consider the case where the true function we are trying to learn is 1-sparse and consists of a single "low frequency" feature, i.e.

$$y = \phi_t^u(x)$$

for $0 \leq t < n$.

Such a restriction is required if we wish to have any hope of linear regresssion to succeed because the first n features together span the entire space of $\mathbb{C}^n$ and can perfectly fit any n training points.

Assume that the training points are evenly spaced, $x_i = \frac{i}{n}$ for $i \in \{0, 1, \ldots, n-1\}$ in the interval $[0, 1)$.

We have access to observations $\mathbf{y} \in \mathbb{C}^n$ where

$$\mathbf{y} = \mathbf{\Phi}_t^u + \boldsymbol{\epsilon}_c$$

where $\mathbf{\Phi}_t^u \in \mathbb{C}^n$ is the column entry with entries $\phi_t^u(x_0), \phi_t^u(x_1), \ldots, \phi_t^u(x_{n-1})$. Here $\boldsymbol{\epsilon}_c$ refers to the noise in the training data and we assume $\boldsymbol{\epsilon}_c \sim \mathcal{CN}(0, \sigma^2 \mathbf{I}_n)$.

Note: If $\boldsymbol{\epsilon}_c \sim \mathcal{CN}(0, \sigma^2 \mathbf{I}_n)$ then it can be expressed as $\boldsymbol{\epsilon}_c = \boldsymbol{\epsilon}_{Re} + j\boldsymbol{\epsilon}_{Im}$ where $\boldsymbol{\epsilon}_{Re}$ and $\boldsymbol{\epsilon}_{Im}$ are i.i.d distributed as $\mathcal{N}(0, \frac{1}{2}\sigma^2 \mathbf{I}_n)$. Further, the k^{th} entry of $\boldsymbol{\epsilon}_c$ is distributed as $\epsilon_k \sim \mathcal{CN}(0, \sigma^2)$ and ϵ_k are independent across k.

Instead of performing linear regression using this featurization we first scale the features to obtain *scaled features*,

$$\phi_k(x) = c_k \phi_k^u(x)$$

where $c_k \in \mathbb{R}$ for $k = 0, 1, \ldots, d-1$. By scaling features appropriately we can favor some features (e.g. low frequency features) and then hope to preserve the true signal in these features. Note that these scalings c_k are *chosen*, not learned. Let $\mathbf{\Phi}_t$ denote the column vector with entries $\phi_t(x_0), \phi_t(x_1), \ldots, \phi_t(x_{n-1})$, and similarly $\mathbf{\Phi}_t^u$ the unscaled version of the same thing. To do learning, we solve the minimum norm interpolation problem using the scaled features:

$$\hat{\boldsymbol{\beta}} = \arg\min_{\boldsymbol{\beta}} \left\| \boldsymbol{\beta} \right\|_2$$
$$\text{s.t. } \mathbf{y} = \sum_{k=0}^{d-1} \beta_k \mathbf{\Phi}_k,$$

Having found $\hat{\boldsymbol{\beta}}$ we convert it into an equivalent set of coefficients $\hat{\boldsymbol{\alpha}}$ to obtain,

$$\mathbf{y} = \sum_{k=0}^{d-1} \hat{\alpha}_k \mathbf{\Phi}_k^u,$$

where $\hat{\alpha}_k = c_k\hat{\beta}_k$.

The test data $x_{\text{test}}, y_{\text{test}}$ is drawn from the distribution p and is noiseless, i.e $y_{\text{test}} = \phi_t^u(x_{\text{test}})$. Recall that the features $\{\phi_k^u\}_{k=0,\ldots,d-1}$ are orthornormal with respect to the test distribution.

We make a prediction at test time using the coefficients $\hat{\boldsymbol{\alpha}}$,

$$y_{\text{pred}} = \sum_{k=0}^{d-1} \hat{\beta}_k \phi_k(x_{\text{test}}).$$

or equivalently,

$$y_{\text{pred}} = \sum_{k=0}^{d-1} \hat{\alpha}_k \phi_k^u(x_{\text{test}}).$$

You may ask why we would think about predicting using $\hat{\boldsymbol{\alpha}}$ and the normalized features $\phi_k^u(x_{\text{test}})$ rather than the scaled $\phi_k(x_{\text{test}})$ and directly learned $\hat{\boldsymbol{\beta}}$. In practice, we would always use $\hat{\boldsymbol{\beta}}$ since that's what we see when we train, but for the purpose of this problem the prediction using $\hat{\boldsymbol{\alpha}}$ is identical, and error analysis using $\hat{\boldsymbol{\alpha}}$ is easier. We measure how well we learn the function by computing,

$$\mathcal{E}_{\text{pred}} = \mathbb{E}\left[|y_{\text{test}} - y_{\text{pred}}|^2\right],$$

where the expectation is both over the randomness in the test point and the randomness in the training noise.

To obtain low prediction error and for linear regression to succeed in this regime, we must be able to preserve the coefficient for the true feature and dissipate/absorb noise harmlessly across the rest of the coefficients. In other words, we need the contamination to stay small.

In this problem we will see that using an appropriate bi-level scaling model that favors lower frequency features we can achieve both.

The most notable thing about this feature family and the particular choice of training points that we have chosen is that the features are arranged into families of perfect aliases as far as these training points are concerned. If t is the index of a particular feature in the first n features, then $A(t)$ captures all the indices of the perfect aliases of that feature. And $R(t)$ includes the original feature as well.

$$A(t) = \{t + n, t + 2n, \ldots, t + Mn\}$$
$$R(t) = \{t\} \cup A(t).$$

and notice that:

$$\boldsymbol{\Phi}_k^u = \boldsymbol{\Phi}_t^u, \; k \in A(t) \tag{8}$$

$$\langle \boldsymbol{\Phi}_t^u, \boldsymbol{\Phi}_k^u \rangle = (\boldsymbol{\Phi}_k^u)^H \boldsymbol{\Phi}_t^u = \begin{cases} n, & k \in R(t) \\ 0, & k \notin R(t) \end{cases}. \tag{9}$$

Here we use the $(\boldsymbol{\Phi}_k^u)^H$ to denote the conjugate-transpose of the vector $\boldsymbol{\Phi}_k^u$.

(a) **1-sparse noiseless setting.** First consider the case where the true function is $y = f(x) = \phi_t^u(x)$ and we have no noise in the training data, i.e $\mathbf{y} = \mathbf{\Phi}_t^u$. **Show that**

$$\hat{\alpha}_k = \begin{cases} \frac{c_k^2}{V}, & k \in R(t), \\ 0, & \text{otherwise}, \end{cases}$$

where

$$V = \sum_{k \in R(t)} c_k^2.$$

(*Hint: The following steps might be useful. First show that it suffices to consider the problem restricted to the indices $k \in R(t)$ and transform the problem to:*

$$\hat{\xi} = \arg\min_{\xi} \left\|\xi\right\|_2$$
$$s.t. \quad \sum_{j=0}^{M} \xi_j \tilde{c}_j = 1,$$

where, $\tilde{c}_k = c_{t+kn}$ for $k = 0, 1, \ldots, M$.
Then either use Cauchy-Schwartz, solve the optimization problem using the Lagrangian, or recognize this as a particularly simple case where the Moore-Penrose Pseudoinverse applies and has a particularly trivial form.)

(b) **Pure noise.** Next consider the case where the true function $y = f(x) = 0$ but the observations are noisy. Thus we have pure noise, $\mathbf{y} = \boldsymbol{\epsilon}_\mathbf{c} \sim \mathcal{CN}(0, \sigma^2 \mathbf{I}_n)$, i.e. **Show that** we can represent $\boldsymbol{\epsilon}_\mathbf{c}$ using the first n features as

$$\boldsymbol{\epsilon}_c = \mathbf{\Phi}^u \boldsymbol{\delta},$$

where

$$\mathbf{\Phi}^u = \begin{bmatrix} \mathbf{\Phi}_0^u & \mathbf{\Phi}_1^u & \ldots & \mathbf{\Phi}_{n-1}^u \end{bmatrix},$$

and $\boldsymbol{\delta} \sim \mathcal{CN}\left(0, \frac{1}{n}\sigma^2 \mathbf{I}_n\right)$.

(*Hint: You may use without proof that if $\boldsymbol{\epsilon}_\mathbf{c} \sim \mathcal{CN}(0, \sigma^2 \mathbf{I}_n)$ then $\mathbf{B}\boldsymbol{\epsilon}_\mathbf{c} \sim \mathcal{CN}(0, \sigma^2 \mathbf{B}\mathbf{B}^H)$. Notice the connection between the property here and the orthogonality of the appropriate DFT matrix.*)

(c) **1-sparse noisy setting. Computing $\hat{\boldsymbol{\alpha}}$**
Finally consider the case where the true function is $y = f(x) = \phi_t^u(x)$ and the observations are noisy, $\mathbf{y} = \mathbf{\Phi}_t^u + \boldsymbol{\epsilon}_c$. **Show that** the entries of $\hat{\boldsymbol{\alpha}}$ are given by,

$$\hat{\alpha}_k = \frac{c_k^2}{V(q)}(1 + \delta_t), k \in R(q), q = t$$
$$\hat{\alpha}_k = \frac{c_k^2}{V(q)}\delta_q, k \in R(q), q \neq t,$$

where $V(q) = \sum_{k \in R(q)} c_k^2$ and q ranges from 0 to $n-1$. Recall that $0 \le t < n-1$. Here $\boldsymbol{\delta}$ is as in the previous part, i.e $\boldsymbol{\epsilon}_c = \boldsymbol{\Phi}^u \boldsymbol{\delta}$.

(HINT: Invoke the linearity of the Moore-Penrose pseudoinverse and leverage the previous two parts.)

(d) *1-sparse noisy setting. Computing the expected squared error on test points.*
Consider the same setting as the previous part. The prediction error is defined as,

$$\begin{aligned}\mathcal{E}_{\text{pred}} &= \mathbb{E}\left[|y_{\text{test}} - y_{\text{pred}}|^2\right] \\ &= \mathbb{E}\left[\left|\phi_t^u(x_{\text{test}}) - \sum_{k=0}^{d-1} \hat{\alpha}_k \phi_k^u(x_{\text{test}})\right|^2\right],\end{aligned}$$

where the expectation is both over the randomness in the test point and the randomness in the training noise. Recall that $0 \le t < n-1$ and $V(q) = \sum_{k \in R(q)} c_k^2$.

Show that

$$\mathcal{E}_{\text{pred}} = (1 - \mathsf{SU})^2 + \mathsf{CN}_s^2 + \mathsf{CN}_\epsilon^2,$$

where

$$\begin{aligned}\mathsf{SU} &= \frac{c_t^2}{V(t)}, \\ \mathsf{CN}_s^2 &= \frac{1}{V^2(t)} \sum_{k \in A(t)} c_k^4, \\ \mathsf{CN}_\epsilon^2 &= \frac{\sigma^2}{n}\left(\sum_{q=1}^{n} \frac{1}{V^2(q)} \sum_{k \in R(q)} c_k^4\right).\end{aligned}$$

The first term SU captures how much of the true signal survives the learning process. The second term CN_s^2 captures the impact of (external) feature contamination that is coming from the learning process here, and has nothing to do with any noise in the training data. The last term CN_ϵ^2 captures the impact of both internal and external feature contamination that is the effect of observation noise in the training data.

Hint 1: Compute conditional expectation conditioned on the training noise first and use the iterated expectation property $\mathbb{E}[\mathbb{E}[X|Y]] = \mathbb{E}[X]$.
Hint 2: Recall that the features $\boldsymbol{\Phi}_k^u$ *are orthonormal with respect to the distribution of the test points.*
Hint 3: $|\mathbf{y}|^2 = \mathbf{y}^H \mathbf{y}$.
Hint 4: If $\epsilon_i \sim \mathcal{CN}(0, \sigma^2)$ *then* $\mathbb{E}[|\epsilon_i|^2] = \sigma^2, \mathbb{E}[\epsilon] = \mathbb{E}[\epsilon^H] = 0$.

(e) *Bi-level scaling model* Now consider a bi-level scaling model with parameters (γ, s) that selects a fraction $\gamma \in (\frac{s}{d}, 1)$ of energy to put on the favored set of $s < n$ features. The idea of this

caricature model is to capture the idea that there are some dominant directions of variation in our features and that the true pattern is representable in terms of those directions. This is the simplest model that exhibits the interesting behavior that we want to understand.

Specifically, we assume that the scalings are

$$c_k = \begin{cases} \sqrt{\frac{\gamma d}{s}}, & 0 \le k < s \\ \sqrt{\frac{(1-\gamma)d}{d-s}}, & \text{otherwise.} \end{cases}$$

From this point forward, assume that the index corresponding to the true feature $t < s$.

To make our calculations easier we will make the assumption that the number of features d is large relative to the number of training data points (and as a consequence large relative to s). Thus we approximately have

$$M = \frac{d}{n} - 1 \approx \frac{d}{n}$$
$$\frac{d-s}{d} \approx 1.$$

Show that with the bi-level weighting model we have

$$\mathsf{SU} \approx \frac{1}{1+\lambda}$$
$$\mathsf{CN}_s^2 \approx \frac{n}{d}\left(\frac{\lambda^2}{(1+\lambda)^2}\right)$$
$$\mathsf{CN}_\epsilon^2 \approx \frac{\sigma^2 s}{n}\left(\frac{1+\frac{n\lambda^2}{d}}{(1+\lambda)^2}\right) + \frac{\sigma^2(n-s)}{d},$$

where

$$\lambda = \frac{s(1-\gamma)}{n\gamma}.$$

Notice here that $\lambda \approx M\frac{c_{small}^2}{c_{big}^2}$ and so reflects the degree of overparameterization as compared to the ratio between the scalings of the unfavored vs favored features.

(f) *Asymptotically perfect prediction.* Finally, suppose we can vary n, γ, d, s (and thus λ) to send prediction squared error to zero. (The idea is to grow n to infinity, while growing both s and d along with it. The question is how these quantities must scale together if we want the prediction error to go to zero as the number of noisy training points n goes to infinity.) Here, the observation noise variance σ^2 is a fixed constant.

Show the following conditions together are necessary and sufficient:

$$\lambda \to 0$$
$$\frac{s}{n} \to 0$$

$$\frac{n}{d} \to 0.$$

In other words, tell us why if any one of these conditions is violated, the prediction squared error will not go to zero, and that if all the conditions are satisfied, then the prediction squared error will go to zero. That is what it means for a collection of conditions to be necessary and sufficient.

Notice that each of these conditions has an intuitive meaning. The condition $\frac{s}{n} \to 0$ is the same as $s = o(n)$ which means that the number of favored features must be sublinear in n. The condition $\frac{n}{d} \to 0$ says that the degree of overparameterization has to increase with n — understandable because the noise energy has to be absorbed somewhere in an interpolating solution, and that somewhere is in the extra aliases. The condition $\lambda \to 0$ is to prevent over-regularization by the extra features and it translates into a condition that γ cannot shrink too fast — in particular, it must stay greater (in an order sense) than $\frac{s}{n}$ which means that the potentially true features must be favored significantly over the other features for this kind of interpolative learning to succeed. Since $\lambda \approx M\frac{c_{small}^2}{c_{big}^2}$, this means that the favored features have to be scaled heavily enough relative to the regularizing extra features so that combined regularizing effect of the sheer quantity of them does not overwhelm any signal that might be present in the favored features.

(g) Suppose that instead of being 1-sparse,

$$\mathbf{y}_{true} = a_1\mathbf{\Phi}^u_{t_0} + a_2\mathbf{\Phi}^u_{t_1} + \boldsymbol{\epsilon}_c$$

with $t_0, t_1 < s$. **Comment on the behavior of the prediction error in this case. Is it still possible to drive the error to 0?**

(*Hint: change the features in some way that preserves orthonormality in the test set while returning the problem to its original formulation.*)

(h) In the Jupyter notebook, **implement least-squares regression to find $\hat{\alpha}$ then explore the behavior of overparameterized regression with n, d, s, γ, and σ, then comment on the results as instructed.**

(i) Finally in the Jupyter notebook, **go over the demo parts that relate the regularizating effect of features to ridge regression and explore the effect that a training point has on the learned function via the impulse response.**

5 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:

- Anant Sahai
- Fanny Yang
- Josh Sanz
- Mark Velednitsky
- Peter Wang
- Philipp Moritz
- Vignesh Subramanian