EECS 189 Introduction to Machine Learning
Fall 2020 HW4

This homework is due **Wednesday, September 30 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 Tikhonov Regularization and Weighted Least Squares

In this problem, we look at Tikhonov regularization from a probabilistic standpoint.

The main goal is to deepen your understanding of how priors and thus the aligned regularization impact the MAP estimator. First, you will work out how introducing a certain probabilistic prior before maximizing the posterior is equivalent to adding the Tikhonov regularization term: by adding the Tikhonov regularization term, we effectively constrain our optimization space. Similarly, using a probabilistic prior drives our optimization towards solutions that have a high (prior) probability of occurring. In the second half of the problem you will then do some simulations to see how different priors influence the estimator explicitly, as well as how this effect changes as the number of samples grows.

(a) Let $\mathbf{x} \in \mathbb{R}^d$ be a d-dimensional vector and $Y \in \mathbb{R}$ be a one-dimensional random variable. Assume a linear model: $Y = \mathbf{x}^\top \mathbf{w} + Z$ where $Z \in \mathbb{R}$ is a standard Gaussian random variable $Z \sim N(0, 1)$ and $\mathbf{w} \in \mathbb{R}^d$ is a d-dimensional Gaussian random vector $\mathbf{w} \sim N(0, \mathbf{\Sigma})$. $\mathbf{\Sigma}$ is a known symmetric positive-definite covariance matrix. Note that $\mathbf{w}$ is independent of the observation noise. **What is the conditional distribution of Y given x and w?**

(b) (Tikhonov regularization) Let us assume that we are given n training data points $\{(\mathbf{x}_1, Y_1), (\mathbf{x}_2, Y_2), \cdots, (\mathbf{x}_n, Y_n)\}$ which we know are generated i.i.d. according to the model of $(\mathbf{x}, Y)$ in the previous part, i.e. we draw one $\mathbf{w}$ and use this to generate all Y_i given distinct but arbitrary $\{\mathbf{x}_i\}_{i=1}^n$, but the observation noises Z_i vary across the different training points in an i.i.d. fashion. **Derive the posterior distribution of w given the training data. Based on your result, what is the MAP estimate of w? Comment on how Tikhonov regularization is a generalization of ridge regression from a probabilistic perspective.**

Note: $\mathbf{w}$ and $\mathbf{Y} = (Y_1, Y_2, \ldots, Y_n)$ are jointly Gaussian in this problem given $\{\mathbf{x}_i\}_{i=1}^n$.

[Hint: (You may or may not find this useful) If the probability density function of a random variable is of the form

$$f(\mathbf{v}) = C \cdot \exp\{-\frac{1}{2}\mathbf{v}^\top \mathbf{A}\mathbf{v} + \mathbf{b}^\top \mathbf{v}\},$$

where C is some constant to make $f(\mathbf{v})$ integrates to 1, then the mean of $\mathbf{v}$ is $\mathbf{A}^{-1}\mathbf{b}$. This can be used to help complete squares if you choose to go that way.

(c) We have so far assumed that the observation noise has a standard normal distribution. While this assumption is nice to work with, we would like to be able to handle a more general noise model. In particular, we would like to extend our result from the previous part to the case where the observation noise variables Z_i are no longer independent across samples, i.e. $\mathbf{Z}$ is no longer $N(\mathbf{0}, \mathbb{I}_n)$ but instead distributed as $N(\boldsymbol{\mu}_z, \mathbf{\Sigma}_z)$ for some mean μ_z and some covariance $\mathbf{\Sigma}_z$ (still independent of the parameter $\mathbf{w}$). **Derive the posterior distribution of w by appropriately changing coordinates.** We make the reasonable assumption that the $\mathbf{\Sigma}_z$ is invertible, since otherwise, there would be some dimension in which there is no noise.

(Hint: Write $\mathbf{Z}$ as a function of a standard normal Gaussian vector $\mathbf{V} \sim N(\mathbf{0}, \mathbb{I}_n)$ and use the result in (b) for an equivalent model of the form $\widetilde{\mathbf{Y}} = \widetilde{\mathbf{X}}\mathbf{w} + \mathbf{V}$.)

(d) (Compare the effect of different priors) In this part, you will generate plots that show how different priors on $\mathbf{w}$ affect our prediction of the true $\mathbf{w}$ which generated the data points. Pay attention to how the amount of data used and the choice of prior relative to the true $\mathbf{w}$ we use are related to the final prediction.

Do the following for $\mathbf{\Sigma} = \mathbf{\Sigma}_1, \mathbf{\Sigma}_2, \mathbf{\Sigma}_3, \mathbf{\Sigma}_4, \mathbf{\Sigma}_5, \mathbf{\Sigma}_6$ respectively, where

$$\mathbf{\Sigma}_1 = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}; \quad \mathbf{\Sigma}_2 = \begin{bmatrix} 1 & 0.25 \\ 0.25 & 1 \end{bmatrix}; \quad \mathbf{\Sigma}_3 = \begin{bmatrix} 1 & 0.9 \\ 0.9 & 1 \end{bmatrix};$$
$$\mathbf{\Sigma}_4 = \begin{bmatrix} 1 & -0.25 \\ -0.25 & 1 \end{bmatrix}; \quad \mathbf{\Sigma}_5 = \begin{bmatrix} 1 & -0.9 \\ -0.9 & 1 \end{bmatrix}; \quad \mathbf{\Sigma}_6 = \begin{bmatrix} 0.1 & 0 \\ 0 & 0.1 \end{bmatrix}$$

Under the priors above, the coordinates of the (random) vector $\mathbf{w}$ are: (1) independent with large variance, (2) mildly positively correlated, (3) strongly positively correlated, (4) mildly negatively correlated, (5) strongly negatively correlated, and (6) independent with small variances respectively.

With the starter code in the corresponding part of the iPython notebook, generate $n \in \{5, 6, \ldots, 500\}$ data points $Y = x_1 + x_2 + Z$ with $x_1, x_2 \sim N(0, 5)$ and $Z \sim N(0, 1)$ as training data (all of them independent of each other). Here, the true $\mathbf{w}$ is thus $\begin{bmatrix}1 & 1\end{bmatrix}^\top$). Note that the randomness of x_i here is only for the generation of the plot but in our probabilistic model for parameter estimation we consider them as fixed and given. The starter code helps you generate an interactive plot where you can adjust the covariance prior and the number of samples used to calculate the posterior. **Include 6 plots of the contours of the posteriors on w for various settings of $\boldsymbol{\Sigma}$ and number of data points. Write the covariance prior and number of samples for each plot. What do you observe as the number of data points increases?**

(e) (Influence of Priors) For our simulations, we will generate n training data samples from $Y = x_1 + x_2 + Z$ where again $x_1, x_2 \sim N(0, 5)$ and $Z \sim N(0, 1)$ (all of them independent of each other) as before. Notice that the true parameters $w_1 = 1, w_2 = 1$ are moderately large and positively correlated with each other. We want to quantitatively understand how the effect of the prior influences the mean square error as we get more training data. This should corroborate the qualitative results you saw in the previous part.

In this case, we could directly compute the "test error" for a given estimator $\widehat{\mathbf{w}}$ of the parameter $\mathbf{w}$ (our prediction for Y given a new data point $\mathbf{x} = (x_1, x_2)^\top$ is then $\widehat{Y} = \widehat{w}_1 x_1 + \widehat{w}_2 x_2$). Specifically, considering $\widehat{\mathbf{w}}$ now fixed, the expected error for a randomly drawn Y given the true (but unknown) parameter vector $\mathbf{w} = (1, 1)^\top$ is equal to $\mathbb{E}_{Z,\mathbf{x}}(Y - \widehat{Y})^2 = 5(\widehat{w}_1 - 1)^2 + 5(\widehat{w}_2 - 1)^2 + 1$. We call this the *theoretical average test error*. For simplicity, we refer to it as theoretical MSE. Note that here by our choice of definition, the expectation for new test samples is over $\mathbf{x}$ as well, although our estimator is not taking the randomness of $\mathbf{x}$ in the training data into account.

In practice, the expectation with respect to the true conditional distribution of Y given $\mathbf{w}$ cannot be computed since the true $\mathbf{w}$ is unknown. Instead, we are only given a finite amount of samples from the model (which we call the *test set*, which independent of the training data, but identically distributed) so that it is only possible to compute

$$\frac{1}{n}\sum_{i=1}^{n}(Y_i - \hat{Y}_i)^2$$

which we call the *empirical average test error*. For simplicity, we refer to it as empirical MSE. Again, note that here, $\hat{Y}_i = \mathbf{x}_i^\top \widehat{\mathbf{w}}$ where $\mathbf{x}_i \in \mathbb{R}^2$ and $\widehat{\mathbf{w}}$ in your model is the solution to the least square problem with Tikhonov regularization given the training data.

With the starter code in the corresponding part of the iPython notebook, generate a test set of 500 **data points** $(\mathbf{x}_i, Y_i)$ **from the above model. Plot the empirical and theoretical mean square error between** $\hat{Y}_i$ **and** Y_i **over the test data with respect to the size of training data** n **(increase** n **from** 5 **to** 200 **in increments of** 5**).**

Note: If we just plotted both the empirical and theoretical average test errors with respect to the amount of training data for one "round" or training data, the results would still look jagged. In order to give a quantitive statement about the test error with respect to the training data n with a "smoother" plot, what we really want to know is the expectation of the theoretical average test error with respect to $\mathbf{w}$ and the training samples $(\mathbf{x}_i, Y_i)$, i.e. $\mathbb{E}_{(\mathbf{x}_1,Y_1),\ldots,(\mathbf{x}_n,Y_n)}\mathbb{E}_{Z,\mathbf{x}}(Y-\widehat{Y})^2$ (note that in this term, only $\widehat{Y}$ depends on the training data $(\mathbf{x}_1, Y_1), \ldots, (\mathbf{x}_n, Y_n)$ whereas $(\mathbf{x}, Y)$ is an independent fresh test sample). Consequently, as an approximation, it is worth replicating the entire experiment a few times (say 100 times) to get an empirical estimate of this quantity. (It is also insightful to look at the spread.) **Compare what happens for different priors as the amount of training data increases. Try plotting the theoretical MSE with logarithmic x and y-axes and explain the plot. What constitutes a "good" prior and which of the given priors are "good" choices for our particular $\mathbf{w} = (1, 1)^\top$? Describe how the influence of different priors changes with the number of data points.**

3 Kernel Ridge Regression: Theory

In traditional ridge regression, we are given a vector $\mathbf{y} \in \mathbb{R}^n$ and a matrix $\mathbf{X} \in \mathbb{R}^{n\times\ell}$, where n is the number of training points and ℓ is the dimension of the raw data points. In most practical settings we don't want to work with just the raw feature space, so we lift/distill the data points using features and replace $\mathbf{X}$ with $\mathbf{\Phi} \in \mathbb{R}^{n\times d}$, where $\phi_i^\top = \phi(\mathbf{x}_i) \in \mathbb{R}^d$. Then we solve a well-defined optimization problem that involves the matrix $\mathbf{\Phi}$ and $\mathbf{y}$ to find the parameters $\mathbf{w} \in \mathbb{R}^d$. Note the problem that arises here. If we have polynomial features of degree at most p in the raw ℓ dimensional space, then there are $d = \binom{\ell+p}{p}$ terms that we need to optimize, which can be very very large (often larger than the number of training points n). Wouldn't it be useful if instead of solving an optimization problem over d variables, we could solve an equivalent problem over n variables (where n is potentially much smaller than d), and achieve a computational runtime independent of the number of augmented features? As it turns out, the concept of kernels (in addition to a technique called the kernel trick) will allow us to achieve this goal.

When people talk about "the kernel trick," they usually mean the following: imagine that we have some machine learning algorithm, which only uses data vectors to compute scalar products with other data vectors. Then we can substitute the operation of taking inner products $\langle \mathbf{x}_i, \mathbf{x}_j \rangle$ with some other function $K(\mathbf{x}_i, \mathbf{x}_j)$ and obtain a new ML algorithm! There is however, another interesting consideration: if our algorithm only operates with inner products, and we do featurization, maybe we could use features so that those scalar products $\langle \phi(\mathbf{x}_i), \phi(\mathbf{x}_j) \rangle =: K(\mathbf{x}_i, \mathbf{x}_j)$ were easy to compute (e.g. without explicitly writing out long vectors $\phi(\mathbf{x}_i)$ and $\phi(\mathbf{x}_j)$). As it turns out, those ideas are closely connected: kernels and features provide two different views of the same thing, and it may sometimes be more convenient to think about one and sometimes about the other.

In this problem we will derive this connection between features and kernels, observe which features correspond to some particular kernels and vice versa, and see how kernel perspective can help with computation.

(a) As we already know, the following procedure gives us the solution to ridge regression in feature

space:

$$\arg\min_{\mathbf{w}} \|\mathbf{\Phi w} - \mathbf{y}\|_2^2 + \lambda \|\mathbf{w}\|_2^2 \tag{1}$$

To obtain the prediction for a test point $\mathbf{x}$ one needs to compute $\phi(\mathbf{x})\hat{\mathbf{w}}$, where $\hat{\mathbf{w}}$ is the solution to (1). In this part we will show how $\phi(\mathbf{x})\hat{\mathbf{w}}$ can be computed using only the kernel $K(\mathbf{x}_i, \mathbf{x}_j) = \phi(\mathbf{x}_i)\phi(\mathbf{x}_j)^\top$. Denote the following two objects:

$$\mathbf{k}(\mathbf{x}) := [K(\mathbf{x}, \mathbf{x}_1), K(\mathbf{x}, \mathbf{x}_2), \ldots, \mathbf{K}(\mathbf{x}, \mathbf{x}_n)]^\top, \quad \mathbf{K} = \{K(\mathbf{x}_i, \mathbf{x}_j)\}_{i,j=1}^n$$

Where the matrix $\mathbf{K}$ is the matrix filled with pairwise kernel similarity computations among the training points.

Show that

$$\phi(\mathbf{x})\hat{\mathbf{w}} = \mathbf{k}(\mathbf{x})^\top (\mathbf{K} + \lambda I)^{-1}\mathbf{y}.$$

In other words, if we define $\hat{\alpha} := (\mathbf{K} + \lambda I)^{-1}\mathbf{y}$, then

$$\phi(\mathbf{x})\hat{\mathbf{w}} = \sum_{i=1}^{n} \alpha[i] K(\mathbf{x}, \mathbf{x}_i)$$

— our prediction is a linear combination of kernel functions at different data points.

HINT: use problem 6b of HW1 to write $\hat{\mathbf{w}}$ using the Moore–Penrose inverse (pseudoinverse), and then use the explicit expression for pseudoinverse for matrices with linearly independent rows. After that you will just need to understand how $\mathbf{K}$ and $\mathbf{k}(\mathbf{x})$ can be expressed through $\mathbf{\Phi}$ and $\phi(\mathbf{x})$.

(b) (Polynomial Regression from a kernelized view) In this part, we will show that polynomial regression with a particular Tiknov regularization is the same as kernel ridge regression with a polynomial kernel for second-order polynomials. Recall that a degree 2 polynomial kernel function on $\mathbb{R}^d$ is defined as

$$K(\mathbf{x}_i, \mathbf{x}_j) = (1 + \mathbf{x}_i^\top \mathbf{x}_j)^2, \tag{2}$$

for any $\mathbf{x}_i, \mathbf{x}_j \in \mathbb{R}^\ell$. Given a dataset $(\mathbf{x}_i, y_i)$ for $i = 1, 2, \ldots, n$, **show the solution to kernel ridge regression is the same as a regularized least square solution to polynomial regression (with standard unweighted monomials as features) for $p = 2$ given the right choice of regularization for the polynomial regression.** That is, show for any new point $\mathbf{x}$ given in the prediction stage, both methods give the same prediction $\hat{y}$ with the same training data. *(Hint: you should consider a regularization term of the form $\|\mathbf{Mw}\|_2^2$ for some matrix $\mathbf{M}$. Such regularization is called* **Tikhonov regularization***.)*

(c) In general, we can show the equivalence between kernel ridge regression with a polynomial kernel $(1 + \mathbf{x}^\top \mathbf{z})^p$ of order p and a polynomial regression with pth order polynomial on $\mathbb{R}^\ell$ with an appropriately specified Tikhonov regularization,. **Comment on the computational complexity of doing polynomial regression with this Tikhonov regularization directly and that of doing kernel ridge regression at the training stage.** (That is, the complexity of finding $\boldsymbol{\alpha}$ and finding $\mathbf{w}$.)

Compare with the computational complexity of actually doing prediction as well.

(d) We will now consider kernel functions that do not directly correspond to a finite-dimensional featurization of the input points. For simplicity, we will stick to a scalar underlying raw input x. (The same style of argument can help you understand the vector case as well.) Consider the kernel function

$$k(x,z) = \exp\left(-\frac{|x-z|^2}{2\sigma^2}\right),$$

for some fixed hyperparameter σ. Assume, for a moment, that it corresponds to some featurized Euclidean vectors $\phi(x)$ and $\phi(z)$ of the inputs x and z.

Imagine what these vectors would look like. In particular, recall that $k(x,z) = \phi(x)^T\phi(z)$. **When would $\phi(x)$ and $\phi(z)$ be collinear? When would they be (almost) orthogonal?**

(Hint: think about when $k(x,z)$ is maximized, and when it goes to zero)

This is a qualitative question. If you are having trouble with it, you can just jump ahead to the next part and come back to it.

(e) We will now try to find a concrete and explicit featurization $\phi(x)$ corresponding to our kernel function. It turns out that no such finite-dimensional featurization exists. However, there exists a series $\phi_d(x)$ of d-dimensional features, such that $\phi_d(x)^T\phi_d(z)$ converges as $d \to \infty$ to $k(x,z)$. **By considering Taylor expansions, find $\phi_d(x)$.**

(Hint: focus your attention on the Taylor expansion of $e^{\frac{xz}{\sigma^2}}$ since the cross term is what determines the interaction. You can leverage the property of exponentials to pull the pure terms out so that the kernel function is written as a product involving a pure function of x, a pure function of z, and a function that has x and z interacting. This will let you pull the pure functions into $\phi_d(x)$ and $\phi_d(z)$ respectively.)

(f) Use the Jupyter notebook to implement kernel ridge regression using the ϕ_d features for fixed values of d, as well as the kernel function directly to see the behavior in the infinite-dimensional case. Consider both their effectiveness at fitting functions, as well as their one-hot "impulse" responses. **Comment on your observations.**

4 Learning 1d function with kernel regression

(a) **Do the tasks from part a of the associated jupyter notebook and report the results.**

(b) **Do the tasks from part b of the associated jupyter notebook and report the results.**

(c) **Do the tasks from part c of the associated jupyter notebook and report the results.**

(d) **Do the tasks from part d of the associated jupyter notebook and report the results.**

(e) **Do the tasks from part e of the associated jupyter notebook and report the results.**

(f) **Do the tasks from part f of the associated jupyter notebook and report the results.**

(g) **Do the tasks from part g of the associated jupyter notebook and report the results.**

(h) **Do the tasks from part h of the associated jupyter notebook and report the results.**

(i) **Do the tasks from part i of the associated jupyter notebook and report the results.**

5 Kernel Ridge Regression: Practice

In the following problem, you will implement Polynomial Ridge Regression and its kernel variant Kernel Ridge Regression, and compare them with each other. You will be dealing with a 2D regression problem, i.e., $\mathbf{x}_i \in \mathbb{R}^2$. We give you three datasets, `circle.npz` (small dataset), `heart.npz` (medium dataset), and `asymmetric.npz` (large dataset). In this problem, we choose $y_i \in \{-1, +1\}$, so you may view this question as a classification problem. Later on in the course we will learn more about logistic regression and SVMs, which can solve classification problems much better and can also leverage kernels. (Actually, you might already be able to see how you can leverage kernels for logistic regression since you've seen how it can be solved by iterating least-squares type problems.)

You are only allowed to use `numpy.*`, `numpy.linalg.*`, and `matplotlib` in the following questions. Make sure to include plots and results in your writeups.

(a) **Use `matplotlib` to visualize all the datasets and attach the plots to your writeup.** Label the points with different y values with different colors and/or shapes.

(b) **Implement polynomial ridge regression** (non-kernelized version that you should already have seen in a previous homework) to fit the datasets `circle.npz`, `asymmetric.npy`, and `heart.npz`. Use the first 80% of the data as the training dataset and the last 20% of the data as the validation dataset. **Report both the average training squared loss and the average validation squared for polynomial order $p \in \{1, \ldots, 16\}$. Use the regularization term $\lambda = 0.001$ for all p. Visualize your result and attach the heatmap plots for the learned predictions over the entire 2D domain for $p \in \{2, 4, 6, 8, 10, 12\}$ in your writeup.**

(c) **Implement kernel ridge regression** to fit the datasets `circle.npz`, `heart.npz`, and optionally (due to the computational requirements), `asymmetric.npz`. Use the polynomial kernel $K(\mathbf{x}_i, \mathbf{x}_j) = (1 + \mathbf{x}_i^\top \mathbf{x}_j)^p$. Use the first 80% data as the training dataset and the last 20% data as the validation dataset. **Report both the average training squared loss and the average validation squared loss for polynomial order $p \in \{1, \ldots, 16\}$. Use the regularization term $\lambda = 0.001$ for all p.** Code for generating heatmap plots is included in the notebook. **For `circle.npz`, also report the average training squared loss and validation squared loss for polynomial order $p \in \{1, \ldots, 24\}$ when you use only the first 15% of data as the training dataset and the final 85% of data as the validation dataset. Based on the error, comment on when you want to use a high-order polynomial in linear/ridge regression.**

(d) *Diminishing influence of the prior with growing amount of data:* With increasing amounts of data, the gains from regularization diminish. **Sample a subset of training data from the first 80% of data from `asymmetric.npz` and use the data from the last 20% of data for validation. Make a plot whose x axis is the amount of the training data and y axis is the**

validation squared loss of the non-kernelized ridge regression algorithm. Include 6 curves for hyper-parameters $\lambda \in \{0.0001, 0.001, 0.01\}$ and $p = \{5, 6\}$. Your plot should demonstrate that with same p, the validation squared loss will converge with enough data, regardless of the choice of λ. You can use log scaling on the x axis for clarity, and you need to resample the data multiple times for a given p, λ, and amount of training data in order to get a smooth curve.

(e) A popular kernel function that is widely used in various kernelized learning algorithms is called the radial basis function kernel (RBF kernel). It is defined as

$$K(\mathbf{x}, \mathbf{x}') = \exp\left(-\frac{\|\mathbf{x} - \mathbf{x}'\|_2^2}{2\sigma^2}\right). \tag{3}$$

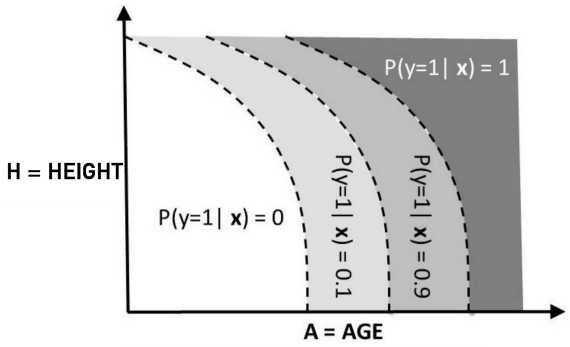
Implement the RBF kernel function for kernel ridge regression to fit the dataset `heart.npz`. Use the regularization term $\lambda = 0.001$. Report the average squared loss, visualize your result and attach the heatmap plots for the fitted functions over the 2D domain for $\sigma \in \{10, 3, 1, 0.3, 0.1, 0.03\}$ in your writeup. You may want to vectorize your kernel functions to speed up your implementation. **Comment on the effect of σ.**

(f) **For polynomial ridge regression, which of your implementation is more efficient, the kernelized one or the non-kernelized one? For RBF kernel, explain whether it is possible to implement it in the non-kernelized ridge regression. Summarize when you prefer the kernelized to the non-kernelized ridge regression.**

6 Nearest Neighbors and Bayes risk

In lecture, you saw an argument for why, if the training data is dense enough, and the underlying pattern is smooth enough, a 1-nearest neighbor estimator is going to basically approach a test squared-error that is the sum of the variance of the training noise and the irreducible noise at test time. You also saw an argument for why k-nearest neighbor estimation will do better by being able to average away the training noise. In this problem, the goal is to help you understand what the counterpart looks like for classification type problems.

A census of Middle Earth is conducted to see the extent to which we can predict that someone is an orc or an elf based on age and height. We design $x = [A, H]$ where A indicates the candidate's age and H their height. The census includes enough data for a posterior probability $P(y|x)$ where y indicates where this being is an elf.



We labels the regions above from left to right R_1 to R_4. For illustrative simplicity, we have made the conditional probability of being an elf piecewise constant so that it is constant over each of those regions.

(a) **What is the Bayes risk in each of the four regions?**

Here, the *Bayes risk* means the probability of error of the optimum Bayes classifier that knows the underlying pattern perfectly. The Bayes risk is the counterpart of the "irreducible error" since it reflects an intrinsic uncertainty that we cannot remove when we are trying to do prediction based on the given information.

(b) Assume that the training data is dense enough so that all the nearest neighbors of a test sample lie in the same region as the test sample itself. Now consider a test sample x which falls in region R_i. **For $i \in 1, 2, 3, 4$, what is the probability that x and its nearest neighbor have different labels y.**

Here, we are assuming that the training data has labels generated by the same random process that generates the labels at test time, and that the labels on training points are independent of the labels on test points.

(c) **What is the 1-nearest neighbor classification error rate in each region?**

(d) Now we generalize the problem above.

$$\begin{aligned} R1 &: P(y = 1|x) = 0 \\ R2 &: P(y = 1|x) = p \\ R3 &: P(y = 1|x) = 1 - p \\ R4 &: P(y = 1|x) = 1 \end{aligned}$$

where p is a number between 0 and 0.5.
Calculate the answers to the previous questions using this new information. Can you see why the classification performance of 1-nearest neighbor with sufficiently dense training data is never worse than twice the Bayes risk?

(e) Now, consider the previous part, but think about k-nearest neighbors where we take the majority vote of the labels for the nearest neighbors. Here, we continue with the assumptions from the previous parts — all the nearest neighbors are in the same region as the test point. Remember, in the model given above, the training labels of data points are iid conditioned on which region that they are found in, and the test point's label is also identically distributed and independent of all the training labels.

What is the probability of classification error for this k-nearest neighbor classifier in each of the regions? Can you see why the performance will approach the Bayes risk as $k \to \infty$?

7 Quantifying contamination in ridge regularization

In this problem, we want to make sure that you can understand why contamination is generically an issue with non-trivial ridge regularization, and where it is coming from at a fundamental level. Here, we are going to be leveraging the "feature augmentation" perspective on ridge-regression since that provides the key intuition.

For simplicity throughout this problem, we are going to assume that the true pattern is a proportion w^* of the first feature $x[1]$ in each sample. So $y_i = w^* x_i[1]$ when we have training data $\{(\mathbf{x}_i, y_i)\}_1^n$.

(a) We will start with something that might initially feel totally unrelated. Suppose that there were only two features, and the first and second features are just multiples of each other. Suppose $x[2] = \beta x[1]$ where β is some nonzero constant. In this case, the second feature is spiritually a perfect "alias" of the first feature for the training data. **Compute what the minimum norm solution will be to the equation $\mathbf{Xw} = \mathbf{y}$. How does the survival of the true weight in the first coefficient depend on β?**

(b) Now, we will return to ridge regression from the "augmented feature" perspective that you saw in an earlier homework. Here, assume that there are m features in $\mathbf{x}$ and the second feature is no longer a scaled version of the first feature. In the "augmented feature" perspective, you construct a new matrix $\mathbf{A} = [\mathbf{X}\ \sqrt{\lambda}\mathbf{I}]$ and solve a minimum norm problem of the type $\mathbf{A}\boldsymbol{\eta} = \mathbf{y}$ where the $\mathbf{y}$ is just the first feature column $\mathbf{X}[1]$.

Consider the new matrix $\widetilde{\mathbf{A}}$ that is obtained by dropping the first column from $\mathbf{A}$. Let $\widetilde{\boldsymbol{\eta}} = \begin{bmatrix} \widetilde{\mathbf{w}} \\ \boldsymbol{\xi} \end{bmatrix}$ and consider the minimum norm problem $\widetilde{\mathbf{A}}\widetilde{\boldsymbol{\eta}} = \mathbf{X}[1]$.

Give an expression for $\widetilde{\boldsymbol{\eta}}$ that is the minimum norm solution.

(HINT: This is just a plug-in the formula type problem. The interesting thing is that you can actually get the answer using two different formulas. The Moore-Penrose pseudo-inverse is the straightforward approach. But you can also go via the traditional solution to ridge regression by reinterpreting the dropped-column problem back as a ridge regression with a smaller parameter space.)

(c) In the previous part, you have effectively computed how for this particular ridge parameter λ, the last features together with the "ridge features" permit you to construct a synthetic perfect alias for the first feature as far as these training points are concerned. **Prove that the solution to the original ridge regression is going to have contamination that is spread across features other than the first feature in a manner that is proportional to the solution you found in the previous part.**

(HINT: The minimum-norm solution perspective will be helpful. Think about the equality constraint and what that implies in this case.)

(d) Now we want to leverage the first part of this problem. **What is the effective β for the synthetic alias?**

In other words, reinterpret the optimization for the original ridge-regression into two phases: in the first phase, you compute the cheapest alias you can for the first feature. In the second phase, you find the solution for ridge regression by optimally weighing the true feature and this cheapest alias in a manner reminisent of the first part of this problem. *(Understanding this reinterpretation is the heart of this part of the problem.)*

We can further think about $\widehat{\mathbf{w}} = \begin{bmatrix} \widehat{w}[1] \\ \check{\mathbf{w}} \end{bmatrix}$ where the $\check{\mathbf{w}} = \begin{bmatrix} \widehat{w}[2] \\ \vdots \\ \widehat{w}[m] \end{bmatrix}$ represents the learned weights on the contaminating features.

Give expressions for both $\widehat{w}[1]$ and $\check{\mathbf{w}}$ that show their proportionality to w^*.

If the underlying features were orthonormal relative to the test distribution for the input X, then $\sum_{j=2}^{m}(\check{w}[j])^*$ would represent the component of test prediction squared error that is arising from the contamination represented by $\check{\mathbf{w}}$.

Although we do not ask you to do this here, it is interesting to think about how everything above depends on λ.

8 Student Survey

Please fill out the survey located at https://forms.gle/e2s63eC88YKmnzYW6.

9 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
- Fanny Yang
- Jerome Baek
- Josh Sanz
- Kate Sanders
- Manav Rathod
- Peter Wang
- Philipp Moritz
- Rahul Arya
- Soroush Nasiriany
- Yichao Zhou
- bsword