EECS 189 Introduction to Machine Learning
Fall 2020

DIS4

This discussion was released **Friday, September 25**.

This discussion material will cover parts of Problem 3 (Kernel Ridge Regression: Theory) and Problem 4 (Learning $1d$ function with kernel regression) in HW4. It aims to give you a better understanding of kernel ridge regression, and it should get you started on the homework problems.

- We will first start with the problem on (**using Jamboard**) deriving the prediction of kernel ridge regression for a test point $\mathbf{x}$ via the *augmented design matrix perspective*.
  ($\sim$ 10 minutes.)
- Then we will bring everyone back to the main room.
- Next we will move to Jupyter notebook part and play around with $1d$ function with kernel ridge regression. We will try to do some visualizations on kernel ridge regression on $1d$, study the effect of two hyperparameters in kernel ridge regression, as well as using k-fold cross-validation to pick hyperparameter.

As a reminder, if you have questions, we will answer them via the queue at oh.eecs189.org. Once you complete the first part, you could go to the Jupyter notebook.

1 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:

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

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:

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

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

Show that

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

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

Solution: In problem 6b of HW1 we derived that solving (1) is equivalent to solving the following problem:

$$\min_{\mathbf{w}\in\mathbb{R}^d,\, \boldsymbol{\xi}\in\mathbb{R}^n} \|\mathbf{w}\|_2^2 + \|\boldsymbol{\xi}\|_2^2 \text{ s.t. } \begin{bmatrix} \mathbf{\Phi}, & \sqrt{\lambda}\mathbf{I} \end{bmatrix} \begin{bmatrix} \mathbf{w} \\ \boldsymbol{\xi} \end{bmatrix} = \mathbf{y}.$$

We know that the solution to this problem is given by the following formula:

$$\begin{bmatrix} \mathbf{w} \\ \boldsymbol{\xi} \end{bmatrix} = \begin{bmatrix} \mathbf{\Phi}, & \sqrt{\lambda}\mathbf{I} \end{bmatrix}^\dagger \mathbf{y},$$

where $\mathbf{A}^\dagger$ denotes the pseudo-inverse of matrix $\mathbf{A}$. Now let's plug in an explicit expression for the pseudo-inverse: if rows of $\mathbf{A}$ are linearly independent, then $\mathbf{A}^\dagger = \mathbf{A}^\top(\mathbf{A}\mathbf{A}^\top)^{-1}$. Therefore, we get

$$\begin{bmatrix}\mathbf{w}\\ \boldsymbol{\xi}\end{bmatrix} = \begin{bmatrix}\boldsymbol{\Phi}^\top\\ \sqrt{\lambda}\mathbf{I}\end{bmatrix}(\boldsymbol{\Phi}\boldsymbol{\Phi}^\top + \lambda\mathbf{I})^{-1}\mathbf{y}.$$

Finally, we can use the explicit formula to calculate the expression for prediction on a new data point:

$$\begin{aligned}\phi(\mathbf{x})\mathbf{w} =& \phi(\mathbf{x})\boldsymbol{\Phi}^\top(\boldsymbol{\Phi}\boldsymbol{\Phi}^\top + \lambda\mathbf{I})^{-1}\mathbf{y}\\ =& \mathbf{k}(\mathbf{x})^\top(\mathbf{K} + \lambda\mathbf{I})^{-1}\mathbf{y},\end{aligned}$$

where we plugged in $\mathbf{k}(\mathbf{x})^\top = \phi(\mathbf{x})\boldsymbol{\Phi}^\top$ and $\mathbf{X} = \boldsymbol{\Phi}\boldsymbol{\Phi}^\top$.

2 Learning 1d functions using kernel regression

This question is designed to help you to be able to visualize what kernel regression is like so that you can understand the different phenomena that can occur.

(a) **Do the tasks from part a of the associated jupyter notebook and report the results.** **Solution:** Increasing gamma makes the plot narrower, but its height remains the same.

(b) **Do the tasks from part b of the associated jupyter notebook and report the results.** **Solution:** As we saw in the previous part, increasing gamma leaves diagonal elements the same, but makes off-diagonal elements smaller. Because of that, when gamma increases the singular values of $\mathbf{K}$ become close to each other. This aligns well with the experiments. From the experiments we can also see that n has almost no influence the shape of the plot of singular values of $\mathbf{K}$.

(c) **Do the tasks from part c of the associated jupyter notebook and report the results.** **Solution:** To compute $\hat{\alpha}$ one can do the following:

```
alpha = np.linalg.inv(K) @ y
```

This isn't the most efficient implementation or the most stable one, but it is fine.

(d) **Do the tasks from part d of the associated jupyter notebook and report the results.** **Solution:** 1. Impulse response behaves more robustly for the Laplacian kernel. This aligns well with the results of part b as for rbf kernel the singular values of kernel matrix are small, and thus $(\mathbf{K} + \lambda\mathbf{I})^{-1}$ is sensitive to small changes of those eigenvalues.

2. As we increase gamma, the vector $\hat{\alpha}$ also converges to an impulse vector (i.e. to $\mathbf{y}$ in this case). This happens because as gamma increases, $\mathbf{K}$ converges to identity and $\hat{\alpha} = (\mathbf{K}+\lambda\mathbf{I})^{-1}\mathbf{y}$.

3. The impulse response is closest to the step function when gamma is about 30–100 (depends on the choice of point). This suggest that for such gamma and lambda the prediction of kernel ridge regression is close to that of the 1 nearest neighbour regression, as each point propagates its value in its own small vicinity.

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

Before seeing the experiement it would be reasonable to assume, for example, that rbf kernel would be better for a smooth function and the Laplace one for a non-smooth function, based on how the impulse responses for those kernels behave.

1. When lambda is set to minimum, rbf achieves better approximation of $\sin(20x)$ and laplacian kernel — of "piecewise linear function".

2. When λ is of order 10^{-8}, rbf kernel can achieve better approximation for both functions.

3. When γ is of order 10^2, rbf kernel can achieve better approximation for both functions, (rbf kernel needs larger λ for fitting piecewise linear function.) For laplacian kernel, it is hard to fit both functions when γ is of order 10^2.

4. When λ is chosen to be very large, the learned function will be zero function; when γ is chosen to be very large, the learned function will be almost zero function and interpolate the training data points.

5. The initial guess that we made was not correct: it turned out that expressive abilities of the rbf kernel are so good that it can approximate a piecewise function better than lapalacian kernel unless there is no regularization.

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

1. For the rbf kernel the prediction looks (relatively) reasonable for γ from 45 to ~1,450. (at least it doesn't blow up or predict zero everywhere...). Higher values of γ turn out to be more succeptible to noise! This doesn't align with the results of part b as higher values of γ ensure better condition number of $\mathbf{K}$. For laplacian kernel the range of gamma is from 0 to 45. For the whole range the predicion is not very sensitive to noise. In part b we could see that the shape of the spectrum for the laplace kernel also doesn't depend on γ too much, but it's not clear whether it is a coincidence (we cannot really vary both things very much, so it's hard to claim that there is some dependence).

2. For rbf kernel, we could use $(\gamma, \lambda) = (8.0, 1e-8)$. For laplacian kernel, we could use $(\gamma, \lambda) = (2.0, 0.0156)$. Laplacian kernel is less sensitive to the choice of parameters. Because of this stiffness it's hard to make it approximate the function. Rbf kernel is much easier to cast to the desired shape.

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

Solution:

1. The $\lambda^\star$ increases when noise level increases.

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

Solution:

1. When there is no noise, the $\gamma^\star$ is larger than the noisy case. When there is noise in the output, we observe the similar $\gamma^\star$ for a wide range of noise level.

2. The only qualitative difference in terms of the shape of the curve is that if there is no noise, there is no second descent (i.e. the curve is U-shaped for $\sigma = 0$, but it is W-shaped for $\sigma > 0$. We can see what exactly happens if we do the experiements in part f. First, in the noiseless case for small values of γ our kernel is very flat and our learned function is close to zero. Second, if gamma is too large, the kernel is so narrow that our solution has spikes in data points and is again zero everywhere else. Third, for the middle values of gamma kernel regression does something reasonable. That's how we get a U-shaped curve in the noiseless case.

Now imagine that we have only noise and no signal. In this case we want to learn zero, so small and large γ do exactly what we want, but middle values of gamma learn something nontrivial from noise. Therefore we get something like a flipped U-shaped curve ($\bigcap$-shaped curve?).

When we add signal and noise, those two curves add up, and the result is a W-shaped curve.

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

Solution:

1. The picked hyperparameter performs reasonably good on the test data.
2. The picked hyperparameter is pretty stable when we change k.

Contributors:

- Alexander Tsigler
- Anant Sahai
- Josh Sanz
- Philipp Moritz
- Rahul Arya
- Yaodong Yu