Cognitive Algorithms - Exercise Sheet 4

Kernel Methods

Department of Machine Learning - TU Berlin

Disclaimer

For each exercise, but particularly for exercises involving calculations:

Show your work or you will not receive (full) credit!

Furthermore: Exercises marked with an asterisk * are not required and don't contribute towards the credit you receive, but you are welcome to do them because they are fun.

Task 1 - Kernel Ridge Regression Example [9 points]

Consider a data set with three data points

$$x_1 = 1, x_2 = 2, x_3 = 3$$

with respective labels

$$y_1 = 1, y_2 = 1, y_3 = 1.$$

Hint: A matrix $M \in \mathbb{R}^{n \times n}$ is PSD $\iff \forall x \in \mathbb{R}^n \setminus \{0\} : x^T M x \geq 0$.

1. We want to fit a simple linear model $g_1(x) = w \cdot x$ to the data using (Kernel) Ridge Regression - (K)RR. Recall the normal RR solution is obtained as

$$w = \arg\min_w \left(\sum_{i=1}^{n} (y_i - w^T x_i)^2 + \lambda \|w\|^2 \right) = (XX^\top + \lambda I)^{-1} X \mathbf{y}^\top$$

   where in this case, $n = 3$ is the number of data points, $X = [x_1, x_2, x_3]$, $\mathbf{y} = [y_1, y_2, y_3]$, and λ is the regularization parameter. Take $\lambda = 0.0001$; compute w, the corresponding function $g_1(x)$ and describe why this results in a poor fit. **[1 point]**

   Hint: You can do your calculations in a python console using numpy, but please upload a screenshot of this if you do!

2. Now we want to fit a better model, simply by adding a constant term, as well as a squared term: $g_2(x) = w_1 + w_2 \cdot x + w_3 \cdot x^2 = \mathbf{w}^\top \cdot \boldsymbol{\phi}(x)$ where we have defined a mapping $\boldsymbol{\phi} : \mathbb{R} \to \mathbb{R}^3$ with

$$\boldsymbol{\phi}(x) = \begin{bmatrix} 1 \\ x \\ x^2 \end{bmatrix}$$

   and a weight vector $\mathbf{w} = \begin{bmatrix} w_1 \\ w_2 \\ w_3 \end{bmatrix}$; thus,

$$X_2 = [\boldsymbol{\phi}(x_1), \boldsymbol{\phi}(x_2), \boldsymbol{\phi}(x_3)] = \begin{bmatrix} 1 & 1 & 1 \\ x_1 & x_2 & x_3 \\ x_1^2 & x_2^2 & x_3^2. \end{bmatrix}.$$

Compute $\mathbf{w}$ and the corresponding function $g_2(x)$. Is this approximation better? Why? **[1 point]**

Hint: You can do your calculations in a python console using numpy, but please upload a screenshot of this if you do!

3. Now, we want to obtain the same solution as above, but by solving the dual representation. Instead of learning $\mathbf{w}$ directly, (which for some mappings can be cumbersome or downright impossible), we learn a linear combination $\boldsymbol{\alpha} \in \begin{bmatrix} \alpha_1 \\ \alpha_2 \\ \alpha_3 \end{bmatrix}; \alpha_i \in \mathbb{R}$ of the data points, such that $\mathbf{w} = \sum_{i \in \{1,2,3\}} \alpha_i \boldsymbol{\phi}(x_i)$ (possible by the representer theorem). Thus, our prediction for x_{new} is

$$g_3(x_{\text{new}}) = \mathbf{w}^\top \boldsymbol{\phi}(x_{\text{new}}) = \sum_{i=1}^{3} \alpha_i \boldsymbol{\phi}(x_i)^T \boldsymbol{\phi}(x_{\text{new}}) = \sum_{i=1}^{3} \alpha_i k(x_i, x_{\text{new}})$$

where $k : \mathbb{R} \times \mathbb{R} \to \mathbb{R}$ is the kernel function corresponding to $\boldsymbol{\phi} : \mathbb{R} \to \mathbb{R}^3$.

In the lecture, we derived the formula for $\boldsymbol{\alpha}$:

$$\boldsymbol{\alpha} = (K + \lambda I)^{-1} \mathbf{y}^T \tag{1}$$

where K is the kernel matrix (also known as Gram matrix), such that $K_{i,j} = k(x_i, x_j)$, and λ a regularizing parameter, as discussed in the lecture. **Take note of the difference between kernel *function* k and *Gram matrix* K!** Maybe an unfortunate soul got them mixed up. :)

(a) If we wanted to perform something akin to kernel *non*-ridge regression, we would set λ to 0; however, this is not allowed. Why not? **[1 point]**

(b) Compute k corresponding to $\boldsymbol{\phi}$ in part 2 and compute the resulting classifier $g_3(x)$, again for $\lambda = 0.0001$. Why is g_3 the same as g_2? **[2 points]**

Hint: Even more so here, we recommend that you do your calculations in a python console using numpy, but please upload a screenshot of this if you do!

4. The main point of using the kernel trick as in part 3 is for when we cannot compute $\boldsymbol{\phi}$ or when it is very expensive to do so, as for example, in the case of the RBG kernel, defined by

$$k_{\text{RBF}}(x_i, x_j) = \exp\left(-\frac{\|x_i - x_j\|^2}{2\sigma^2}\right).$$

Consider the RBF kernel with kernel width $\sigma_1 = \sqrt{2}$ and $\sigma_2 = \frac{\sqrt{2}}{2}$. Compute the corresponding classifier g_{RBF1} and g_{RBF2} based on the same data, (with the same $\lambda = 0.0001$). What is the difference between them? Denote this in your plot in task 1.5. **[2 points]**

5. Draw a 2D plot with the data points and all of the classifiers you have derived. **[2 points]**

Task 2 - More fun with kernels [9 points]

1. Recall the RBF kernel from task 1. Notice how we did not give a mapping function $\boldsymbol{\phi}$ such that $k(x_i, x_j) = \boldsymbol{\phi}(x_i)^T \boldsymbol{\phi}(x_j)$. The reason for this is that the RBF kernel is an implicit mapping into an infinite-dimensional space. Assume, without loss of generality a constant $\sigma = \sqrt{2}$. Prove that the subspace of $\boldsymbol{\phi}$ would map to an infinite dimensional space. **[3 points]**

Hint 1: $e^x = \sum_{d=0}^{\infty} \frac{x^d}{d!}$.

Hint 2: Recall the definition of the d'th degree homogeneous polynomial kernel $k_d(x_1, x_2) = (x_1^T x_2)^d$.

2. (a) You are a researcher at a natural language processing firm and they have given you the task of classifying greetings into either English or German. You are given a six-greeting data set, three in each language.

English: "yo dude", "What's up", and "hey there what's happening".
German: "Servus," "Grüß Gott", and "Digga was geht ab"

Each of these is encoded as a binary number. Thus, each input greeting is in $\mathbb{R}$. Unfortunately, you cannot easily classify simply based on these binary numbers without overfitting.

Therefore, you construct a function $\boldsymbol{\phi} : \mathbb{R} \to \{0,1\}^{G+E}$, that takes a binary encoded greeting and maps it a one-hot vector with an index for every word in both German and English (G is the number of words in German, and E the number of words in English), under the assumption that no word is present in both.

For example, $\boldsymbol{\phi}(yo\ dude) =$

$$\begin{bmatrix} 0 \\ \vdots \\ 1 \text{ (this is entry for the word yo)} \\ \vdots \\ 0 \\ \vdots \\ 1 \text{ (this is entry for the word dude)} \\ \vdots \\ 0 \end{bmatrix}$$

Recall that for two-class, bias-less classification we want a weight vector $\mathbf{w}$ s.t

$$(\forall x \in C_1 : \mathbf{w}^T x > 0) \wedge (\forall x \in C_2 : \mathbf{w}^T x < 0)$$

or, in the case of a mapping, $\boldsymbol{\phi}$, we want a weight vector $\mathbf{w}_{\boldsymbol{\phi}}$:

$$(\forall x \in C_1 : \mathbf{w}_{\boldsymbol{\phi}}^T \boldsymbol{\phi}(x) > 0) \wedge (\forall x \in C_2 : \mathbf{w}_{\boldsymbol{\phi}}^T \boldsymbol{\phi}(x) < 0)$$

for two classes C_1 and C_2.

We can think of this classification problem as a subproblem of OLS, in which our training data have only two values. Without actually solving the OLS equation however, give a weight vector $\mathbf{w}_{\boldsymbol{\phi}}$ which results in proper classification of each training example, and describe how this mapping into a higher dimension was the key to solving this problem. **[3 points]**

(b) It occurs to you that $\boldsymbol{\phi}$ is terribly expensive to compute and takes up inordinate space. Thus, you have the epiphany to use the kernel trick (nice)! Derive a kernel function k and an expression for a classifier $f_k(x)$. Use the formal definition for $\boldsymbol{\alpha}$ (see Eqn. 1) if you would like (with $\lambda = 0.0001$), or simply give some $\boldsymbol{\alpha}$ that fits the data optimally - and explain how you arrived at it. **[3 points]**

Task 3 - Cross Validation [2 points]

1. You have a model and would like to perform model evaluation using cross validation. You are faced with the decision of how much data to use per fold. Explain the pros and cons to using a lot of data per fold (and having fewer folds), vs using little data per fold (and having many folds).