Homework 4

36-708, Spring 2021

Due May 14 at 5PM EST

Please attach all code to your homework. In an RMarkdown document for example, this can be done in one line (see Yihui Xie's website for how to do this).

1 Kernel calculus and the Gaussian kernel

To hack the so-called kernel trick into a machine learning algorithm, we need to be able come up with (or justify the choices of) valid kernel functions. One approach to do this is to first construct explicit feature maps and then get the kernels using corresponding inner products. An alternate approach is to directly construct kernels that are appropriate for a given application by building them out of simpler kernels using calculus of kernels. In this questions, we play around with some of such calculus rules and use them to show the validity of the popular Gaussian kernel. Consider the following setup.

- $K_1, K_2 : \mathbb{R}^d \times \mathbb{R}^d \to \mathbb{R}$ and $K_3 : \mathbb{R}^D \times \mathbb{R}^D \to \mathbb{R}$ are any valid kernel functions.
- Constants: c_1, c_2 are nonnegative real constants
- Polynomial: p is a polynomial with nonnegative coefficients.
- Generic function: $\phi : \mathbb{R}^d \to \mathbb{R}^D$ is any function.
- Generic points: $u, v \in \mathbb{R}^d$.

Prove that the following are valid kernels:

(a) $c_1 K_1(u, v) + c_2 K_2(u, v)$,

(b) $K_1(u, v) K_2(u, v)$,

(c) $p(K_1(u, v))$,

(d) $\exp(K_1(u, v))$, and

(e) $K_3(\phi(u), \phi(v))$.

(f) Using the above, show that

$$K(u, v) = \exp\left(-\frac{\|u - v\|^2}{2\sigma^2}\right)$$

is a valid kernel.

2 Two flavours of kernel regression

In this question we explore two flavors of kernel regression. One is a local fit perspective and the other is a global fit perspective, but both of them lead to smoothers involving kernels.

2.1 Kernel regression

Consider the following local linear regression setup.

- Data: features $x_i \in \mathbb{R}^d$ and responses $y_i \in \mathbb{R}$ for $i = 1, \ldots, n$.
- Regression estimate: at a point $x \in \mathbb{R}^d$ the regression estimate is denoted by $\widehat{f}(x) = \widehat{\beta}^T x$ where

$$\widehat{\beta} = \operatorname*{argmin}_{\beta} \sum_{i=1}^{n} w_i(x) \left(y_i - \beta^T x_i\right)^2, \quad \text{and}$$

$$w_i(x) := \frac{K(x, x_i)}{\sum_{i=1}^{n} K(x, x_i)},$$

for some kernel K.

(a) Verify that the objective function can be re-written as

$$(y - X\beta)^T \Omega(x)(y - X\beta)$$

where $y = \begin{pmatrix} y_1 & y_2 & \cdots & y_n \end{pmatrix}^T \in \mathbb{R}^n$ and $X = \begin{pmatrix} x_1 & x_2 & \cdots & x_n \end{pmatrix}^T \in \mathbb{R}^{n \times d}$, and $\Omega(x) = \operatorname{diag}(w_1(x), \ldots, w_n(x))$.

(b) Show that $\widehat{f}(x)$ is a linear combination of $\{y_i\}_{i=1}^n$.

2.2 Kernelized ridge regression

The nonparametric kernel regression in the question above performs a local fit around the test point. Let us now investigate the use of kernels for regression in another way. We apply the kernel trick in regular regression as follows. Note that this produces a global fit to the data. Consider the following ridge regression setup.

- Data: features $x \in \mathbb{R}^d$ and responses $y_i \in \mathbb{R}$ for $i = 1, \ldots, n$.
- Feature map: we use a feature mapping function ϕ to map the original d-dimensional feature vector x to a new D-dimensional feature vector $\phi(x)$, where $D \gg d$.
- Regression estimate: the regression estimate at a point $x \in \mathbb{R}^d$ is denoted by $\widehat{f}(x) := \widehat{\beta}^T \phi(x)$, where

$$\widehat{\beta} = \operatorname*{argmin}_{\beta} \frac{1}{2} \|y - \Phi\beta\|^2 + \frac{\lambda}{2} \|\beta\|^2,$$

and $\Phi := \begin{pmatrix} \phi(x_1) & \ldots & \phi(x_n) \end{pmatrix}^T \in \mathbb{R}^{n \times D}$ denotes the kernel design matrix and λ is a regularization parameter.

(a) Show that we can write $\widehat{\beta} = \Phi^T (\Phi\Phi^T + \lambda I_n)^{-1} y$ where I_n is the identity matrix.

(b) Show that we can alternately write $\widehat{\beta} = (\Phi^T \Phi + \lambda I_d)^{-1} \Phi^T y$.

(c) In practice, designing ϕ is nontrivial, and when it is designed, we must compute the inner product $\phi(x)^T \phi(x')$ which may be costly. Instead we may want to use the kernel $K(x, x') = \phi(x)^T \phi(x')$ directly if it is computationally cheaper. Show how to 'kernelize' ridge regression (both in the training and evaluation phases) in the sense that neither $\phi(x)$ nor $\phi(x)^T \phi(x')$ would need to be computed, but $K(x, x')$ would. In particular, show that the predicted value at a new test point $x^\star$ can be computed as

$$y^\star = y^T (M + \lambda I_n)^{-1} c(x^\star)$$

for some matrix $M \in \mathbb{R}^{n \times n}$ and vector $c(x^\star) \in \mathbb{R}^n$. Write down expressions for M and $c(x)$ in terms of the kernel.

3 Clustering and PCA

In this question, we explore the use of unsupervised dimension reduction techniques for some image processing tasks.

3.1 Image segmentation and compression using k-means

In this part, we explore how we can use k-means clustering for image segmentation and compression. The goal of image segmentation is to partition an image into multiple segments, where each segment typically represents an object in the image. Consider the following algorithm for image segmentation.

- Treat each pixel in the image as a point in 3-dimensional space comprising the intensities of the red, blue, and green channels. Treat each pixel in the image as a separate data point.
- Perform k-means clustering (e.g. using Lloyd's algorithm) and identify the clusters.
- All the pixels belonging to a cluster are treated as a segment in the image.

For any value of k we can re-construct the image by replacing each pixel vector with the red, blue, and green intensity triplet given by the center to which that pixel has been assigned.

(a) Take any image (you can be creative here!) and show the image re-constructions obtained using various values of k.

(b) Suppose we require 24 bits to store red, blue, and green intensity values of each pixel. How many bits do we need to store an image with N pixels? Now, suppose we first run k-means with k clusters on the image data and identify each pixel using its cluster ID. How many bits do you need to store the image? What's the compression ratio?

3.2 Implementing PCA for face recognition

In this part. we explore how we can use PCA to recognize faces. We play with AT&T database which has 10 near frontal images of 40 individuals under different illuminations per individual. The data can be downloaded from https://36708.github.io/hw4_q3_data/. We represent each image as a 1024 dimensional vector (the images have 32×32 pixels).

(a) Implement a function to perform PCA.

(b) With the mean ($\widehat{\mu}_n$) and matrix of eigenvectors (V) learned from the training data, you can project other data points into this eigen space. Let x_{test} be a test point. Then, to project it into the eigenspace, simply subtract the mean vector and multiply by the eigenvector matrix (i.e, compute $V(x - \widehat{\mu}_n)$). This will give you a vector of length k. Given any point $z \in \mathbb{R}^k$ in the eigenspace we can reconstruct the point in the original space as $V^\top z + \widehat{\mu}_n$.

Take a test image and project it into the eigenspace (use $k = 5, 10, 20, 40$). Use the projections to reconstruct the test images. Display the reconstructed images along with the original test image.

4 Neural networks

4.1 PCA and autoencoders

In this question, we explore the relation between PCA, kernel PCA and auto-encoder neural networks (trained to output the same vector they receive as input). Consider the following setup.

- Number of data points: n
- Number of features: d

(a) Consider an auto-encoder with a single hidden layer of k nodes. Let w_{ij} denote the weight of the edge from the ith input node to the jth hidden node. Similarly, let v_{ij} denote the weight of the edge from the ith hidden node to the jth output node. Show how you can set the activation functions of hidden and output nodes as well as the weights w_{ij} and v_{ij} such that the resulting auto encoder resembles PCA.

4.2 Kernel PCA and autoencoders

Recall that kernel PCA is a non-linear dimensionality reduction technique where a principal vector v_j is computed as a linear combination of training examples in the feature space

$$v_j = \sum_{i=1}^{n} \alpha_{ij} \phi(x_i)$$

Computing the principal component of a new point x can then be done using kernel evaluations:

$$z_j(x) = \langle v_j, \phi(x) \rangle = \sum_{i=1}^{n} \alpha_{ij} \langle \phi(x_i), \phi(x) \rangle = \sum_{i=1}^{n} \alpha_{ij} k(x_i, x)$$

We show that kernel PCA can be represented by a neural network. First we define a kernel node. A kernel node with a vector w_i of incoming weights and an input vector x computes the output $y = k(x, w_i)$.

(a) Show that, given a data set $x_1, \ldots, x_n$, there exists a network with a single hidden layer and the output of the network is the kernel principal components $z_1(x), \ldots, z_k(x)$ for a given input x. Specify the number of nodes in the input, output and hidden layers, the type and activation function of hidden and output nodes, and the weights of the edges in terms of $\alpha, x_1, \ldots, x_n$.

(b) What is the number of parameters (weights) required to store the network in the previous question?

(c) Another way to do non-linear dimensionality reduction is to train an auto encoder with non-linear activation functions (e.g. sigmoid) in the hidden layers instead of using kernels. State one advantage and one disadvantage of that approach compared to kernel PCA.

4.3 Neural network for classification

In this problem, we play with the TensorFlow playground (http://playground.tensorflow.org), which is a nice visual tool for training simple Multi Layer Perceptrons (MLPs). Your goal in this problem is to carefully select input features to design the "smallest" MLP classifiers that can achieve low test loss for each of the 4 data sets in TensorFlow playground shows the 4 data sets available on TensorFlow playground). Here "smallest" is defined as having least number of neurons in the network. By low test loss we mean a test loss ¡ 0.1 for the swiss roll data set and a test loss of ≈ 0 for the rest of the data sets. Submit screenshots after your networks achieve the required test loss for each of the following data sets.

(a) Circles

(b) Clusters

(c) Squares

(d) Swiss Roll

4.4 Neural network for regression

Finally, we use a neural network on a regression problem. This is the most open-ended question so far and you are essentially free to try anything. You are encouraged to play around with different packages and various settings.

- Dataset: We play with the Ames Housing Dataset. Recall that the raw data is available at http://jse.amstat.org/v19n3/decock/AmesHousing.txt and the dataset description can be found at http://jse.amstat.org/v19n3/decock/DataDocumentation.txt.
- You will need to do some preprocessing (including missing values, possible outliers, etc.). Please feel free to make any reasonable preprocessing decisions, so long as you report them.
- Train and test split: Again use a 3:1 random split.
- Neural network architecture: You are free to play around with any architecture.

(a) Report different choices (preprocessing, architecture, optimizers, etc.) you made throughout and the resulting train and test accuracies.

(b) Comment on relative advantages and disadvantages of training a neural network versus a stacked regressor you trained in a previous homework.