CSCC11 Introduction to Machine Learning, Winter 2024

Assignment 1

This assignment makes use of material from the first 3 weeks of the lectures and tutorials, but some of it can be completed after week 1, and much of it can be completed after week 2 (see Chapters 1-7 of the course notes).

The assignment comprises two parts, a written part and a coding part. The written work asks for you to derive some solutions to problems, and for you to demonstrate an understanding of certain concepts. The written and programming parts will have different deadlines (see below).

Submission instructions can be found at the end of the handout.

We remind you that the work you hand in for the this assignment must be your own. Students should not make use of large language models for programming assistants like Copilot.

Theory Questions (Due Thursday, Feb. 1, 11:59pm)

Question 1. Least Squares Model Fitting (3 marks)

Weighted Least Squares is a variant of Least Squares in which *importance factors* increase or decrease the influence of individual training data points in the objective function. Essentially, it treats each observation as being more or less informative about the underlying relation between inputs and outputs. More important points should have higher importance factors. This is particularly useful when you know, for example, that certain data points are noisier than others, or that some are more important to the model *a priori*. In the case of N training pairs, with D-dimensional inputs and scalar outputs, the weighted least-squares objective function may be expressed as

$$E(\tilde{w}) = \sum_{i=1}^{N} \beta_i (y_i - \tilde{\mathbf{w}}^T \tilde{\mathbf{x}}_i)^2, \qquad \beta_i > 0 \tag{1}$$

where β_i is the non-negative importance factor associated with the ith training point. Using this formulation for weighted least-squares, answer the questions below.

a Modify the objective function in matrix form in Equation (2.13) of the course notes to incorporate importance factors for the data points. What are the properties of the weighting matrix? What can you say about its structure?

b Derive the mathematical form of the weighted LS solution, again in matrix form. To this end, set the gradient of the objective to 0 to obtain the normal equations and then solve them to find the optimal model parameters for this weight LS formulation. Povide a step-by-step derivation to obtain the solution.

Question 2. Regularized Least Squares (3 marks)

As explained in the course notes (Section 3.2), regularized LS regression incorporates an L2 penalty on the magnitudes of the weights (i.e., parameters) of the model. The goal is to help avoid over-fitting by encouraging models to be smooth. This method is sometimes referred to as *ridge regression*. In Deep Learning, an L2 penalty on weights is referred to as *weight decay*. As explained in Section 3.2, the form of the objective function for regularized basis function regression is (see Equation (3.11) of the course notes):

$$E(\mathbf{w}) = \|\mathbf{y} - \mathbf{B}\mathbf{w}\|^2 + \lambda \|\mathbf{w}\|^2 \ . \tag{2}$$

Here, $\lambda > 0$ is often referred to as a regularization parameter; it determines the amount of regularization.

a The regularized LS estimate for $\mathbf{w}$ is given by

$$\mathbf{w}^* \ = \ (\mathbf{B}^T\mathbf{B} + \lambda \mathbf{I})^{-1} \mathbf{B}^T \mathbf{y} \ . \tag{3}$$

Using a simple numerical example with a 2×2 matrix $\mathbf{B}$, show that for $\lambda = 0$, the solution may be extremely sensitive to noise because $(\mathbf{B}^T\mathbf{B})^{-1}$ can be ill-conditioned or not even exist. Using the example, discuss whether the problem remains ill-conditioned or unsolvable when $\lambda \neq 0$.. Are the issues resolved and why (or why not)?

b When the input is orthonormal (i.e., when $\mathbf{B}^T\mathbf{B} = \mathbf{I}$), determine the mathematical relation between the weights computed from regularized LS (ie Equation (3.14) in the course notes) and the LS optimal weights (ie Equation (2.16), without regularization).

c The impact of the regularization term (the second term on the right hand side of (2) above) will also depend on the magnitude of the inputs and basis functions. For example, using Equation (3.14), and assuming orthonormal input matrix, suppose we multiply matrix $\mathbf{B}$ by a scalar value (e.g., 2). How would you need to scale the regularization parameter such that the optimal weights are identical to those you obtain without that scalar factor? Derive the solution with the scalar factor to prove the equivalence.

Question 3. K-Nearest Neighbour Regression (3 marks)

KNN regression relies on identifying training data points that are similar to a test point. The concept of similarity assumes a critical role, the most widely used metric for which is Euclidean distance between input features. Specifically, the distance between input features $\mathbf{a} = (a_1, ...a_D)^T$ and $\mathbf{b} = (b_1, ...b_D)^T$ is

$$d(\mathbf{a}, \mathbf{b}) \;=\; \sqrt{\sum_{i=1}^{D} (a_i - b_i)^2} \;. \tag{4}$$

a Re-write Equation (4) above in matrix-vector form.

b In many circumstances, different features (i.e., elements of the input vector) may have very different influence on the distance calculation. For example, some features may be measured in meters while others are in millimetres. As such, the distance measure may be dominated by features measured in millimetres since all those numbers will be larger. To mitigate this issue with KNN algorithms, we sometimes weight different dimensions of the input differently. To this end, suppose you wanted to modify Equation 3 to incorporating some form of feature weighting (ie by normalizing the different feature dimensions), so all dimensions exert similar influence on the distance calculation over the entire training set. There may be many ways to do this. Your task is to suggest one approach, write it in matrix form, and explain (in 3 or 4 sentences) why the approach will be effective.

c In K-NN regression, the value of K can be important. In a few sentences, discuss how too small a value of K can lead to over-fitting, while too large a value of K can lead to under-fitting.

Programming Questions (Due Thursday, Feb. 8, 11:59pm)

To begin the coding exercises, download `A1-Programming.zip` from the calendar page on the course website. When you unzip the file a directory `A1-Programming` is created. Please do not change the directory structure, nor the headers of the Python functions contained therein.
Inside `A1-Programming`, there are two directories `q1` and `q2` corresponding to the first and second coding questions. Next these, you find a text file `requirements.txt` in which all necessary python packages are listed. We strongly encourage you to use environment manager tools (such as Anaconda or Virtualenv) to create a **python version 3.8** environment and install packages listed in `requirements.txt`. We provide an example on how to setup an new environment and install dependencies in the course website. This section also includes python notebooks that you run locally on the jupyter notebook or import them on Google Colab [1] (with other starter codes uploaded). During tutorial, you see a brief introduction to Google Colab.
Instructions for electronic submission are included below.

Question 1. Polynomial Regression (9 marks)

In this question, you are going to implement 1D polynomial regression model and solve for its parameters for multiple different datasets. Recall that in 1D polynomial regression, the scalar output is expressed as:

$$\hat{y} \;=\; f(x) \;=\; b + \sum_{i=1}^{K} w_i x^i, \tag{5}$$

[1] `https://colab.google/`

where K is the order of the polynomial, w_k is the k-th weight and b is the bias. Given a set of training input/output pairs $\{(x_i, y_i)\}_{i=1}^N$, one can solve for the the parameters using least squares objective (LS) as discussed in the lecture. With the additional regularization term $||w||^2$, we obtain regularized LS solution. Here, the main goal is to explore effects of different factors such as dataset size, order of the polynomial and regularization parameter.

The starter code for this question is divided into the python script `poly_regression.py` which implements the polynomial regression model, and the python notebook `poly_notebook.ipynb` that runs the experiments with the regression model. The script `poly_regression.py` contains the class of `PolynomialRegression` with five methods:

1. `__init__(self, K, l2_coeff)`: This is the constructor of the class. `K` and `l2_coeff` specify the degree of the polynomial and regularization coefficient λ, respectively.
2. `predict(self, X)`: This method predicts the output for a given input, using the model parameters. `X` is a vector of inputs. The method outputs a vector of predicted outputs.
3. `fit(self, train_X, train_Y)`: This method find the LS solution, given training data inputs `train_X` and outputs `train_Y`.
4. `fit_with_l2_regularization(self, train_X, train_Y)`: Similar to the previous method, but finds regularized LS solution considering `l2_coeff` as the regularization coefficient λ.
5. `compute_mse(self, X, observed_Y)`: This method computes the mean squared error between predictions over the input data `X` and observed outputs `observed_Y`.

You need to complete the body of four methods: `predict`, `fit`, `fit_with_l2_regularization` and `compute_mse`. The constructor creates the column vector `self.parameters` of size $K + 1$ to store the bias b and parameters w_i. Please note that the bias b is **the first element** in the vector. Both model fitting methods (`fit`, `fit_with_l2_regularization`) update this parameter while `predict` uses it for prediction. Once you complete the methods, you can run `poly_regression.py` to call basic tests to verify your implementation. Passing test does not guarantee the correctness of your implementation, so you can design your own test for further evaluation.

Once you are confident with your code, start running the python notebook `poly_notebook.ipynb`. This notebook is divided into three section, and you explore the following:

1. Dataset size: We have provided few benchmarks, each comprises of one test dataset and three training datasets with various sizes of small, medium, and large. For each benchmark, the underlying ground-truth function and data distribution is the same. In the first part, notebook uses the implemented regression model to solve for its parameters (not regularized) on three training datasets and then computes and visualizes the mean squared error on test data as well as the corresponding training data. Based on the visualization, discuss the effect of dataset size on both train and test errors and justify your answer.
2. Polynomial order: This factor determines the complexity of the model. In the second part, for the first the benchmark, the notebook fits polynomial models of different orders (ranging from 1 to 10) to training data of different sizes. Similarly, the mean squared error for both train and test data are visualized. Discuss how increasing the order (or complexity) influences the train and test errors. How does the effect of model complexity change when training on datasets of different sizes? Justify our answers.
3. Regularization coefficient: Finally, you fit the polynomial model using regularized least squares with various regularization coefficients for the second benchmark. As before, the mean squared error for both train and test data are visualized. Discuss how the errors change when increasing the regularization coefficient. Also describe if the trends are consistent across datasets of different size. Justifications are required.

Please use the **markdown** to provide your answers in the corresponding sections in the notebook. For this question, **you should submit two files**: `poly_regression.py` and `poly_notebook.ipynb`. For `poly_regression.py`, the body of aforementioned methods must be completed. Make sure that visualizations are included in `poly_notebook.ipynb` and it runs without any error.

Question 2. Image Denoising with RBF Regression (9 marks)

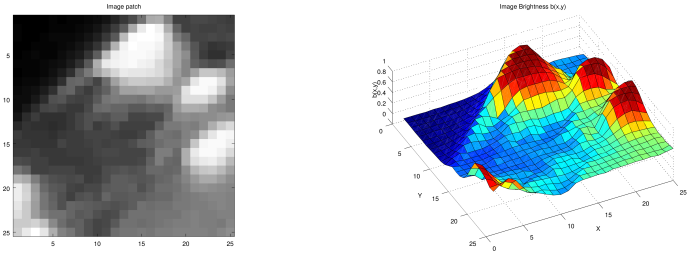

Figure 1: *(left)* Greyscale image. *(right)* Depiction of image as a height map where brighter pixels are higher.

Background An image is typically expressed as a 2D array of pixels. For greyscale images (e.g. left of Fig 1), each pixel is a scalar representing the brightness level, so they image can be viewed as a function that takes 2D position and outputs a greyscale value. Color images have three values per pixel, namely red, green and blue components, but for now let's focus on greyscale ones. One can model an image as a function $I(\mathbf{x})$ mapping from position $\mathbf{x} \in \mathbb{R}^2$ to the brightness level. Pixel values are normalized between 0 (black) and 1 (white).
The function $I(\mathbf{x})$ can be represented using basis functions. Since natural images are smooth and correlated in local regions, we use radial basis functions (RBFs) that enables the brightness level to be localized in each region. Let's formalize this as follows:

$$I(\mathbf{x}) = b + \sum_{i=1}^{K} w_k b_k(\mathbf{x}), \tag{6}$$

where b is the bias term and $b_k(\mathbf{x})$ is k-th basis function, a smooth bump centered at location $\mathbf{c}_k \in \mathbb{R}^2$ with width σ,

$$b_k(\mathbf{x}) = \exp(-||\mathbf{x} - \mathbf{c}_k||^2/2\sigma^2). \tag{7}$$

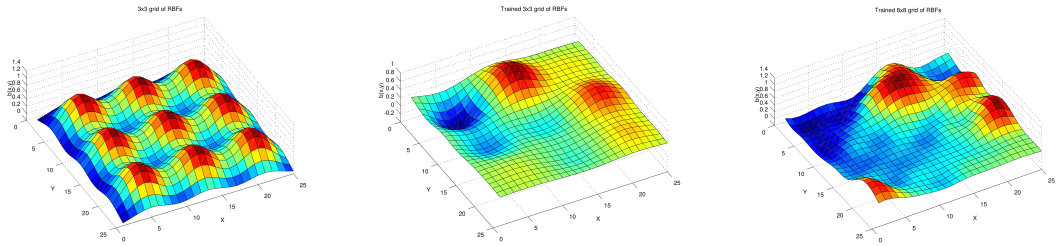

Figure 2: *(left)* An RBF model with all weight $w_k = 1$. *(middle)* A model with 3×3 RBFs with LS regression weights. *(right)* A model with 8×8 RBFs with LS regression weights.

For example, to model the greyscale image in Fig. 1, let's assume the basis functions are evenly spaced on a square grid, all with the same width. Using the regularized least squares regression, one can find the parameters w_k and b. For a 3×3 grid of RBFs, with all weights equal to 1, the model output look as in Figure 2 left. If we fit RBF weights using least squares, we obtain a better approximation to our image patch (Fig. 2 middle). We get an even better approximation if we use more RBFs, e.g. an 8×8 grid as in Fig. 2 right. For colored images, separate RBFs is dedicated to each color channel.

Image Denoising is a long-standing problem in areas of signal processing and computer vision. The measurements in physical processes are typically noisy and noise removal is a crucial step to obtain the underlying ground-truth signal. In this question, we aim to deal with the salt-and-pepper noise that is synthetically added to a clean image of lighthouse (Fig. 3 left). This noise appears as sparsely occurring white and black pixels in the acquired image (Fig. 3 middle). To denoise an image I, we first divide it into square patches and model each patch $I_{i,j}$ as a separate 2D function $f_{i,j}(\mathbf{x})$. Following the discussion in the background, each image patch is particularly represented by RBFs with centers evenly spaced on the patch grid. The widths are also the same. The clean pixels are used as input data to fit the RBF model using regularized least squares and then the fitted RBF can be evaluated at corrupted pixels to reconstruct the missing values. The values at clean pixels are kept

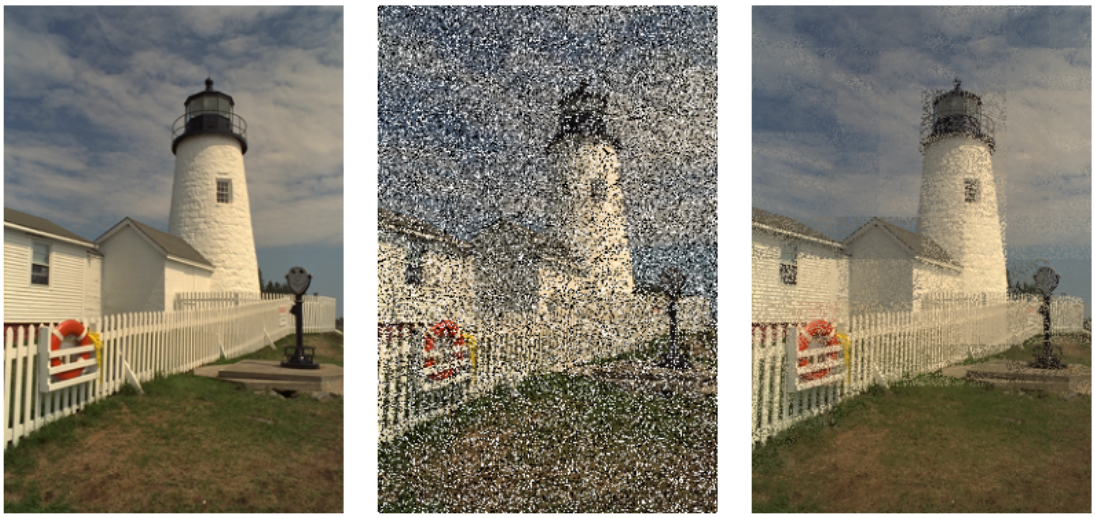

Figure 3: *(left)* The noiseless image of the lighthouse. *(middle)* The noisy image corrupted with the salt-and-pepper noise. *(right)* The output of RBF regression model that performs denoising.

the same. An example denoise image is shown in the right hand side of Fig. 3.

The starter code for this question includes the script `rbf_regression.py` which implement the RBF regression model and the python notebook `image_denoising.ipynb` that runs image denoising experiments. The script `rbf_regression.py` contains the class of `RBFRegression` with four methods:

1. `__init__(self, centers, widths)`: This is the constructor of the class. `centers` is a $K \times 2$ matrix storing coordinates of center of 2D radial basis functions (*bumps*), and `widths` specifies a vector of K corresponding widths.

2. `rbf_2d(self, X, i)`: This method computes the output of the i-th 2D radial basis function given the inputs.

3. `predict(self, X)`: This method predicts the output for the given inputs `X`, using the model parameters.

4. `fit_with_l2_regularization(self, train_X, train_Y, l2_coeff)`: This method fits the parameters to the training input/output pairs `train_X`, `train_Y` using regularized LS with regularization coefficient `l2_coeff`.

You need to complete the body of three methods: `rbf_2d`, `predict`, and `fit_with_l2_regularization`. The constructor creates the column vector `self.parameters` of size $K + 1$ to store the bias b and parameters w_i. Please note that the bias b is **the first element** in the vector. Within `rbf_regression.py`, a few basic tests are also provided to verify your implementation. Passing test does not guarantee the correctness of your implementation, so you can design your own test for further evaluation. When you are confident with your implementation, start running the python notebook `image_denoising.ipynb` which uses the RBF regression for image denoising. This notebook is divided into three section, and you explore the following:

1. Denoised image: For the input image (Fig. 3 left), run the denoising with default setting. The notebook visualizes the clean image, noisy image and the denoised one, respectively. Does the denoiser perform well? Do you see any artifacts in the denoised image?

2. Width: Given a certain spacing, the notebook runs denoising with various widths for the RBFs and computes the mean squared error between denoised and clean image. The error is visualized as a function of the width size. How does the error change when increasing the width? Justify our answer.

3. Spacing: Finally, the notebooks runs an experiment that explores the effect of the spacing between basis functions. As before, the mean squared error is computed and visualized. Discuss how the error changes when increasing the spacing. Justifications are required.

Please use the **markdown** to provide your answers in the corresponding sections in the notebook. For this question, **you should submit two files**: `rbf_regression.py` and `image_denoising.ipynb`. For `rbf_regression.p`, the body of aforementioned methods must be completed. Make sure that visualizations are included in `image_denoising.ipynb` and it runs without any error.

Submission Instructions

Theory Questions (Due Thursday, Feb. 1, 11:59pm) This part may be done with pen and paper or a text editor (eg latex). Formatted answers are easier to read but likely require more time. In any case, your answers should be handed in electronically as a single pdf file (for all 3 questions). The file should be called `solution.pdf`, and it should be submitted to **Markus**. If you do work with pen and paper you could scan or take photos of your work then convert to pdf.

Programming Questions (Due Thursday, Feb. 8, 11:59pm) For this part, you will submit four files in total. There should be two files for the first question (Q1), namely, `poly_regression.py` and `poly_notebook.ipynb`. And there are two files for the second question (Q2), namely, `rbf_regression.py` and `image_denoising.ipynb`. The body of incomplete methods in python scripts `poly_regression.py` and `rbf_regression.py` should be completed by you in the submitted files of course. The two notebooks files, i.e., `poly_regression.py` and `image_denoising.ipynb` should contain the **visualizations and answers to questions in the markdown**.

Note that in Markus you find **separate sections** dedicated to theory and programming parts of this assignment.