MLSP 2022 assignment

Problem 2. (10 points) Consider the "slope-only"(i.e. $\theta_0 = 0$) linear regression model from class which has the following hypothesis and cost function

$$\text{Hypothesis: } h_\theta(x) = \theta_1 x$$
$$\text{Cost function: } J(\theta_1) = \frac{1}{2}\sum_{i=1}^{m}(h_\theta(x^{(i)}) - y^{(i)})^2$$

and consider the following small data set

x	y
1	3
-1	-2
2	4

Solve for the value of θ_1 that minimizes the cost function by substituting the values from the training set into the cost function, setting the derivative equal to zero and solving for θ_1. Show your work.

Problem 3. (10 points.) Now use the same hypothesis and cost function from the previous problem but do NOT substitute the values of the training set into the cost function. Instead, leave the $x^{(i)}$ and $y^{(i)}$ variables, take the derivative with respect to θ_1, set it equal to zero, and solve for θ_1. This will give you a general expression for θ_1 in terms of the training data.

Check your answer by plugging in the training data from the previous problem into your expression for θ_1. Confirm that you get the same value for θ_1 as in the previous problem.

Problem 4. (15 points.) Now consider the more general hypothesis and cost function:

$$\text{Hypothesis: } h_\theta(x) = \theta_0 + \theta_1 x$$
$$\text{Cost function: } J(\theta_0, \theta_1) = \sum_{i=1}^{m}(h_\theta(x^{(i)}) - y^{(i)})^2$$

Take the partial derivatives with respect to both θ_0 and θ_1 and set them to zero. Then solve the system of two equations to come up with expressions for θ_0 and θ_1 in terms of the training data (in terms of x and y).

4a. Solve for θ_0 and θ_1 when $m = 2$. Hint: If you encounter a lot of nasty algebra, this means you are doing the problem correctly.

4b. When solving for an arbitrary m, would you prefer this method or gradient descent? Why?

Problem 1. (10 points) Mathematical Foundations.

The exercises in this section are designed to help you practice linear algebra fundamentals.

1. Write the result of the following matrix-matrix multiplication. Your answer should be written in terms of u, v, a, and b.
$$\begin{bmatrix} 3 & -1 \\ 2 & 5 \\ -2 & 2 \end{bmatrix} \cdot \begin{bmatrix} u & a \\ v & b \end{bmatrix}$$
2. Suppose $A \in \mathbb{R}^{2\times 2}, B \in R^{2\times 4}$. Does the product $A \cdot B$ exist? If so, what size is it?

3. Suppose $A \in \mathbb{R}^{3\times 5}, B \in R^{4\times 1}$. Does the product $A \cdot B$ exist? If so, what size is it?
4. Suppose $A \in \mathbb{R}^{3\times 2}, y \in \mathbb{R}^3$. Is $y^T A$ a row vector or a column vector?
5. Supposed $A \in \mathbb{R}^{3\times 2}, x \in \mathbb{R}^2$. Is Ax a row vector or a column vector?
6. (5 points) Suppose $(B\mathbf{x}+\mathbf{y})^{\mathbf{T}}\mathbf{A}^{\mathbf{T}} = \mathbf{0}$, where A and B are both invertible $n \times n$ matrices and $\mathbf{x}$ and $\mathbf{y}$ are vectors in $\mathbb{R}^n$, and $\mathbf{0}$ is a vector of all zeros. Use the properties of matrix multiplication, transpose, and inverse to show that $\mathbf{x} = -\mathbf{B}^{-\mathbf{1}}\mathbf{y}$. Show your work.

Problem 2. (12 points) Vectorized Code Fundamentals.

Create a jupyter notebook called `problem_2.ipynb` and write Python code for the following problems. (Note: you *must* name your notebook exactly `problem_2.ipynb` or `pprint_hw2.py` will not print it correctly.)

1. Create the following variables as NumPy arrays
$$A = \begin{bmatrix} -2 & -3 \\ 1 & 0 \end{bmatrix}, B = \begin{bmatrix} 1 & 1 \\ 1 & 0 \end{bmatrix}, x = \begin{bmatrix} -1 \\ 1 \end{bmatrix}$$
2. Compute $C = A^{-1}$
3. Confirm that $A \cdot C = I$ and $C \cdot A = I$ where I is a 2×2 identity matrix.
4. Compute Ax
5. Compute $A^T A$
6. Compute $(AB)^T$
7. Compute $Ax - Bx$
8. Compute $||x||$
9. Compute $||Ax - Bx||$
10. Print the first column of A (do NOT use a loop—use array "slicing" instead)
11. Assign the vector x to the first column of B and call this new variable D. Do NOT use a loop—use "array slicing" instead.
12. Compute the element-wise product between the first column of A and the second column of A.

Problem 3. (40 points). Polynomial Regression.

In this problem you will implement methods for multivariate linear regression and use them to solve a polynomial regression problem. The purpose of this problem is:

1. To practice writing vectorized versions of algorithms in Python.
2. To understand how feature expansion can be used to fit non-linear hypotheses using linear methods
3. To understand feature normalization and its impact on numerical optimization for machine learning.

Open the notebook `multivariate-linear-regression.ipynb` and follow the instructions to complete the problem.

In this assignment, you will implement the K-means algorithm. You will then use it to perform image clustering, to test your implementation.

Part I: Clustering (25 points)

Write a function `my_kmeans.ipynb` to implement a basic version of the K-means algorithm.

Inputs: [5 pts for correct format of input/output]

- an NxD data matrix `A` where N is the number of samples and D is the dimensionality of your feature representation,
- the number `K` denoting how many clusters to output, and
- a value `iters` saying how many iterations to run for K-means.

Outputs:

- an Nx1 output `ids` containing the data membership IDs of each sample (denoted by indices randing from 1 to K, where K is the number of clusters),
- a KxD matrix `means` containing the mean/center for each cluster, and
- a scalar `ssd` measuring the final SSD error of the clustering, i.e. the sum of the squared distances between points and their assigned means, summed over all clusters.

Instructions:

1. [5 pts] First, initialize the cluster means randomly. Get the range of the feature space, separately for each feature dimension (compute max and min and take the difference) and use this to request random numbers in that range. Look for the function for generating `random numbers .`
2. [5 pts] Then, iterate over the following two steps. The first step is to compute the memberships for each data sample. Write a function (similar to Matlab function `pdist2)` to efficiently compute distances (check its documentation to see what inputs it expects). Then for each sample, find the min distance and the cluster that gives this min distance.
3. [5 pts] The second step is to recompute the cluster means, simply taking the average across samples assigned to that cluster, for each feature dimension.
4. [5 pts] Finally, compute the overall SSD error. It helps to keep track of the min distance per sample as you iterate.

Part II: Random restarts (5 points)

K-means is sensitive to the random choice of initial clusters. To improve your odds of getting a good clustering, implement a wrapper function `restarts.ipynb` to do `R` random restarts and return the clustering with the lowest SSD error.

Inputs: same as for `my_kmeans.ipynb` plus

- a scalar `R` denoting how many random restarts to perform.

Outputs: same as for `my_kmeans.ipynb`, but

- `ssd` is the lowest SSD across all random restarts.

Part III: Image segmentation using clustering (20 points)

You will next test your implementation by applying clustering to segment and recolor an image. Write your code in a script `segment.ipynb` .

1. [5 pts] Download the following images: `panda`, `cardinal`. Load them in python using image reading function. This will return a HxWx3 matrix per image, where H and W denote height and width, and the image has three channels (R, G, B). Convert the image to `double` format. To avoid a long run of your code, downsample the images (reduce their size) e.g. using `image resize(im, [100 100])function ;.`
2. [5 pts] To perform segmentation, you need a representation for every image pixel. We will use a three-dimensional feature representation for each pixel, consisting of the R, G and B values of each pixel. Use `im = reshape(im, H*W, 3);` to convert the 3D matrix into a 2D matrix with pixels as the rows and channels (features) as the columns. Use the random restarts function you wrote above, to perform clustering over the pixels of the image.
3. [5 pts] Then recolor the pixels of each image according to their cluster membership. In particular, replace each pixel with the average R, G, B values for the cluster to which the pixel belongs (i.e. recolor using the cluster means). Show the recolored image using `imshow`, but convert it to format `uint8` before displaying.
4. [5 pts] Experiment with at least five different combinations of settings for `K, iters, R`. Write a brief report (`report.pdf` or `report.docx`) documenting your findings about these, and include the image results inside the document.