Assignment (1)

Artificial Intelligence and Machine Learning (24-787)

Due date: 09/13/2022 @ 11:59 pm EST

Note: In case a problem requires programming, it should be programmed in Python. While programming, you should use plain Python, unless otherwise stated. For example, if the intention of a problem is to gain familiarity with the numpy library, it will be clearly noted in that problem to use numpy.

File Submission Structure

1. **HW1 Writeup**: Submit a combined pdf file to Gradescope. This file should contain the answers to the theoretical questions as well as the pdf form of the FILE.ipynb notebooks.

- Convert the jupyter notebook to a pdf file. Ensure that the submitted notebooks have been run and the cell outputs are visible - Hint: ***Restart and Run All*** option in the Kernel menu. Make sure all plots are visible in the pdf. For more details on how to convert a jupyter notebook to a PDF file, please reference this link.
- If an assignment has theoretical and mathematical derivation, you may scan your handwritten solution, or type your solution using Latex or Word.
- Then concatenate them all together in your favorite PDF viewer/editor. The file name should be formatted as **HW-assignmentnumber-andrew-ID.pdf**. For example for assignment 1, your filename should be HW-1-andrewid.pdf
- Submit this final PDF on Gradescope, and **make sure to tag the questions correctly!**

2. **HW1 Code**: Submit a ZIP folder to Gradescope, containing the FILE.ipynb notebooks for each of the programming questions.

- The ZIP folder containing your iPython notebook solutions should be named as **HW-assignmentnumber-andrew-ID.zip**
- You can refer to the numpy documentation while working on this assignment. Any deviations from the submission structure shown below would attract penalty to the assignment score. Please use Piazza for any questions on the assignment.

PROBLEM 1

Topic: Linear Regression

a) (Theoretical problem)- 5 points: Find the equation of the regression line that fits the points (-2, 2), (2, 4), (3, 8), (5, 11), and (4, 17) by using and solving the normal equations. (To submit your response to this question, you may handwrite your answer, or typeset your equations with Latex or Word, and then convert it to a PDF document.)

b) (Programming problem)- 5 points: Write a program which takes n numbers of (x, y) as input, and outputs b_0 and b_1, for the linear regression equation $y = b_0 + b_1 x$ that fits the (x, y) points. Use the normal equation. Verify the function you write by printing b_0 and b_1 within the Jupyter cell and comparing the results with (a). You can use the numpy and math libraries in Python.

c) (Programming problem)- 10 points: Using the dataset provided in this assignment (p1_data.csv) and the program you wrote in part b, calculate b_0 and b_1. You can use the numpy and math libraries in Python, do not use Pandas to load the csv file.

PROBLEM 2

Topic: Python Programming

Note: For programming problems, write the function with given title. You should write the code your own way and need to use the given function name.

a) (Programming problem)- 10 points: Write a function called `includes` which accepts a collection, a value, and an optional starting index. The collection can be a string, a list or a dictionary. The function should return `True` if the value exists in the collection. Otherwise it should return `False`. If the starting index is provided to the function, the search process should begin from that index in the collection, and exclude elements before the starting index. If no starting index is given, the search process should begin from the first index of the collection. If the collection is a dictionary, the starting index will be ignored and the entire dictionary will be searched, since objects in dictionaries have no intrinsic order. You can test your code and show your results in your jupyter notebook with the examples below.

```
def includes(item,val,start_ind=None):
    pass


includes([2, 3, 4], 2, 0) # True
includes([2, 3, 4], 2, 1) # False
includes([2, 3, 4], 4, 1) # True
includes({'a':1,'b':2}, 1) # True
includes({'a':1,'b':2}, 'a') # False
includes('abcd', 'b') # True
```

b) (Programming problem)- 10 points: Create a function `moving_average`. When the function is passed a value, the function returns the current average of all previous function calls. Round all answers to the 1st decimal place. You can test your code with the example below to see if it returns the correct values. You may use numpy but do not use `numpy.ma.average`. *Hint: You might need an inner function for this if no parameter is passed in the function.

```
def moving_average():
    pass

mAvg = moving_average()
print(mAvg(10)) #10.0
print(mAvg(11)) #10.5
print(mAvg(12)) #11.0
```

c) (Programming problem)- 10 points: Write a function called `same_frequency` which accepts two numbers and returns `True` if they contain the same frequency of the digits. Otherwise, return `False`. You can test your code with the examples given.

```
def same_frequency(num1,num2):
    pass

same_frequency(551122,221515) # True
same_frequency(321142,3212215) # False
same_frequency(12345,31354) # False
same_frequency(1212, 2211) # True
```

d) (Programming problem)- 10 points: K-means is a clustering algorithm that aims to partition n observations into k clusters, where each observation belongs to the cluster with the nearest mean (cluster center or cluster centroid). In this question, you are given a **.npz** file which contains the data points, cluster assignments, and

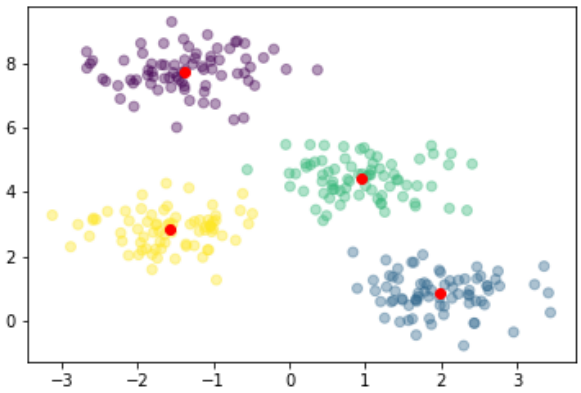

Fig. 1: Cluster assignment and centroids for Kmeans

centroids of the clusters with field names as 'data', 'pred' and 'centers' respectively. You are required to plot the data with cluster information using different colors, along with the centroids of the respective cluster.

You may use **numpy.load** to load the npz file, and extract the compressed fields in the npz file with the expression **npzfile['field_name']**. You may use **matplotlib.pyplot** for plotting. The output should look similar to Fig 1. For more information on .npz files, you may refer to `https://numpy.org/doc/stable/reference/generated/numpy.savez.html`.

e) (Programming problem)- 10 points: Implement the outer product function for numpy arrays. Your function should take two 1-dimensional numpy array as input and return a 2-dimensional numpy array. The outer product (also known as the tensor product) of 1-dimensional vectors x and y is defined as

$$x \bigotimes y = \begin{bmatrix} x_1y_1 & x_1y_2 & \cdots & x_1y_n \\ x_2y_1 & x_2y_2 & \cdots & x_2y_n \\ \vdots & \vdots & \ddots & \vdots \\ x_my_1 & x_my_2 & \cdots & x_my_n \end{bmatrix} \tag{1}$$

Write your code to achieve the results and use the numpy built-in function `numpy.outer()` to test your answers. Make sure to test your function with the following code. Show the correctness of your implementation of the product function by comparing with the matrix obtained by the built-in function. Also, explain in two lines why the inbuilt implementation is faster than the function you wrote.

```
numpy.random.seed(24787)
X = np.random.randint(-1000, 1000, size=3000)
Y = np.random.randint(-1000, 1000, size=3000)
def NUMPY_outer(X,Y):
    pass
```

PROBLEM 3

Python Programming- Linear Regression using Gradient Descent **[40 points]**

The goal of this exercise is to find the weights associated with the linear regression fitting process, using Gradient Descent instead of the Normal Equation that you used in Problem 1.

Write a program that performs Linear Regression using Gradient Descent. Find the b_0 and b_1 for the linear regression equation $y = b_0 + b_1x$ that fits (x, y).

We suggest that you implement two functions to achieve this. The first function should compute the cost (loss) and gradients (using least squares loss), based on the input arguments (training data, b_0 and b_1). The second function should update the weights, using the gradient descent parameter update, based on the arguments (training data, initial b_0, initial b_1, learning rate, number of iterations). Ideally, the second function would call the first function. Use the file **p3_data.npy**. The first column corresponds to the **x** values, and the second column contains the **y** values.
You can use **numpy**, **matplotlib** libraries in this problem.

☊Find the b_0 and b_1. Plot (x,y) data and the fitted line in two different colors. (Hint= set learning rate to 0.0001).

☊Plot the cost (loss) vs. iterations for learning rates (0.0002, 0.0006, 0.01,10).

☊What in your opinion is the best learning rate and why? Use the next cell in the jupyter notebook to write down your answer.

☊Derive the Normal equation that you used to solve the above problem. Also, explain Gradient Descent and describe three types of Gradient Descent.

☊What is the approximate upper bound of learning rate (i.e before divergence occurs)? Plot the cost function within the Jupyter cell to show your evidence of divergence occurring. Write down your conclusion in a new cell.

☊Use Mini-Batch Gradient Descent with batch size 20 and plot the cost per mini-batch. Choose an appropriate learning rate. What is your observation on this plot? Write it down in a new cell.