COM761 Machine Learning Coursework 1

2022/23

This is an individual assignment: University regulations on plagiarism will apply.

Enter your solutions directly into this Jupyter notebook.

For each question, you should provide your answer **in the cell immediately below the question**.

If a question requires a **numerical answer**, your cell should include python code carrying out the relevant calculations, and include a `print` statement to make clear what your answer to the question is.

If a question requires a **graphical plot**, your cell should include python code that results in the required plot being printed to screen.

You need to submit BOTH of the following via Blackboard before the deadline:

1. **This Jupyter notebook (containing your solutions), AND**
2. **A pdf of this Jupyter notebook (containing your solutions).**

   You can convert your Jupyter notebook to pdf by:

   - printing directly from your browser to pdf
   - if the above does not work in your browser, go to File-> Download As... -> HTML (.html). Then open the html file in Google chrome, and print from chrome to pdf.

Question 1

Application of Multiple Linear Regression

Total marks for question: 10

Download physical.txt from http://www.statsci.org/data/oz/physical.html (http://www.statsci.org/data/oz/physical.html)

(Note that the first row of this text file contains header information, i.e. the names of the variables.)

Throughout this question, you can use functions from `scikit-learn` .

Use all the data for training - you do not need to construct a validation set.

(a) Build a least squares multiple linear regression model to predict *mass* from the other ten attributes (*fore*, *bicep*, etc). Print out the weights (coefficients and intercept) for the model. **[2 marks]**

(b) Suppose you wish to reduce your model down to using only *one* feature x. Use recursive feature elimination from scikit-learn to determine which feature you should use. **[2 marks]**

(c) Using the feature x from **(b)**, construct a polynomial regression model of the form:

$$f(\mathbf{x}) = w_0 + w_1 x + w_2 x^2$$

Print out the weights for this model. **[3 marks]**

(d) Plot the polynomial regression model from **(c)**. **[3 marks]**

In [ ]:

```
## Enter code for part (a) below [2 marks]
```

In [ ]:

```
## Enter code for part (b) below [2 marks]
```

In [ ]:

```
## Enter code for part (c) below [3 marks]
```

In [ ]:

```
## Enter code for part (d) below [3 marks]
```

Question 2

The Normal Equation for Multiple Least Squares Linear Regression

Total marks for question: 10

The optimal coefficients for a least squares linear regression model $f(\mathbf{x}) = \mathbf{w}^T \mathbf{x}$ can be derived analytically, by differentiating the mean squares cost function with respect to the weight and setting the derivative equal to zero [1].

Doing this, the vector $\mathbf{w}$ of weights that minimizes the mean squares cost function is given by the *normal equation*:

$$\mathbf{w} = (\mathbf{X}^T \mathbf{X})^{-1} \mathbf{X}^T \mathbf{y}$$

where $\mathbf{X} = \begin{bmatrix} \mathbf{x}_1^T \\ \mathbf{x}_2^T \\ \vdots \\ \mathbf{x}_N^T \end{bmatrix} = \begin{bmatrix} 1 & x_1^{(1)} & x_2^{(1)} & \cdots & x_n^{(1)} \\ 1 & x_1^{(2)} & x_2^{(2)} & \cdots & x_n^{(2)} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & x_1^{(N)} & x_2^{(N)} & \cdots & x_n^{(N)} \end{bmatrix}$

where $x_i^{(j)}$ is the i-th feature of example j.

In this question, you are going to calculate the weights for the multiple linear regression model in Q1 (a), using the normal equation above.

(a) Using the data in physical.txt , construct a vector $\mathbf{y}$ that contains the values of *mass*, and a matrix $\mathbf{X}$ that contains the values of the other ten attributes (*fore*, *bicep*, etc) for x_i $(1 \leq i \leq 10)$. **[3 marks]**

(b) Write code to calculate the optimal weight vector $\mathbf{w}$ from the normal equation above. **[5 marks]**

(c) Print the value of $\mathbf{w}$ to screen, and compare to the weights you obtained using scikit-learn for this model in Q1 (a) (i.e. are they the same or not?) **[2 marks]**

[1] pp. 9-14, 22-27 of *A First Course in Machine Learning*, 2nd Edition.

In [ ]:

```
## Enter code for part (a) below [3 marks]


## Enter code for part (b) below [5 marks]


## Enter code and comment for part (c) below [2 marks]
```

Question 3

Experiments with Random search

Total marks for question: 10

In this question, you will use the random_search algorithm introduced in the lectures. The code is already provided in the cell below.

(a) Write a function called booth that implements the *Booth function*:

$$g(x_1, x_2) = (x_1 + 2x_2 - 7)^2 + (2x_1 + x_2 - 5)^2$$

This function should take in a numpy column vector representing $\mathbf{x} = \begin{bmatrix} x_1 \\ x_2 \end{bmatrix}$ and return the value of $g(x_1, x_2)$. **[2 marks]**

(b) Amend the random_search function to implement another option for alpha_choice called decay . In this setting, alpha should be set to $\alpha = e^{-k/5}$, where $e = 2.718281828\ldots$ is the natural base. **[2 marks]**

(c) Perform an experiment to investigate the effect of alpha_choice when optimizing the Booth function with the following settings:

- Number of steps $K = 50$
- Number of random directions at each step $P = 1000$
- Initial point $\mathbf{w}^0 = \begin{bmatrix} -2 \\ -2 \end{bmatrix}$

You should investigate three settings for α: fixed steplength of $\alpha = 1$, 'diminishing' and 'decay'.

Show the results from your experiments by plotting (on the same figure) the cost history (i.e. history of objective function values) against iteration number for the three settings of the random search. **[4 marks]**

(d) Comment on the worst performing setting of the random search in **(c)**, giving an explanation of why it does not perform as well as the other two settings. **[2 marks]**

In [ ]:

```
## Enter code for part (a) below [2 marks]



# random search function - ammend for part (b) [2 marks]
def random_search(g,alpha_choice,max_its,w,num_samples):
    # run random search
    weight_history = []          # container for weight history
    cost_history = []            # container for corresponding cost function history
    alpha = 0
    for k in range(1,max_its+1):
        # check if diminishing steplength rule used
        if alpha_choice == 'diminishing':
            alpha = 1/float(k)
        else:
            alpha = alpha_choice

        # record weights and cost evaluation
        weight_history.append(w.T)
        cost_history.append(g(w.T))

        # construct set of random unit directions
        directions = np.random.randn(num_samples,np.size(w))
        norms = np.sqrt(np.sum(directions*directions,axis = 1))[:,np.newaxis]
        directions = directions/norms

        ### pick best descent direction
        # compute all new candidate points
        w_candidates = w + alpha*directions

        # evaluate all candidates
        evals = np.array([g(w_val.T) for w_val in w_candidates])

        # if we find a real descent direction take the step in its direction
        ind = np.argmin(evals)
        if g(w_candidates[ind]) < g(w.T):
            # pluck out best descent direction
            d = directions[ind,:]

            # take step
            w = w + alpha*d

    # record weights and cost evaluation
    weight_history.append(w.T)
    cost_history.append(g(w.T))
    return weight_history,cost_history

w_init = np.array([[0,0]])

# Enter code for part (c) below [4 marks]



#Enter comment below for part (d); or alternatively type your answer into a new markdown ce
```

Question 4

Implementation of Mean Absolute Deviations Linear Regression

Total marks for question: 10

In the cell below you are given:

- data stored in x4 (feature) and y4 (label)
- code to compute the least squares cost function using this data
- code for training a least squares linear regression model using this data

(a) Implement code for the *Mean Absolute Deviation* (MAD) cost function, given by:

$$\text{MAD}(\mathbf{w}) = \frac{1}{N} \sum_{i=1}^{N} |y_i - \text{model}(x_i, \mathbf{w})|$$

where N is the number of examples we are training from. **[3 marks]**

(b) Use the random search algorithm from Q3 to find the weights w_0, w_1 that optimize the Mean Absolute Deviation for a simple linear regression model trained on the data stored in x4 and y4 . Print the values of the optimal model weights found using MAD. (Other than cost function, you should keep the settings passed to random_search the same as those used below for the least squares model). **[2 marks]**

(c) On the same figure, plot the MAD model and the least-squares model (as well as the data), and comment on the cause of the difference in models. **[5 marks]**

In [ ]:

```
## Data for use in Q4
x4 = np.array([[-0.372180347],[-0.630104555],[-1.150723051],[0.933616921],[-0.831975033],
                [-1.656988490],[1.124082010],[-0.393019792],[-0.732695333],[0.452214528]])
y4 = np.array([[-1.174803460],[-1.856390758],[-2.256630546],[-9.994548305],[-1.710868702],
               [-2.682265692],[0.191515393],[-1.247013155],[-1.684018106],[-0.624466506]])

# Code provided for Q4
def model(x,w):
    a = w[0] + w[1]*x[0]
    return a

def least_squares(w):
    cost = 0
    for i in range(len(x4)):
        cost += (model(x4[i],w)-y4[i])**2

    return cost/float(np.size(y4))

# the next two lines find the optimal weights for the simple linear regression model using
w_init = np.array([[0,0]])
wh_MSE,sh1_MSE = random_search(g=least_squares,alpha_choice='diminishing',max_its=50,w=w_in
print('model parameters found using least squares:',wh_MSE[-1])

## Enter code for part (a) below [3 marks]



## Enter code for part (b) below [2 marks]



## Enter code and comment for part (c) below [5 marks]
```