Programming Exercise 1

Shivang Ahuja - shivang.ahuja.18csc@bmu.edu.in - 9529058813

Aim:

1. Write a function to generate an m+1 dimensional data set, of size n, consisting of m continuous independent variables (X) and one dependent variable (Y) defined as **yi = xiβ + e**
2. Write a function that learns the parameters of a linear regression line given inputs
    a. X: An n × m numpy array of independent variable values
    b. Y : The n × 1 numpy array of output values
    c. k: the number of iterations (epochs)
    d. τ : the threshold on change in Cost function value from the previous to current iteration
3. The function should implement the Gradient Descent algorithm as discussed in class that initialises β with random values and then updates these values in each interaction by moving in the the direction defined by the partial derivative of the cost function with respect to each of the coefficients. The function should use only one loop that ends after a number of iterations (k) or a threshold on the change in cost function value (τ). The output should be an m + 1 dimensional vector of coefficients and the final cost function value.

Libraries Required and Used:

1. Numpy - for mathematical operations on matrices
2. Matplotlib - for plotting graphs
3. Scipy - for calculating cosine similarity

Function to generate a random dataset

```
[193] def generateRandom(dimension, number, varience):
        np.set_printoptions(precision=4)
        X = np.random.randn(number, dimension+1)
        X[:,0] = 1
        B = np.random.randn(dimension+1, 1)
        Y = (X.dot(B) + np.random.normal(loc=1, scale=varience))
        return X, B, Y
```

Function of Linear Regression

```
[194] def linerRegression(X, Y, epoch=10, lr=0.05, threshold=0.00001):
        dimension = np.shape(X[0])[0]
        Bp = np.zeros(shape=(dimension,1))
        costs = [0]
        Blist = []
        for iter in range(epoch):
          h = X.dot(Bp)
          loss = h - Y
          gradient = X.T.dot(loss) / len(Y)
          Bp = Bp - lr*gradient
          Blist.append(cosine(B, Bp))
          cost = cost_function(X, Y, Bp)
          if abs(cost-costs[-1]) < threshold:
            print("Threshold met at epoch: %2d" %(len(costs),))
            break
          costs.append(cost)
        #plot(range(len(Blist)), Blist)            #plot how Similirity between B an Bp increases
        #plot(range(len(costs)), costs)            #plot how Cost function is decreasing
        return Bp, costs
```

Function to calculate cost, cosine similarity and plot a graph

```
[195] def cost_function(X, Y, B):
        m = len(Y)
        J = np.sum((X.dot(B) - Y) ** 2)/(2 * m)
        return J
```

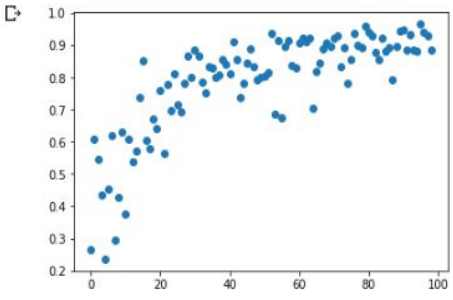
```
[196] def cosine(B, Bp):
        result = 1 - spatial.distance.cosine(B, Bp)
        return result
```

```
[197] def plot(X, Y):
        #plt.plot(X, Y)
        plt.scatter(X, Y)
        plt.show()
```

Variation in similarity between B (generated) and B (from linear regression function) with change in number of data points(value of n)

```
[203] cosineVal = []
      for i in range(1, 100):
        X, B, Y = generateRandom(20, i, 1)
        Bp, cost = linerRegression(X, Y, epoch = 100, lr = 0.005, threshold=0)
        cosineVal.append(cosine(B, Bp))

      plot(range(len(cosineVal)), cosineVal)
```

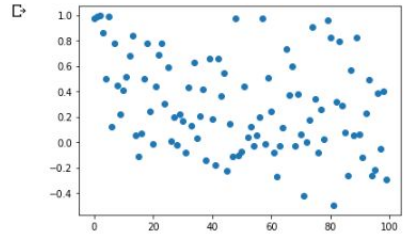


As per the graph increasing the number of data points (increase in value of n) leads to a better similarity between B (generated) and B (from Linear Regression Function)

Variation in similarity between B (generated) and B (from linear regression function) with change in the standard deviation of unexplained error.

```
[200] cosineVal = []
      for i in range(0, 100):
        X, B, Y = generateRandom(10, 1000, i)
        Bp, cost = linerRegression(X, Y, epoch = 100, lr = 0.005, threshold=0)
        cosineVal.append(cosine(B, Bp))

      plot(range(len(cosineVal)), cosineVal)
```

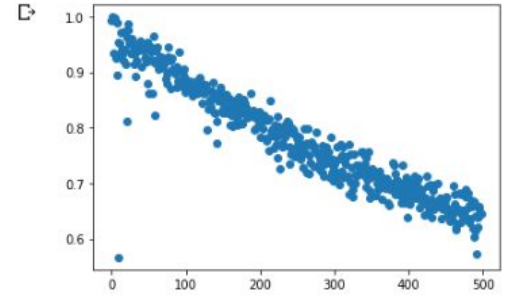


As per the graph increasing the value of standard deviation of unexplained error the similarity between B (generated) and B (from linear regression function) reduces. Though it is not that much prominent from this graph. But still the graph gives an estimate.

Variation in similarity between B (generated) and B (from linear regression function) with change in the number of parameters(value of m)

```
[201] cosineVal = []
      for i in range(2, 500):
        X, B, Y = generateRandom(i, 250, 1)
        Bp, cost = linerRegression(X, Y, epoch = 100, lr = 0.005, threshold=0)
        cosineVal.append(cosine(B, Bp))

      plot(range(len(cosineVal)), cosineVal)
```



From the graph it is clear that as we increase the number of parameters(value of m) while keeping the number of data points (n = 250) consent. The similarity between B (generated) and B (from linear regression function) reduces drastically.

This implies that if we keep the number of parameters less than the number of data points we can get a better estimate of B(coefficients of X) from Linear regression function.

Result: The following things were observed.

1. Increasing the number of data points will give a better estimate of B(coefficients of X) from Linear regression function. While keeping number of parameters constant
2. Increasing the standard deviation of unexplained error in the value of Y will reduce the accuracy of Linear Regression Model. Thus will not give a good estimate of B(coefficients of X) from Linear regression function. Here number of parameters(m) and number of data points(n) were kept constant

3. Increasing the number of parameters (more than number of data points) will not give a good estimate of B(coefficients of X) from Linear regression function.

Link to code:

https://colab.research.google.com/drive/1d2Oa4i-SbA-Kara--ibggi1l42x2eajb?usp=sharing