Madushani Lekam Wasam Liyanage

CSCI381 – Homework1

Report

In this project, we are computing pairwise Euclidean distance matrix and correlation matrix using nested loops and numpy matrix operations (no loops). Then we use three data sets from sklearn to test the functions we implemented. Also, we take the computation times of with-loop and non-loop methods and draw a bar chart to compare the times.

Problem 1 – Computing the Distance Matrix

1. With Loops

In this function, we use two nested for-loops to iterate over rows in the matrix. Then we calculate the pairwise distance between any two rows and put it in the output matrix.

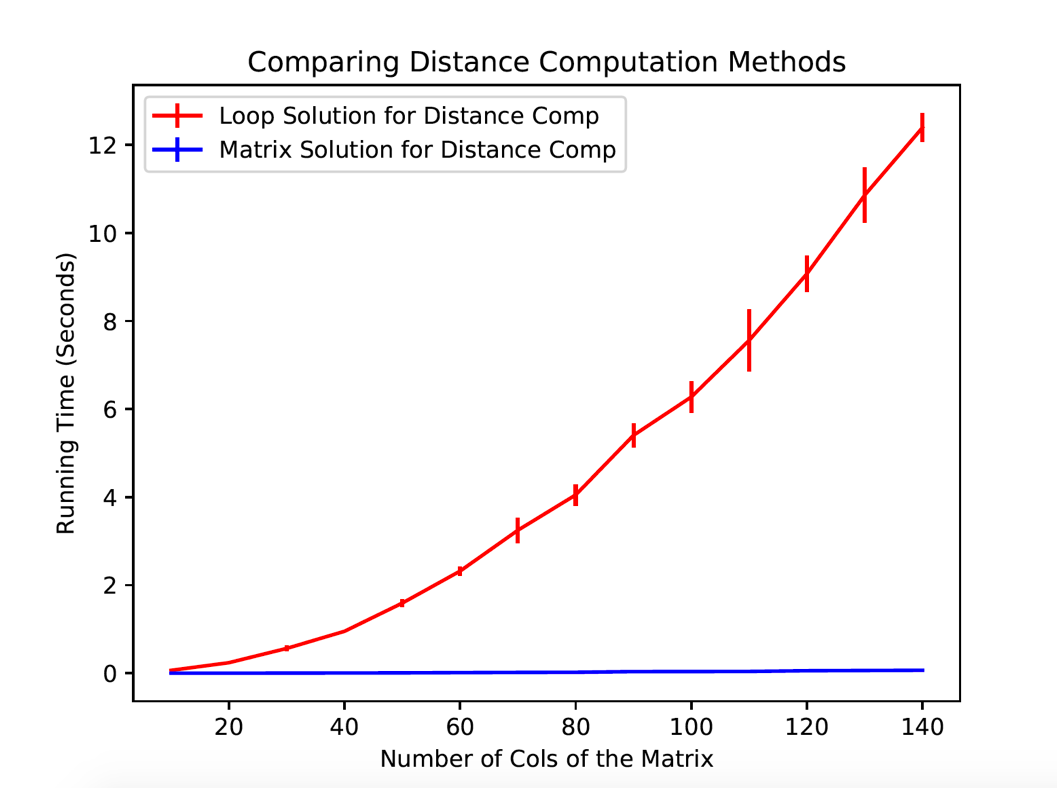
**def** compute\_distance\_naive(X):  
 N = X.shape[0] *# num of rows* D = X[0].shape[0] *# num of cols* M = np.zeros([N,N])  
 **for** i **in** range(N):  
 **for** j **in** range(N):  
 xi = X[i,:]  
 xj = X[j,:]  
 dist = np.linalg.norm(xi-xj)  
 M[i,j] = dist  
  
 **return** M

1. Without Loops (Numpy matrix operations)

In this function, we only used numpy matrix operations. Distance between two vectors x and y can be calculated by,

||x-y|| =

**def** compute\_distance\_smart(X):  
 N = X.shape[0] *# num of rows* D = X[0].shape[0] *# num of cols  
   
 # use X to create M* M = np.zeros([N, N])  
  
 sum = np.sum(np.multiply(X,X), axis=1)  
  
 x2 = (sum \* np.ones([N,1]))  
 y2 = np.transpose(x2)  
 xy = np.dot(X, X.T)  
  
 M = np.sqrt(abs(x2 - 2 \* xy + y2))  
 **return** M



Problem 2 – Computing the Correlation Matrix

1. With Loops

This naïve method is implemented with two nested for loops iterating over rows and columns of a given matrix. In the given matrix, the mean(mu) of each pair of column is computed. Then that mean(mu) matrix is used to compute the covariance. Then from that the standard deviation and variance matrices. At the end correlation matrix is computed.

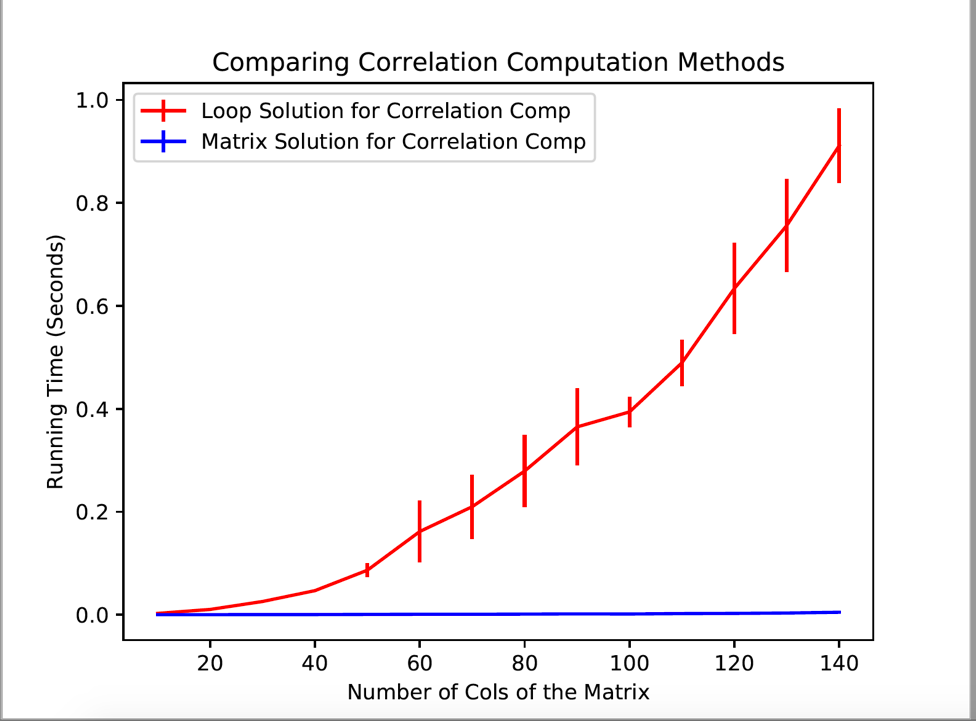
**def** compute\_correlation\_naive(X):  
 N = X.shape[0] *# num of rows* D = X[0].shape[0] *# num of cols  
  
 # use X to create M* M = np.zeros([D, D])  
  
 **for** i **in** range(D):  
 **for** j **in** range(D):  
 xi = X[:, i]  
 xj = X[:, j]  
  
 mui = np.sum(xi).astype(float) / N  
 muj = np.sum(xj).astype(float) / N  
 xni = xi - mui  
 xnj = xj - muj  
 sij = (np.dot(xni, xnj)).astype(float) / (N - 1)  
  
 sigmai = np.sqrt(np.dot(xni, xni).astype(float) / (N - 1))  
 sigmaj = np.sqrt(np.dot(xnj, xnj).astype(float) / (N - 1))  
 sigma = sigmai \* sigmaj  
 cmatrix = sij.astype(float) / sigma  
  
 M[i, j] = cmatrix  
  
 **return** M

1. Without Loops (Numpy matrix operations)

In this function, we solely use numpy matrix operations to compute correlation matrix without using any loops.

Mean(mu) vector is calculated first and then a matrix is created from that vector. Then covariance is calculated from taking the dot product of Original Matrix – mean matrix and transpose matrix of the Original Matrix – mean and dividing it by number of rows.

**def** compute\_correlation\_smart(X):  
 N = X.shape[0] *# num of rows* D = X[0].shape[0] *# num of cols  
  
 # use X to create M* M = np.zeros([D, D])  
  
 vectormu = (np.sum(X, axis=0)).astype(float) / N  
 matrixmu = vectormu \* np.ones([N, 1])  
 x = X - matrixmu  
 xT = np.transpose(x)  
 covarience = (np.dot(xT, x)).astype(float) / (N - 1)  
 x2 = np.multiply(x, x)  
 varience = (np.sum(x2, axis=0)).astype(float) / (N - 1)  
 sigma = np.sqrt(varience)  
 denmatrix = np.outer(sigma, sigma)  
  
 **with** np.errstate(divide=**'ignore'**, invalid=**'ignore'**):  
 c = np.power(denmatrix,-1)  
 c[c == np.inf] = 0  
 c = np.nan\_to\_num(c)  
 M = np.multiply(covarience, c)  
  
 **return** M



With the generated pdfs, we can see how runtime exponentially grows in loop calculations according to the number of columns of the matrix, while runtime stays constant in matrix solution.

Problem 3

We used the above implemented methods to compute Euclidean distance matrix and Covariance matrix of data sets of different sizes and plotted the results. With the pdf file generated, we can see how the compute-time largely varies between loop method and non-loop method when it comes to larger data sets. Therefore, the conclusion is, it’s more efficient to use Numpy operations for matrix computations.

