a)

**Finding The best fit line:**

Y =mx+c is the equation that represents a straight line along 2-Dimensional data, i.e., x-axis and y-axis.

**Calculating the slope of the line:**

|  |  |  |  |
| --- | --- | --- | --- |
| **X** | **y** | **xy** | **x2** |
| 2 | 8 | 16 | 4 |
| 5 | 25 | 125 | 25 |
| 3 | 9 | 27 | 9 |
| 10 | 40 | 400 | 100 |

m = (4(568) - (20) (82))/ (4(138) - (20)2)

m = (2272 - 1640)/ (552 - 400)

m = 632/ (152)

m = 4.1579

**calculating the y-intercept:**

The y-intercept of a line is the value of y at the point where the line crosses the y-axis.

c= (∑y – m∑x)/(n) = (82 – (4.1579) (20))/4 = (82 – 83.158)/ (4) = (-1.158)/ (4) = -0.2895

**Substitute the values in the final equation y= mx + c**

y-mean **=** 20.5

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **X** | **y** | **Y\_pred=mx+c** | **error** | **Root Mean Square Error (rmse)**  **(error)2** | **ss\_tot**  **(y- (y\_mean))2** |
| 2 | 8 | (4.1579) (2) – 0.2895 = 8.0263 | -0.0263 | (-0.0263)2 = 0.00069169 | (-12.5)2 = 156.25 |
| 5 | 25 | (4.1579) (5) – 0.2895 = 20.5 | 4.5 | (4.5)2 = 20.25 | (4.5)2=20.25 |
| 3 | 9 | 12.1842 | -3.1842 | (-3.1842)2 = 10.13912964 | (-11.5)2= 132.25 |
| 10 | 40 | 41.2895 | -1.2895 | (-1.2895)2= 1.66281025 | (19.5)2 = 380.25 |

**rmse** = √∑rmse/n = √32.0526315/4) = √8.01315788 = 2.830752175659325

**r2** = 1- (∑rmse/∑ss\_tot) = 1 – (32.0526315/689) = 1 – 0.04652051 = 0.95347949

b)

Form of the required equation is **θ = (xTx)-1xTy**

xTx==

(xTx)-1=

xTy = =

θ =(xTx)-1xTy = =

c)

**Normal Equation** is the Closed-form solution for the Linear Regression algorithm which means that we can obtain the optimal parameters by just using a formula that includes a few matrix multiplications and inversions.

d)

Graphical user interface, text, application

Description automatically generated

Graphical user interface, text, application

Description automatically generated

Graphical user interface

Description automatically generated with low confidence

**2. PCA Analysis**

#Import Libraries

import numpy as np

import pandas as pd

import matplotlib.pyplot as plt

import seaborn as sns

import random

from mnist import MNIST

from sklearn.preprocessing import StandardScaler

from scipy.linalg import eigh

The MNIST Dataset is currently being loaded.

mndata = MNIST('C:\\Users\\ramne\\OneDrive\\Documents\\mnist')

training\_images, training\_labels = mndata.load\_training()

test\_images, test\_labels = mndata.load\_testing()

print("the shape of the training\_images data is : ",

np.array(training\_images).shape)

print("the shape of the training\_labels data is : ",

np.array(training\_labels).shape)

print("the shape of the test\_images data is : ",

np.array(test\_images).shape)

print("the shape of the test\_labels data is : ",

np.array(test\_labels).shape)

#From the training dataset, extracting the column label

label = training\_labels

ind = np.random.randint(0,20000)

plt.figure(figsize = (20,5))

grid\_data =

np.array(pd.DataFrame(training\_images).iloc[ind]).reshape(28,28)

#Using the matplotlib imshow() method to plot a random sample data point

from the training dataset.

plt.imshow(grid\_data, interpolation = None, cmap = 'gray')

plt.show()

#Column standardization of the training dataset via the

sklearn.preprocessing module's standardScalar class.

#Because after our data has been column standardized, the mean of each

feature is 0 (zero) and the variance is 1.

#As a result, we begin PCA with the origin point.

scalar = StandardScaler()

std\_df = scalar.fit\_transform(training\_images)

print("Shape of the dataset after the column standardization:",

std\_df.shape)

#Using the numpy matmul method, find the co-variance matrix AT \* A.

covar\_mat = np.matmul(std\_df.T, std\_df)

print("the dimensions of co-variance matrix after multiplication",

covar\_mat.shape)

#The parameter 'eigvals' is defined (low value to high value)

#Finding the top two eigen-values and related eigen vectors for projection

onto a 2D surface

# The eigen values will be returned in ascending order by the eigh

function.

values, vectors = eigh(covar\_mat, eigvals = (782,783))

print("Dimensions of eigen vector:", vectors.shape)

vectors= vectors.T

print("Dimensions of eigen vector:", vectors.shape)

#Find two major components by multiplying the two top vectors by the covariance matrix. PC1 and PC2 are two different types of computers.

final\_df = np.matmul(vectors, stu\_df.T)

print("vectors:", vectors.shape, "n", "std\_df:", std\_df.T.shape,"n",

"final\_df:", final\_df.shape)

#Stack final df and label vertically, then transpose them to find the

NumPy data table.

#Using PCA, I was able to convert 60000 \* 784 data to 60000\*4.

final\_dfT = np.vstack(final\_df, label).T

dataFrame = pd.DataFrame(final\_dfT, columns = ['pca\_1', 'pca\_2', 'label'])

print(dataFrame)

#Now let's use the seaborn Facet Grid technique to visualize the final

data.

sns.FacetGrid(dataFrame, hue = 'label', size = 8)\

.map(sns.scatterplot, 'pca\_1','pca\_2')\

.add\_legend()

plt.show()

A screenshot of a computer

Description automatically generated with medium confidence

A picture containing text, clipart

Description automatically generated

Text, letter

Description automatically generated

Text

Description automatically generated

[6000 rows \* 3 columns]

Chart, scatter chart

Description automatically generated

2 a)

x from mnist import MNIST

import numpy as np

import matplotlib.pyplot as plt

mnist = MNIST('C:\\Users\\ramne\\OneDrive\\Documents\\mnist')

X\_train,y\_train = mnist.load\_training()

M = np.zeros(28,28,10)

S = np.zeros(28,28,10)

for i in range(9) :

 X\_subset = X\_train[y\_train == i]

 M[:,:,i+1] = np.mean(X\_subset, axis = 0)

 S[:,:, i+1] = np.mean(X\_subset, axis = 0)

 plt.subplot(2,10,i+1)

 plt.imshow(M[:,:,i+1])

 plt.subplot(2,10,i+11)

 plt.imshow(S[:,:,i+1])

plt.show()

A picture containing text, monitor, indoor, display

Description automatically generated

2. b)

**Solution:**

Let X be NxD matrix

X = UΣVT (SVD of X)

C = (1/ (N−1)) XXT (dxd covariance matrix)

Now, XTX = VΣUT UΣVT

XTX = VΣΣVT

XTX = VΣ2VT

C = V (Σ2/N−1) VT

But C is symmetric, Hence C = VΛVT

Therefore, the eigen vectors of C were same as matrix V and eigen values of C can be derived from

λi = (σ2i) /(N−1)

3. c)

Step 1: Define a function, and pass Principal Components (PC), and Mean of Dimension Vectors (dM) as output arguments. Pass Input Matrix D as input argument.

Step 2: Find the Mean between Input Matrix (d) and 2 and assign the value to a variable called **mn.**

Step 3: Assign the size(D,2) to m.

Step 4: Assign the value of D - repmat(mn,1,m) to D

Step 5: Assign the value of D'/sqrt(m-1) to y

Step 6: [u,S,PCno] = svd(Y);

Step 7: PC = PCno(:,1:k);

Step 8: Assign the mean value of mn to dM

Step 9: End the function

Step 10: Load MNIST Train Images

Step 11: Reshape the image Dataset to (28,28,60000)

Step 12: Apply PCA\_SVD

Step 13: Reduce the train data to 10 dimensions

Step 14: Reshape principal components to display as image

Step 15: Display the images doing contrast stretching.

3. d)

from mnist import MNIST

import numpy as np

import matplotlib.pyplot as plt

def pcs\_svd(i\_train, i):

 PC = np.dot(i\_train, i\_train.T)

 reconn = np.dot(PC, i\_train)

 normal = np.dot(reconn, PC)

 dif = np.sum(normal=i\_train)

 dif = dif/(60000\*784)

 diff[i] = dif

 if(name\_ == '\_main\_'):

 i\_train =

np.loadtxt('C:\\Users\\ramne\\OneDrive\\Documents\\mnist\\t10k-imagesidx3-ubyte')

 diff = np.zeros(784)

 for i in range(784)

 pcs\_svd(i\_train, i)

 plt.plot(diff)

 plt.show

4. a)

Gaussian distribution in one dimension is a classic distribution over a single scalar random variable x and parameterized by a mean, µ and a standard deviation σ or equivalently its variance Sigma squared.

N (x; µ, σ) = (1)/ ( ) exp [((-1/2) (x-µ)2)/σ2]

Parameters: mean µ, variance σ2

The multivariate Gaussian is simply a vectorized version of the ordinary Gaussian. Suppose we have a random vector variable x follows a mean-centered multivariate Gaussian distribution μ as well as a covariance matrix Σ, the probability density function (pdf) of this object is given by:

Text

Description automatically generated with medium confidence



Finally, the linear transformations affect the parameters of a Gaussian.



We can use this to "standardize" any Gaussian to have a mean of 0 and a variance of 1 by using this formula.

A picture containing text

Description automatically generated

**Covariance Matrix**

Covariance is the most important feature of a multivariate Gaussian distribution.

The covariance matrix has two key properties:

The covariance matrix is positive semi-definite.

In a multivariate Gaussian distribution, the covariance matrix is positive definite.

### **Covariance matrix in multivariate Gaussian distribution is positive definite**

It is necessary for the covariance matrix to be invertible.

Because Σ is invertible, it must have full rank, and the linear system Σ

x=0 has just one solution, which is x = 0.

Because Σ is symmetric, we know that it may be decomposed as Σ = BT B according to Lemma 2.

As a result, any non-zero vector y,yT Σ y=0 if and only if By=0. Because yT Σ y=yyBTBy=(By)T(By). If By=0, Σy=BTBy=0. Because Σ is full rank and y! =0, there is no solution for linear system yT Σ y=0. Therefore, yT Σ y>0 and Σ is positive definite.

**b)**

**Definition:**

Any random variable X with a multivariate Gaussian distribution can be interpreted as the result of applying a linear transformation (X = BZ + µ) to some collection of n independent standard normal random variables (Z).

**C)**

Let’s consider a density function,

F(x/σ) = (1)/ (2 σ) exp (-|x|/ σ)

The log-likelihood function is

l(σ) = ∑ [-log2-log σ- (|xi|/ σ)]

Let the derivate with respect to θ be a zero –

lI(σ) = ∑ [(-1/ σ) +(|xi|/ σ2] = (-n/ σ) +( ∑|xi|/ σ2) = 0

This gives the MLE for σ as: σ=(∑|xi|/(n))

**d)**

We can assume that the probability density is without substantial loss of generality (or mass) f(xi) for any observation xi (out of n observations) is strictly positive, allowing it to be written as an exponential.



for a parameter vector θ=(θj).

Equating the log likelihood function's gradient to zero (which discovers stationary points of the likelihood, including all interior global maxima if one exists) produces a set of equations of the type

A picture containing text

Description automatically generated

one for every j We'd want to be able to isolate the xi terms from the words to have a ready answer for any of them. The most general method is to write the equations in the form:



The answer is achieved by solving the simultaneous equations for known functions nj, Tj, and αj.

A picture containing text, watch

Description automatically generated

Text

Description automatically generated

