INF 552 Machine Learning for Data Informatics

Homework #2

Group Members

Tanmay Diwan

USCID: 5047-3478-66

Kshitija Godse

USCID: 6833-7845-04

Table of Contents **Page No.**

1. Problem 1:- Expectation Maximization Algorithm

1.1 Abstract:

Gaussian Mixer Models are probabilistically-grounded way of doing soft clustering. Each cluster is a generative model i.e. Gaussian or multinomial. Mean/covariance are unknown parameters. Expectation Maximization algorithm automatically discovers all parameters for the K sources. This algorithm works for higher dimensions and discrete data. It consists of two steps E-step and M-step. M-step maximizes likelihood of the data. This algorithm converges when change in P(x1,..xn) is sufficiently small. Inthis instances can come from multiple clusters. Co-variance is a notion of distance changes over time.

$$P(X_i|C) \quad = \frac{1}{\sqrt{2\pi|\sum c|}} + e\{-\tfrac{1}{2}(X_i - \mu_i).T \sum{}^{-1} (X_i - \mu_i)\}$$

$$P(C|X_i) = \frac{P(X_i|C)P(C)}{\sum_{c=1}^{k} P(X_i|C)P(C)}$$

$$W_{ic} = \frac{P(C|X_i)}{P(C|X_1) + \cdots + P(C|X_n)}$$

$$\mu_{ca} = W_{c1}X_{c1} + \cdots + W_{ca}X_{na}$$

1.2 Approach & procedure:

1.2.1 Data Structure and Procedure-

- newMatrix() – This function initializes co-variance matrices which calculates each element in the co-variance matrix.
- expFunc() – Expectation step is computed in this function.
- maximization()- this function is for maximization step
- em()- Expectation Maximization Algorithm is computed in this function.
- We did this programming in Python language, using numpy and math functions.

1.2.2 Pseudo Code-

```
Create an array of first columns of data points
X= Store it in a matrix form
centroidlist= 3 random indexes for initial centroids
define function newMatrix()
    initialize a,b as a list
    x11=sum([(dataPoints[i][0]-newCentroidsList[k][0])**2 for i in range(0,len(dataPoints))])
    x11= x11/(len(dataPoints))
    a.append(x11)
    do same for x22 and append it to b
    x12=sum([(dataPoints[i][0]-newCentroidsList[k][0])*(dataPoints[i][1]-
    newCentroidsList[k][1]) for i in range(0,len(dataPoints)) ])
    x12= x12/(len(dataPoints))
    a.append(x12)
    do same for x21 and append it to b
    A = matrix([a,b])
Return A

Define expFunc()
    Declare variable finale=[]
    For k from 0 to length of data points
        A, e =[]
        For j from 0 to length of newcentroidList
            l=np.asmatrix(lisMat[j])
            t=1.0/((2*np.pi*np.linalg.det(l))**0.5)
            y=(-0.5)*((X[k]- mu[j])*l.I*(X[k]- mu[j]).T)
            append y into a
            append (t*math.exp(y)) into e
        append e into finale
    fE= np.array(finalE)

    P_Cx=fE/fE.sum(axis=0)
    return P_Cx
define maximization()
    declare temp as a list
    for I from 0 to 3
        append  (P_Cx[0:len(dataPoints),i]  *X)/(sum(P_Cx[0:len(dataPoints),i])*150  )  into
temp
    convert temp into array
    declare temp1 as a list
    for i from 0 to 3
        append temp into temp1
    declare variable lisMat as a list
    for k from 0 to 3
        declare a, b = []
        x11=sum([P_Cx[0:len(dataPoints),k][i]*(dataPoints[i][0]-
mu[k][0])**2foriinrange(0,len(dataPoints))])
        x11= x11/(len(dataPoints))
        append x11 into a
        do the same for x22 and append it to b
        x12=sum([P_Cx[0:len(dataPoints),k][i]*(dataPoints[i][0]-
        mu[k][0])*(dataPoints[i][1]-mu[k][1])for i in range(0,len(dataPoints)) ])
        x12= x12/(len(dataPoints))
        append x12 into a
        do the same for x21 and append it to b
        s=array(a,b)
        append array of a,b into lisMat
return temp1, lisMat
define em()
    declare variable count
    increase count by +1
    call expFunc and store the value in ResExp
    call maximization and store the result in muCentroids, coMatrix
    if muCentroids and mutemp arrays are equal or count=10 then
        for i upto 3
            print muCentroids and coMatrix
            return muCentroids, coMatrix
        call em()
    convert newCentroidList into array and save it to mu
    declare variable lisMat1 as a list
    for i upto 3
        A=newMatrix(newCentroidsList,i)
        Append A into lisMat1
    Call em()
```

1.3 Code level optimization:

- List Comprehensions: Most of the functions use Python's in built method of "List comprehensions" to generate lists.
- Arrays and Matrices: Higher level objects such as matrices and arrays optimize operations on these datasets to make the program acceptably fast

1.4 Result:

Input Matrix:

```
[[-1.05327784 -1.52514851]
 [-1.15427326 -1.63669976]
 [-0.77614321 -0.27192839]
 [-1.39725873 -1.37222178]
 [-0.43275423 -0.71180799]
 [ 0.36791474  0.16035139]
 [-0.62683434  0.2692529 ]
 [-0.08187465  0.73112803]
 [-0.32592173 -1.16427876]
 [ 0.44190604  0.22068693]
 [-0.5602329  -0.17368193]
 [-0.23015493  0.51436157]
 [ 1.36048579 -0.25169782]
 [ 0.39393455  0.18988241]
 [-1.12746565  0.43049151]
 [ 0.37459467 -0.72061083]
 [-1.12043209 -0.75723674]
 [-0.43719234 -1.40137091]
 [-0.23279108 -0.48001267]
 [ 0.19189515 -0.15130048]
 [-0.73500182 -0.96656475]
 [-0.051041   -0.36162415]
 [ 0.84997416 -0.67956683]
 [-1.83813945 -0.73543868]
 [-1.23198857  0.16326425]
 [ 0.42955969  0.19071943]
 [-0.65943686  0.43708275]
 [-0.47236228 -1.22023169]
 [ 0.23134149 -0.51609533]
 [ 1.18154264  0.07106786]
 [-1.39666475 -1.37496123]
 [-1.10928832  1.01476331]
```

```
[-0.50112326 -0.24198133]
[-0.4440651  -0.81954177]
[-0.36628273  0.35137601]
[-0.67297981 -0.13520382]
[ 0.44762988 -0.33399959]
[-1.10970563 -0.91984898]
[-0.2952468  -1.72518342]
[ 0.27947804 -1.13014602]
[-0.17001171 -0.89420508]
[-0.89947112 -0.07925568]
[-0.35626267  0.04094325]
[-0.67658679 -0.99785329]
[-0.88305643  0.19951456]
[ 0.54527912  0.08021451]
[-0.58901561 -0.50821315]
[-0.86719796 -1.12098931]
[-0.99736616  0.24690816]
[-0.35579169 -0.51134534]
[-1.10304721 -0.71708584]
[-0.68901454 -0.57218207]
[-0.60027316 -0.66009996]
[-1.24963478 -1.03578438]
[-0.16103813 -0.98539054]
[-1.47448989 -1.17208318]
[-1.30642145 -0.41608608]
[-0.37116204 -0.79579787]
[-0.66517328 -1.02991141]
[-0.91183433 -0.53465158]
[-0.76774427 -0.97567656]
[-0.53232388 -0.62063955]
[-1.05546663 -0.22338008]
[-0.46793737 -0.43817408]
[-0.5434458  -0.84555919]
[-0.78630597 -0.82837036]
[-0.70899239 -0.06735582]
[-0.97720501 -0.84950575]
[-1.05173458 -0.67438129]
[-0.89868485 -0.66517172]
[-0.43511336 -0.44296793]
[-0.84994959 -0.70624624]
[-0.91926265 -0.65023293]
[-1.09185457 -0.66659573]
[-1.1422867   0.21162983]
[-0.76430162 -0.46106333]
[-0.43127615 -1.11135393]
[-0.36872026 -1.23786672]
[-0.58830327 -1.38532011]
[-0.87262355 -0.10336482]
```

```
[-0.89178023 -1.08999459]
[-0.65193907 -0.41799653]
[-0.46490218 -0.5880563 ]
[-0.4528633  -1.35262575]
[-1.46463504 -0.0355444 ]
[-0.78673591 -0.58119994]
[-1.08777031 -0.11076288]
[-0.92568238 -0.65899155]
[-0.20394636 -0.71950383]
[-1.02586659 -1.16435065]
[-1.32230562 -0.77217914]
[-0.71363197 -0.73525061]
[-0.81225381 -1.35052009]
[-0.90746488 -0.79315023]
[-0.48386752 -0.72555748]
[-0.58635746 -1.05371495]
[-0.64857531 -0.63377527]
[-0.86212863 -1.23607977]
[-0.6570205  -0.99901965]
[-0.86682359 -0.7470245 ]
[ 0.84823771  0.40897727]
[ 1.43002458  0.7525725 ]
[ 0.98550593  1.292017  ]
[ 0.69582441  1.60213898]
[ 1.71885233  1.16915197]
[ 1.53101101  1.12293118]
[ 1.51476152  1.30988898]
[ 1.57920652  2.7742926 ]
[ 1.76294517  1.28295883]
[ 1.75243573  1.15960061]
[ 1.21014347  2.08317451]
[ 0.93595469  1.1266805 ]
[ 1.12836351  0.63661598]
[ 1.89236726  1.66804012]
[ 0.51240611  0.83912049]
[ 0.87129758  1.19901677]
[ 1.78214452  1.63227936]
[ 1.67035992  1.58838294]
[ 0.91030906  0.03869755]
[ 1.07388306 -0.44891609]
[ 1.10278942  0.61816051]
[ 1.03502873  2.13077678]
[-0.19317142  1.22020673]
[ 0.94895546  0.97160123]
[ 1.36615478  0.96894321]
[ 2.58000065  1.73633864]
[ 0.36240135  1.28152553]
[ 0.83098875  1.37492177]
```

```
 [ 0.97266566 -0.27439602]
 [ 0.65720689  0.95728099]
 [ 0.8093286   1.08146174]
 [ 0.10087197  0.5164066 ]
 [ 1.12736561  1.0543807 ]
 [ 0.89729351  0.68203198]
 [ 1.49415139  1.62819925]
 [ 1.22358058  0.70227952]
 [ 1.79489678  1.90762265]
 [ 2.10928745  1.04172323]
 [ 1.26009248  1.64467051]
 [ 0.48964161  2.53756352]
 [ 1.32936585  1.11207856]
 [ 1.98324191  2.02967417]
 [ 1.13696543  0.37987743]
 [ 1.41066444  2.24095889]
 [ 0.64358867  0.29249536]
 [ 2.04608839  1.02710393]
 [ 1.41984711  0.85907835]
 [ 0.95189109  0.3305925 ]
 [ 0.24263447  0.22847626]
 [ 0.80147786  1.48454794]]
Intial means:
[[-0.433871895, -1.721150679], [-1.391116494, 0.905055556], [-0.085368179, -2.169919876]]


Final Means with Covariance Matrices:
Mean 1
[-0.90977865 -1.15900432]
Covariance Matrix
[[ 1.29526143  0.09425688]
 [ 0.09425688  0.98331287]]
Mean 2
[ 4.30415011  3.34448181]
Covariance Matrix
[[ 3.93264202  2.32424515]
 [ 2.32424515  4.98812531]]
Mean 3
[-1.31571783  1.23296349]
Covariance Matrix
[[ 0.80128781 -0.21315653]
 [-0.21315653  1.19859329]]
```

2. Problem 2:- Principal Component Analysis

2.1 Abstract:

Datasets are typically high dimensional, but the true dimensions are often much lower. Machine learning provides Dimensionality reduction technique. It only structures data that affects class separation. This analysis defines a set of principal components. First m<<d components become m new dimensions. It uses covariance matrix & eigenvectors for the computations. Principal components are eigenvectors with largest eigenvalues.

$$\text{Eigenvalue} = \det(\sum - \lambda |) \quad \text{Eigenvector} = \sum e_i = \lambda_i e_i$$

Using original coordinates and new coordinates center of the data is calculated. This mean value is further projected on all the data points and new dimensions are computed. Such dimension is selected which maximizes the variance.

2.2 Approach and Procedure:

2.2.1 Data Structure and Functions-

- Array data structure is used to store the data points. Numpy function .linalg.eig is used to calculate eigenvalues and eigenvectors.
- reverse(), sort(), reshape() these functions are used to reverse the list, sort the list and reshape the list respectively.
- Numpy function .hstack is used to stack the sequence of arrays horizontally to make a single array.

2.2.2 Pseudo Code-

```
Take input data points into an array a
Calculate mean vector mean_vec
Calculate covariance matrix = (a - mean_vec).T.dot((a - mean_vec)) / (a.shape[0]-1)
Print covariance matrix
Calculate eigenvalues and eigenvectors = .linalg.eig(covariance matrix)
Make single list eig_pairs consisting of eigenvalues and eigenvectors
Sort this eig_pairs list
Reverse this eig_pairs list
Form a matrix matrix_w by stacking the arrays of eig_pairs
Take a dot product of data points and matrix_w
Print the result
```

2.3 Code level optimization:

- List Comprehensions: Most of the functions use Python's in built method of "List comprehensions" to generate lists.
- Function .linalg.eig is used to calculate the eigenvalues.
- Function .reshape Gives a new shape to an array without changing its data.
- Function .hstack to stack arrays horizontally in a single array.
- Matrix Manipulation: A matrix is a specialized 2-D array that retains its 2-D nature through operations. Matrix multiplication is optimized.

2.4 Result:

```
#######Input Vector##########
[[-1.98793769 -0.74538471 -0.17029177 -0.09803062]
 [-1.42961762 -1.46092472 -1.21433657 -3.45998017]
 [ 1.80783608  0.67599408  0.97147546 -0.18889123]
 [ 3.29772301  2.10358676  2.62706108  3.05998598]
```

```
[-0.75943517  0.79793136  0.83288684  2.22314027]
[-2.23485798  0.03902535 -0.17455733  0.92476861]
[ 1.71141252  0.69028557  1.25537412  0.48962121]
[ 0.01959636 -0.18458676 -0.15380813 -1.51179925]
[-4.2019328  -2.13935218 -1.60571136 -2.14884214]
[ 4.72157406  3.12058589  2.75853392  2.91607966]
[ 2.69293979  1.270206    1.22523812  0.0277424 ]
[-3.38353061 -1.69356337 -1.42517573 -2.16038433]
[ 3.39635393  2.45595154  2.32148217  2.70256192]
[-1.32594826 -0.74166322 -0.06301759 -0.54175005]
[-0.99746067  0.70941161  0.84752415  2.40192054]
[ 3.51386292  2.12831706  1.84642452  1.30730312]
[-1.0583326  -1.05918767 -0.99412821 -2.98911155]
[-1.79808487 -0.39720441 -0.07846954  0.24394152]
[ 1.26134577  0.88039868  1.18461391  0.98828058]
[ 4.93885291  4.38874317  3.43667692  5.32324417]
[ 0.73533921  1.80395995  1.69169912  3.45201885]
[-0.81651255  0.76618359  0.626007    1.83471014]
[ 0.4112564   0.54632534  0.68343185  0.50193246]
[-0.7896201   0.15673691  0.33551328  0.61738365]
[ 3.9413336   3.71014544  3.20089484  5.17060159]
[-0.99077755 -0.15425289  0.69863893  1.23380243]
[-2.34752614 -0.66514723  0.0951759   0.87273074]
[ 1.2237241   1.21173532  1.89255516  2.77312156]
[ 0.87118277  1.29149389  0.61573555  0.65178229]
[ 2.29789895  2.99394243  2.71835257  5.13274863]
[ 4.16093716  3.46634803  2.42601516  3.15744131]
[ 4.54768464  4.2404643   3.31824655  5.32927275]
[ 2.37273966  2.83461187  1.55257596  2.56702408]
[ 1.09247697  2.14832045  1.56637661  3.1885967 ]
[ 3.00420666  2.60184273  2.57138028  3.74039662]
[-1.40931793 -1.28703327 -0.65693607 -2.1915874 ]
[-1.21031979  0.51861676  0.03963256  0.80820167]
[-2.50204452 -1.71683292 -1.27916546 -2.7731194 ]
[ 2.98457486  2.23021109  1.4308265   1.10728923]
[ 2.08812577  1.0330417   1.05723363  0.05938334]
[ 1.21041341  0.36010081  0.9402102   0.03010778]
[-0.08031812 -0.70317448 -0.47049667 -2.56384972]
[ 2.39689924  1.03861695  1.41119412  0.46410603]
[ 0.18525014 -0.50771664 -0.27324556 -2.23945786]
[ 0.32125453  0.00562737 -0.02615018 -1.36792753]
[-3.06510561 -1.28140089 -0.97503882 -1.16637299]
[-1.11339861 -0.11786267 -0.00894353 -0.02235124]
[ 1.85310391  0.6562346   0.78846611 -0.61993709]
[ 0.31367948  0.39450558  0.53302795  0.14688199]
[-0.90042099 -0.49894952  0.0092078  -0.58011281]
[ 2.16706964  1.13149484  1.85822212  1.68086935]
[-0.65906586  0.19402923  0.09495835  0.04301196]
```

```
[-0.44478604  0.29948716  0.1033373  -0.04905216]
[ 1.43630686  1.14138536  1.72610374  2.15728584]
[ 2.26829453  1.88899105  2.12329724  2.86729105]
[-2.0027185   0.05720017 -0.19478463  0.67034933]
[-0.36086478 -0.69333252  0.55621499 -0.22003772]
[-2.52589224 -1.90857413 -0.58109634 -1.54487456]
[ 0.36441545  0.04306776  0.00874119 -1.30386537]
[ 2.02055483  1.54280908  1.61708386  1.75642181]
[ 1.61454963  1.19262642  0.56012827 -0.30166662]
[ 2.57647143  1.88547427  2.25047124  2.80994528]
[ 3.65594277  2.76871597  2.90177775  3.91632861]
[ 3.77587727  2.03870216  1.66975939  0.60234373]
[ 3.45764505  2.23272626  1.77625287  1.32758689]
[ 4.81701301  4.10549619  3.23191843  4.75232011]
[ 4.48256098  2.32895504  1.94745282  0.74129971]
[ 3.194422    2.05844501  2.00261392  1.86925073]
[ 1.49682805  0.5207682   1.11521714  0.25437457]
[ 0.3836045  -0.40108784 -0.26188861 -2.30846953]
[ 1.11308037  0.53179367  0.70984946 -0.16158774]
[ 2.82530376  1.83950126  1.438163    0.8905234 ]
[-1.56991438 -0.86720069 -1.05025379 -2.39779398]
[ 1.51133166  1.09460242  1.81107171  2.2054142 ]
[ 1.32776657  1.6916207   1.0453111   1.45447636]
[ 2.70282459  0.98318616  1.24486497 -0.22990842]
[ 0.14581204  1.6708127   1.10463692  2.73427451]
[ 0.26458484  1.63714853  0.64342471  1.65941308]
[ 0.8373412   0.42166378  1.32506156  1.23444576]
[ 1.1101889   1.04189972  0.94725445  0.82621991]
[ 1.17229768  2.402052    1.2977327   2.82521976]
[ 1.70419154  0.43045863  0.79398969 -0.68575351]
[-1.51769806 -1.10183257 -0.77137263 -2.12687979]
[-2.44275045 -0.32047437 -0.31102055  0.50023505]
[ 1.2077581   2.05439831  1.01264625  1.87193265]
[ 1.43535033  1.84669518  1.46530186  2.34194858]
[ 5.11574563  2.89368682  2.88330725  2.54455574]
[-1.35317178 -1.36122378 -0.61222933 -2.23251056]
[ 3.72425173  3.64258732  2.81290385  4.54414339]
[ 0.66816841  1.47185816  1.40572441  2.61513872]
[ 0.65088007  1.27507978  0.45479584  0.53379151]
[ 2.28497869  1.1493746   0.95225893 -0.23108637]
[ 1.15010803  1.0811195   0.87578301  0.68257733]
[ 1.63139748  0.98979728  1.34397341  1.04634653]
[ 0.64509059  1.01188119  1.46733562  2.30146171]
[-1.17644764 -0.43222134 -0.68712125 -1.63001626]
[-1.64197926 -1.99640248 -1.237335   -3.82909328]
[ 1.89078924  2.82108915  2.16330758  4.25691509]
[ 1.81644845  0.05129638  0.80336689 -1.15841824]
[ 3.54998993  2.31221808  1.8488006   1.45982932]]
```

```
#####Covariance matrix #######
[[ 4.44791479  2.74483743  2.2486726   2.79426785]
 [ 2.74483743  2.17666469  1.64106293  2.71395314]
 [ 2.2486726   1.64106293  1.39917388  2.1907381 ]
 [ 2.79426785  2.71395314  2.1907381   4.30116149]]
##########Eigenvectors#######
[[ 0.5871257   0.70346768 -0.37796447  0.13251222]
 [ 0.44447908 -0.01493583  0.37796448 -0.8120087 ]
 [ 0.35827357  0.00392202  0.75592894  0.54790154]
 [ 0.57390052 -0.71055947 -0.37796447  0.15128216]]
##########
Eigenvalues##########
[  1.06293757e+01   1.57973675e+00   6.26920820e-18   1.15802423e-01]
########Eigenvalues in descending order:##########
10.6293756788
```

3. Problem 3: FastMap Algorithm

3.1 Abstract:

An idea for fast searching in traditional and multimedia databases is to map objects into points in k-dimensional space such that the dis-similarities are preserved. It is useful for visualization and data mining. This algorithm is much faster and allows indexing. It takes input as objects and their distance matrix. Computation of Euclidian distance between two feature vectors as the distance function between the corresponding objects is done. The Goal is to Project these points on K mutually orthogonal directions only by using distances. Basic assumption here is Distance is metric. Benefit of mapping objects is help with visualization, clustering, document retrieval and data-mining. Time complexity of this algorithm is O(Nk) linear in time.

$$x_i = \frac{d_{a,i}^2 + d_{a,b}^2 - d_{b,i}^2}{2d_{a,b}}$$

Applications of Fast algorithm are in Image and multimedia databases, medical databases, string databases(OCR), time series(financial data). This accelerate the search time for queries, by employing SAMs like R*-trees and z-ordering.

3.2 Approach and Procedure:

3.2.1 Data structure and Functions-

- Furthest()- this function returns the Maximum/furthest distance amongst all the distances.
- pickPivot()- this function finds the two most distant points and for fixed number of iterations it calls function furthest(). Finally returns the two distant points.
- fastmap()- this function gives call to pickPivot(0 and cosine() functions.
- cosine()- This function projects the ith point onto the line defined by x and y
- distance()- It recursively computes the distance based on previous projections.
- euclidean()- Calculates distance using Euclidean distance formula.
- distmatrix()- It recursively calls the function euclidean() to compute the distances
- distortion()-
- final()- this function prints distance matrix by giving a call to distmatrix() and do the mapping by calling function fastmap() and prints the reduced dimensions.

3.2.2 Pseudo Code-

```
Initialie variable DISTANCE_ITERATIONS = 5
Define function furthest()
    Generate teo random numbers and store it
    while DISTANCE_ITERATIONS are greater than zero
            call furthest() store the return value in o
            if o is equal to o2=-1
                    break
            update o2
            if o is equal to o1
                    break
            update o1
            decrease iterations
    store the pivots
    return the pivots
```

```
define fastmap()
        call function pickPivot and store the results
        if distance is equal to zero
                return
        for I upto length(dist)
                call function cosine() and store the value
        increase col by 1
define cosine()
        dix = distance(i, x,col,dist)
         diy = distance(i, y,col,dist)
         dxy = distance(x, y,col,dist)
         return (dix + dxy - diy) / 2 * math.sqrt(dxy)

define function distance()
        if k=0 then
                return absolute value od dist[x,y]
        rec = distance(x, y, k - 1,dist)
        resd = (res[x][k] - res[y][k])
        return abs(rec - resd)
define function euclidean()
        return value of math.sqrt(sum((x-y)**2))
define function distmatrix()
        foe x uopto length(p)
                foe y from x to length(p)
                        if x is equal to y
                                conyinue
                        store euclidean between x and y in dist[x,y]
                        dist[y,x] = dist[x,y]
        return dist
define function distortion()
        return the value scipy.sum(((d1 / d1.max()) - (d2 / d2.max())) ** 2) / d1.size
define function final()
        dist = distmatrix(dist,points)
        print distance matrix
         if dist.max() is greater than 1
                dist /= dist.max
        call function fastmap()
        take a product of res and 10
        print reduced dimensions
call final()
```

3.3 Code level optimization:

- numpy.zeros- Return a new array of given shape and type, filled with zeros.
- We have defined different functions for each computation. So that the stored results can be repeatedly used. For example, euclidean() function computes Euclidean distance between given points.
- List Comprehensions: Most of the functions use Python's in built method of "List comprehensions" to generate lists.

- Scipy - provides convenient and fast N-dimensional array manipulation
- Arrays and Matrices: Higher level objects such as matrices and arrays optimize operations on these datasets to make the program acceptably fast.

3.4 Result:

Distance matrix

```
[[ 0.          3.63544401   4.21190031 ...,  7.22326079   4.14493519
   6.82064798]
 [ 3.63544401   0.          5.5249729  ..., 10.01597209   4.71084622
   8.52176231]
 [ 4.21190031   5.5249729    0.        ...,  5.07877659   1.16557476
   3.03320093]
 ...,
 [ 7.22326079  10.01597209   5.07877659 ...,  0.          6.23318156
   3.30673557]
 [ 4.14493519   4.71084622   1.16557476 ...,  6.23318156   0.
   4.00812737]
 [ 6.82064798   8.52176231   3.03320093 ...,  3.30673557   4.00812737
   0.        ]]
```

Mapping

K = 2

Reduce Dimensions

```
[[ 2.50606735   2.50606735]
 [ 1.52529367   1.52529367]
 [ 4.87332668   4.87332668]
 [ 7.50717369   7.50717369]
 [ 4.61460068   4.61460068]
 [ 3.03742972   3.03742972]
 [ 5.13804541   5.13804541]
 [ 3.15357791   3.15357791]
 [ 0.          0.        ]
 [ 8.30370947  8.30370947]
 [ 5.54632086  5.54632086]
 [ 0.59615772  0.59615772]
 [ 7.4586958   7.4586958 ]
 [ 2.67536608  2.67536608]
 [ 4.5540982   4.5540982 ]
 [ 6.73619475  6.73619475]
 [ 1.89563495  1.89563495]
 [ 2.8441582   2.8441582 ]
 [ 5.16432273  5.16432273]
 [10.         10.        ]
 [ 6.21722999  8.79249097]
 [ 4.38907427  6.20708835]
 [ 4.394179    6.21430754]
 [ 3.69497428  5.22548274]
 [ 9.19139686 12.9985981 ]
 [ 3.82754919  5.41297197]
 [ 2.82331647  3.99277245]
 [ 6.06121189  8.57184805]
 [ 4.86575722  6.88121985]
 [ 8.04389509 11.37578554]
 [ 8.26207637 11.68434045]
 [ 9.72700708 13.75606533]
 [ 6.91653157  9.78145275]
 [ 6.36733488  9.00477134]
 [ 7.79702846 11.02666339]
 [ 1.87478652  2.65134852]
 [ 3.62406818   5.12520637]
 [ 0.99210164   1.40304359]
 [ 6.39353075   9.0418179 ]
 [ 5.20306226   7.35824122]
 [ 4.58864803   6.48932827]
 [ 2.63548905   3.72714436]
 [ 5.55657794   7.85818789]
 [ 2.92618853   4.1382555 ]
 [ 3.41802      4.83381024]
 [ 1.27840315   1.80793508]
 [ 3.15340025   4.4595814 ]
 [ 4.70076235   6.64788187]
 [ 4.14440183   5.86106928]
 [ 2.9493186    4.17096637]
 [ 6.03885147   8.54022565]
 [ 3.5033885    4.95453953]
 [ 3.60358761   5.09624247]
 [ 5.87370738   8.30667663]
 [ 6.82587086   9.65323915]
 [ 3.0344688    4.29138693]
 [ 3.39725022   4.80443734]
 [ 1.34461732   1.90157605]
 [ 3.47516194   4.91462114]
 [ 6.08073696   8.59946068]
 [ 4.81281873   6.80635353]
 [ 6.96358841   9.84800117]
 [ 8.26307055  11.68574643]
 [ 6.48864947   9.17633608]
 [ 6.73895489  9.5303214 ]
 [ 9.56633152 13.52883577]
 [ 6.91901392  9.78496332]
 [ 6.85218511  9.69045312]
 [ 4.87952459  6.90068985]
 [ 3.03323551  4.2896428 ]
 [ 4.47861998  6.33372511]
 [ 6.13409739  8.67492372]
 [ 1.78849994  2.52932087]
 [ 5.92828967  8.38386765]
 [ 5.58169875  7.89371407]
 [ 5.37946227  7.6077085 ]
 [ 5.52726743  7.81673656]
 [ 5.07513359  7.17732275]
 [ 4.95826264  7.01204228]
 [ 5.03403079  7.11919462]
 [ 6.28878525  8.89368539]
 [ 4.55126998  6.43646774]
 [ 1.85915023  2.62923547]
 [ 2.64265181  3.73727402]
 [ 5.78577369  8.18231962]
 [ 6.10281838  8.63068852]
 [ 8.19897779 11.59510558]
 [ 1.88532018  2.66624537]
 [ 8.79510725 12.43815996]
 [ 5.72793523  8.10052368]
 [ 4.68401575  6.6241986 ]
 [ 5.19377596  7.34510841]
```

```
 [  4.99528237   7.06439607]
 [  5.41653174   7.66013265]
 [  5.47585672   7.74403084]
 [  2.36359769   3.34263191]
 [  1.29044864   1.82496998]
 [  7.43235613  10.51093885]
 [  4.34390916   6.14321524]
 [  6.86542583   9.70917832]]
[Finished in 1.7s]
```

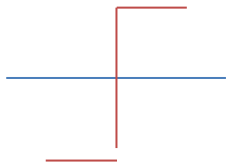
4. Problem 4:- Perceptron Learning Algorithm

4.1 Abstract-

Perceptron learning algorithm models dataset. It receives n inputs, sums up those inputs, checks the results and produces an output. It is used to classify linearly separable classes, often for binary classification. It consists of weights, the summation processor and activation functions. A perceptron takes a weighted sum of inputs and outputs. 1 If the sum is > some adjustable threshold value (theta) otherwise 0. The inputs and connection weights are real values. It has another input known as the bias. The bias allows us to shift the transfer function curve horizontally (left/right) along the input axis while leaving the shape unaltered. The weights determine the slope.

$$\text{Summation} = \sum_{i=1}^{n} W_i X_i$$

Transfer



4.2 Approach and Procedure-

4.2.1 Data structures and Functions:

- calOutput()- It takes theta, weight[], x, y, z as an input. Then calculates summation of product of weight vector and data points.
- perceptron()- This function calls calOutput() recursively and stores the value of each iteration. This value is further used to calculate error rate. Weight vector calculated by taking product of rate & error & each data point.

4.2.2 Pseudo Code:

```
unit_step = lambda x: 1 if x >= 0 else -1
calOutput()
Initialize variable sum
    sum= x*weights[0]+y*weights[1]+z*weights[2]+weights[3]
return unit_step(sum)
perceptron()
    while true
        for each data point in input data do
            call calOutput
            calculate error = expected output – actual output
            if error is not equal to 0
                increase error count by 1
                for each point in datapoint
                    update weight[] by taking product of rate, error &
    datapoint
                weight[3] = product of rate & error
        print error count
        if error count equals to zero then
            print number of iterations
call perceptron
```

print final weight vector and equation of hyperplane

4.3 Code Level Optimization-

- List Comprehensions: Most of the functions use Python's in built method of "List comprehensions" to generate lists.
- Lambda functions: Python supports the creation of anonymous functions (i.e. functions that are not bound to a name) at runtime, using a construct called "lambda". To calculate unit_step we used this function.

4.4 Result-

```
#######Input Vector##########
[[-1.98793769 -0.74538471 -0.17029177 -0.09803062]
 [-1.42961762 -1.46092472 -1.21433657 -3.45998017]
 [ 1.80783608  0.67599408  0.97147546 -0.18889123]
 [ 3.29772301  2.10358676  2.62706108  3.05998598]
 [-0.75943517  0.79793136  0.83288684  2.22314027]
 [-2.23485798  0.03902535 -0.17455733  0.92476861]
 [ 1.71141252  0.69028557  1.25537412  0.48962121]
 [ 0.01959636 -0.18458676 -0.15380813 -1.51179925]
 [-4.2019328  -2.13935218 -1.60571136 -2.14884214]
 [ 4.72157406  3.12058589  2.75853392  2.91607966]
 [ 2.69293979  1.270206    1.22523812  0.0277424 ]
 [-3.38353061 -1.69356337 -1.42517573 -2.16038433]
 [ 3.39635393  2.45595154  2.32148217  2.70256192]
 [-1.32594826 -0.74166322 -0.06301759 -0.54175005]
 [-0.99746067  0.70941161  0.84752415  2.40192054]
 [ 3.51386292  2.12831706  1.84642452  1.30730312]
 [-1.0583326  -1.05918767 -0.99412821 -2.98911155]
 [-1.79808487 -0.39720441 -0.07846954  0.24394152]
 [ 1.26134577  0.88039868  1.18461391  0.98828058]
 [ 4.93885291  4.38874317  3.43667692  5.32324417]
 [ 0.73533921  1.80395995  1.69169912  3.45201885]
 [-0.81651255  0.76618359  0.626007    1.83471014]
 [ 0.4112564   0.54632534  0.68343185  0.50193246]
 [-0.7896201   0.15673691  0.33551328  0.61738365]
 [ 3.9413336   3.71014544  3.20089484  5.17060159]
 [-0.99077755 -0.15425289  0.69863893  1.23380243]
 [-2.34752614 -0.66514723  0.0951759   0.87273074]
 [ 1.2237241   1.21173532  1.89255516  2.77312156]
 [ 0.87118277  1.29149389  0.61573555  0.65178229]
 [ 2.29789895  2.99394243  2.71835257  5.13274863]
 [ 4.16093716  3.46634803  2.42601516  3.15744131]
 [ 4.54768464  4.2404643   3.31824655  5.32927275]
 [ 2.37273966  2.83461187  1.55257596  2.56702408]
 [ 1.09247697  2.14832045  1.56637661  3.1885967 ]
 [ 3.00420666  2.60184273  2.57138028  3.74039662]
 [-1.40931793 -1.28703327 -0.65693607 -2.1915874 ]
 [-1.21031979  0.51861676  0.03963256  0.80820167]
```

```
[-2.50204452 -1.71683292 -1.27916546 -2.7731194 ]
[ 2.98457486  2.23021109  1.4308265   1.10728923]
[ 2.08812577  1.0330417   1.05723363  0.05938334]
[ 1.21041341  0.36010081  0.9402102   0.03010778]
[-0.08031812 -0.70317448 -0.47049667 -2.56384972]
[ 2.39689924  1.03861695  1.41119412  0.46410603]
[ 0.18525014 -0.50771664 -0.27324556 -2.23945786]
[ 0.32125453  0.00562737 -0.02615018 -1.36792753]
[-3.06510561 -1.28140089 -0.97503882 -1.16637299]
[-1.11339861 -0.11786267 -0.00894353 -0.02235124]
[ 1.85310391  0.6562346   0.78846611 -0.61993709]
[ 0.31367948  0.39450558  0.53302795  0.14688199]
[-0.90042099 -0.49894952  0.0092078  -0.58011281]
[ 2.16706964  1.13149484  1.85822212  1.68086935]
[-0.65906586  0.19402923  0.09495835  0.04301196]
[-0.44478604  0.29948716  0.1033373  -0.04905216]
[ 1.43630686  1.14138536  1.72610374  2.15728584]
[ 2.26829453  1.88899105  2.12329724  2.86729105]
[-2.0027185   0.05720017 -0.19478463  0.67034933]
[-0.36086478 -0.69333252  0.55621499 -0.22003772]
[-2.52589224 -1.90857413 -0.58109634 -1.54487456]
[ 0.36441545  0.04306776  0.00874119 -1.30386537]
[ 2.02055483  1.54280908  1.61708386  1.75642181]
[ 1.61454963  1.19262642  0.56012827 -0.30166662]
[ 2.57647143  1.88547427  2.25047124  2.80994528]
[ 3.65594277  2.76871597  2.90177775  3.91632861]
[ 3.77587727  2.03870216  1.66975939  0.60234373]
[ 3.45764505  2.23272626  1.77625287  1.32758689]
[ 4.81701301  4.10549619  3.23191843  4.75232011]
[ 4.48256098  2.32895504  1.94745282  0.74129971]
[ 3.194422    2.05844501  2.00261392  1.86925073]
[ 1.49682805  0.5207682   1.11521714  0.25437457]
[ 0.3836045  -0.40108784 -0.26188861 -2.30846953]
[ 1.11308037  0.53179367  0.70984946 -0.16158774]
[ 2.82530376  1.83950126  1.438163    0.8905234 ]
[-1.56991438 -0.86720069 -1.05025379 -2.39779398]
[ 1.51133166  1.09460242  1.81107171  2.2054142 ]
[ 1.32776657  1.6916207   1.0453111   1.45447636]
[ 2.70282459  0.98318616  1.24486497 -0.22990842]
[ 0.14581204  1.6708127   1.10463692  2.73427451]
[ 0.26458484  1.63714853  0.64342471  1.65941308]
[ 0.8373412   0.42166378  1.32506156  1.23444576]
[ 1.1101889   1.04189972  0.94725445  0.82621991]
[ 1.17229768  2.402052    1.2977327   2.82521976]
[ 1.70419154  0.43045863  0.79398969 -0.68575351]
[-1.51769806 -1.10183257 -0.77137263 -2.12687979]
[-2.44275045 -0.32047437 -0.31102055  0.50023505]
[ 1.2077581   2.05439831  1.01264625  1.87193265]
```

```
 [ 1.43535033  1.84669518  1.46530186  2.34194858]
 [ 5.11574563  2.89368682  2.88330725  2.54455574]
 [-1.35317178 -1.36122378 -0.61222933 -2.23251056]
 [ 3.72425173  3.64258732  2.81290385  4.54414339]
 [ 0.66816841  1.47185816  1.40572441  2.61513872]
 [ 0.65088007  1.27507978  0.45479584  0.53379151]
 [ 2.28497869  1.1493746   0.95225893 -0.23108637]
 [ 1.15010803  1.0811195   0.87578301  0.68257733]
 [ 1.63139748  0.98979728  1.34397341  1.04634653]
 [ 0.64509059  1.01188119  1.46733562  2.30146171]
 [-1.17644764 -0.43222134 -0.68712125 -1.63001626]
 [-1.64197926 -1.99640248 -1.237335   -3.82909328]
 [ 1.89078924  2.82108915  2.16330758  4.25691509]
 [ 1.81644845  0.05129638  0.80336689 -1.15841824]
 [ 3.54998993  2.31221808  1.8488006   1.45982932]]
#####Covariance matrix #######
[[ 4.44791479  2.74483743  2.2486726   2.79426785]
 [ 2.74483743  2.17666469  1.64106293  2.71395314]
 [ 2.2486726   1.64106293  1.39917388  2.1907381 ]
 [ 2.79426785  2.71395314  2.1907381   4.30116149]]
##########Eigenvectors#######
[[ 0.5871257   0.70346768 -0.37796447  0.13251222]
 [ 0.44447908 -0.01493583  0.37796448 -0.8120087 ]
 [ 0.35827357  0.00392202  0.75592894  0.54790154]
 [ 0.57390052 -0.71055947 -0.37796447  0.15128216]]
##########
Eigenvalues##########
[  1.06293757e+01   1.57973675e+00   6.26920820e-18   1.15802423e-01]
########Eigenvalues in descending order:##########
10.6293756788
1.57973674988
0.115802422687
6.26920820271e-18
######Matrix of top 2 eigen vectors#######
[[ 0.5871257   0.70346768]
 [ 0.44447908 -0.01493583]
 [ 0.35827357  0.00392202]
 [ 0.57390052 -0.71055947]]
########Final Dimensionally reduced vector
[[ -1.61574808e+00  -1.31832829e+00]
 [ -3.90946485e+00   1.46988934e+00]
 [  1.60154146e+00   1.39968632e+00]
 [  5.56851234e+00   1.24424139e-01]
 [  1.48304259e+00  -2.12256263e+00]
 [ -8.26610692e-01  -2.23052095e+00]
 [  2.04239300e+00   8.50632032e-01]
 [ -9.93267197e-01   1.09016240e+00]
 [ -5.22646560e+00  -1.40338844e+00]
[ 6.82104205e+00  1.21363724e+00]
[ 2.60056597e+00  1.86051732e+00]
[-4.48975975e+00 -8.25417697e-01]
[ 5.46843316e+00  4.41317501e-01]
[-1.44164026e+00 -5.36985928e-01]
[ 1.41179278e+00 -2.41566037e+00]
[ 4.42085868e+00  1.51842594e+00]
[-3.16378357e+00  1.39135958e+00]
[-1.12036629e+00 -1.43260474e+00]
[ 2.12347791e+00  1.76580463e-01]
[ 9.13671511e+00 -3.60228923e-01]
[ 3.82076551e+00 -1.95588606e+00]
[ 1.13837995e+00 -1.88704922e+00]
[ 1.01720426e+00 -7.28266613e-02]
[ 8.05823628e-02 -9.95185121e-01]
[ 8.07734725e+00 -9.44279217e-01]
[ 3.08110581e-01 -1.56862601e+00]
[-1.13897732e+00 -2.26122805e+00]
[ 3.52661928e+00 -1.12029305e+00]
[ 1.68019578e+00  1.32844243e-01]
[ 6.59950130e+00 -2.06468106e+00]
[ 6.66494665e+00  6.41277117e-01]
[ 8.80217265e+00 -6.37936649e-01]
[ 4.68248550e+00 -1.91125633e-01]
[ 3.98743345e+00 -1.52310892e+00]
[ 5.98818477e+00 -5.73187606e-01]
[-2.89262212e+00  5.82489951e-01]
[-2.06889895e-03 -1.43328674e+00]
[-4.28189680e+00  2.30984195e-01]
[ 3.89170396e+00  1.28505880e+00]
[ 2.09801673e+00  1.41545076e+00]
[ 1.22485343e+00  8.28402475e-01]
[-1.99966439e+00  1.77392368e+00]
[ 2.64086890e+00  1.34638835e+00]
[-1.50002700e+00  1.72809696e+00]
[-6.03305207e-01  1.19779943e+00]
[-3.38787087e+00 -1.31211071e+00]
[-7.22124045e-01 -7.65632754e-01]
[ 1.30639183e+00  1.73739186e+00]
[ 6.34784239e-01  1.12493272e-01]
[-1.08006106e+00 -2.13724085e-01]
[ 3.40567174e+00  3.20494000e-01]
[-2.42006921e-01 -4.96719651e-01]
[-1.19157579e-01 -2.82105926e-01]
[ 3.20709940e+00 -5.32762143e-01]
[ 4.57765213e+00 -4.61594930e-01]
[-8.35495580e-01 -1.88678909e+00]
[-4.47047416e-01 -8.49698445e-02]
```

```
 [ -3.42613330e+00  -6.52931249e-01]
 [ -5.12056887e-01   1.18221941e+00]
 [  3.45943583e+00   1.56651983e-01]
 [  1.50559360e+00   1.33451953e+00]
 [  4.76967989e+00  -2.03503563e-01]
 [  6.66434759e+00  -2.40919018e-01]
 [  4.06699108e+00   2.20430572e+00]
 [  4.42075965e+00   1.46263101e+00]
 [  8.53826926e+00  -3.68364024e-02]
 [  4.79015171e+00   2.59945233e+00]
 [  4.58071061e+00   8.96068556e-01]
 [  1.65583532e+00   8.68817701e-01]
 [ -1.37171072e+00   1.91512169e+00]
 [  1.05147427e+00   8.92675033e-01]
 [  3.50275591e+00   1.33290608e+00]
 [ -3.05956304e+00   6.08224428e-01]
 [  3.28841702e+00  -5.13150695e-01]
 [  2.74068797e+00  -1.20617097e-01]
 [  2.33796111e+00   2.05491104e+00]
 [  2.79321507e+00  -1.86091315e+00]
 [  2.06588293e+00  -1.01491344e+00]
 [  1.86222887e+00  -2.89205625e-01]
 [  1.92846733e+00   1.82057151e-01]
 [  3.84228638e+00  -1.21360002e+00]
 [  1.08281573e+00   1.68279711e+00]
 [ -2.87780091e+00   4.57054476e-01]
 [ -1.40099101e+00  -2.07027603e+00]
 [  3.05935040e+00  -5.07213196e-01]
 [  3.53257288e+00  -6.76206139e-01]
 [  6.78310364e+00   1.75879234e+00]
 [ -2.90010199e+00   6.52348728e-01]
 [  7.42133313e+00  -6.52366159e-01]
 [  3.05097238e+00  -1.40464683e+00]
 [  1.41817926e+00   6.13218343e-02]
 [  2.06099129e+00   1.75817719e+00]
 [  1.86129438e+00   3.11339475e-01]
 [  2.47978854e+00   3.94631618e-01]
 [  2.67502694e+00  -1.19088339e+00]
 [ -2.06448055e+00   3.34391266e-01]
 [ -4.49223042e+00   1.59068421e+00]
 [  5.58214779e+00  -1.72833296e+00]
 [  7.12292024e-01   2.10332250e+00]
 [  4.61219609e+00   1.43272379e+00]]
[Finished in 1.8s]
```

5 Problem 3:- Pocket Algorithm

5.1 Abstract-

If the learning set is not linearly separable the perceptron learning algorithm does not terminate. However, in many cases in which there is no perfect linear separation, we would like to compute the linear separation which correctly classifies the largest number of dataset. variant of the perceptron learning algorithm capable of computing a good approximation to this ideal linear separation. The main idea of the algorithm is to store the best weight vector found so far by perceptron learning (in a "pocket") while continuing to update the weight vector itself. If a better weight vector is found, it supersedes the one currently stored and the algorithm continues to run.

5.2 Approach and Procedure-

5.2.1 Data Structure and Functions:

- calOutput()- It takes theta, weight[], x, y, z as an input. Then calculates summation of product of weight vector and data points.
- perceptron()- This function calls calOutput() recursively and stores the value of each iteration. This value is further used to calculate error rate. Weight vector calculated by taking product of rate & error & each data point. Best weight vector is stored in *finalsuccess* variable.

5.2.2 Pseudo Code:

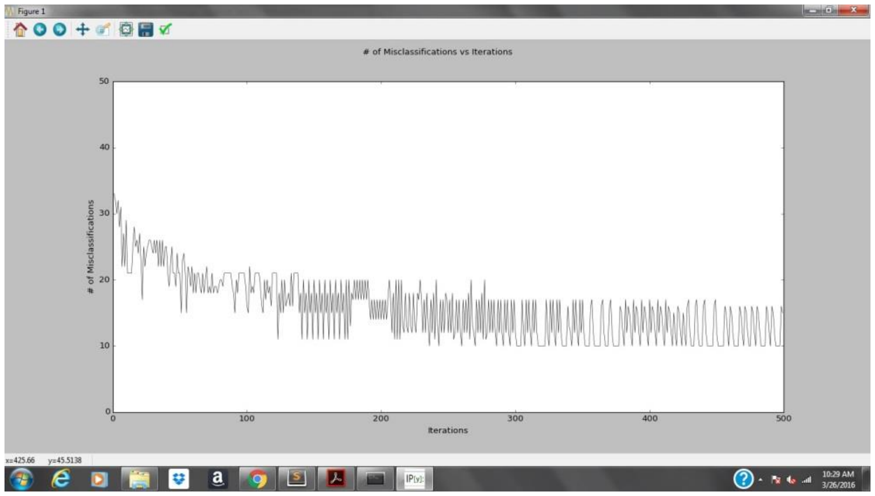
```
unit_step = lambda x: 1 if x >= 0 else -1
calOutput()
Initialize variable sum
    sum= x*weights[0]+y*weights[1]+z*weights[2]+weights[3]
return unit_step(sum)
perceptron()
    while true
            for each data point in input data do
                    call calOutput
                    calculate error = expected output – actual output
                    if error is not equal to 0
                            increase error count by 1
                            for each point in datapoint
                                    update weight[] by taking product of rate, error &
datapoint
                            weight[3] = product of rate & error
                    else
                            increase success by 1
            append list of weights into finalweights
            append success into finalsuccess

            if iterations equal to 500 then
                    print number of iterations
                    print maximum of finalsuccess
                    print finalweights[finalsuccess.index(max(finalsuccess))]
call perceptron
print final weight vector and equation of hyperplane
```

5.2 Code level optimization:

- List Comprehensions: Most of the functions use Python's in built method of "List comprehensions" to generate lists.
- Lambda functions: Python supports the creation of anonymous functions (i.e. functions that are not bound to a name) at runtime, using a construct called "lambda". To calculate unit_step we used this function.
- We are storing values of weights and success into finalweights and finalsuccess respectively. So that they can be used for future reference.
- Max function: This method returns the elements from the list with maximum value.

5.3 Result:

```
Number of Iterations 500
Final Maximum classifications
40
finalweights
[-1.7893265255001023, -1.6755371306059959, 0.6739182990200135, 0.5419999999999996]
Equation of hyperplane -1.7893265255x + -1.67553713061y +0.67391829902z +0.542 = 0
```



6. Problem 6:- Linear Regression

6.1 Abstract-

Regression analysis is concept of fitting straight lines to patterns of data. In a linear regression model, the "dependent" variable is predicted from k other "independent" variables using a linear equation. Consider Y denotes the dependent variable, and X1, ..., Xk, are the independent variables, then the assumption is that the value of Y on row t in the data sample is determined by the linear equation.

$$(Y_t) = b_0 + b_1X_{1t} + b_2X_{2t} + \cdots + b_kX_{kt}$$

Where the b's are constants. b_0 is the intercept of the model which is the expected value of Y when all the X's are zero and b_i is the multiplier of the variable X_i.

6.2 Approach and Procedure-

6.2.1 Data Structure & Functions:

- We started with dividing given dataset into two lists. Column 1 & 2 in one list and column 3 in another list.
- We formed list X such as 1st column being value 1 and other 2 columns from given dataset. Similarly formed list Y.
- For convenient arithmetic operations we converted all the lists into matrix.
- We did this programming in Python language, using numpy and math functions.

6.2.2 Pseudo Code:

```
dataPoints[[]]- list of dataset 1, column 1 & column 2
dataPoints1[[]]- list of dataset column 3
X- convert list dataPoints into an array
Y- convert list dataPoints1 into an array
a= Inverse of dot product of X transpose  and X
b= Dot product of X transpose and Y
c= Dot product of a and b
print c
```

6.3 Code Level optimization-

- List Comprehensions: Most of the functions use Python's in built method of "List comprehensions" to generate lists.
- Arrays and Matrices: Higher level objects such as matrices and arrays optimize operations on these datasets to make the program acceptably fast

6.4 Result-

```
Input Matrix
[[ 1.         16.83413438   2.25005563]
 [ 1.         16.16018143   2.52837564]
 [ 1.         14.80749844   2.2651013 ]
 [ 1.         14.33371275   1.29106839]
 [ 1.         13.23638125   1.00369145]
 [ 1.         15.96362121   2.39582104]
 [ 1.         15.0271781    2.26970918]
 [ 1.         13.79970165   1.241663  ]
 [ 1.         14.7087742   1.15562141]
 [ 1.         13.67476111  1.34204643]
 [ 1.         16.00677836  2.35912361]
 [ 1.         14.47340337  2.34436439]
 [ 1.         16.76727118  1.8413657 ]
 [ 1.         13.06869016  1.84365927]
 [ 1.         16.31622364  1.27203654]
 [ 1.         15.5063639   2.09826712]
 [ 1.         15.15498606  2.29251589]
 [ 1.         15.60203054  1.58317811]
 [ 1.         15.90651813  1.59243122]
 [ 1.         13.37795426  1.63132482]
 [ 1.         16.51029574  1.25807837]
 [ 1.         13.05744858  2.05075064]
 [ 1.         14.17721051  2.50732905]
 [ 1.         13.71965951  2.5170518 ]
 [ 1.         16.70517707  2.19439389]
 [ 1.         13.27272175  1.22102301]
 [ 1.         15.32437291  2.07927568]
 [ 1.         15.54860489  0.96366133]
 [ 1.         15.60507706  0.94700167]
 [ 1.         16.45848805  1.08180571]
 [ 1.         13.22381012  0.9914642 ]
 [ 1.         16.2674207   1.87008723]
 [ 1.         15.11568987  0.84672338]
 [ 1.         15.77740224  2.34465244]
 [ 1.         13.84961959  2.14580595]
 [ 1.         15.17311978  2.55171608]
 [ 1.         15.81008109  1.6852772 ]
 [ 1.         16.82573823  2.0097658 ]
 [ 1.         14.77816865  2.48024696]
 [ 1.         13.34159132  1.77705115]
 [ 1.         13.22936059  1.26441033]
 [ 1.         15.51780194  2.65191504]
 [ 1.         16.18471627  0.77285103]
 [ 1.         15.76476533  1.35248915]
 [ 1.         14.38123145  2.64602725]
 [ 1.         16.78726667  1.43006525]
 [ 1.         15.08076127  1.31829924]
 [ 1.         16.8152521   0.94182477]
 [ 1.         13.29438254  2.53153141]
 [ 1.         13.82812779  0.97095641]]
Final weight vector
[[-1.50767563]
 [ 2.20658575]
 [ 4.52402743]]
[Finished in 1.1s]
```

linearRegO.txt

7. Problem 7:- Logistic Regression

7.1 Abstract:

Logistic regression is a statistical method for analyzing a dataset in which there are one or more independent variables that determine an outcome. It is a special type of regression where binary response variable is related to a set of explanatory variables, which can be discrete or continuous. The expected values of the response variable are modeled based on combination of values taken by the predictors. The goal of logistic regression is to find the best fitting model to describe the relationship between the dependent variable and a set of independent (predictor) variables. Logistic regression generates the coefficients of a formula to predict a logit transformation of the probability of presence of the characteristic of interest.

$$logit(P) = b_0 + b_1 X_{1t} + b_2 X_{2t} + \cdots + b_k X_{kt}$$

where p is the probability of presence of the characteristic of interest.

The sigmoid function ϴ(s) = es/(1+es)

7.2. Approach and Procedure:

7.2.1. Data structure and Functions-

- logistic_func()- It takes input values such as theta and data points and computes sigmoid function.
- log_gradient()- Calculates the gradient by taking input values such as theta data points and class lables.
- cost_func()- It computes the cost function by giving a call to logistic_func()
- grad_desc()- It takes input values as theta values, data points, class labels, learning rate & convergence value. This function updates the values of cost function, theta values and computes the gradient descent.
- pred_values()- It takes input as theta and predicted value of data point. This function classifies data points into two classes 1 & -1.

7.2.2. Pseudo Code-

```
Form an array of all the data point and class labels
Initialize the theta values and form an array
Define safe_ln()
    Return np.log(x.clip(min=minval))
Define function logistic_func()
    Computes the sigmoid function 1.0 / (1 + np.exp(-x.dot(theta)))
    Returns the value
Define function log_gradient()
    first_calc = logistic_func(theta, x) - np.squeeze(y)
    final_calc = first_calc.T.dot(x)
    return final_calc
define cost_func()
    og_func_v = call logistic_func()
    step1= product of class lables y and log_func_v
    step2= (1-y) * safe_ln(1 - log_func_v)
    final = -step1-step2
```

```
            return mean value of final
        define grad_desc()
            Normaliza data points
            While(change_cost>converge_change)
                    Store cost in old_cost variable
                    Update the theta value theta_values = theta_values - (lr * log_gradient(theta_values,
        X, y))
                    Update the cost
                            cost = cost_func(theta_values, X, y)
                            cost_iter.append([i, cost])
                             change_cost = old_cost - cost
                              i+=1
            return theta values
         a, b = call grad_desc()
         print a – theta vector
         print b – cost at each iteration
        define pred_values()
            Xpred = normalize data points
            onesCol = np.ones(s)
            concatenate onesCol and Xpred
            pred_prob = call logistic_func()
            print pred_prob
            pred_value = np.where(pred_prob >= 0.5, 1, 0)
            if hard=true
                    return the pred-value
            return pred_prob
    s = pred_values(a,Xpred)
    count = 0
    for i from 0 to 50
            if s[i] is equal to y[i]
                    increase count by 1
    print count
```

7.3. Code level Optimization:

- List Comprehensions: Most of the functions use Python's in built method of "List comprehensions" to generate lists.
- Arrays and Matrices: Higher level objects such as matrices and arrays optimize operations on these datasets to make the program acceptably fast.
- We used functions like .append, .concatenate for fast computations.

7.4. Result:

```
this is final theta vector
[-0.01168987 -1.4068922  -1.26521     3.24464594]
this is cost at each iteration
[[  0.          0.78007056]
 [  1.          0.7487228 ]
 [  2.          0.71964103]
 [  3.          0.69284811]
 [  4.          0.66833169]
 [  5.          0.6460418 ]
 [  6.          0.62589137]
 [  7.          0.60775971]
 [  8.          0.59149883]
 [  9.          0.5769417 ]
 [ 10.          0.5639114 ]
 [ 11.          0.55222997]
 [ 12.          0.54172604]
 [ 13.          0.53224057]
 [ 14.          0.52363049]
 [ 15.          0.51577044]
 [ 16.          0.50855291]
 [ 17.          0.50188724]
 [ 18.          0.49569794]
 [ 19.          0.4899227 ]
 [ 20.          0.48451033]
 [ 21.          0.47941882]
 [ 22.          0.4746136 ]
 [ 23.          0.47006611]
 [ 24.          0.46575256]
 [ 25.          0.46165292]
 [ 26.          0.45775018]
 [ 27.          0.45402972]
 [ 28.          0.4504788 ]
 [ 29.          0.44708621]
 [ 30.          0.443842  ]
 [ 31.          0.44073722]
 [ 32.          0.43776378]
 [ 33.          0.43491428]
 [ 34.          0.43218196]
 [ 35.          0.42956056]
 [ 36.          0.42704428]
 [ 37.          0.42462775]
 [ 38.          0.42230593]
 [ 39.          0.42007413]
 [ 40.          0.41792793]
 [ 41.          0.41586321]
 [ 42.          0.41387608]
 [ 43.          0.41196289]
 [ 44.          0.41012017]
 [ 45.          0.40834468]
 [ 46.          0.40663335]
 [ 47.          0.40498328]
 [ 48.          0.40339172]
 [ 49.          0.40185608]
 [ 50.          0.4003739 ]
 [ 51.          0.39894286]
 [ 52.          0.39756075]
 [ 53.          0.39622547]
 [ 54.          0.39493505]
 [ 55.          0.3936876 ]
 [ 56.          0.39248134]
 [ 57.          0.39131456]
 [ 58.          0.39018565]
 [ 59.          0.38909308]
 [ 60.          0.38803538]
 [ 61.          0.38701117]
 [ 62.          0.38601913]
 [ 63.          0.38505799]
 [ 64.          0.38412656]
 [ 65.          0.38322369]
 [ 66.          0.38234829]
 [ 67.          0.38149932]
 [ 68.          0.38067578]
 [ 69.          0.37987673]
 [ 70.          0.37910127]
 [ 71.          0.37834852]
 [ 72.          0.37761766]
 [ 73.          0.37690791]
 [ 74.          0.3762185 ]
 [ 75.          0.37554871]
 [ 76.          0.37489786]
 [ 77.          0.37426527]
 [ 78.          0.37365031]
 [ 79.          0.37305238]
 [ 80.          0.37247088]
 [ 81.          0.37190528]
 [ 82.          0.37135502]
 [ 83.          0.37081959]
 [ 84.          0.37029851]
 [ 85.          0.3697913 ]
 [ 86.          0.36929749]
 [ 87.          0.36881667]
 [ 88.          0.3683484 ]
 [ 89.          0.36789229]
 [ 90.          0.36744795]
 [ 91.          0.367015  ]
 [ 92.          0.36659309]
 [ 93.          0.36618188]
 [ 94.          0.36578102]
 [ 95.          0.36539021]
 [ 96.          0.36500913]
 [ 97.          0.36463749]
 [ 98.          0.36427501]
 [ 99.          0.3639214 ]
 [ 100.          0.36357641]
 [ 101.          0.36323977]
 [ 102.          0.36291125]
 [ 103.          0.36259061]
 [ 104.          0.36227761]
 [ 105.          0.36197203]
 [ 106.          0.36167367]
 [ 107.          0.36138231]
 [ 108.          0.36109776]
 [ 109.          0.36081982]
 [ 110.          0.36054831]
 [ 111.          0.36028305]
 [ 112.          0.36002386]
 [ 113.          0.35977058]
 [ 114.          0.35952305]
 [ 115.          0.35928111]
 [ 116.          0.35904461]
 [ 117.          0.35881339]
 [ 118.          0.35858733]
 [ 119.          0.35836627]
 [ 120.          0.3581501 ]
 [ 121.          0.35793867]
 [ 122.          0.35773187]
 [ 123.          0.35752957]
 [ 124.          0.35733166]
 [ 125.          0.35713803]
 [ 126.          0.35694855]
 [ 127.          0.35676314]
 [ 128.          0.35658167]
 [ 129.          0.35640406]
 [ 130.          0.35623021]
 [ 131.          0.35606002]
 [ 132.          0.3558934 ]
 [ 133.          0.35573026]
 [ 134.          0.35557052]
 [ 135.          0.35541408]
 [ 136.          0.35526088]
 [ 137.          0.35511083]
 [ 138.          0.35496386]
 [ 139.          0.35481989]
 [ 140.          0.35467885]
 [ 141.          0.35454067]
 [ 142.          0.35440527]
 [ 143.          0.35427261]
 [ 144.          0.3541426 ]
 [ 145.          0.35401519]
 [ 146.          0.35389032]
 [ 147.          0.35376792]
```

```
 [ 148.          0.35364795]
 [ 149.          0.35353034]
 [ 150.          0.35341504]
 [ 151.          0.35330199]
 [ 152.          0.35319116]
 [ 153.          0.35308247]
 [ 154.          0.3529759 ]
 [ 155.          0.35287138]
 [ 156.          0.35276888]
 [ 157.          0.35266834]
 [ 158.          0.35256973]]
Equation of hyperplane -1.40689219662x + -1.26520999809y +3.24464594241z +-
0.0116898725501 = 0
44  Points are classified
[Finished in 1.1s]
```

8. Challenges:

Programs required a lot of vector/matrix manipulation. Most of them use numpy arrays and matrices. The matrix operations such as inverse, transpose and multiplication were achieved swiftly through the numpy library.