Structured Analysis of High Dimensional FMR Model

VLSI Signal Processing Architecture

Project Report

Submitted to:

Prof. Rajiv Kapoor

Submitted by:

Sumedha

M.Tech S.P.D.D (Second Semester)

ROLL No. : 2K19/SPD/17

CANDIDATE'S DECLARATION

I hereby declare that the work for project "Structured Analysis of High Dimensional FMR Model" submitted to the Department of Electronics & Communication Engineering of Delhi Technological University, is an authentic record of my own work. The matter represented in this report has not been submitted by me for award of any other degree for this or any other institute/university.

Date: 14/07/2020

SUMEDHA

M.Tech S.P.D.D. (Second Semester)

ROLL No. : 2K19/SPD/17

ACKNOWLEDGEMENT

It Is Indeed a Great Pleasure and a moment of immense satisfaction for me to express my gratitude towards Prof. Rajiv Kapoor for his priceless help and moral support during the completion of the project. I would like to extend thanks to him for his valuable help during the project. I would also like to thank friends and all the people who directly or indirectly have contributed to the completion of this project.

Date: 14/07/2020

SUMEDHA

M.Tech S.P.D.D. (Second Semester)

ROLL No. : 2K19/SPD/17

Contents

1:

Introduction

1.1 Introduction to Finite Mixture of Regression (FMR):

Finite mixture of regression models are widely used for the modelling and analysis of data from a heterogeneous population. Data of this kind can be subject to some upper and/or lower detection limits because of the restriction of experimental apparatus. If we could use the same approach as with clustering, but instead of gaussians around means, we might have each sub-group defined by a linear model. In this case, as long as we can produce a likelihood estimate for a model, we can use the E-M algorithm to identify and estimate the groups. That is, we can fit a model to a subgroup, obtain the maximum likelihood estimate, then place each observation in the model it is best described by, and repeat until things settle down.

To review: E-M stands for Expectation-Maximization. It is an iterative process whereby we apply two complementary processes. Suppose we assume there are two sub-groups. First, randomly assign members to either of the sub-groups. Next, we compute a maximum likelihood (ML) estimate for each sub-group. Now, because of random variation, there are likely to be members of one group that are better described by the other group (in terms of likelihood). We then apply the ‘maximization’ step to re-sort data into the group that better describes them. We repeat this process, re-estimating our models and re-sorting members until nobody is better described by a model that they are not in. Original research on the E-M algorithm proved that it would converge to a local maximum; we might not get the global maximum however, so we typically repeat the process many times and pick the best outcome we find. This is possible to do by hand, and in this case is not too complicated.

The finite mixture distribution is given by:

$H(y|x, w, \Theta) = X\ K\ k=1\ \pi k\ (w, \alpha)Fk\ (y|x,\vartheta k\ )$

with $X\ K\ k=1\ \pi k\ (w, \alpha) = 1$ and $\pi k\ (w, \alpha) > 0\ \forall k$. In the following it is assumed that the component specific density functions fk exist and determine the mixture density h.

1.2 Methods for analysis of FMR:

1.2.1 Structured Analysis Method:

Structured Analysis is a development method that allows the analyst to understand the system and its activities in a logical way. It is a systematic approach, which uses graphical tools that analyze and refine the objectives of an existing system and develop a new system specification which can be easily understandable by user. It has following attributes:

- It is graphic which specifies the presentation of application.

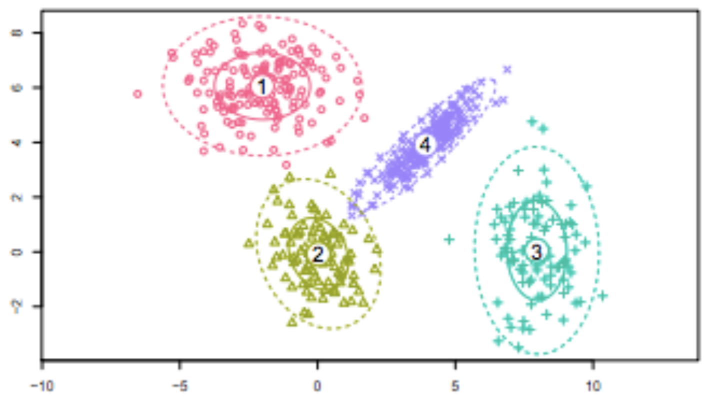
- It divides the processes so that it gives a clear picture of system flow.
- It is logical rather than physical i.e., the elements of system do not depend on vendor or hardware.
- It is an approach that works from high-level overviews to lower-level details.

Some structured analysis tools used are:

- Data Flow
- Diagrams
- Data Dictionary
- Decision Trees
- Decision Tables
- Structured English Pseudo-codes

The structured concepts reached their peak in the structured analysis approach, which is currently in existence in many different forms. In the structured analysis approach, the current application system was captured in the "data flow diagram." The technique itself advocated the separation of the logical design and physical implementation. To achieve this, the existing data store was viewed as the old physical model, and a new logical model was derived from it. If there were no previous system in place, then the manual process would be analyzed as if it were one and documented as so. This new logical design was then focused on what was done rather than how it was done. Changes could then be applied to the logical model that encompassed the client's desired changes. The changed model would become an even newer model and be translated into a new physical model for implementation. As a result of the impact this approach had on the evolution of the relationship between the business problem and the program solution, the concept of modularization was refined. This refinement gave uniformity to program module structure, interface and communication restrictions between modules, and quality measurements. Later, some of the significant findings during this time were useful in forming the conceptual roots of object oriented design, which we will cover in more detail elsewhere.

After analyzing FMR using Structured Analysis output should look like:



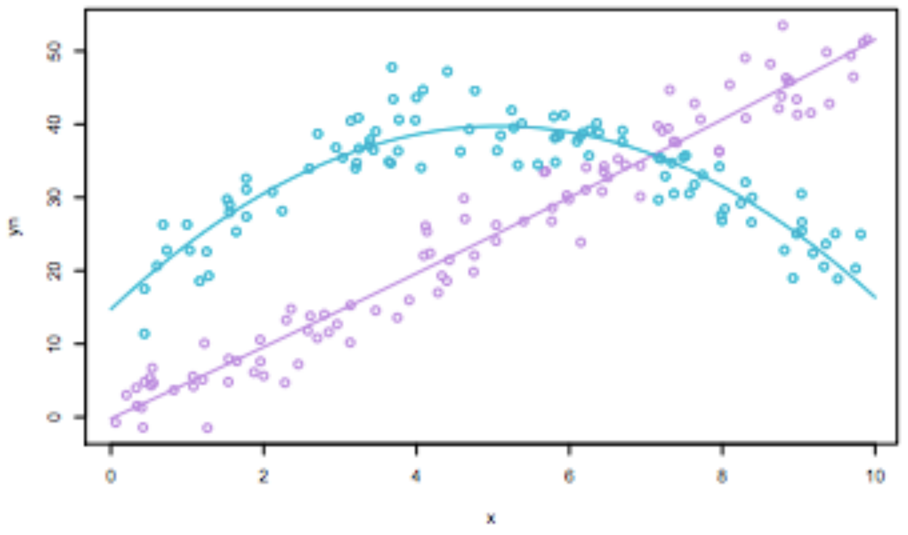
1.2.2 Component specific method:

Existing FMR analysis methods rely heavily on mixtures of linear models, where the linear predictor must be given as an input. A flexible FMR model is presented using a combination of the random forest learner and a penalized linear FMR. The performance of the new method is assessed by predictive log-likelihood in extensive simulation studies. The method is shown to achieve equal performance with the existing FMR methods when the true regression functions are in fact linear and superior performance in cases where at least one of the regression functions is nonlinear. The method can handle a large number of covariates, and its predictive ability is not greatly affected by surplus variables.

The component-label vectors zn = (znk)k=1,...,K are treated as missing data. It holds that: znk ∈ {0, 1} and PK k=1 znk = 1 for all k = 1, . . . , K.

The complete log-likelihood is given by log Lc(Θ) = X K k=1 X N n=1 znk [log πk (wn, α) + log fk (yn |xn,ϑk)]

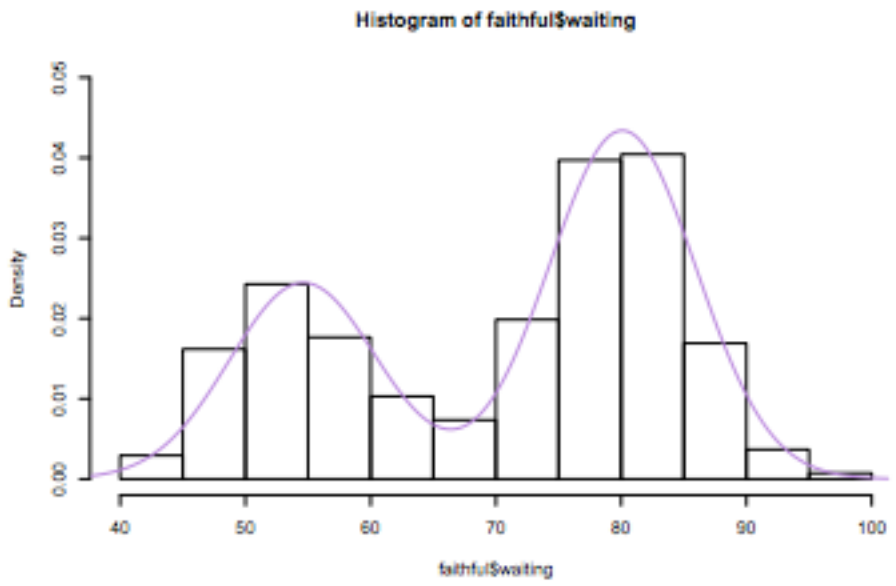
After analyzing FMR using Component Specific method output should look like:



1.2.3 Concomitant variable method:

The next parameter estimate is given by: Θ (i+1) = arg max Θ Q(Θ; Θ(i)). The estimates for the component sizes are given by: α (i+1) = arg max α X N n=1 zˆ (i) nk log πk (wn, α). ⇒ weighted ML estimation of the concomitant variable model. If the component sizes are assumed to be constant, they are given by π (i+1) k = 1 N X N n=1 zˆ (i) nk .

After analyzing FMR using Concomitant variable method output should look like:



1.3 Heterogeneous and Homogeneous Covariates:

In data analysis, a set of data is considered homogeneous if the variables are one type (i.e. binary or categorical); if the variables are mixed (i.e. binary + categorical), then the data set is heterogeneous.Heterogeneous data are not uncommon. Multiple factors can lead to heterogeneity. When only a small proportion of subjects behave differently and such subjects are not of interest, robust estimation can be conducted, focusing on the majority of homogeneous subjects. When all subjects are of interest, mixture models have been commonly adopted. In the context of regression analysis, finite mixture of regression (FMR) models have been popular and extensively used in biology, genetics, engineering, marketing, and other fields. For relevant discussions on the methodology and application of FMR.

Sometimes the definitions of homogeneous and heterogeneous covariates may get more complicated. For example, a covariate may have equal nonzero coefficients in some but not all Q datasets. That is, there is a possibility of "partially homogeneous" covariates. With Q > 2, as the newly added penalty has a pair-wise form, it is expected that the computational algorithm developed below can be applied with very minor revisions

2:

FMR implementation using Structured Analysis

2.1 Implementing Equations:

<u>Python, html and ipynb files of equations and dataset implemented are present in the folder sent</u>

<u>Data, model, and estimation</u>:

<u>Equation 1:</u> The conditional density of Y given X has the form:
fξ(Y|X) = ∑µqg(Y;h(βq0 +XTβq),σq).

$$f_{\xi}(Y|\mathbf{X}) = \sum_{q=1}^{Q} \mu_q g\left(Y; h(\beta_{q0} + \mathbf{X}^T \beta_q), \sigma_q\right).$$

Here, Y is the response variable and X is the length p vector of covariates.

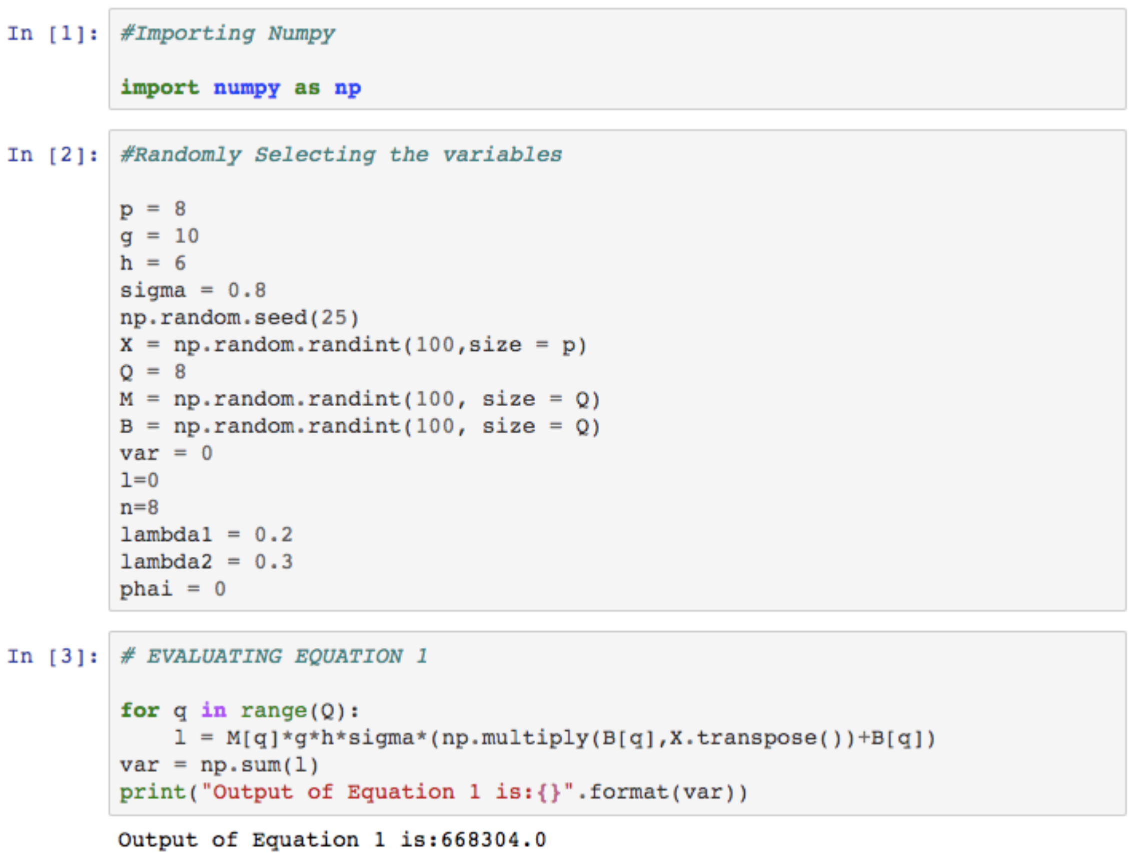
Q is the number of mixture components (subpopulations); µq's are the mixture weights and satisfy µq ≥ 0 and ∑µq = 1; g is the known density function; h is the known link function; βq0 is the unknown intercept; βq = (βq1 , . . . , βqp).

T is the length p regression parameter vector; σq is an unknown parameter usually corresponding to variance; and ξ=(µ1,…,µQ−1,β10,…,βQ0,β1T,...,βQT,σ1,...,σQ).

T is the vector of all unknown parameters. In the literature, multiple data distributions have been considered, including Binomial, Gaussian, Poisson, and others.

Q, the number of mixture components, needs to be determined and may not be trivial.

<u>Python Implementation</u>:



```
In [1]: #Importing Numpy

        import numpy as np
```

```
In [2]: #Randomly Selecting the variables

        p = 8
        g = 10
        h = 6
        sigma = 0.8
        np.random.seed(25)
        X = np.random.randint(100,size = p)
        Q = 8
        M = np.random.randint(100, size = Q)
        B = np.random.randint(100, size = Q)
        var = 0
        l=0
        n=8
        lambda1 = 0.2
        lambda2 = 0.3
        phai = 0
```

```
In [3]: # EVALUATING EQUATION 1

        for q in range(Q):
            l = M[q]*g*h*sigma*(np.multiply(B[q],X.transpose())+B[q])
        var = np.sum(l)
        print("Output of Equation 1 is:{}".format(var))

        Output of Equation 1 is:668304.0
```

Equation 2:

Denote the new vector of unknown parameters as θ = (φ1,φ2,ρ1,ρ2,μ1). Assume n independent observations Z = {(xi, yi) : i = 1, . . . , n}. Then the log-likelihood function is:

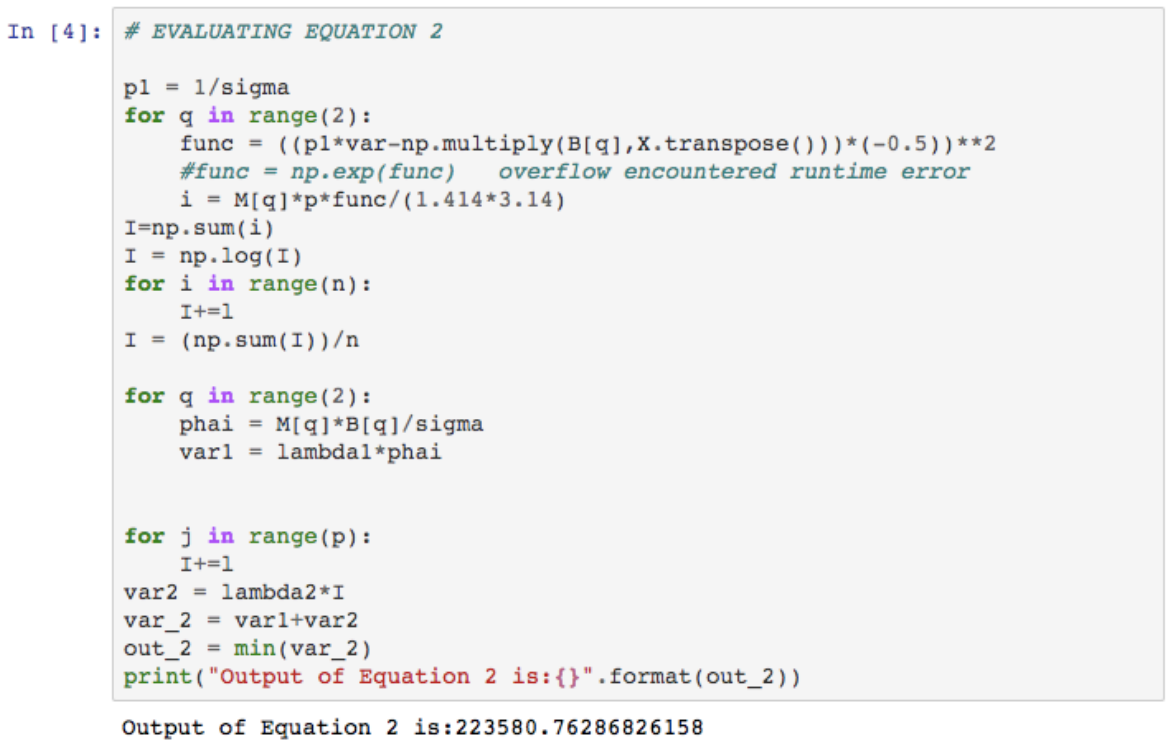
$$l(\theta;\mathbb{Z}) = n^{-1}\sum_{i=1}^{n} \log\left(\sum_{q=1}^{2} \mu_q \frac{\rho_q}{\sqrt{2\pi}} e^{-\frac{1}{2}(\rho_q y_i - \mathbf{x}_i^T \phi_q)^2}\right).$$

We propose the penalized estimate

$$\check{\theta}_\lambda^\gamma = \arg\min\left\{-l(\theta;\mathbb{Z}) + \lambda_1 \sum_{q=1}^{2} \mu_q^\gamma \left\|\phi_q\right\|_1 + \lambda_2 \sum_{j=1}^{p} I(\phi_{1j} \neq \phi_{2j})\right\}, \qquad (2)$$

where λ = (λ1, λ2) are data-dependent tunings, γ is a parameter designed to accommodate unbalance in data, μ2 = 1 − μ1, ‖·‖1 is the l1 norm, and I(·) is the indicator function. Important and unimportant covariates correspond to the nonzero and zero components of φˇq's, respectively. If φˇ1j = φˇ2j/= 0, then covariate j is a homogeneous one. Heterogeneous covariates can be identified accordingly. The mixture probability can be inferred from μˇ1. Specifically, for a certain subject, its posterior probability of belonging to a particular subpopulation can be calculated using the Bayesian rule. The formula is provided in (4) using the obtained estimates.

Python Implementation:



```
In [4]: # EVALUATING EQUATION 2

        p1 = 1/sigma
        for q in range(2):
            func = ((p1*var-np.multiply(B[q],X.transpose()))*(-0.5))**2
            #func = np.exp(func)   overflow encountered runtime error
            i = M[q]*p*func/(1.414*3.14)
        I=np.sum(i)
        I = np.log(I)
        for i in range(n):
            I+=1
        I = (np.sum(I))/n

        for q in range(2):
            phai = M[q]*B[q]/sigma
            var1 = lambda1*phai


        for j in range(p):
            I+=1
        var2 = lambda2*I
        var_2 = var1+var2
        out_2 = min(var_2)
        print("Output of Equation 2 is:{}".format(out_2))

        Output of Equation 2 is:223580.76286826158
```

Equation 3:

The indicator function is not continuous, making optimization challenging. To simplify computation, we further propose the estimate:

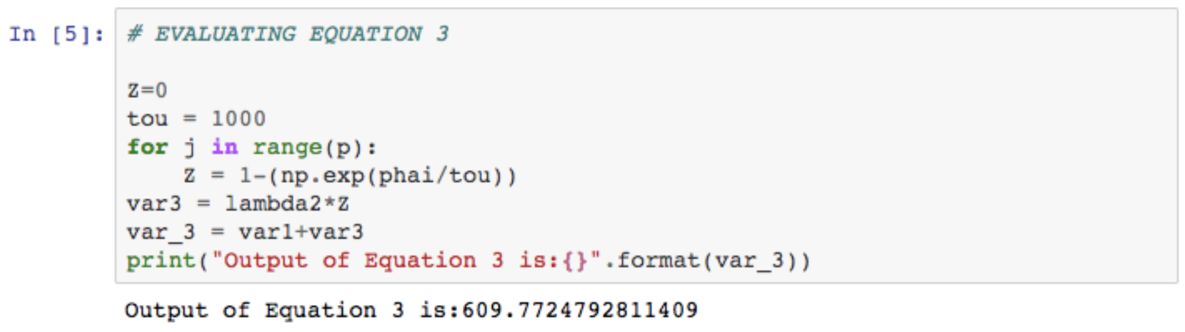
$$\hat{\theta}_{\lambda}^{\gamma} = \arg\min \left\{ -l(\theta;\mathbb{Z}) + \lambda_1 \sum_{q=1}^{2} \mu_q^{\gamma} \left\| \phi_q \right\|_1 + \lambda_2 \sum_{j=1}^{p} \left[1 - e^{-\frac{(\phi_{1j}-\phi_{2j})^2}{\tau}} \right] \right\}$$

here τ is a small positive number that controls the goodness of the approximation.

<u>We conclude that a covariate is homogeneous if its estimates in the two subpopulations are sufficiently close.</u>

To accommodate cases with Q > 2, we propose further extending the second penalty which can be approximated with λ2 j=1,...,p;1≤q1≤q2≤Q 1−e τ.

Python Implementation:



```
In [5]: # EVALUATING EQUATION 3

        Z=0
        tou = 1000
        for j in range(p):
            Z = 1-(np.exp(phai/tou))
        var3 = lambda2*Z
        var_3 = var1+var3
        print("Output of Equation 3 is:{}".format(var_3))

        Output of Equation 3 is:609.7724792811409
```

Computation:

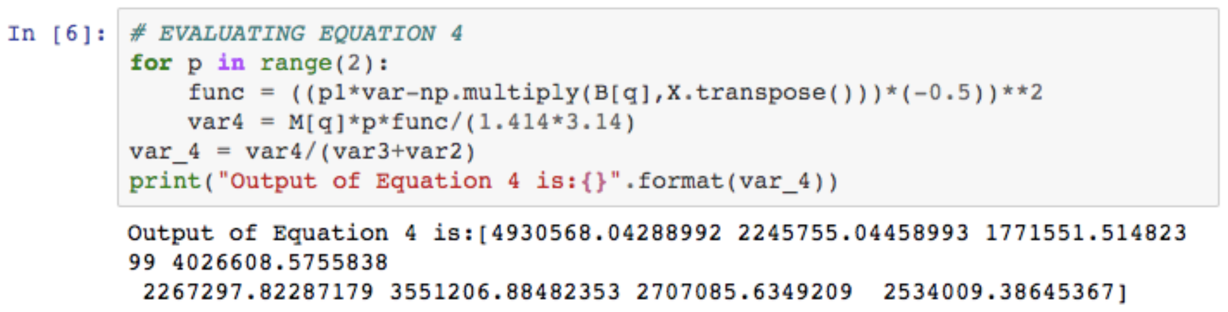
Equation 4:

A GEM (Generalized Expectation–Maximization) algorithm for optimization is used because in the EM algorithm, sometimes the complete-data maximum likelihood estimation can be overly complicated. One way to reduce computational complexity is to increase the value of the objective function rather than maximizing it in each M-step, leading to the GEM technique. For subject i(= 1, . . . , n), denote (Δi,1 , Δi,2) as the unobserved mixture membership indicators. Specifically, Δi,1 = 1 if subject i belongs to the first subpopulation, and 0 otherwise. Δi,2 is defined for the second subpopulation in the same way. Denote Δ = {(Δi,1 , Δi,2) : i = 1, . . . , n}. The complete-data log-likelihood function is:

$$l_c(\theta;\mathbb{Z},\Delta)=\sum_{i=1}^{n}\sum_{q=1}^{2}\left\{\Delta_{i,q}\log\left(\frac{\rho_q}{\sqrt{2\pi}}e^{-\frac{1}{2}(\rho_q y_i-\mathbf{x}_i^T\phi_q)^2}\right)+\Delta_{i,q}\log(\mu_q)\right\}$$

$$\hat{\delta}_{i,q}=E_{\theta^{(m)}}[\Delta_{i,q}|\mathbb{Z}]=\frac{\mu_q^{(m)}\rho_q^{(m)}e^{-\frac{1}{2}\left(\rho_q^{(m)}y_i-\mathbf{x}_i^T\phi_q^{(m)}\right)^2}}{\mu_1^{(m)}\rho_1^{(m)}e^{-\frac{1}{2}\left(\rho_1^{(m)}y_i-\mathbf{x}_i^T\phi_1^{(m)}\right)^2}+\mu_2^{(m)}\rho_2^{(m)}e^{-\frac{1}{2}\left(\rho_2^{(m)}y_i-\mathbf{x}_i^T\phi_2^{(m)}\right)^2}}.$$

Python Implementation:



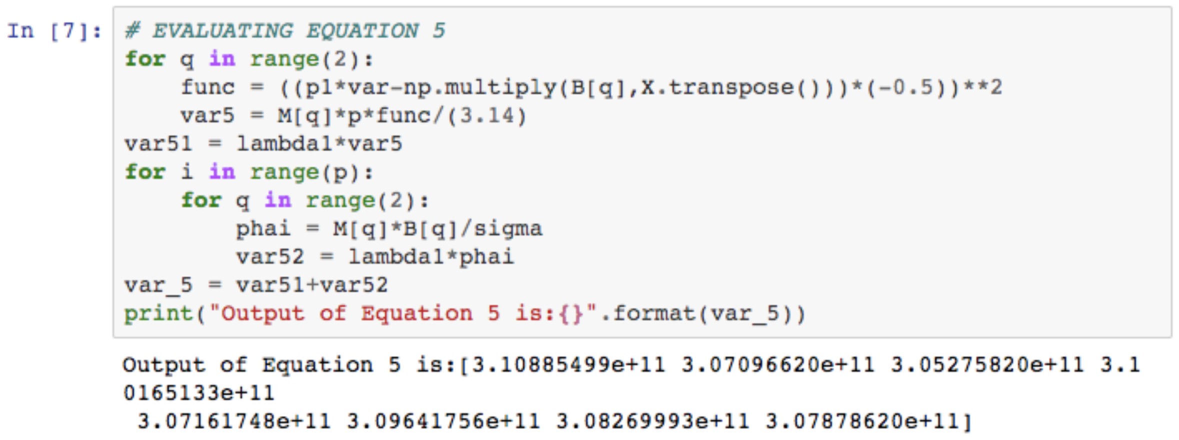
```
In [6]: # EVALUATING EQUATION 4
        for p in range(2):
            func = ((p1*var-np.multiply(B[q],X.transpose()))*(-0.5))**2
            var4 = M[q]*p*func/(1.414*3.14)
        var_4 = var4/(var3+var2)
        print("Output of Equation 4 is:{}".format(var_4))

        Output of Equation 4 is:[4930568.04288992 2245755.04458993 1771551.514823
        99 4026608.5755838
         2267297.82287179 3551206.88482353 2707085.6349209  2534009.38645367]
```

Equation 5:

Generalized M-Step: Optimizes (θ|θ(m)) with respect to θ. (a) Optimize with respect to

$$-n^{-1}\sum_{i=1}^{n}\sum_{q=1}^{2}\hat{\delta}_{i,q}\log(\mu_q)+\lambda_1\sum_{q=1}^{2}\mu_q^{\gamma}\left\|\phi_q^{(m)}\right\|_1 .$$

Python Implementation:



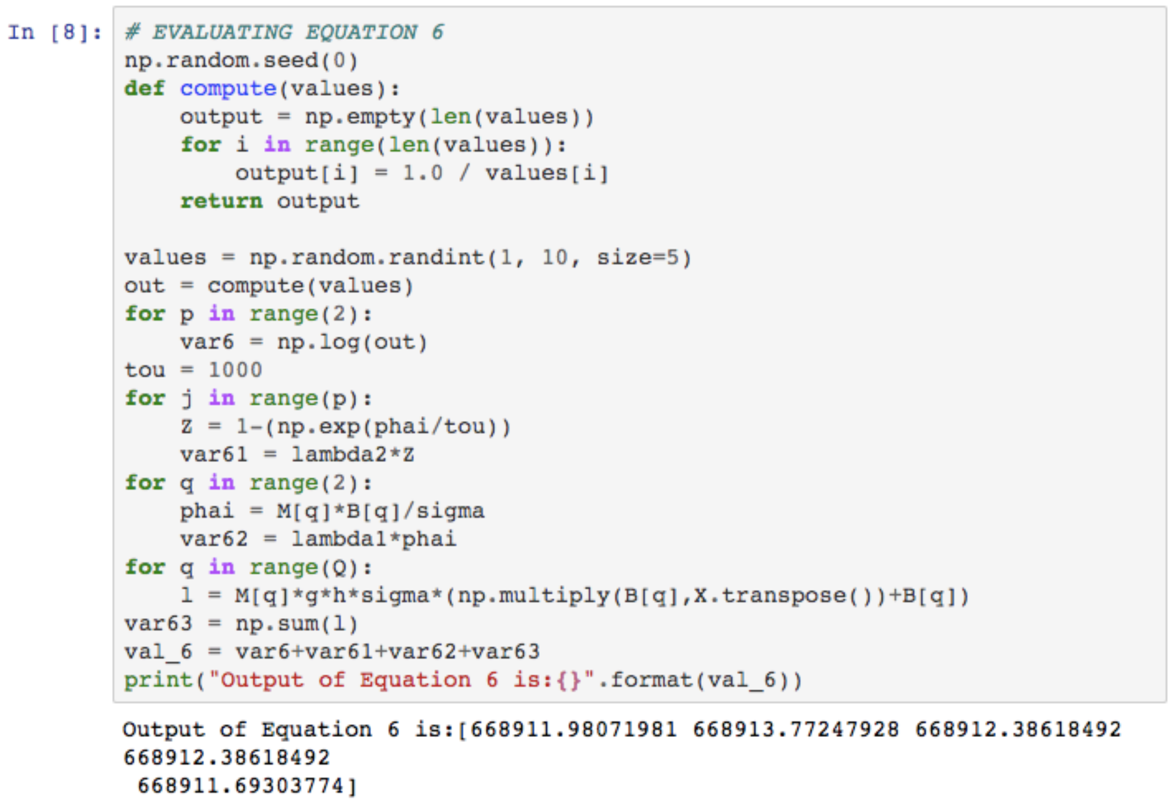
```
In [7]: # EVALUATING EQUATION 5
        for q in range(2):
            func = ((p1*var-np.multiply(B[q],X.transpose()))*(-0.5))**2
            var5 = M[q]*p*func/(3.14)
        var51 = lambda1*var5
        for i in range(p):
            for q in range(2):
                phai = M[q]*B[q]/sigma
                var52 = lambda1*phai
        var_5 = var51+var52
        print("Output of Equation 5 is:{}".format(var_5))

        Output of Equation 5 is:[3.10885499e+11 3.07096620e+11 3.05275820e+11 3.1
        0165133e+11
         3.07161748e+11 3.09641756e+11 3.08269993e+11 3.07878620e+11]
```

Equation 6:

$$\sum_{q=1}^{2}-\frac{n_q}{n}\log(\rho_q)+\frac{1}{2n}\left\|\rho_q\tilde{y}-\tilde{\mathbf{x}}^T\phi_q\right\|^2+\lambda_1\sum_{q=1}^{2}\mu_q^{\gamma}\left\|\phi_q\right\|_1+\lambda_2\sum_{j=1}^{p}\left[1-e^{-\frac{(\phi_{1j}-\phi_{2j})^2}{\tau}}\right],$$

where y˜ and x˜ are composed of (y˜i,x˜i)’s and (y˜i,x˜i) = √δˆi,q(yi,xi).

Python Implementation:



```
In [8]: # EVALUATING EQUATION 6
        np.random.seed(0)
        def compute(values):
            output = np.empty(len(values))
            for i in range(len(values)):
                output[i] = 1.0 / values[i]
            return output

        values = np.random.randint(1, 10, size=5)
        out = compute(values)
        for p in range(2):
            var6 = np.log(out)
        tou = 1000
        for j in range(p):
            Z = 1-(np.exp(phai/tou))
            var61 = lambda2*Z
        for q in range(2):
            phai = M[q]*B[q]/sigma
            var62 = lambda1*phai
        for q in range(Q):
            l = M[q]*g*h*sigma*(np.multiply(B[q],X.transpose())+B[q])
        var63 = np.sum(l)
        val_6 = var6+var61+var62+var63
        print("Output of Equation 6 is:{}".format(val_6))
```

```
Output of Equation 6 is:[668911.98071981 668913.77247928 668912.38618492
668912.38618492
 668911.69303774]
```

<u>Equation 7:</u>

As opposed to fully optimizing (6), we minimize it in a coordinate-wise manner, update one coordinate, and hold the q=1 q=1 j=1 other coordinates at their current estimates. The closed-form coordinate updates can be obtained as:

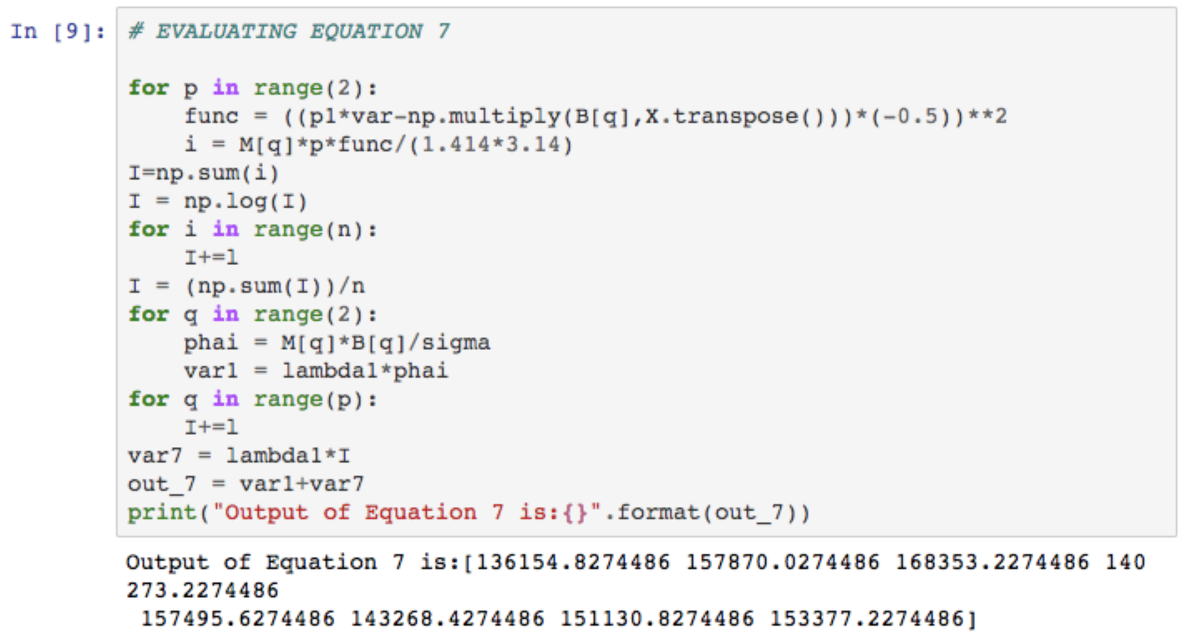
$$\rho_q^{(m+1)} = \frac{\left\langle \tilde{y}, \tilde{\mathbf{x}}^T \phi_q^{(m)} \right\rangle + \sqrt{\left\langle \tilde{y}, \tilde{\mathbf{x}}^T \phi_q^{(m)} \right\rangle^2 + 4 \left\| \tilde{y} \right\|^2 n_q}}{2 \left\| \tilde{y} \right\|^2}, \quad q = 1, 2,$$

$$\phi_{1.j}^{(m+1)} = \begin{cases} (-M_{1.j} - n\lambda_1(\mu_1^{(m+1)})^\gamma + L\phi_{2.j}^{(m)})/(\left\|\tilde{\mathbf{x}}_j\right\|^2 + L) & if \quad L\phi_{2.j}^{(m)} > M_{1.j} + n\lambda_1(\mu_1^{(m+1)})^\gamma, \\ (-M_{1.j} + n\lambda_1(\mu_1^{(m+1)})^\gamma + L\phi_{2.j}^{(m)})/(\left\|\tilde{\mathbf{x}}_j\right\|^2 + L) & if \quad L\phi_{2.j}^{(m)} < M_{1.j} - n\lambda_1(\mu_1^{(m+1)})^\gamma, \\ 0 & otherwise, \end{cases}$$

$$\phi_{2.j}^{(m+1)} = \begin{cases} (-M_{2.j} - n\lambda_1(\mu_1^{(m+1)})^\gamma + L\phi_{1.j}^{(m)})/(\left\|\tilde{\mathbf{x}}_j\right\|^2 + L), & if \quad L\phi_{1.j}^{(m)} > M_{2.j} + n\lambda_1(\mu_1^{(m+1)})^\gamma \\ (-M_{2.j} + n\lambda_1(\mu_1^{(m+1)})^\gamma + L\phi_{1.j}^{(m)})/(\left\|\tilde{\mathbf{x}}_j\right\|^2 + L), & if \quad L\phi_{1.j}^{(m)} < M_{2.j} - n\lambda_1(\mu_1^{(m+1)})^\gamma \\ 0 & otherwise, \end{cases}$$

$\phi^{(m)} \quad \phi^{(m)},2$

Python Implementation:



```
In [9]: # EVALUATING EQUATION 7

        for p in range(2):
            func = ((p1*var-np.multiply(B[q],X.transpose()))*(-0.5))**2
            i = M[q]*p*func/(1.414*3.14)
        I=np.sum(i)
        I = np.log(I)
        for i in range(n):
            I+=1
        I = (np.sum(I))/n
        for q in range(2):
            phai = M[q]*B[q]/sigma
            var1 = lambda1*phai
        for q in range(p):
            I+=1
        var7 = lambda1*I
        out_7 = var1+var7
        print("Output of Equation 7 is:{}".format(out_7))
```

```
Output of Equation 7 is:[136154.8274486 157870.0274486 168353.2274486 140
273.2274486
 157495.6274486 143268.4274486 151130.8274486 153377.2274486]
```

2.2 Implementing Dataset:

Dataset used:

Breast Cancer dataset from kaggle.

Dataset, python, html and ipynb files are present in the folder sent

Code and output:

```
In [1]:
## Importing Libraries

import numpy as np
import pandas as pd
import csv
from scipy.stats import norm
from multiprocessing import Pool
import matplotlib
import matplotlib.pyplot as plt
import matplotlib.pylab as pylab
import random
from sklearn.cluster import KMeans
from sklearn.datasets.samples_generator import make_blobs
%matplotlib inline
```

In [2]:

```
## Loading the data from the file

data = pd.read_csv('/Users/sumedha/Desktop/VLSI_Project/
dataset.csv')
data.head()
```

Out[2]:

	id	diagnosis	radius_mean	texture_mean	perimeter_mean	area_mean	smoothness_mear
0	842302	M	17.99	10.38	122.80	1001.0	0.1184
1	842517	M	20.57	17.77	132.90	1326.0	0.0847
2	84300903	M	19.69	21.25	130.00	1203.0	0.1096
3	84348301	M	11.42	20.38	77.58	386.1	0.1425
4	84358402	M	20.29	14.34	135.10	1297.0	0.1003

In [3]:

```
## Loading the data for each subset

def load_data(filename):
    for line in reader:
      subset_id = int(float(line[0]))
      x = np.array([float(a) for a in line[1:-1]])
      y = float(line[-1])
      if subset_id not in data:
        data[subset_id] = ([], [])

      data[subset_id][0].append(x)
      data[subset_id][1].append(y)

      ## Converting lists to numpy arrays
      for subset_id in data.iterkeys():
        x, y = data[subset_id]
```

```
                data[subset_id] = (np.array(x), np.array(y))
            return data
```

In [4]:

```
## Creating pipelines for the model

class MixtureModel(object):
    def __init__(self, assignments, component_weights,
coefficients, variances):
        self.assignments = assignments
        self.component_weights = component_weights
        self.coefficients = coefficients
        self.variances = variances
```

In [5]:

```
## Creating Pipelines for the Model result

class MixtureResults(object):
    def __init__(self, num_components):
        self.num_components = num_components
        self.iterations = []
        self.log_likelihoods = []
        self.best = None

    def add_iteration(self, assignments, component_weights,
coefficients, variances, data_log_likelihood):
        self.iterations.append(MixtureModel(assignments,
component_weights, coefficients, variances))
        self.log_likelihoods.append(data_log_likelihood)

   def finish(self):
        self.log_likelihoods = np.array(self.log_likelihoods)
        self.best =
self.iterations[np.argmax(self.log_likelihoods)]
```

In [6]:

```
## Assigning weights to the dataset
```

```
def weights(x, y, weights, coefficients):
    result = 0
    for i in xrange(len(y)):
        result += weights[i] * (y[i] -
x[i].T.dot(coefficients)) ** 2
    return result / weights.sum()
```

In [7]:

```
## Assigning each set of points to a component

from sklearn.cluster import KMeans
def calculate_assignments(assignment_weights, stochastic):

    if stochastic:
        return np.array([np.random.choice(len(row),p=row) for
row in assignment_weights])
    return np.argmax(assignment_weights, axis=1)
```

In [8]:

```
## Determining probability for each component to generate each
set of points

defcalculate_assignment_weights(data, keys, component_weights,
coefficients, variances):

    num_components = len(component_weights)

    # Initializing the new assignment weights
    assignment_weights = np.ones((len(data), num_components),
dtype=float)

    # Calculating the new weights for every set of points
    for i,key in enumerate(keys):
        x, y = data[key]
        for xi, yi in zip(x, y):
            mu = np.array([xi.dot(b) for b in coefficients])
            sigma = np.array([np.sqrt(v) for v in variances])
            temp_weights = norm.pdf(yi, loc=mu, scale=sigma)
            assignment_weights[i] *= temp_weights / 
temp_weights.sum()
            assignment_weights[i] /= 
assignment_weights[i].sum()
        assignment_weights[i] *= component_weights
        assignment_weights[i] /= assignment_weights[i].sum()
    return assignment_weights
```

In [9]:

```
## Calculating the parameter values that maximize the 
likelihood of the data

def maximum_likelihood_parameters(data, keys, num_components, 
num_features, assignments, assignment_weights):
    # Calculating the weight of each component in the mixture
    component_weights = np.array([(assignments == i).sum() for 
i in xrange(num_components)]) / float(len(assignments))
    
    # Calculating the regression coefficients and variance for 
each component
    coefficients = np.zeros((num_components, num_features))
    variances = np.zeros(num_components)
    for i in xrange(num_components):
        points = np.where(assignments == i)[0]
        subset_weights = assignment_weights[points][:,i]
        if len(points) == 0:
            points = np.random.choice(len(assignments), 
size=np.random.randint(1, len(assignments)), replace=False)
            subset_weights = np.ones(len(points)) / 
float(len(points))
        component_x = []
        component_y = []
        weights = []
        for key, subset_weight in zip(keys[points], 
subset_weights):
            x, y = data[key]
            component_x.extend(x)
            component_y.extend(y)
            weights.extend([subset_weight / float(len(y))] *
len(y))
        component_x = np.array(component_x)
        component_y = np.array(component_y)
        weights = np.array(weights)
        coefficients[i] =
weighted_linear_regression(component_x, component_y, weights)
        variances[i] =
weighted_regression_variance(component_x, component_y,
weights, coefficients[i])
    return (component_weights, coefficients, variances)
```

```
In [10]:
```

```
## Fitting the Mixture Model

def fit_with_restarts_worker(worker_params):
    data, keys, num_components, max_iterations, num_restarts,
stochastic, verbose = worker_params
    return fit_with_restarts(data, keys, num_components,
max_iterations, num_restarts, stochastic=stochastic,
verbose=verbose)
```

```
In [11]:
```

```
## Saving the result

def save_results(results, keys, filename):
    with open(filename, 'wb') as f:
        writer = csv.writer(f)
        writer.writerow([results.num_components])
        best = results.best
        print('component_weights: {0} coefficients: {1}
variances: {2}'.format(best.component_weights.shape,
best.coefficients.shape, best.variances.shape))
        rows = np.zeros((results.num_components,
best.coefficients.shape[1] + 2))
        rows[:,0] = best.component_weights
        rows[:,1:1+best.coefficients.shape[1]] =
best.coefficients
        rows[:,1+best.coefficients.shape[1]] = best.variances
        writer.writerows(rows)
        for key, assignment in zip(keys, best.assignments):
            writer.writerow([key, assignment])
```

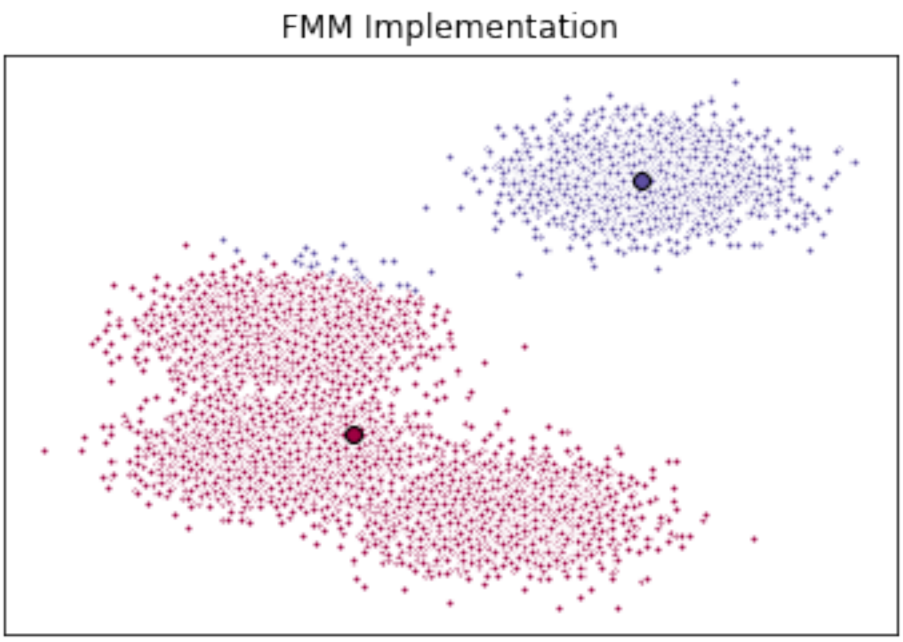
In [12]:

```
## Chooseing parameter values for the algorithm

# Number of mixture components
num_components = 3
# Maximum iterations per run
max_iterations = 20
# Number of random restarts to try
num_restarts = 5
# Using stochastic process
stoachastic = False
# Number of worker processes to use
num_workers = 4
```

In [13]:

```
X, y = make_blobs(n_samples=8000, centers=[[-1,3], [4, 8], [2,
-3], [-1, -1]], cluster_std=0.9)
fmm = KMeans(init = "k-means++", n_clusters = 2, n_init = 100)
fmm.fit(X)
fmm_labels = fmm.labels_
fmm_labels
fmm_centers = fmm.cluster_centers_
fmm_centers
fig = plt.figure(figsize=(6, 4))
colors = plt.cm.Spectral(np.linspace(0, 1,
len(set(fmm_labels))))

# Create a plot
ax = fig.add_subplot(1, 1, 1)
for k, col in zip(range(len([[4,4], [-2, -1], [2, -3], [1,
1]])), colors):
    my_members = (fmm_labels == k)
    cluster_center = fmm_centers[k]
    ax.plot(X[my_members, 0], X[my_members, 1], 'w',
markerfacecolor=col, marker='.')
    ax.plot(cluster_center[0], cluster_center[1], 'o',
markerfacecolor=col,  markeredgecolor='k', markersize=6)

# Title of the plot
ax.set_title('FMM Implementation')
ax.set_xticks(())
ax.set_yticks(())

# Show the plot
plt.show()
```

Output:



- **Points with Red colour represent Heterogeneous Covariates**
- **Points with Blue colour represent Homogeneous Covariates**

3:

Applications

Structured Analysis of High Dimensional FMR Model has become a fundamentally important problem in recent statistical literature.

• Sometimes, in applications, many variables are introduced to reduce possible modelling biases, but the number of variables a model can accommodate is often limited by the amount of data available. In other words, the number of variables considered depends on the sample size, which reflects the estimability of the parametric model.

• The problem of feature selection in finite mixture of regression models is used when the number of parameters in the model can increase with the sample size. A penalized likelihood approach for feature selection in these models is used. Under certain regularity conditions, our approach leads to consistent variable selection. We carry out extensive simulation studies to evaluate the performance of the proposed approach under controlled settings.

• It is also applied for tele-monitoring of Parkinson's disease (PD), where the problem concerns whether dysphoric features extracted from the patients' speech signals recorded at home can be used as surrogates to study PD severity and progression.

• Another is on breast cancer prognosis, in which one is interested in assessing whether cell nuclear features may offer prognostic values on long–term survival of breast cancer patients.

4:

Advantages and Disadvantages

4.1 Advantages:

• Most people understand structured methods better than object oriented methods. As one of main reasons of modelling a system is for communication with users and customers, there is a benefit in providing structured models for information exchange with customers or user groups.

• Specifications are typically in form of a simple English language statement of Work. Thus, the system to be built, understood in terms of requirements (functions the system has to perform), which is why this naturally leads to a structured analysis, at least at the upper most level.

• Specially structured methods (functional decomposition) provide a natural vehicle for modelling, discussing and deriving the requirements of the system.

4.2 Disadvantages:

• The difficulty with structured methods is that they do not readily support use of reusable modules.

• It procedure works well for new development, but does not provide mechanisms for designing in use of existing components.

• This does not lead to a set of requirements which map well to existing components. When requirements do not map cleanly, we have two options: either to not use existing components, or to force fit requirements to the existing components and deal with requirements which are only partly covered by existing components, which does not lead to a good successful system.

5:

Conclusion and Future Scope

5.1 Conclusion:

Structured Analysis of High Dimensional FMR Models can be used to fit finite mixtures of regressions to datasets used in the literature to illustrate these models. The results can be reproduced and additional insights can be gained using visualization methods available in python. A suitable methods have been implemented for objects of class which are returned.

5.2 Future Scope:

In the future it would be desirable to have more diagnostic tools available to analyze the model fit and compare different models. The use of resampling methods would be convenient as they can be applied to all kinds of mixtures models and would therefore suit well the purpose of the package which is flexible modelling of various finite mixture models. Furthermore, an additional visualization method for the fitted coefficients of the mixture would facilitate the comparison of the components.

6:

Refrences

1. Structured analysis of the high-dimensional FMR model by Mengque Liu a, Qingzhao Zhang b,c, Kuangnan Fang b, Shuangge Ma
2. A General Maximum Likelihood Analysis of Overdispersion in Generalized Linear Models by Aitkin M
3. Statistics and Computing, 6, 251–262 by Aitkin M (1999a)
4. Meta-Analysis by Random Effect Modelling in Generalized Linear Models
5. Statistics in Medicine by Follmann DA, Lambert D (1989)
6. MCLUST: Software for Model-Based Clustering, Discriminant Analysis and Density Estimation by Fraley C, Raftery AE (2002).
7. Technical Report 415, Department of Statistics, University of Washington, Seattle, WA, USA.
8. http://www.expertsmind.com/questions/advantages-and-disadvantages-of-structured-analysis-301100493.aspx
9. https://cran.r-project.org/web/packages/flexmix/vignettes/regression-examples.pdf
10. https://agris.fao.org/agris-search/search.do?recordID=US201900256573