FIT 5202: ASSIGNMENT2

Latent Variable models and Neural networks

SUBMITTED BY:

ROHAN SINGH
3004246
rsin0021@student.monash.edu

1.1 Document Clustering using Hard Expectation Maximization

(*The derivation has been taken from Alexandria and Lecture Content)

Generative Story:

For each document d_n :

- Toss the K-face dice (with the probability parameter φ) to choose the face k (i.e., the cluster) that the $n\ th$ document belongs to.

- For each word placeholder in the document dn, generate the word by tossing the dice (with the probability parameter μk) corresponding to the face k

Parameters:

- The clusters proportion $\varphi = \varphi_1, \varphi_2, \dots, \varphi_K, \varphi_k \geq 0, \sum_{k=1}^{k} \varphi k = 1$
- The word proportion $\mu_k = \mu_{k,1}, \mu_{k,2}, \dots, \mu_k, \mathcal{A}, \mu_{k,w} \geq 0, \sum_{w \in \mathcal{A}} \mu k, w = 1$

Objective:

The probability of generating a document and its cluster is given as:

$$p(k,d) = p(k)p(d|k) = \varphi_k \prod_{w \in d} \mu_{k,w} = \varphi_k \prod_{w \in \mathcal{A}} \mu_{k,w}^{c(w,d)}$$

In practice, the document cluster labels are not given to us. So, we use latent variables $\boldsymbol{z}_n$ to denote the cluster assignments for n^{th} document. The probability of generating n documents and their respective clusters is given as:

$$p(d_1, \dots, d_N) = \prod_{n=1}^{N} p(d_n) = \prod_{n=1}^{N} \sum_{k=1}^{K} \left(\varphi_k \prod_{w \in \mathcal{A}} \mu_{k,w}^{c(w,d_n)} \right)$$

where,

$c\ (w, d)$ is the number of occurrences of the word w in the document d

To summarise, the log-likelihood is:

$$\begin{aligned}\ln p(d_1,\ldots,d_N) = \sum_{n=1}^{N} \ln p(d_n) &= \sum_{n=1}^{N} \ln \sum_{k=1}^{K} p(z_{n,k}=1, d_n) \\ &= \sum_{n=1}^{N} \ln \sum_{k=1}^{K} \left(\varphi_k \prod_{w\in\mathcal{A}} \mu_{k,w}^{c(w,d_n)} \right)\end{aligned}$$

To maximise the above incomplete data log-likelihood objective, we resort to the EM Algorithm.

Q Function:

The Q function for EM Algorithm is given as:

$$Q(\boldsymbol{\theta}, \boldsymbol{\theta}^{\text{old}}) = \sum_{n=1}^{N}\sum_{k=1}^{K} \gamma(z_{n,k}) \left(\ln \varphi_k + \sum_{w\in\mathcal{A}} c(w,d_n) \ln \mu_{k,w} \right)$$

where $\boldsymbol{\theta} := (\boldsymbol{\varphi}, \boldsymbol{\mu}_1, \ldots, \boldsymbol{\mu}_K)$ is the collection of model parameters, and $\gamma(z_n, k) := p(z_{n,k} = 1 | d_n, \boldsymbol{\theta}^{\text{old}})$ are the responsibility factors.

To maximise the Q function, we form the Lagrangian to enforce the constraints, and set the derivatives to zero which leads to the following solution for the model parameters:

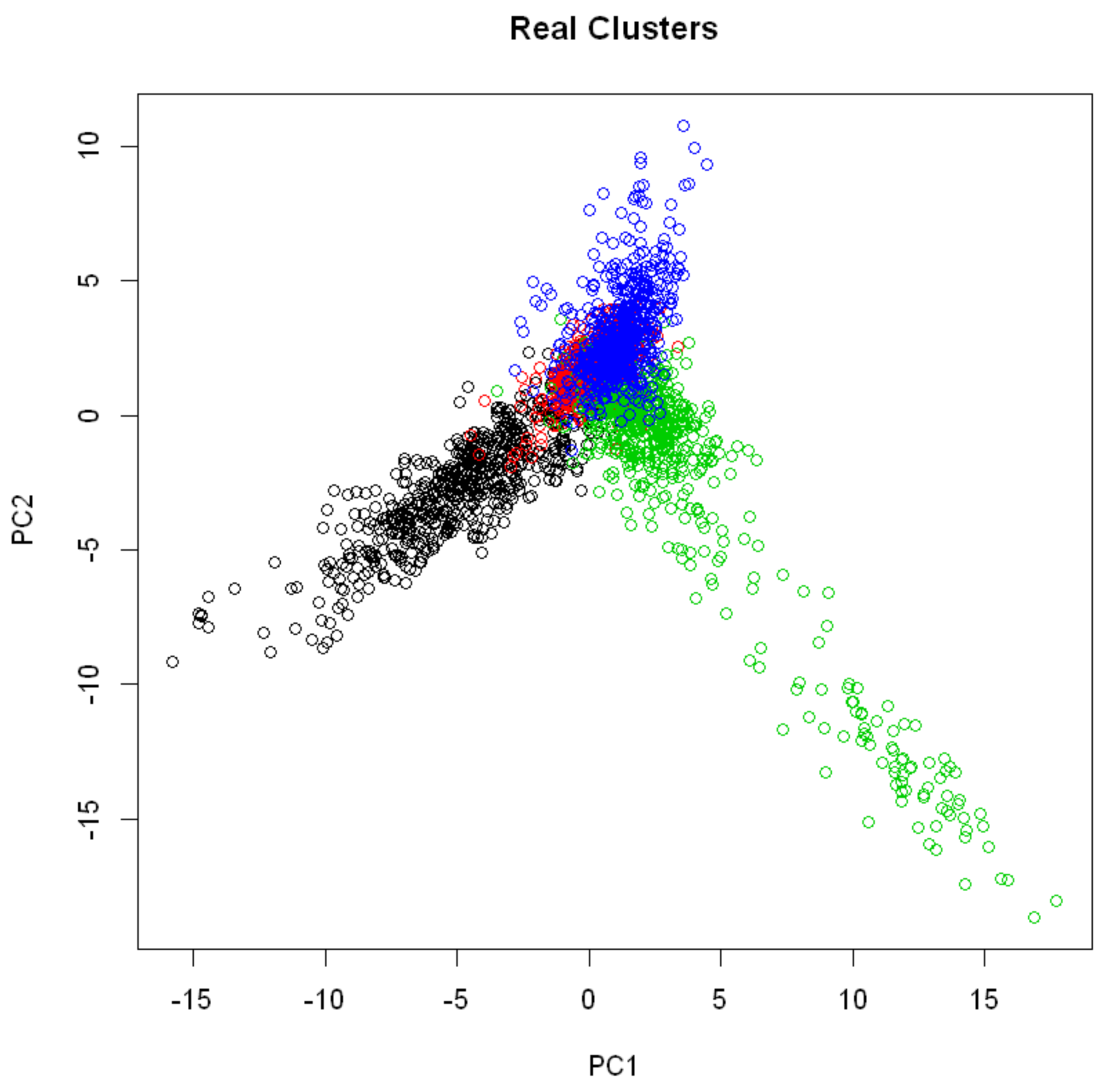
- The mixing components: $\varphi_k = \frac{N_k}{N}$ where $N_k := \sum_{n=1}^{N} \gamma(z_{n,k})$
- The word proportion parameters for each cluster: $\mu_{k,w} = \frac{\sum_{n=1}^{N} \gamma(z_{n,k}) c(w,d_n)}{\sum_{w'\in\mathcal{A}} \sum_{n=1}^{N} \gamma(z_{n,k}) c(w',d_n)}$

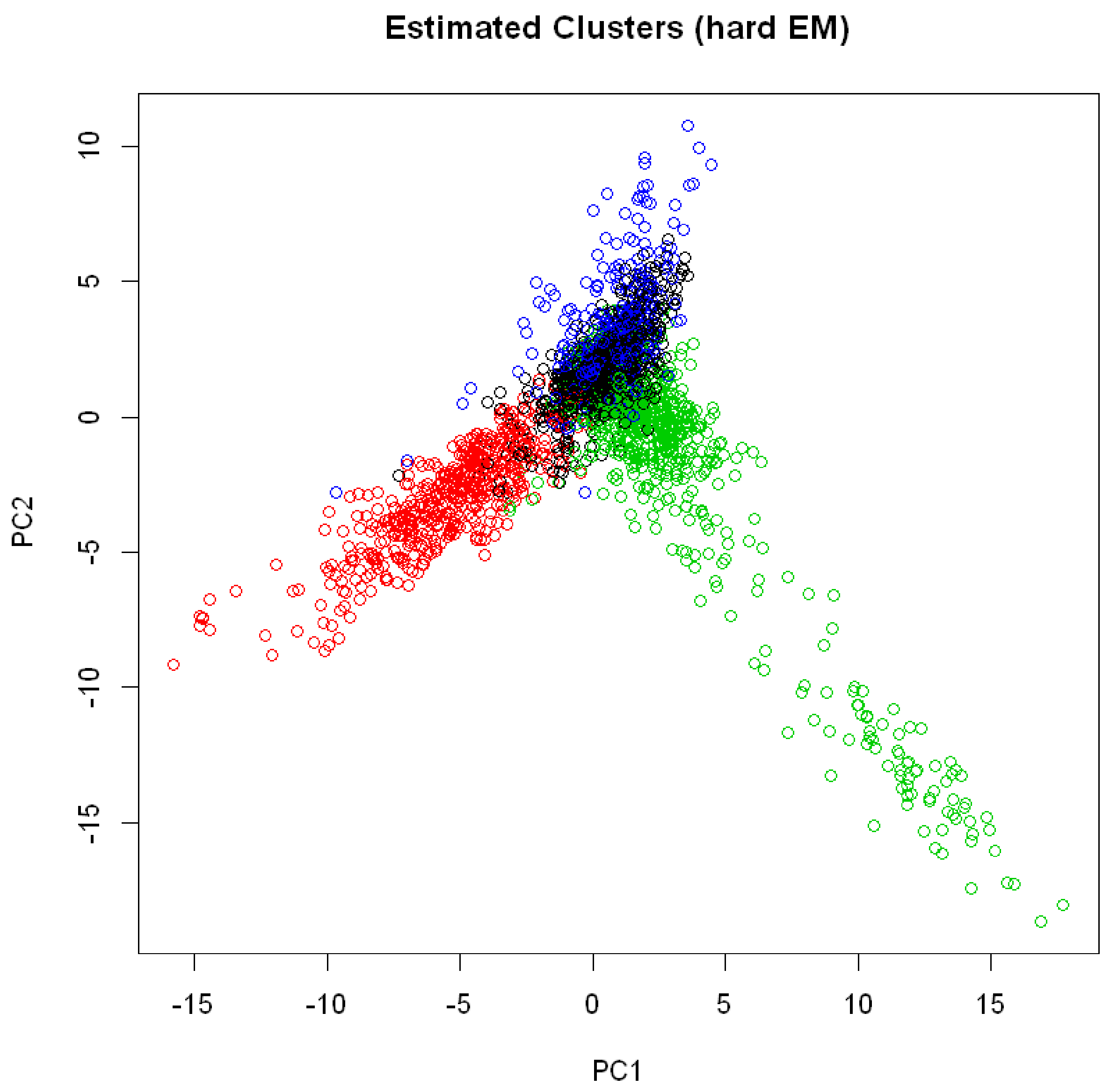
The Hard EM Algorithm:

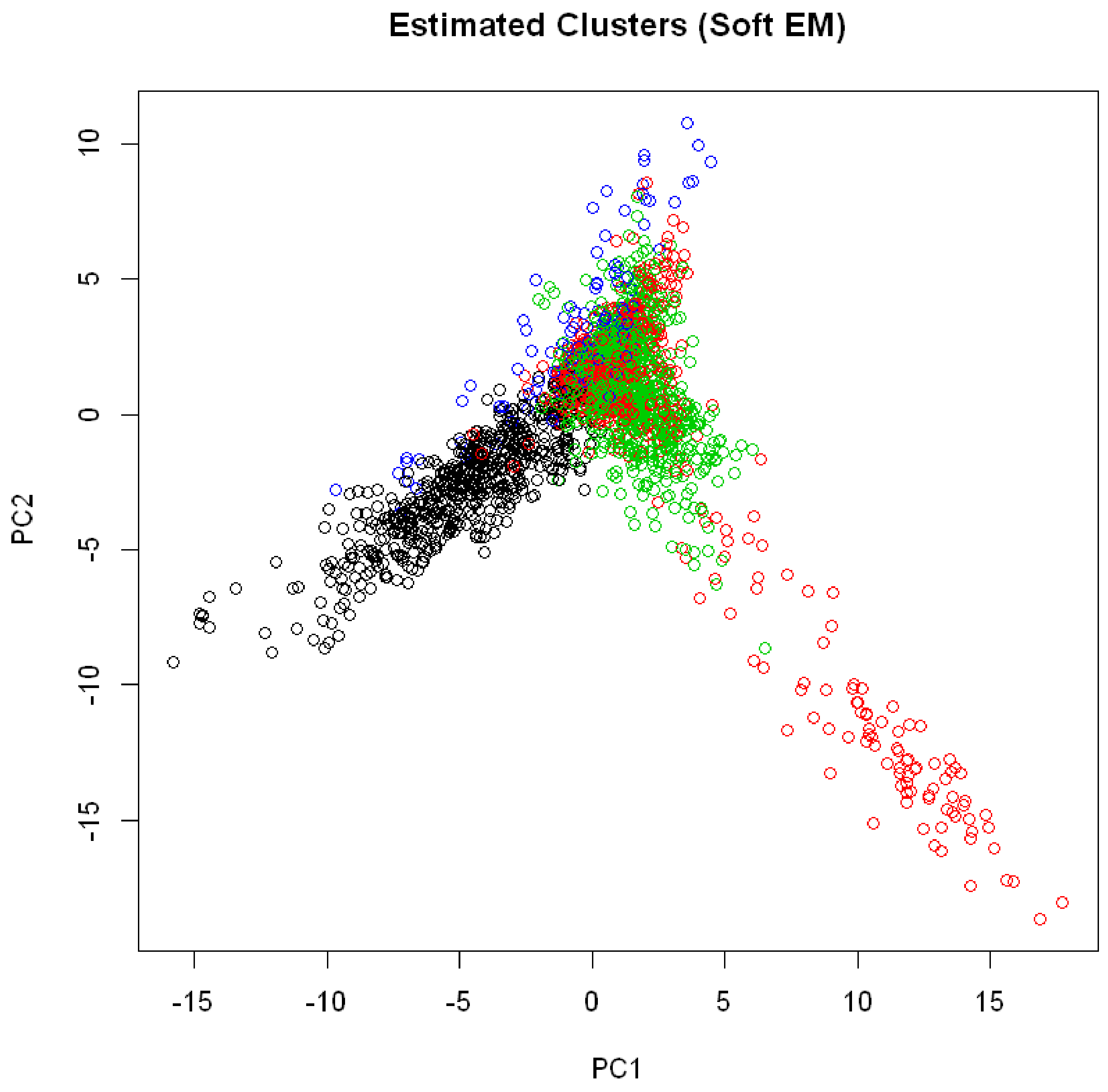
In the Hard-EM Algorithm, there is no expectation in over the latent variables in the definition of the Q function. Instead, only the most probable value for the latent variable is chosen to define the Q function. The algorithm works as follows:

- Choose an initial setting for the parameters $\boldsymbol{\theta}^{\text{old}} = (\boldsymbol{\varphi}^{\text{old}}, \boldsymbol{\mu}_1^{\text{old}}, \ldots, \boldsymbol{\mu}_K^{\text{old}})$
- While the convergence is not met:
  - **E step:** Set $\boldsymbol{Z}^* \leftarrow \arg\max_{\boldsymbol{Z}} p(\boldsymbol{Z}|\boldsymbol{X}, \boldsymbol{\theta}^{\text{old}})$
  - **M Step:** Set $\boldsymbol{\theta}^{\text{new}} \leftarrow \arg\max_{\boldsymbol{\theta}} \ln p(\boldsymbol{X}, \boldsymbol{Z}^*|\boldsymbol{\theta})$
  - $\boldsymbol{\theta}^{\text{old}} \leftarrow \boldsymbol{\theta}^{\text{new}}$

1.4 Cluster Plots:

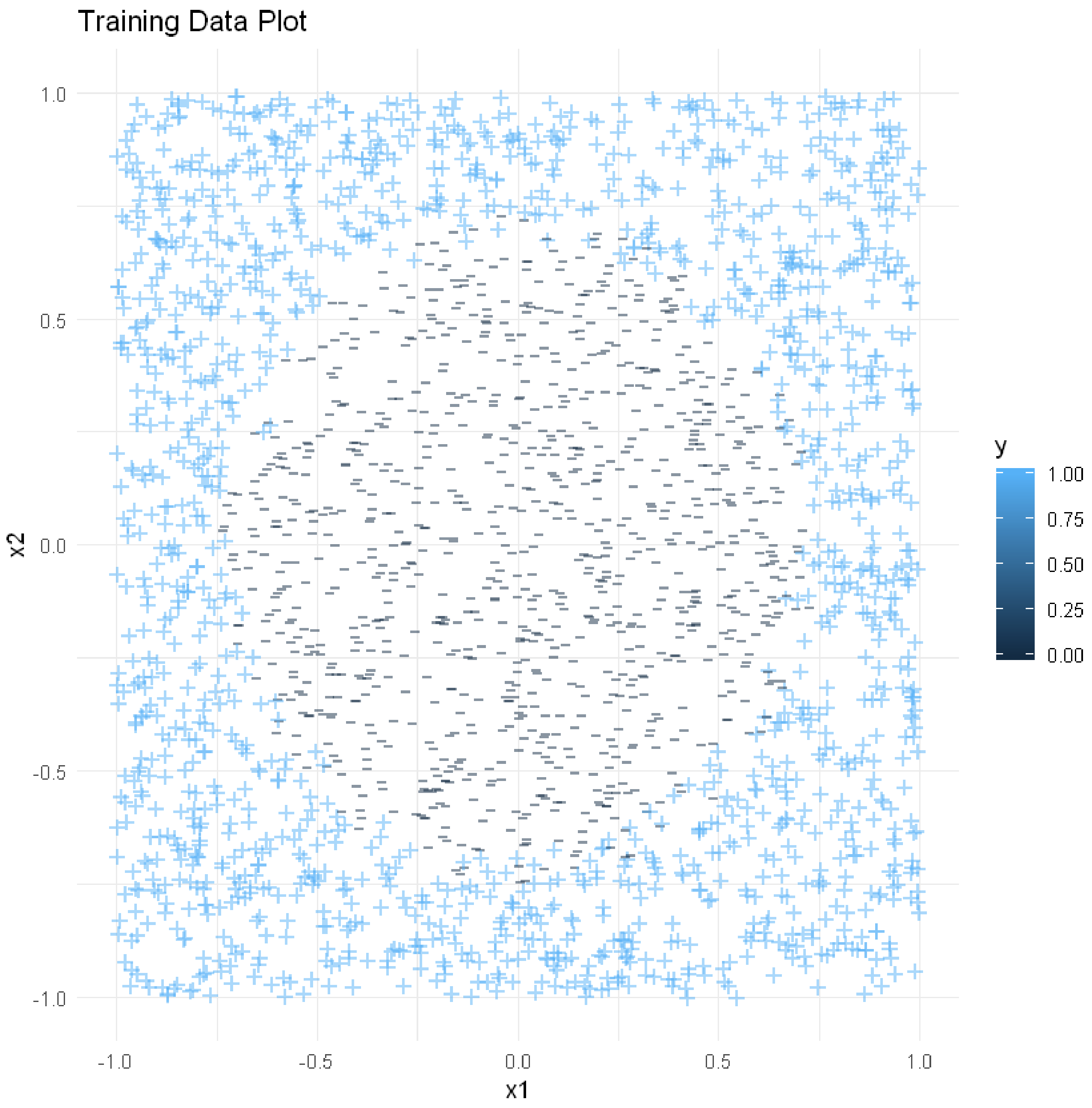



Estimated Clusters (hard EM)
PC2
PC1
10
5
0
-5
-10
-15
-15
-10
-5
0
5
10
15

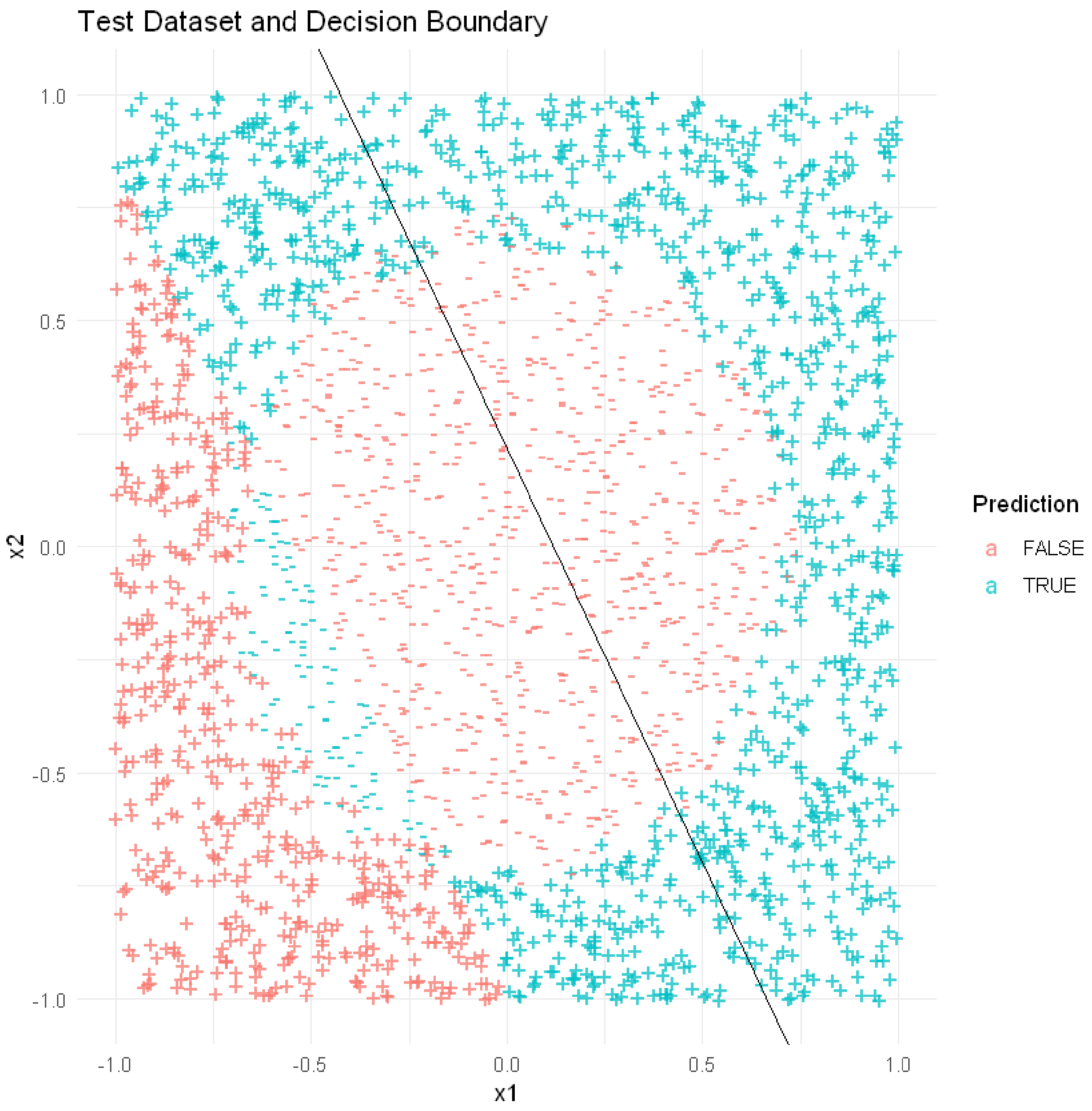


Estimated Clusters (Soft EM)
PC2
PC1
10
5
0
-5
-10
-15
-15
-10
-5
0
5
10
15

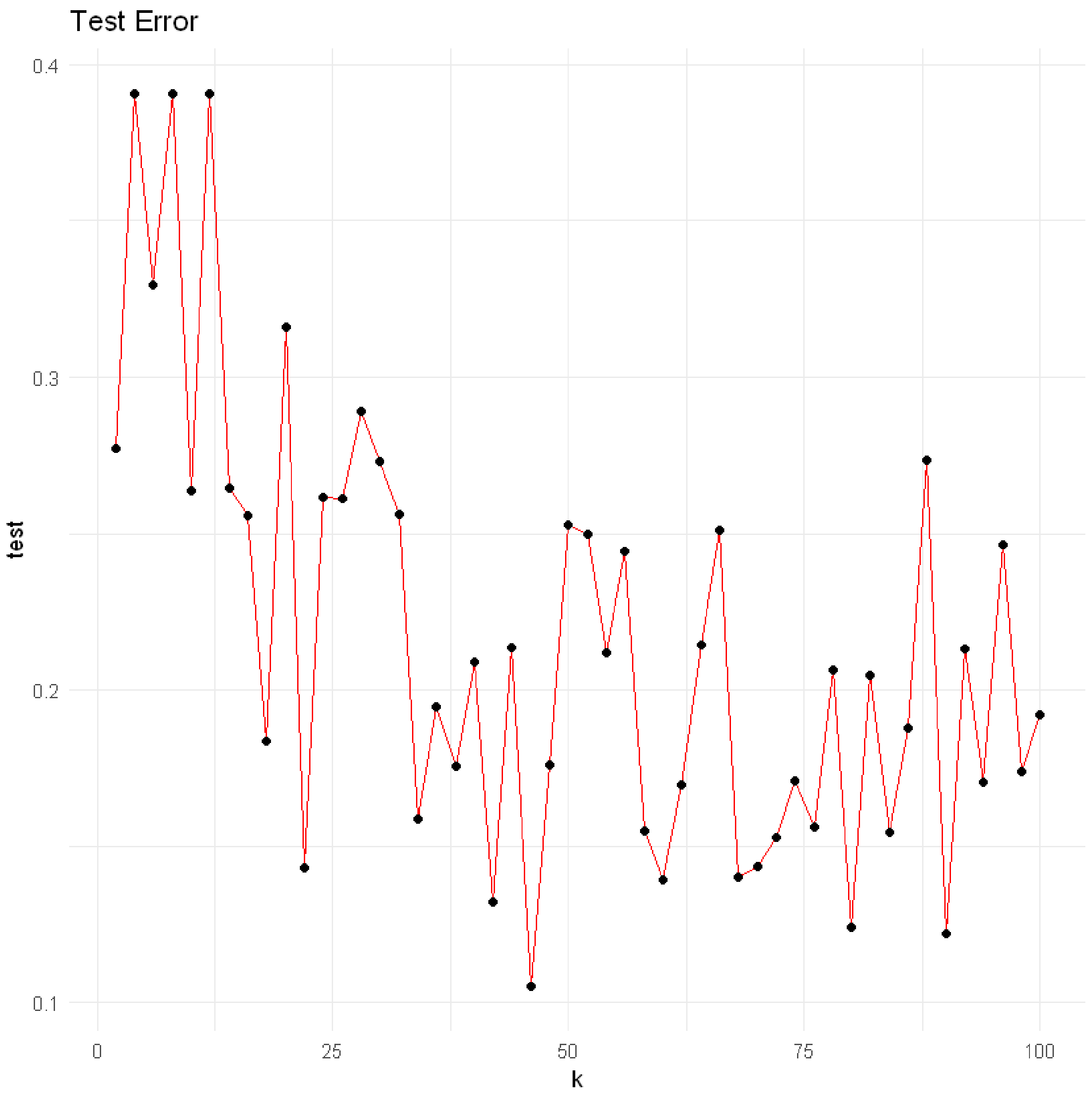
2.1 Plot of Data:



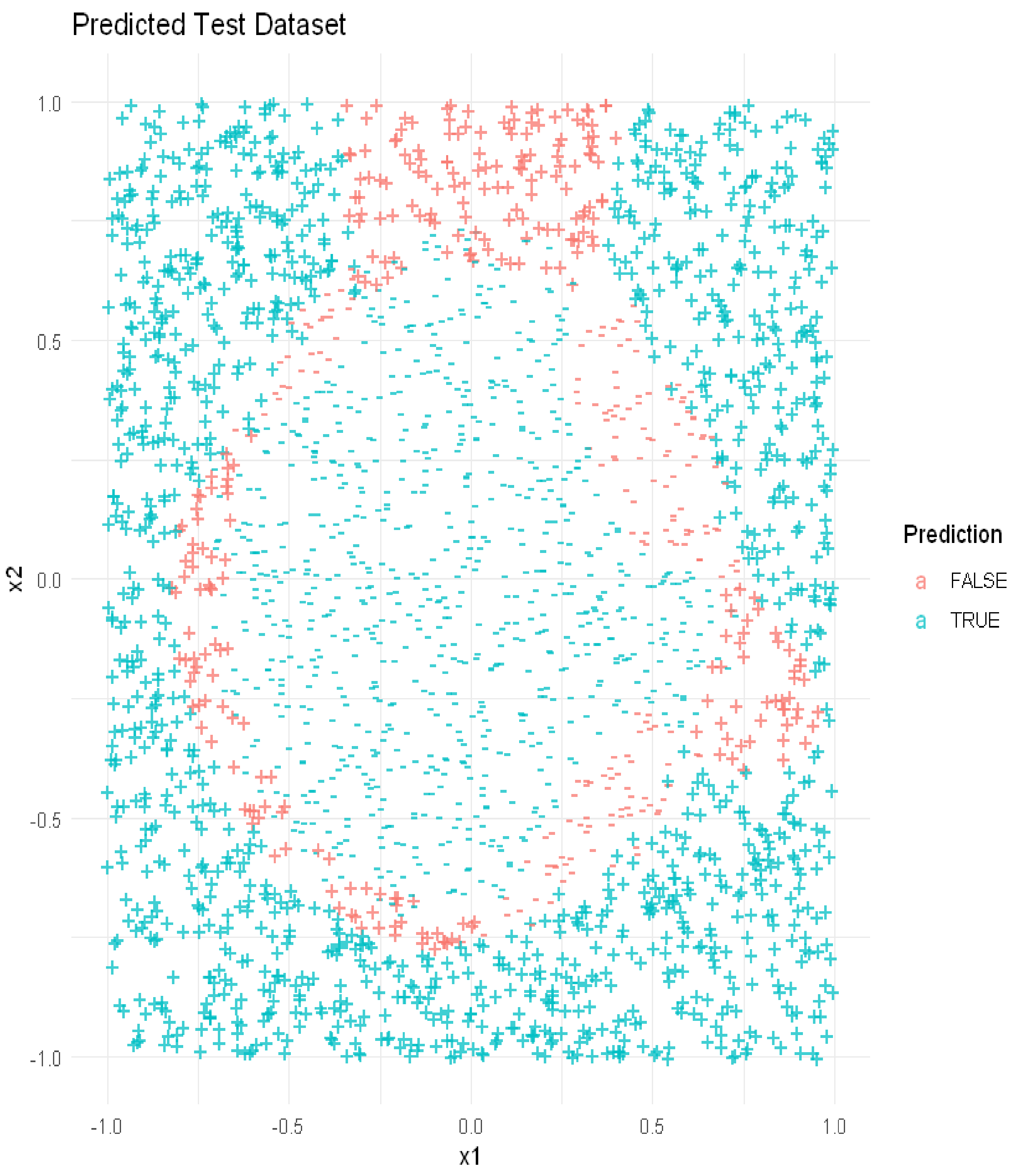
2.2 Perceptron plot:



2.3 NN Error vs K plot:



Plot for test prediction for Best K=46:



2.4 Error rates obtained by the perceptron and all variants of NN.

A data.frame: 50 × 3

k	NN_test_error	perceptron_test_error
<dbl>	<dbl>	<dbl>
2	0.2772	0.5432
4	0.3908	0.5432
6	0.3296	0.5432
8	0.3908	0.5432
10	0.2636	0.5432
12	0.3908	0.5432
14	0.2644	0.5432
16	0.2556	0.5432
18	0.1836	0.5432
20	0.3160	0.5432
22	0.1432	0.5432
24	0.2616	0.5432
26	0.2612	0.5432
28	0.2892	0.5432
30	0.2732	0.5432
32	0.2560	0.5432
34	0.1588	0.5432
36	0.1944	0.5432
38	0.1756	0.5432
40	0.2088	0.5432
42	0.1324	0.5432
44	0.2136	0.5432
46	0.1052	0.5432

A data.frame: 50 × 3

k	NN_test_error	perceptron_test_error
<dbl>	<dbl>	<dbl>
48	0.1760	0.5432
50	0.2528	0.5432
52	0.2500	0.5432
54	0.2120	0.5432
56	0.2444	0.5432
58	0.1548	0.5432
60	0.1392	0.5432
62	0.1696	0.5432
64	0.2144	0.5432
66	0.2512	0.5432
68	0.1404	0.5432
70	0.1436	0.5432
72	0.1528	0.5432
74	0.1708	0.5432
76	0.1564	0.5432
78	0.2064	0.5432
80	0.1240	0.5432
82	0.2048	0.5432
84	0.1544	0.5432
86	0.1880	0.5432
88	0.2736	0.5432
90	0.1220	0.5432
92	0.2132	0.5432
94	0.1704	0.5432
96	0.2464	0.5432

A data.frame: 50 × 3

k	NN_test_error	perceptron_test_error
<dbl>	<dbl>	<dbl>
98	0.1740	0.5432
100	0.1920	0.5432

Best Value of K=46, Test error= 10.52% :

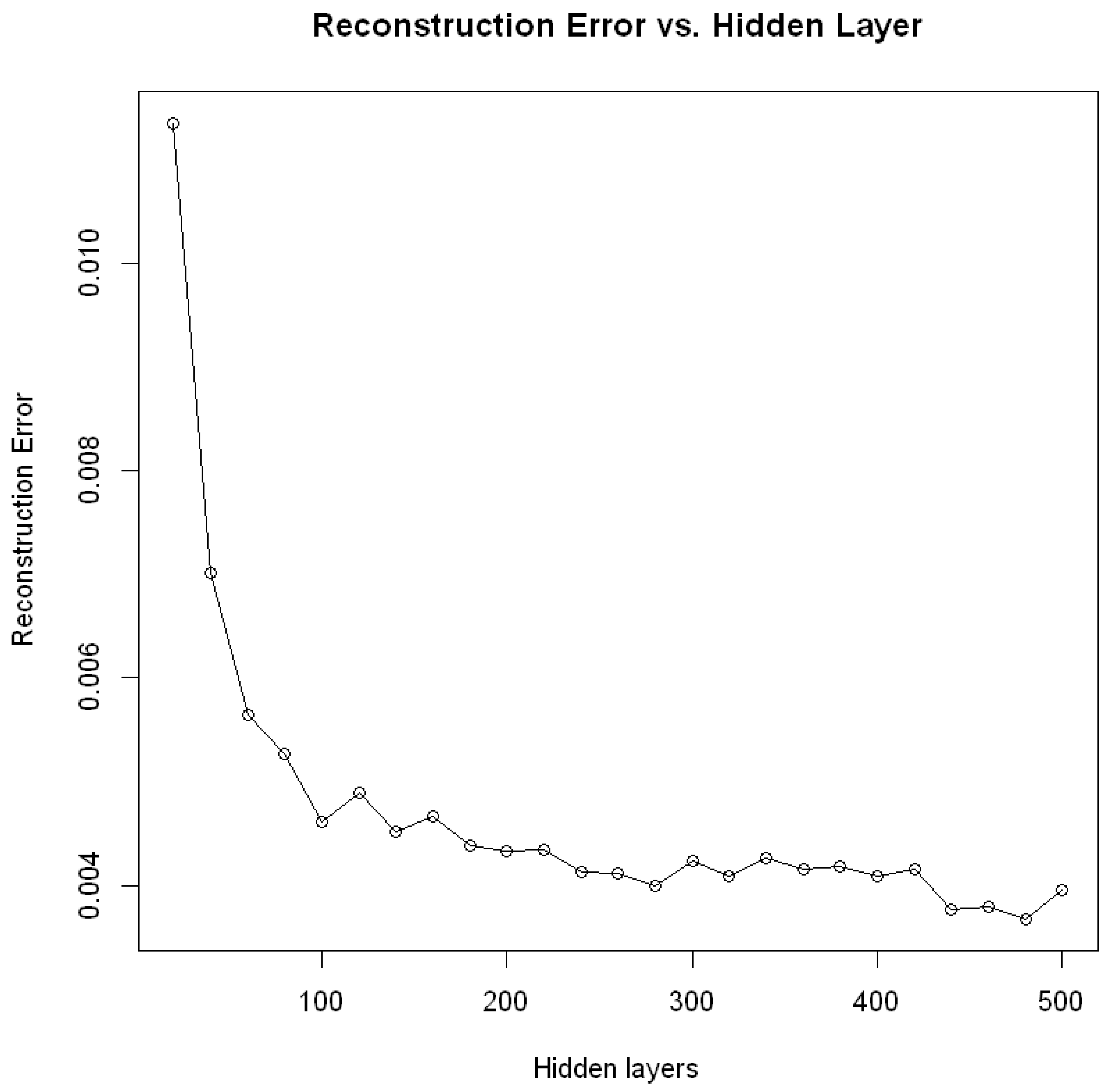
A data.frame: 1 × 3

	k	NN_test_error	perceptron_test_error
	<dbl>	<dbl>	<dbl>
23	46	0.1052	0.5432

2.5 In your PDF report explain the reason(s) responsible for such difference between perceptron and a 3-layer NN.

- The neural network comprises of several stages, each of which resembles a perceptron. .Hence it is also called Multilayer Perceptron.
- Here we get best test error of Neural Network as 10.52% for k=46 whereas for perceptron we get the test error as 54.32%. It is evident that the performance of Neural network is far better than the perceptron.
- The main reason for such a difference in performance is that a neural network uses continuous nonlinearities in the hidden units (e.g. σ(.) or tanh), whereas the perceptron uses step-function nonlinearities.
- This layered architecture enables the neural network to model the data which is not lineraly separable and minimise the test error to a much greater extent as compared a perceptron

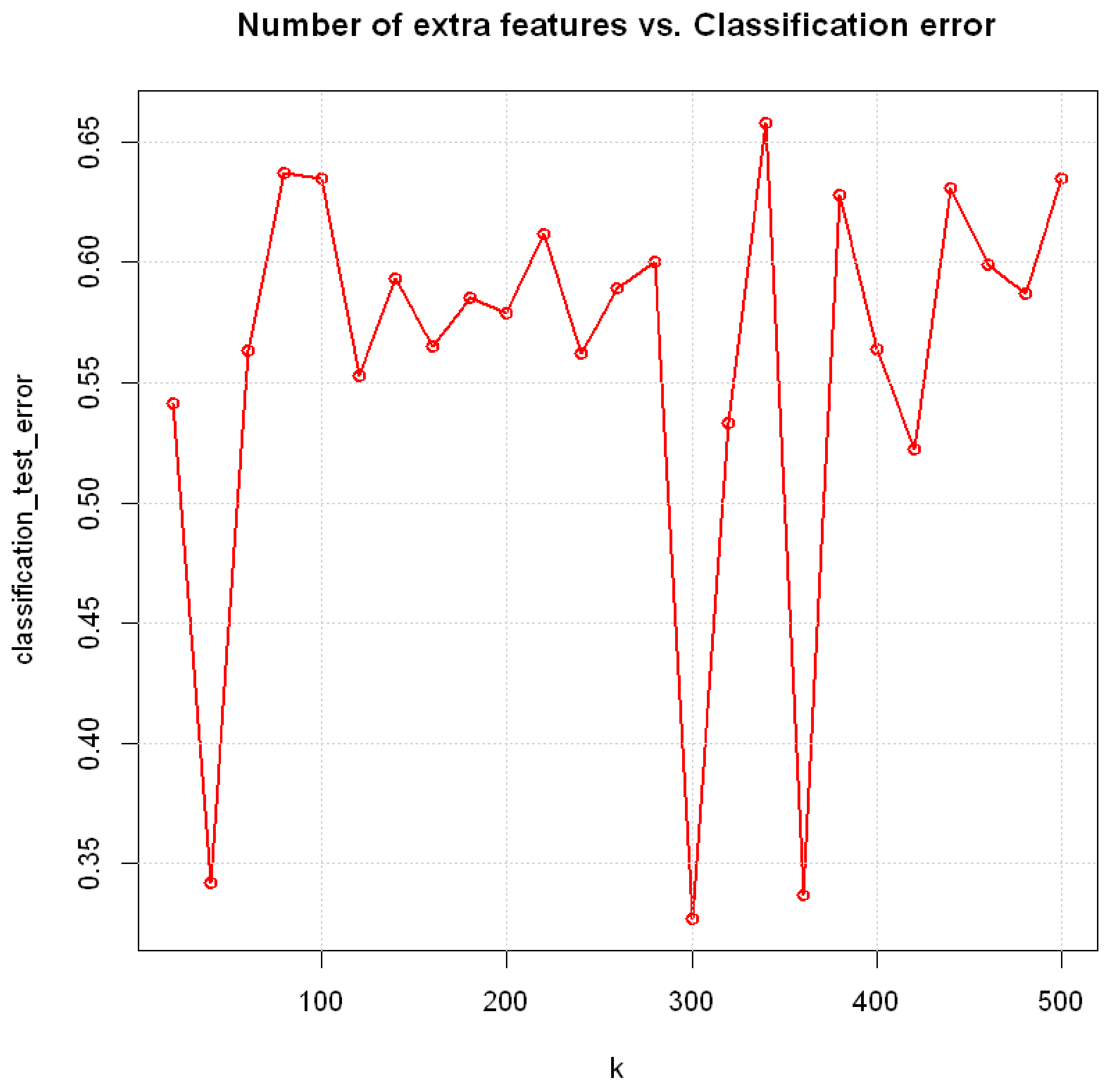
3.3 Plot for Reconstruction Error vs K (Number of neurons in hidden layer)



Findings:

- We get the minimum error of 0.003673218 at k=480
- It can be seen from the below plot that the reconstruction error follows the general trend,i.e it decreases with increase in the number of neurons in the middle layer.

3.6 Plot for Classification Error vs Number of Extra Features:



3.7

- **Optimum number of units in the middle layer in terms of reconstruction error = 480**

A data.frame: 1 × 2

	k	reconstruction_test_error
	<dbl>	<dbl>
24	480	0.003673218

- **Optimum number of units in the middle layer in terms of classification error= 300**

A data.frame: 1 × 2

	k	classification_test_error
	<dbl>	<dbl>
15	300	0.327

3.8 Comparing the plot from Step III and VI, do you observe any relation between the reconstruction error and misclassification error?

- We get a minimum reconstruction error of 0.003673218 for 480 neurons in task3, while in task 6 we get a minimum classification error of 0.327 for 300 extra featues.
- The plot in task 3 suggests that generally the reconstruction error decreases with the increase in number of neurons in the middle layer.
- But the plot in task6, i.e classification error v.s the extra features, shows no such general behaviour. There are erratic fluctuations in the classification error with respect to the increase in number of extra features and the classification test error is much higher than the reconstruction error.
- The main reason for this could be that adding extra features makes the model very complex and hence it leads to overfitting. Therefore the overfitting results in high test error.
- One of the ways to improve this could be to increase the number of neurons of the classifier while we add extra features to it, rather than fixing it to 100.
- In the autoencoder we are continuously increasing the number of neurons and hence diminishing the error whereas in the classifier we have fixed the neurons to 100 and we keep on increasing the complexity (features), which leads to poor test results.