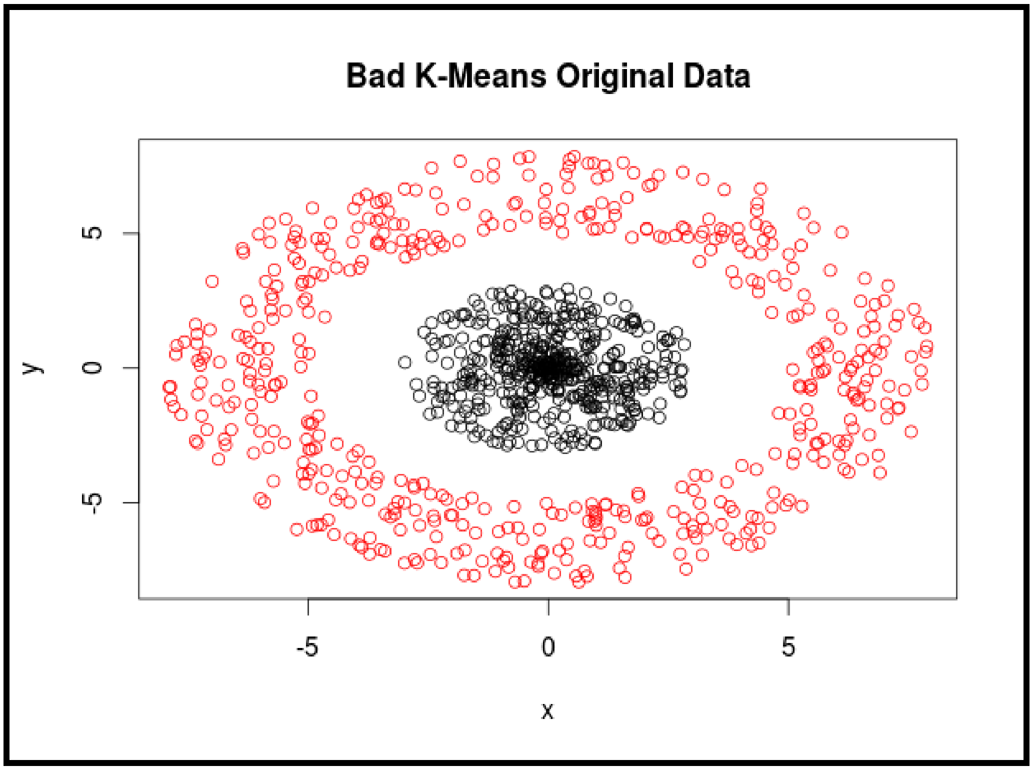
Project: Kernelization, Kernel Tricks
Unity Id: akagrawa

Note: Please refer the scripts inside the folder codes. The scripts are written for kmeans, SVM, PCA and pipeline separately based on exercise 1 to 4.

Exercise 1: ***Generating the data sets*. Write a script (in R, Matlab, or SAS) that generates three data sets in a 2-dimensional space, defined as follows.**

(a) BAD_kmeans: The data set for which the kmeans clustering algorithm will not perform well.

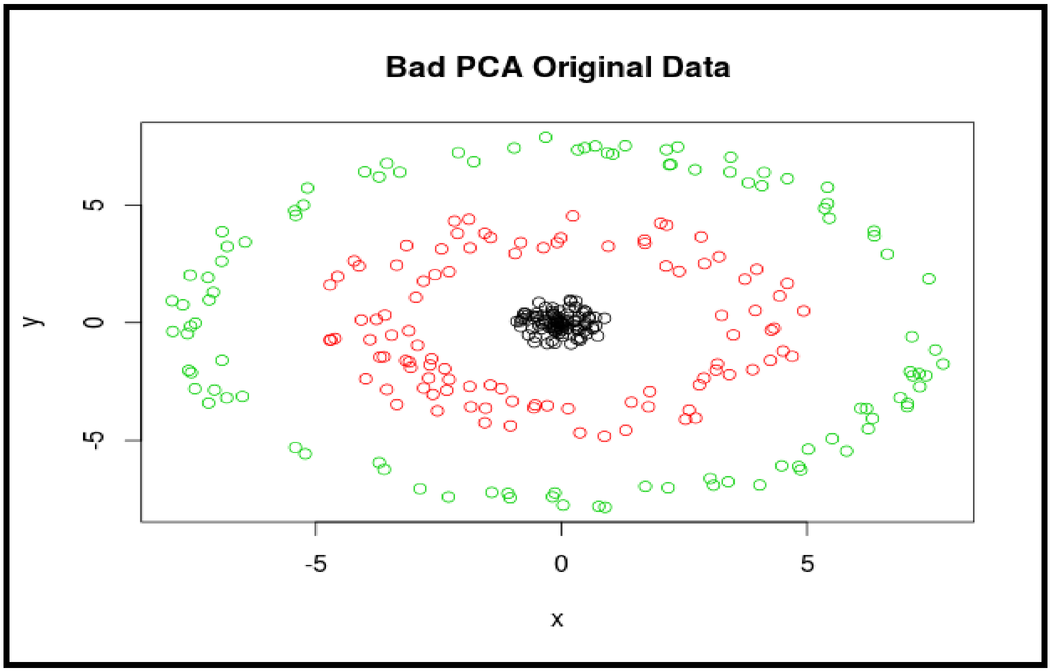
Please refer p3_kmeans.R script for the code to generate and plot the data.



The kmeans will not be able to identify the natural cluster in the above data in 2 Dimension.

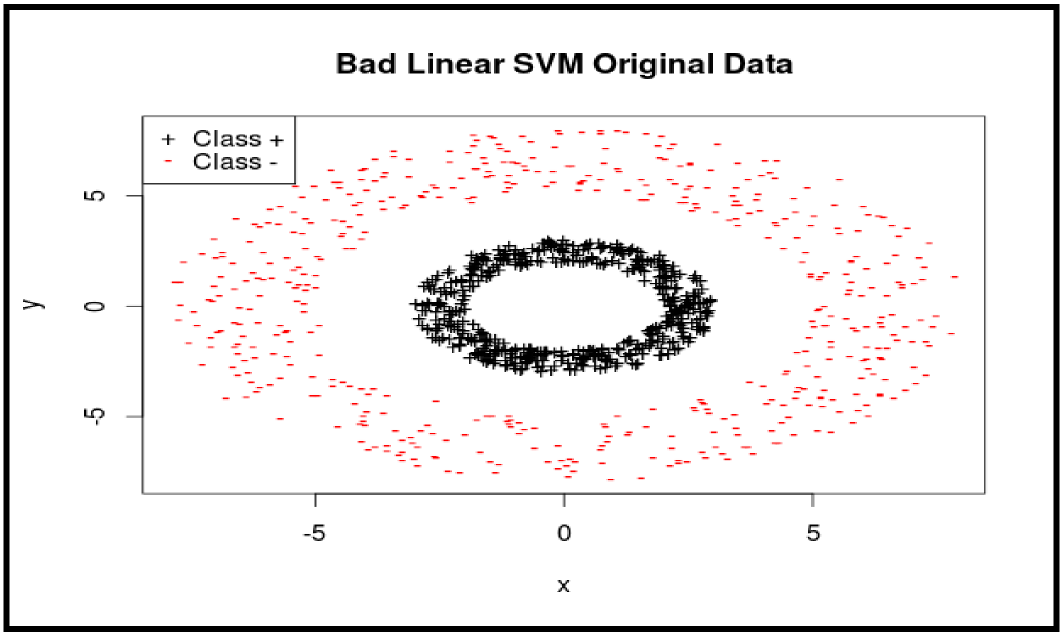
(b) BAD_pca: The data set for which the Principal Component Analysis (PCA) dimension reduction method upon projection of the original points into 1-dimensional space (i.e., the first eigenvector) will not perform well.

Please refer p3_PCA.R script for the code to generate and plot the data. For this data set the linear PCA will not be able to find linear separation wrt to its eigen vector plots.



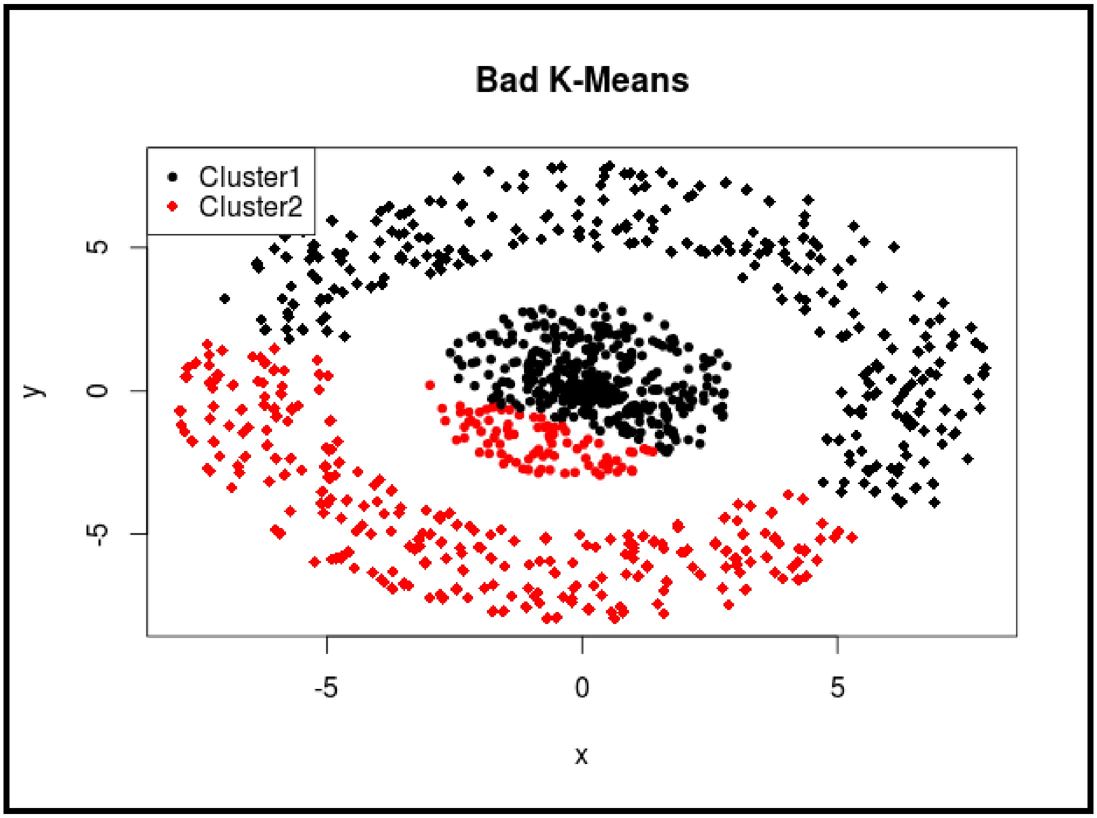
(c) BAD_svm: The data set for which the *linear* Support Vector Machine (SVM) supervised classification method using two classes of points (positive and negative) will not perform well.

Please refer p3_SVM.R script for the code to generate and plot the data.



In the plot, linear SVM will not be able to find linear boundaries for classification.

Exercise 2: ***Evaluating the "badness" of the data mining methods***. Write a script that uses the BAD data set in Exercise 1, runs the corresponding data mining method, produces the output from the method, and evaluates how bad the performance of this method is.



a) Bad K-means : Please refer p3_kmeans.R script for the performance evaluation code.

Performance Measure:

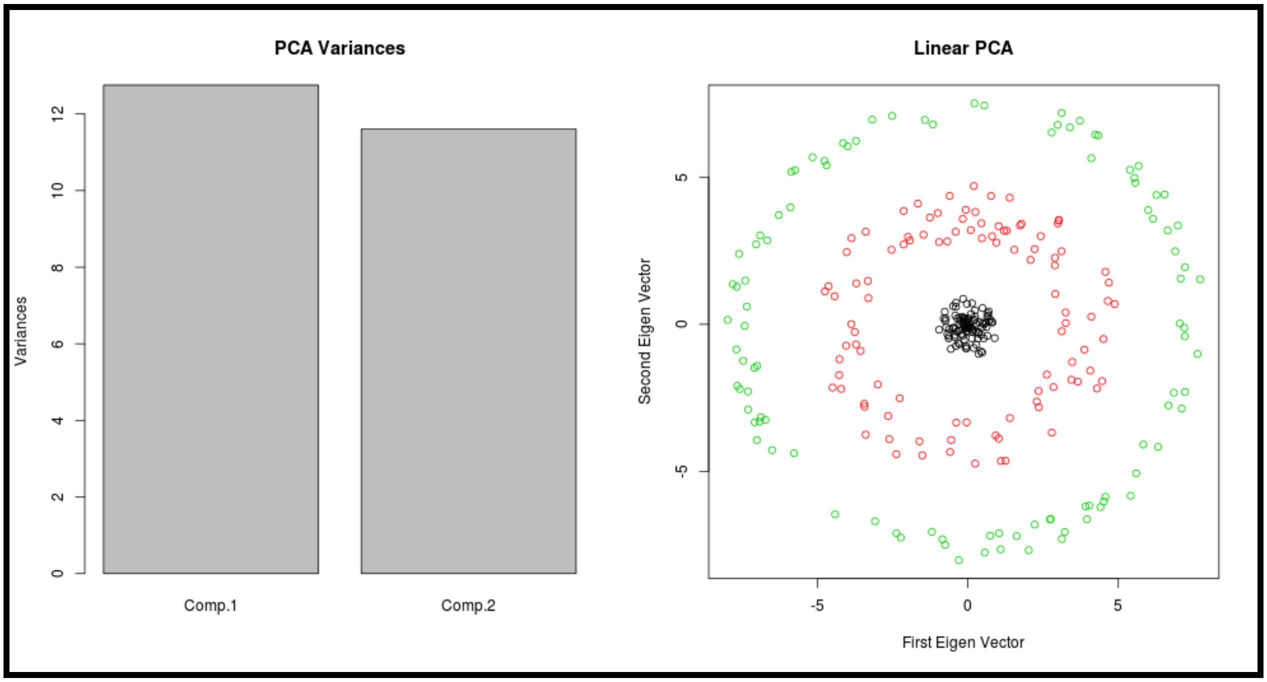
Confusion Matrix

	cl1	**cl2**
GT1	329	171
GT2	247	253

Accuray	**0.582**
Error Rate	0.418
True Positive Rate	0.658
True Negative Rate	0.506
False Positive Rate	0.494
False Negative Rate	0.342
Precision+	0.5711806
Precision-	0.5966981
F-measure+	0.6115242

F-measure- 0.547619
G-Mean 0.5770165

b) Bad PCA : Please refer p3_pca.R script for the performance evaluation code.



Performance Metrics :

	Comp.1	Comp.2
SS loadings	1.0	1.0
Proportion Var	0.5	0.5
Cumulative Var	0.5	1.0

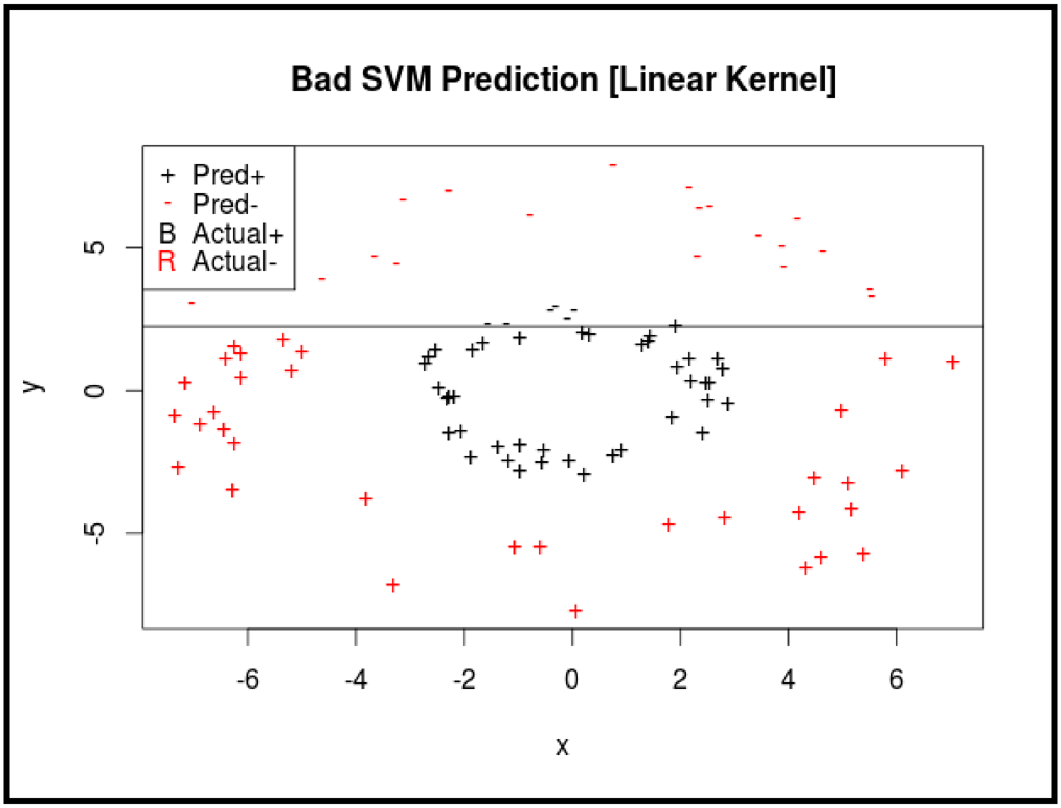
Importance of components:

	Comp.1	Comp.2
Standard deviation	3.571250	3.406975
Proportion of Variance	**0.523528**	**0.476472**
Cumulative Proportion	0.523528	1.000000

Clearly, not a linear separation of the data after applying linear PCA.

c) Bad SVM : Please refer p3_SVM.R script for the performance evaluation code.

Below is the plot of the prediction done by linear SVM, on the model trained on the training data. 90% of the data is taken for training the data, while 10% of the remaining is taken as test data for cross validation. Please find the performance metrics below.



<u>Performance Metrics:</u>

<u>Confusion Matrix</u>

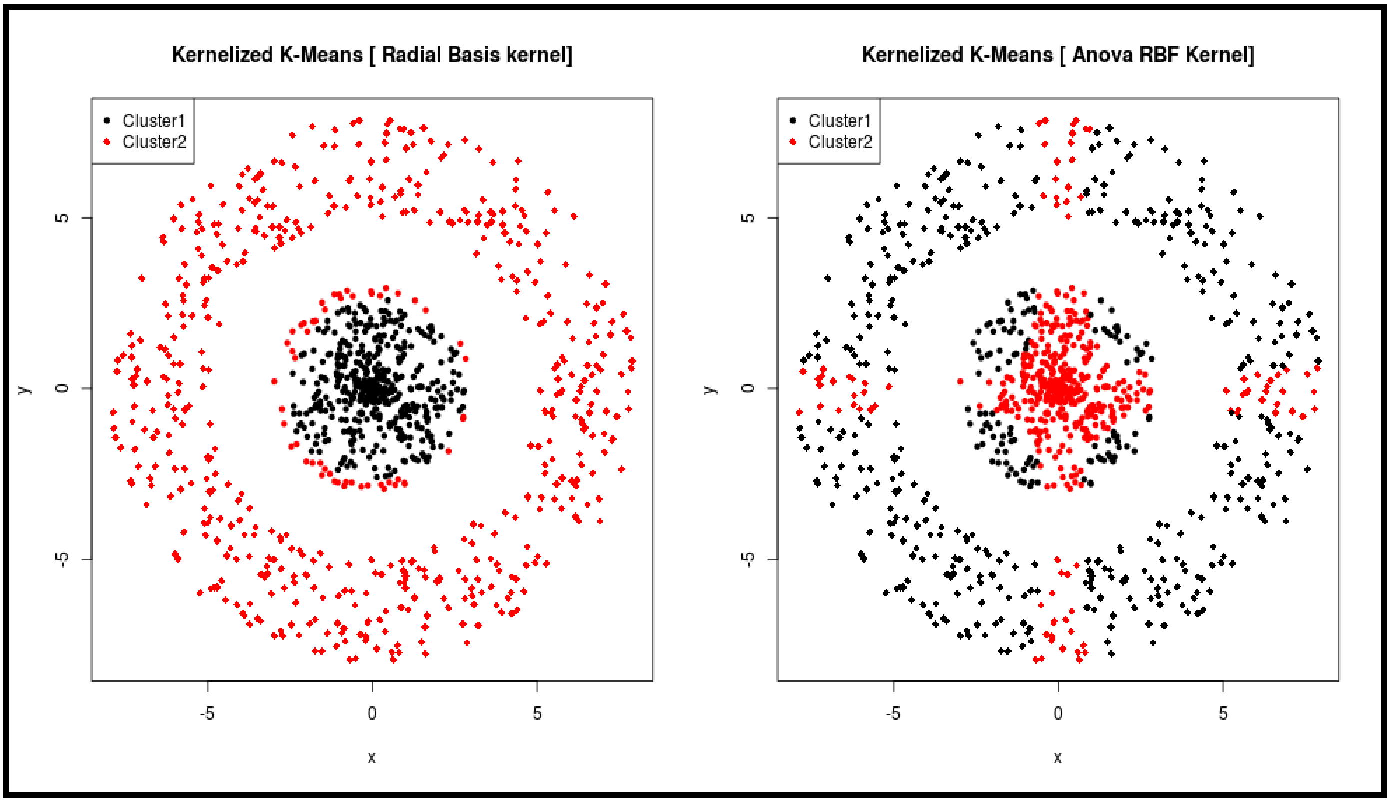
	Pred+	Pred-
GT+	40	6
GT-	33	21

Accuray	**0.61**
Error Rate	0.39
True Positive Rate	0.8695652
True Negative Rate	0.3888889
False Positive Rate	0.6111111
False Negative Rate	0.1304348
Precision+	0.5479452
Precision-	0.7777778
F-measure+	0.6722689
F-measure-	0.5185185
G-Mean	0.5815189

Exercise 3: ***Kernelizing the methods.*** **Write a script that uses the *kernalized* version of each of the data mining method in Exercise 2.**

KKmeans : RBF Kernel & ANOVA RBF Kernel

(a) Choose at least two kernels for each of the methods.



Above is the plot for Kernelized Kmeans for RBF and ANOVA RBF kernel and the corresponding cluster labels.

(b) Use the same performance metrics as in Ex. 3, and compare the performance obtained by the methods after applying the kernel trick versus the original un-kernelized versions of the techniques.

Performance Metrics	**Original Kmeans**	**RBF Kernelized Kmeans**	**ANOVA Kernelized Kmeans**
Confusion Matrix	cl1 cl2 GT1 329 171 GT2 247 253	cl1 cl2 GT1 446 54 GT2 0 500	cl1 cl2 GT1 123 377 GT2 415 85
Accuracy	0.61	**0.946**	0.208

Error Rate	0.39	0.054	0.792
True Positive Rate	0.8695652	0.892	0.246
True Negative Rate	0.3888889	1	0.17
False Positive Rate	0.6111111	0	0.83
False Negative Rate	0.1304348	0.108	0.754
Precision+	0.5479452	1	0.2286245
Precision-	0.7777778	0.9025271	0.1839827
F-measure+	0.6722689	0.9429175	0.2369942
F-measure-	0.5185185	0.9487666	0.1767152
G-Mean	0.5815189	0.9444575	0.2044994

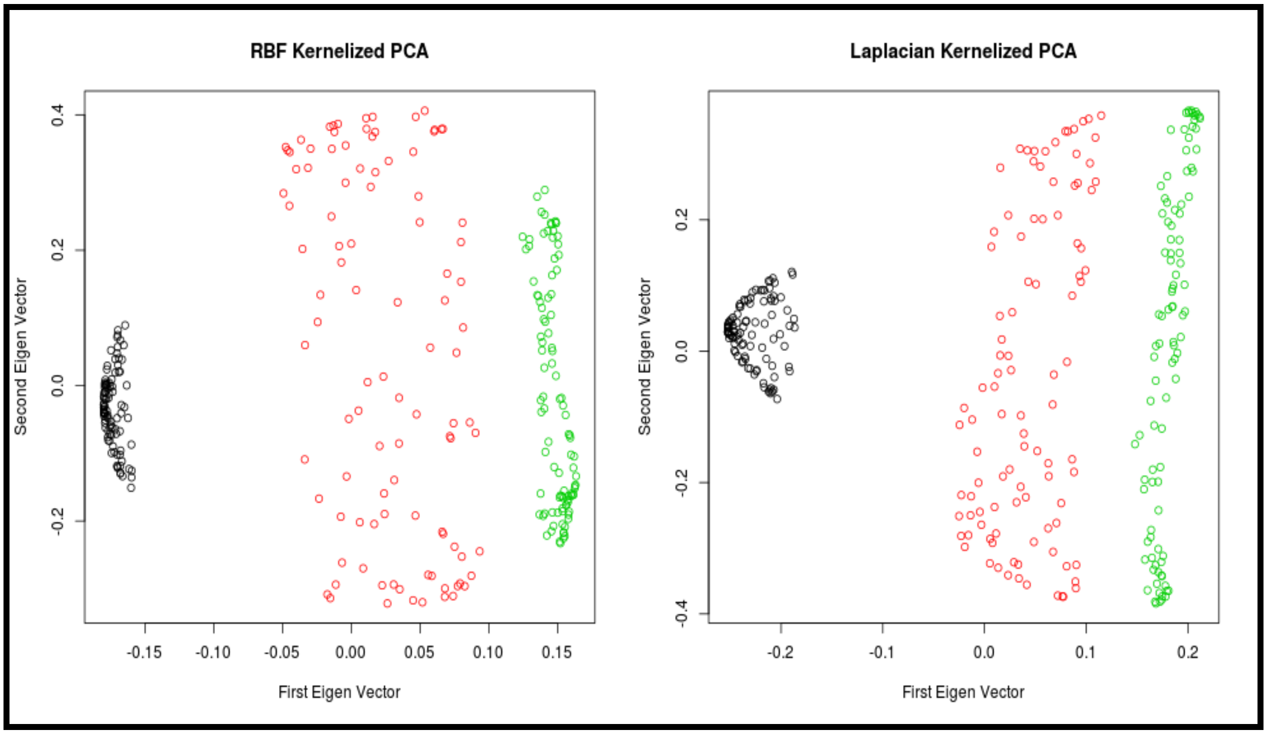
From the above comparision we can say that RBF Kernelized Kmeans performs much better than original and ANOVA Kernelized Kmeans.

(c) Do you observe the difference in performance when you use different kernels?
Yes, based on the kernel methods, the performance differs a lot. In the above table, we can clearly see that not all kernel methods tend to have accurate prediction, it is highly dependent on the data orientation and distribution in space.

(d) What are the best performance results do you get by playing with different kernels and kernel parameters?
Depending on the values of kernels parameters different results are observed. In the comparison table above we can see the performance of different kernels for the default parameters. Among those RBF Kernel performs best for automatic kernel parameter.

<u>PCA : RBF Kernel & Laplacian RBF Kernel</u>
(a) Choose at least two kernels for each of the methods.

Below is the plot for Kernelized Kmeans for RBF and ANOVA RBF kernel and the corresponding cluster labels.



The data points are easily linearly separable after applying the kernelized PCA with RBF kernel and Laplacian Kernel.

(b) Use the same performance metrics as in Ex. 2, and compare the performance obtained by the methods after applying the kernel trick versus the original un-kernelized versions of the techniques.

Importance of eigenvectors

Linear PCA

	comp1	**comp2**
Proportion of Variance	**0.52353**	**0.47647**
Cumulative Proportion	0.52353	1.00000

RBF Kernelized PCA

	comp1	**comp2**	**comp3**	**comp4**	**comp5**	**comp6**
Proportion of Variance	0.25494	0.13477	0.12670	0.081437	0.069097	0.050095
Cumulative Proportion	0.25495	0.38972	0.51642	0.59786	0.66696	0.71705

Laplacian Kernelized PCA

	comp1	**comp2**	**comp3**	**comp4**	**comp5**	**comp6**
Proportion of Variance	0.18404	0.12894	0.12596	0.05614	0.0470	0.03525
Cumulative Proportion	0.18404	0.31299	0.43896	0.49510	0.5421	0.57741

From the above metric comparison, we can say that for the given data with the appropriate kernel parameters the kernel functions can linearly separate the data. It seems like they doesn’t capture much of a variability of the data in first few eigenvalues, but maximum variability is captured in first few dimensions as taking only first 2 dimensions as first 2 eigenvectors, the kernel functions are able to separate the data set linearly.

(c) Do you observe the difference in performance when you use different kernels?
Yes, based on the kernel methods, the performance differs a lot. In the above metrics, we can clearly see that not all kernel methods tend to have same number of eigen vectors capturing equal ratio of dataset variability. RBF performance is high as compare to Laplacian since it captures the variability in lower dimension as compare to Laplacian Kernel with linear separability.

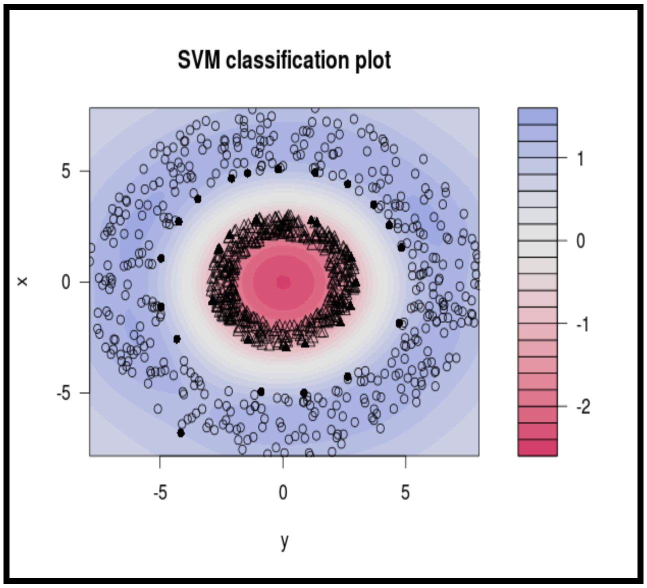
(d) What are the best performance results do you get by playing with different kernels and kernel parameters?
For the RBF Kernel function , sigma = 0.06 , eigen vectors = 39 , captures= 71.7% variability of the dataset in top 6 eigenvalues
For the Laplacian Kernel function , sigma = 0.19, eigen vectors =215, captures= 57.7% variability of the dataset in top 6 eigenvalues

With the increase of sigma (inverse kernel width), the number of eigen vectors tend to increase.

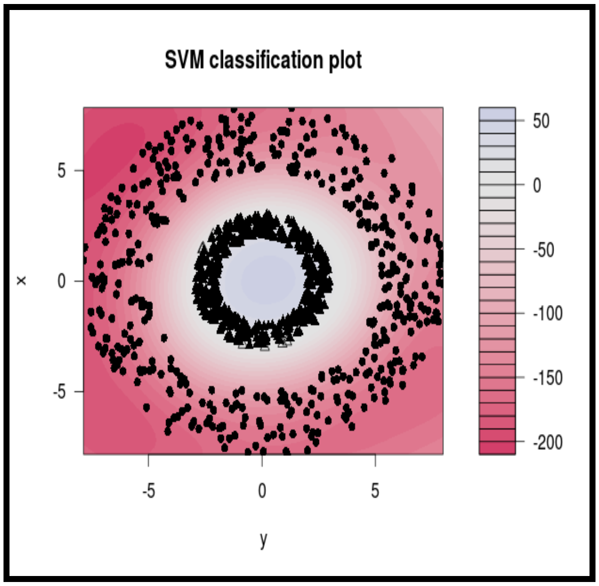
SVM : RBF Kernel & Hyperbolic Tangent Kernel

(a) Choose at least two kernels for each of the methods.

RBF Kernel SVM

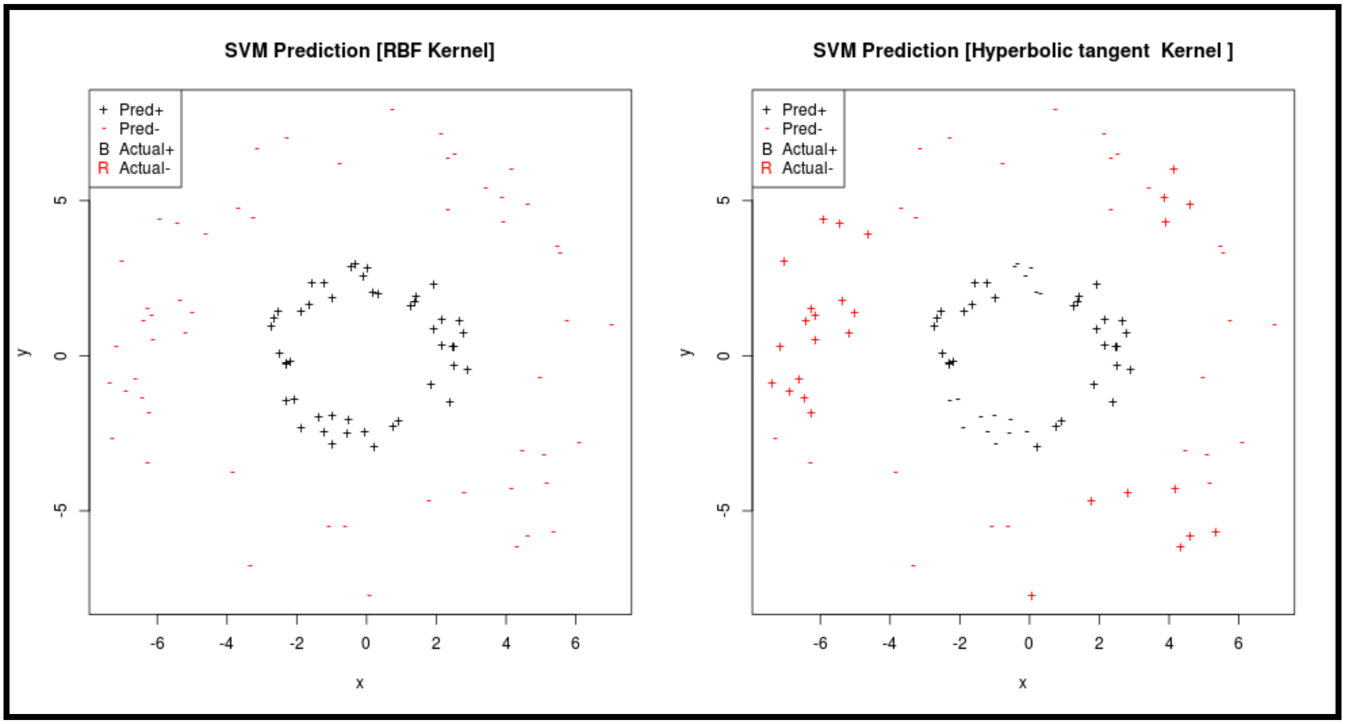


Hyperbolic tangent Kernel SVM



Above is the plot for Kernelized SVM for RBF and Hyperbolic tangent kernel and the corresponding support vectors in dark black symbols.

(b) Use the same performance metrics as in Ex. 3, and compare the performance obtained by the methods after applying the kernel trick versus the original un-kernelized versions of the techniques.



We can observe that both the above kernel functions have increased the accuracy with respect to linear SVM.

Performance Metrics	Linear SVM	RBF Kernelized SVM	Hyperbolic tangent Kernelized SVM
Confusion Matrix	Pred+ Pred- GT+ 40 6 GT- 33 21	Pred+ Pred- GT+ 46 0 GT- 0 54	Pred+ Pred- GT+ 30 16 GT- 28 26
Accuracy	**0.61**	**1**	**0.56**
Error Rate	0.39	0	0.44
True Positive Rate	0.8695652	1	0.6521739

True Negative Rate	0.3888889	1	0.4814815
False Positive Rate	0.6111111	0	0.5185185
False Negative Rate	0.1304348	0	0.3478261
Precision+	0.5479452	1	0.5172414
Precision-	0.7777778	1	0.6190476
F-measure+	0.6722689	1	0.5769231
F-measure-	0.5185185	1	0.5416667
G-Mean	0.5815189	1	0.5603657

From the above comparison we can say that RBF Kernelized SVM performs much better than original and hyperbolic tangent Kernelized SVM.

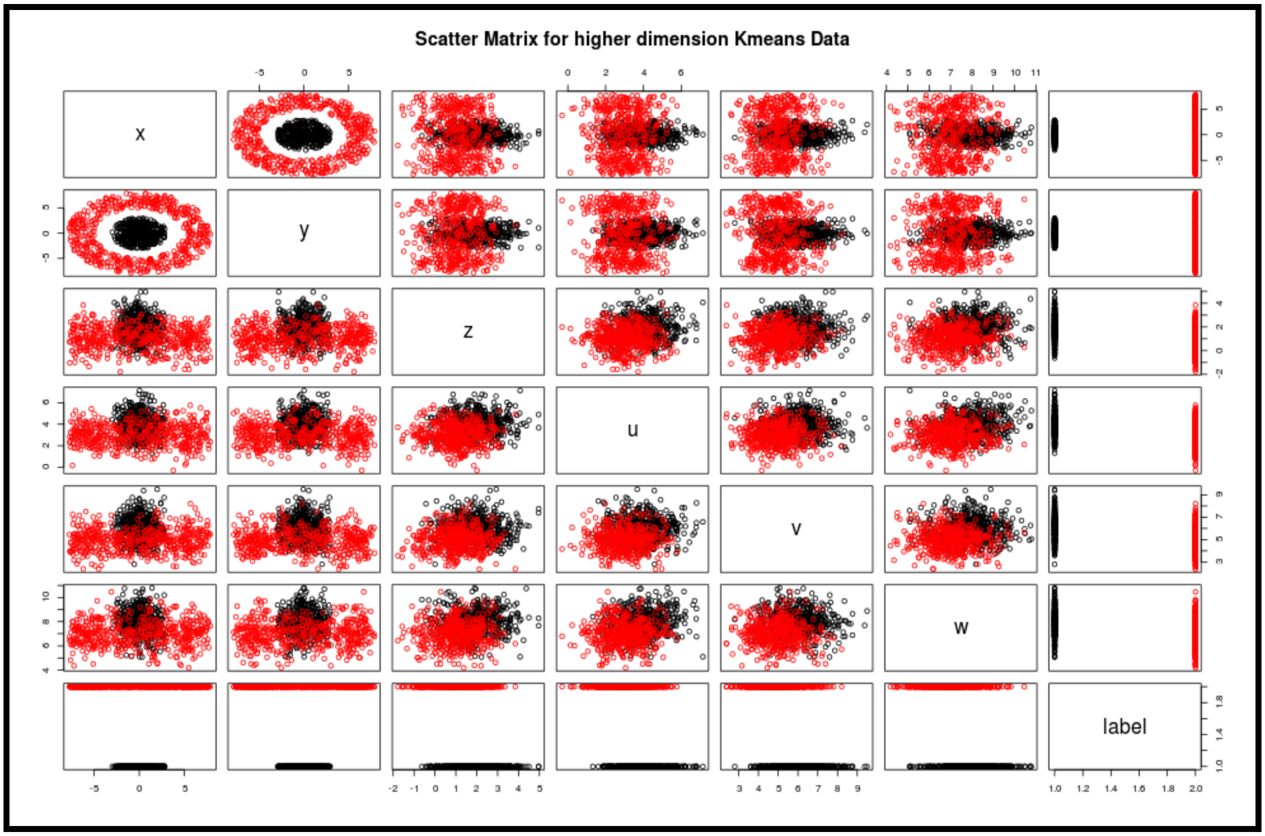
(c) Do you observe the difference in performance when you use different kernels?
Yes, based on the kernel methods, the performance differs a lot. In the above table, we can clearly see that not all kernel methods tend to have accurate prediction, it is highly dependent on the data orientation and distribution in space.

(d) What are the best performance results do you get by playing with different kernels and kernel parameters? Also, make sure to report the number of support vectors for the SVM (the good rule of thumb is to strive for no more than 35%-50% support vectors to avoid model overfitting.

Best performance is achieved by RBF Kernel SVM with accuracy 100%, and minimum number of Support Vectors. Support Vectors : **44**
Hyperbolic tangent SVM is having 56% accuracy, which is less than linear SVM in this case. For this kernel SVM, the kernel parameter “scale” is tuned to 3.3 in order to obtain support vectors within 35-50% of the total data in training model. Support Vectors : **318**

Exercise 4: *Pipelining*. Dimension reduction is often used as the key data preprocessing step to other data mining techniques downstream of end-to-end data analysis. In this exercise we will use unsupervised kernel PCA as a preprocessing step to clustering. Later in the course, we will use *supervised dimension reduction methods* as a preprocessor to the supervised classification methods.

(a) Generalize your BAD_kmeans data set to very high-dimensional space (d>>2).



Dimension increased , d=6

(b) Show that the kmeans clustering method does not perform well on that data.
The below metrics will show that the kmeans in this higher dimensional data will equally perform bad.

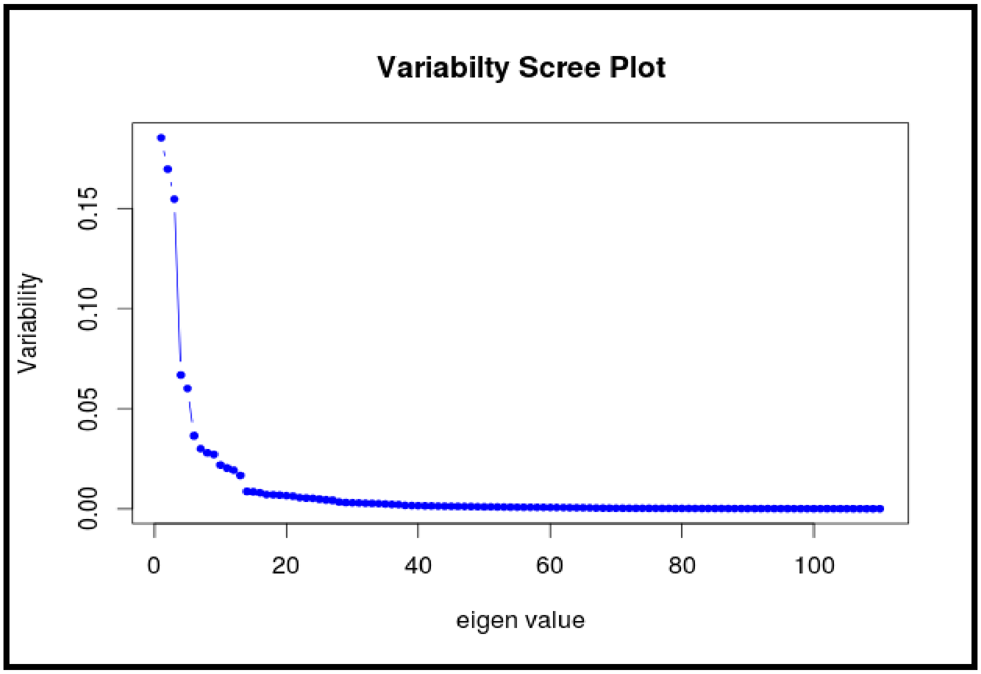
confusion Matrix

	cl1	cl2
GT1	363	137
GT2	267	233

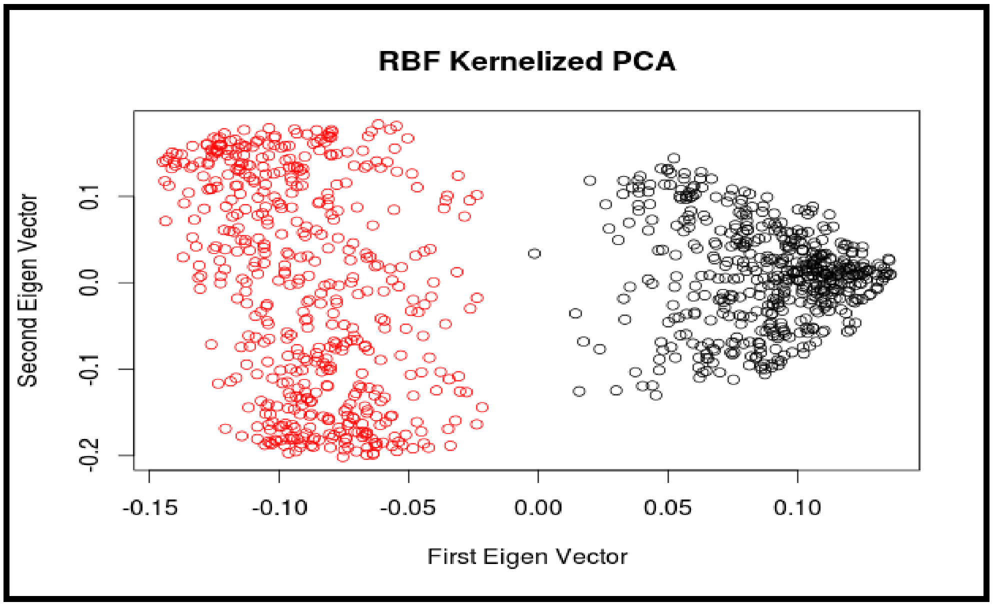
Accuray	**0.596**
Error Rate	0.404
True Positive Rate	0.726
True Negative Rate	0.466
False Positive Rate	0.534
False Negative Rate	0.274
Precision+	0.5761905
Precision-	0.6297297
F-measure+	0.6424779
F-measure-	0.5356322
G-Mean	0.5816494

(c) Apply the kernel PCA method to this high dimensional data and identify the number (*m*<<*d*) of principal components that provide a reasonably good low-dimensional approximation to your data.How much total variability of the data will be preserved upon using this low-dimensional representation?

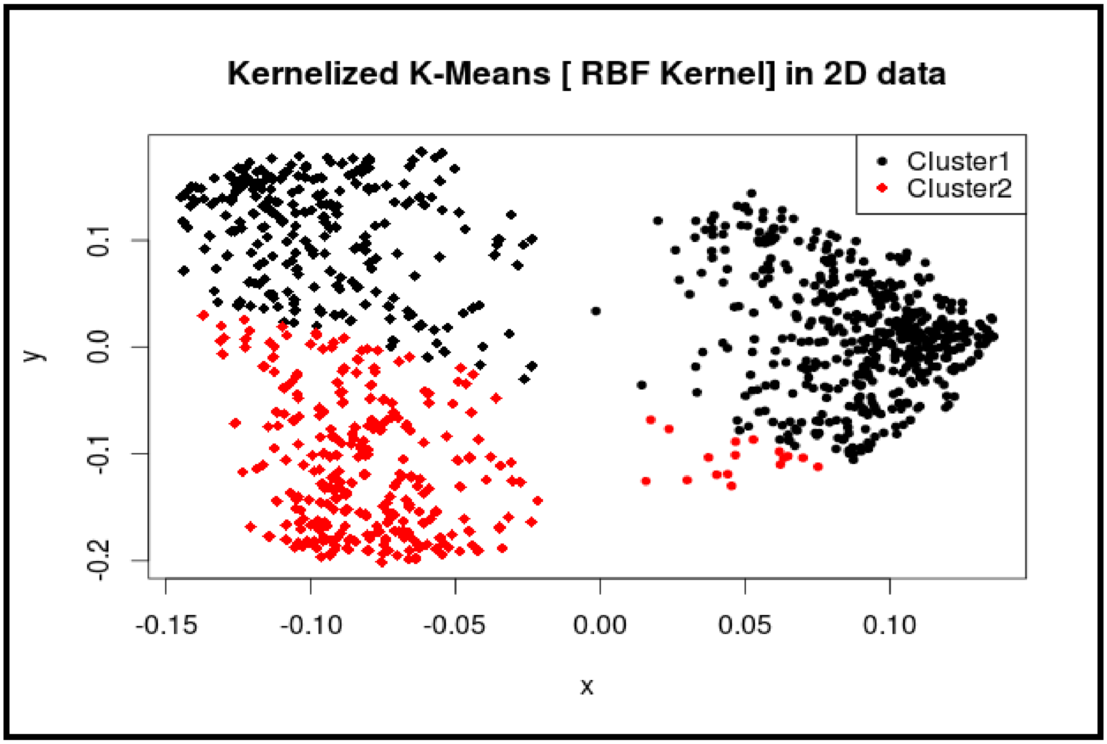
After applying the RBF kernel PCA, the top two eigenvectors captures 35.48% of variability of the entire dataset. Hence m =2.



(d) Project your original data onto the top *m* eigenvectors corresponding the largest eigenvalues. For 1st and 2nd highest eigen value



(e) Run the kmeans clustering algorithm on the projected low dimensional data.



(f) Compare the performance of the kmeans on *d*-dimensional original data vs. the *m*-dimensional projected data. Has the performance improved?

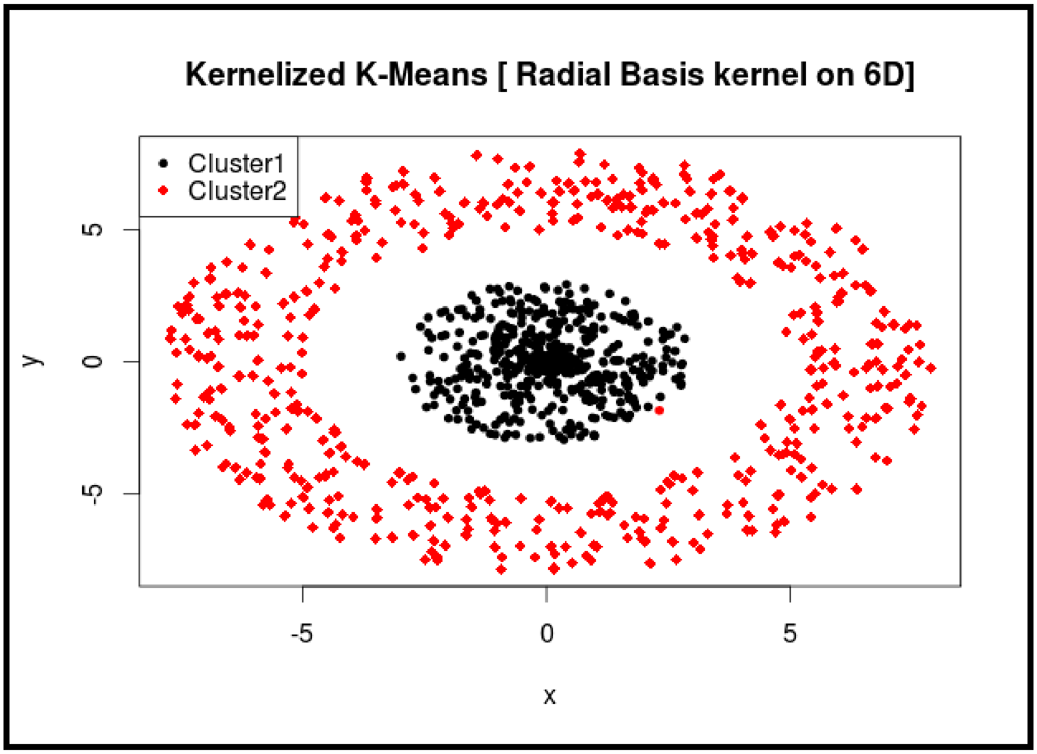
Below is the performance comparison table.

Performance Metrics	**High Dimension Kmeans**	**RBF Kernelized 2 D kmeans**
Confusion Matrix	cl1 cl2 GT1 363 137 GT2 267 233	cl1 cl2 GT1 483 17 GT2 238 262
Accuracy	**0.596**	**0.745**
Error Rate	0.404	0.255
True Positive Rate	0.726	0.966
True Negative Rate	0.466	0.524
False Positive Rate	0.534	0.476
False Negative Rate	0.274	0.034

Precision+	0.5761905	0.6699029
Precision-	0.6297297	0.9390681
F-measure+	0.6424779	0.7911548
F-measure-	0.5356322	0.6726573
G-Mean	0.5816494	0.7114661

Yes. the performance has improved with the dimensionality reduction and performing kmeans.

(g) If you run the kernel kmeans clustering method on the original data, will get better/worse performance? Can you discuss the pros and cons of using kernel kmeans on the original data directly versus applying the kernel pca as the pre-processing step and then running the kmeans on the low-dimensional data.



We will get the better performance on applying kernel kmeans on the higher dimension data for this dataset using RBF kernel.

Accuracy	0.999
Error Rate	0.001

True Positive Rate	0.998
True Negative Rate	1
False Positive Rate	0
False Negative Rate	0.002
Precision+	1
Precision-	0.998004
F-measure+	0.998999
F-measure-	0.999001
G-Mean	0.9989995

Pros: 1) Using kernelized kmeans in higher dimension, will preserve the variability of data, as no dimensionality reduction is applied in this case.

2) More accuracy can be achieved, observed as per the experiment above.

Cons: 1) Curse of dimensionality, as for higher dimension, kernelized operation will be costly for higher dimension

References :

[1] "Practical Graph Mining with R (CRC Press)." Practical Graph Mining with R (CRC Press). N.p., n.d. Web. 19 Oct. 2014.
[2] "SAS Programming for Data Mining." : SAS Implementation of Kernel PCA. N.p., n.d. Web. 19 Oct. 2014.
[3] Alexandros Karatzoglou, Alex Smola, Kurt Hornik. "Kernlab - An S4 Package for Kernel Methods in R." (n.d.): n. pag. Web.

*************** END OF REPORT ******************************