SVM3

Irene Volpe
r0740784

Exercises

1 Kernel PCA

The Principal component analysis (Figure 1) is a dimension reduction technique. It is a technique for creating new variables which are linear combinations of the original variables. The new variables are referred to as the *principal components* and are selected such that they are uncorrelated with each other.

Furthermore, the *first principal component* accounts for the *maximum variance* in the data, the second principal component accounts for the maximum of the variance not yet explained by the first component, and so on.

The maximum number of new variables that can be created is equal to the number of original variables. These set of new variables are probably not all needed and a selected number of principal components is used. The above steps can be summarized as follows: *1)*The data are mean centered.

2) Decide on whether the data need to be standardized or not.

3) First component is selected in that direction where the observations establish most of the data variability

4)·Second component is selected in that direction that is orthogonal to the first component and that accounts for most of the remaining variance in the data.

5) Procedure continuous until the number of principal components equals the number of variables. The total number of new variables accounts for the same amount of variability as the set of original variables.

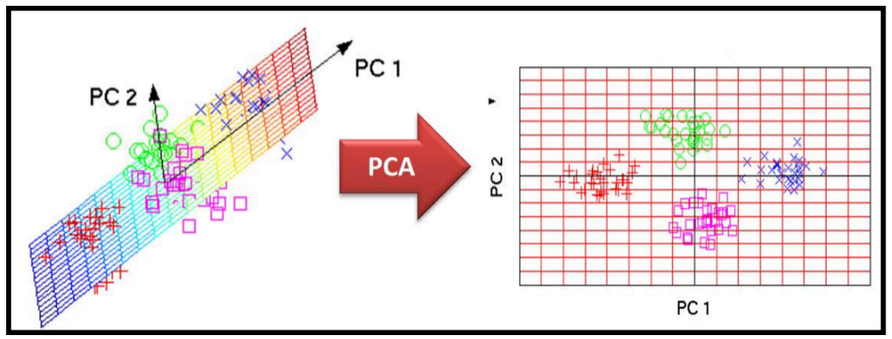


fig 1

1.0 A numerical example: Food Data

Let us consider a numerical example using the food data (FIgure 2)

	City	Bread	Hamburger	Butter	Apples	Tomatoes
1	Anchorage	70.9	135.6	155	63.9	100
2	Atlanta	36.4	111.5	144.3	53.9	95
3	Baltimore	28.9	108.8	151	47.5	104
4	Boston	43.2	119.3	142	41.1	96
5	Buffalo	34.5	109.9	124.8	35.6	75
6	Chicago	37.1	107.5	145.4	65.1	94
7	Cincinnati	37.1	118.1	149.6	45.6	90
8	Cleveland	38.5	107.7	142.7	50.3	83
9	Dallas	35.5	116.8	142.5	62.4	90
10	Detroit	40.8	108.8	140.1	39.7	96
11	Honolulu	50.9	131.7	154.4	65	93
12	Houston	35.1	102.3	150.3	59.3	84
13	Kansas City	35.1	99.8	162.3	42.6	87

fig 2

Figure 3: Use standardized data and obtain the eigenvalues;

To decide the number of principal components to extract we can apply the eigenvalue-greater-than-one rule since we are working with standardized data.With one principal component, we only explain 48% of the total variability (which is too low). The screenplot in figure suggests to take 2 principal components, so we get to explain 67% of the total variability.

Figure 4: Interpretation of principal components

Once we decided how many component to take, we redo the PCA with only 2 components; we therefore ask for correlations between the (PC1, PC2) and (original variables); The higher the correlation, the more influence it has in the formation of the principal component and vice versa.

Traditionally, one use (in absolute value) 0.5 as the cutoff point to decide which variables are influential.

Comp1 can be interpreted as the price index for non-fruit items (i.e. not apples). It is a measurement of the prices of bread, hamburger, butter and tomatoes across the cities.

Comp2 can be interpreted as the price index for fruit (here only apples). We can then visualize the original variables in this two-dimensional plane (with the variables factor map).

```
> names(food2)
[1] "City"      "Bread"     "Hamburger" "Butter"    "Apples"    "Tomatoes"
> result <- PCA(food2[2:6], scale.unit=TRUE)
> # take a look at the eigenvalues
> round(result$eig,2)
       eigenvalue percentage of variance cumulative percentage of variance
comp 1       2.44                 48.79                             48.79
comp 2       0.93                 18.59                             67.38
comp 3       0.83                 16.66                             84.04
comp 4       0.53                 10.66                             94.70
comp 5       0.26                  5.30                            100.00
```

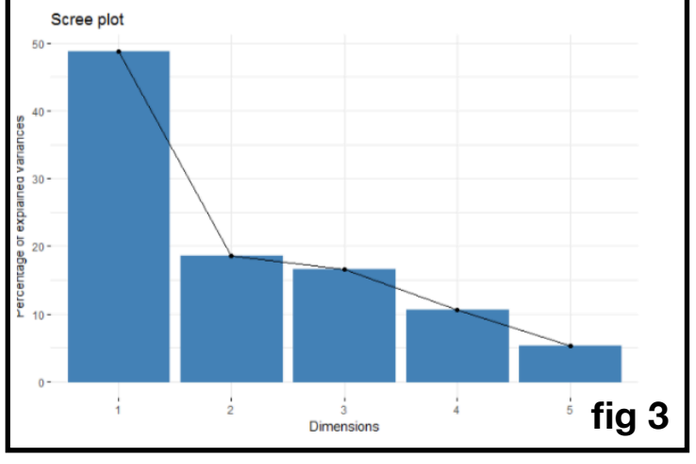


fig 3

```
> result$var$cor
             Dim.1      Dim.2
Bread     0.7964257  0.05447088
Hamburger 0.8122708 -0.26766433
Butter    0.6205371  0.09583809
Apples    0.4544061  0.84532396
Tomatoes  0.7441281 -0.36224623
```

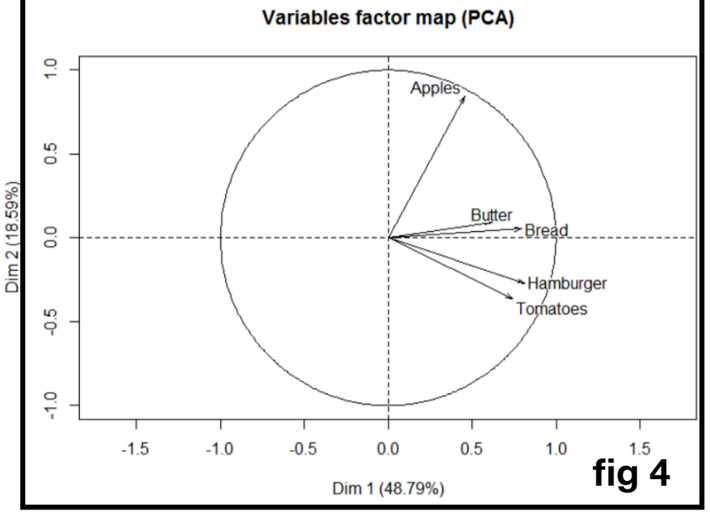


fig 4

The coordinates for the variable used to create the plot are shown in Figure 5.

The coordinate of a variable on a principal component is the correlation between the variable and the principal component. The quality of representation of the variables on a factor map is given by cos2. For a given variable, the sum of the cos2 on all the principal components is equal to one (Figure 6).

When we look at cos2 for the first two dimensions (Figure 7), the sum of squared cos2 for a variable is the communality of the variable in a plane of dim1 and dim2, and represents the quality of the variable in such a plane; then the sum of values for one dimension is the sum up to the first eigenvalue, and is the total variance explained by the first dimension.

Figure 8 shows the contribution of the variables to the variance of the new dimensions. The variables highly correlated with the principal components are the most important in explaining the variability.

Figure 9 shows the relationships between the original variables and the principal components (variable factor plot/ variable correlation plot). In such plot positively correlated variables are grouped together, controversially negative correlated variables are positioned on opposite sides of the plot origin; the length of the variable measures the quality of the variable on the factor map (cos2). Variables with a long vector are well represented.
One can easily deduce that the second principal component is mainly represented by the variable Apples, that the first principal component by the variables Hamburger Tomatoes and Bread, and that the variable Butter is not so well represented in the plot.

Figure 10 shows the principal component scores (these are obtained as the projection of the observation on this two dimensional plane), and Figure 11 visualizes such scores in a plot. One can deduce that Anchorage is a city with high price for all non-fruit items (high on factor 1), Buffalo is a city with low price for all food items, Detroit has low price for factor 2 (apples) and average price for factor 1 (Hamburger Tomato and Bread), and Atlanta (in the ceter) has average price for factor 1 and factor 2 items.

To detect outliers one can ask for contributions (Figure 12) and visualize those in a biplot (Figure 13), ie a visual representation of an (nxp) matrix in 2 dimensions.

```
> result$var$coord
                Dim.1      Dim.2
Bread      0.7964257 0.05447088
Hamburger  0.8122708 -0.26766433
Butter     0.6205371 0.09583809
Apples     0.4544061 0.84532396
Tomatoes   0.7441281 -0.36224623
```

fig 5

```
> round(result$var$cos2, 3)
           Dim.1 Dim.2 Dim.3 Dim.4 Dim.5
Bread      0.634 0.003 0.134 0.151 0.077
Hamburger  0.660 0.072 0.138 0.003 0.128
Butter     0.385 0.009 0.492 0.099 0.015
Apples     0.206 0.715 0.004 0.074 0.001
Tomatoes   0.554 0.131 0.064 0.207 0.044
```

fig 6

```
$cos2
                Dim.1       Dim.2
Bread      0.6342939 0.002967077
Hamburger  0.6597838 0.071644191
Butter     0.3850662 0.009184940
Apples     0.2064849 0.714572603
Tomatoes   0.5537266 0.131222329
```

fig 7

```
$contrib
                Dim.1      Dim.2
Bread      26.002519  0.3191808
Hamburger  27.047464  7.7070648
Butter     15.785573  0.9880623
Apples      8.464734 76.8695583
Tomatoes   22.699710 14.1161338
```

fig 8

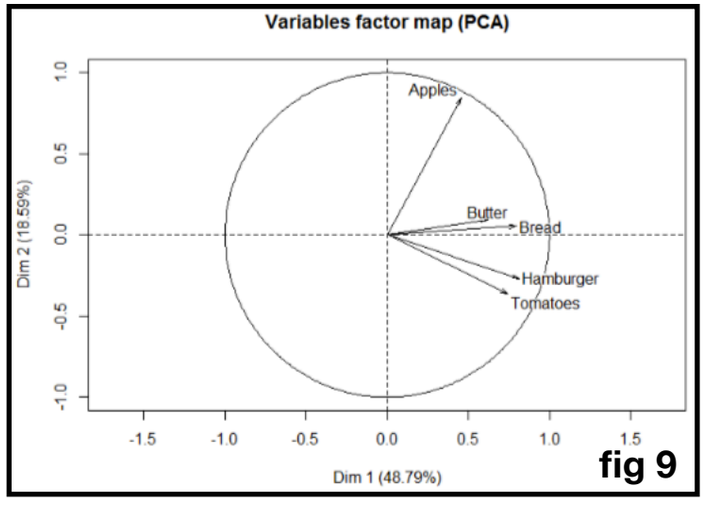


fig 9

```
> head(round(result$ind$coord,3))
           Dim.1  Dim.2
Anchorage  4.675  0.540
Atlanta    0.324 -0.082
Baltimore  0.366 -1.126
Boston     0.586 -1.653
Buffalo   -2.692 -1.194
Chicago    0.519  1.321
```

fig 10

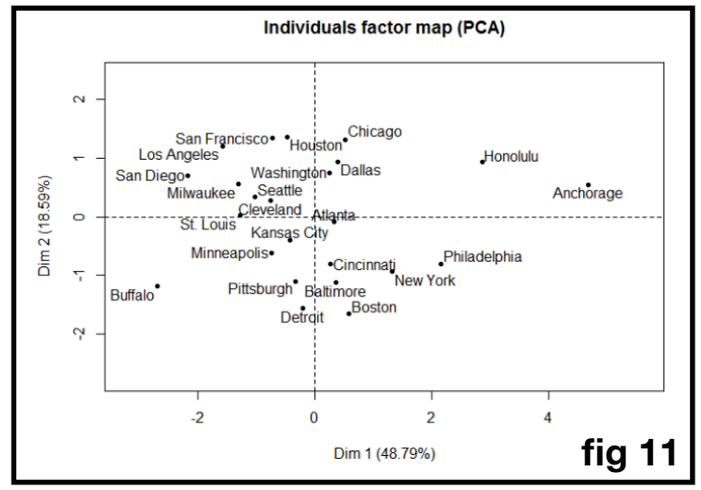


fig 11

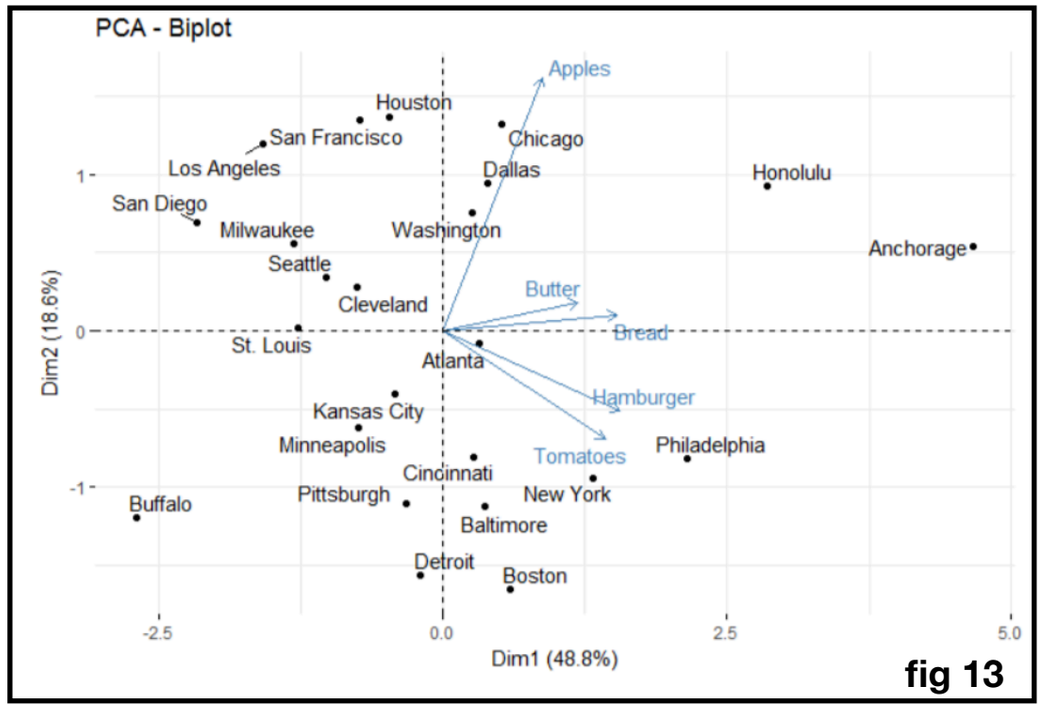


fig 13

```
> head(cbind(city, result$ind$contrib))
  city        Dim.1                Dim.2
1 "Anchorage" "37.3241441866544"  "1.3068880480434"
2 "Atlanta"   "0.178796037269315" "0.0300117766716827"
3 "Baltimore" "0.228993344915314" "5.68119748705364"
4 "Boston"    "0.587168985563456" "12.2463651909984"
5 "Buffalo"   "12.3763382897188"  "6.38792253129751"
6 "Chicago"   "0.460482127566553" "7.81938566055003"
```

fig 12

1.1 Denoising PCA

By reducing the dimension of data, we reduce the noise or the unwanted parts of the data. Dimensionality reduction can be done either through feature selection or feature mapping. Like the non-linear SVM Kernel PCA also use feature maps for dimensionality reduction i.e. the model selects the best and most relevant features itself. Often the data is structured such that the variance in the dimensions does not vary much. In such a dataset a linear PCA cannot eliminate most of the eigenvectors in order to keep the reconstruction error at a minimum. Therefore, by incorporating a feature map into the linear PCA, a projection of the original data into a higher dimensional space can be performed where just a few of the new dimensions would end up having most of the variance. Then applying linear PCA on this mapped dataset gives the desired results. This does, however, mean that the final dimensionality of the projected data is now no longer restricted to be smaller than or equal to the original dataset.

In case of linear PCA the first step dataset is to standardize or zero-center the data, which is done by subtracting the mean from the dataset.

A covariance matrix of the standardized data is made, which is then used to compute the eigenvalues and eigenvectors.

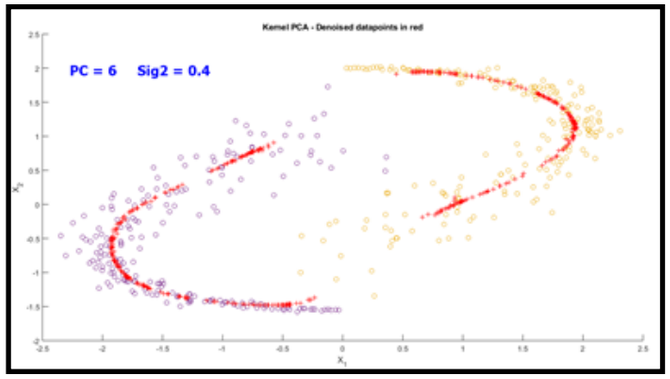
Eigenvectors - The eigenvectors are used to represent the data in the lower dimensionality. The greater the number of eigenvectors that are chosen, the lesser the reconstruct error will be. However, we want to keep in mind that one whole goal of applying this operation is dimensionality reduction, so an optimal number of eigenvectors is to be chosen that correctly represents the greater part of the original data.

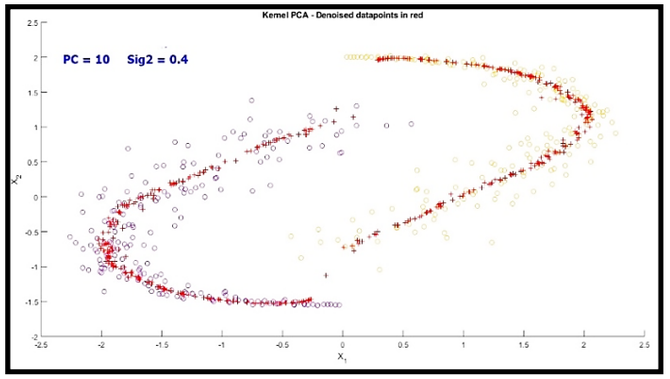
Eigenvalues - The eigenvectors with higher eigenvalues represent the maximal variance of the original data and lower eigenvalues represent the noise in the dataset. Keeping only the first few highest eigenvalues and their corresponding eigenvectors, we can effectively reduce the data dimensions and eliminate noise from out data. This results in the dimensionality of the projected data to be always lower than or, in the worst-case scenario, at least equal to the original data.

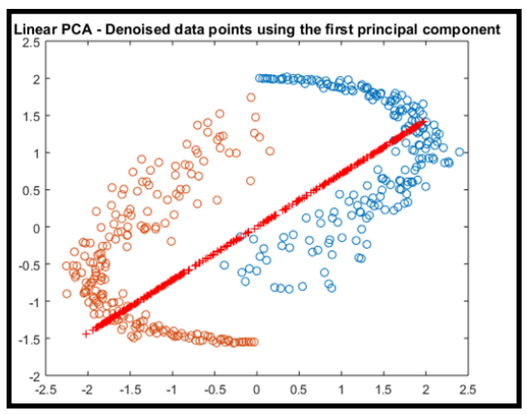
1.2 Increasing number of PC

As the kernel PCA uses feature maps its performance is influenced by the number of components selected and the various kernel hyperparameters. Figure 1 shows results of increasing number of principle components and changing the sig^2 hyperparameter.

By decreasing the number of principle components, only the components with very high variance are selected and the dimensionality reduction becomes much more evident. This way we can maintain the representation of the original data. Similarly, increasing the number of components increases the vectors in the projected data to a point where the projected points exactly represent the datapoints in the original dataset. A high value of sig2 causes the projected point to spread in the overall direction of the dataset distribution.

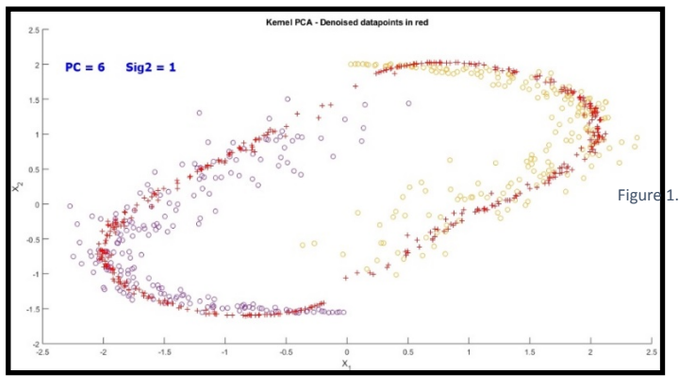






Linear PCA on the dataset

fig 14



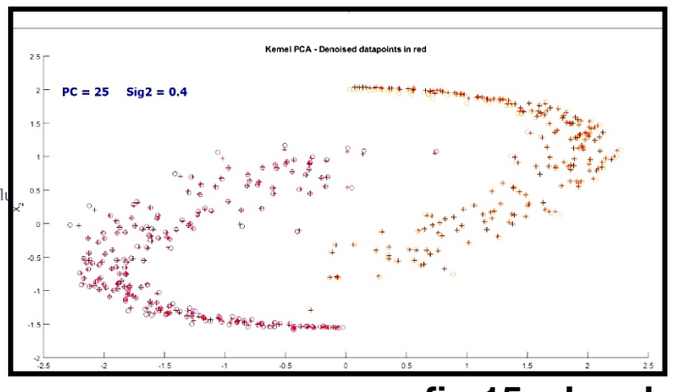


Figure 1. Influ

fig 15 a,b,c,d

1.3 Difference between PCA and Kernel PCA

Standard PCA finds linear principal components to represent only linearly separable data in lower dimension. Figure 2 shows that it fails to separate datapoint classes when used on our dataset. Kernel PCA (KPCA) can be used to overcome this limitation by using kernel trick to find principal components in different space and possibly a high dimensional space.

PCA finds new directions based on covariance matrix of original variables. It can extract maximum P (number of features) eigen values. KPCA finds new directions based on kernel matrix. It can extract n (number of observations) eigenvalues.

PCA allow us to reconstruct pre-image using few eigenvectors from total P eigenvectors. It may not be possible in KPCA.

The computational complexity for KPCA to extract principal components takes more time compared to Standard PCA.

1.4 Number of PC we can obtain

Number of components obtained in PCA, is equal to the number of variables being analyzed. For example, analyzing data of customers with 5-variables (age, gender, country, state, profession) would result in five components. The first component can be expected to account for a fairly large amount of the total variance. Each succeeding component will account for progressively smaller amounts of variance. Although many components may be extracted in this way, only the first few components will be important enough to be retained for interpretation (such as in multiple regression analyses).

The next step then in PCA analysis, is to determine out of the obtained components how many meaningful components should be retained for interpretation. This depends on following criteria:

i. Eigenvalue> 1 are chosen and automatically sorted from the highest eigenvalue, i.e. highest explanatory power, to the one with the lowest (Figure 3).

ii. Amount of variance between variables: should have >=70- 80% of variance.

iii. Scree plot /Figure 4): With the scree test, we can plot the eigenvalues associated with each component and look for a "break" between the components with relatively large eigenvalues and those with small eigenvalues. The components that appear before the break are assumed to be meaningful and are retained for rotation; those appearing after the break are assumed to be unimportant and are not retained. Some examples are given below. In first graph without break, components 1-2 will be considered more important to be retained, but in this graph but it is difficult to decide exactly where to draw the line. In Broken stick model to retain number of PCA, the number of components to retain is computed as the largest integer k for which the first k components each explain more variance than the broken-stick model (null model).

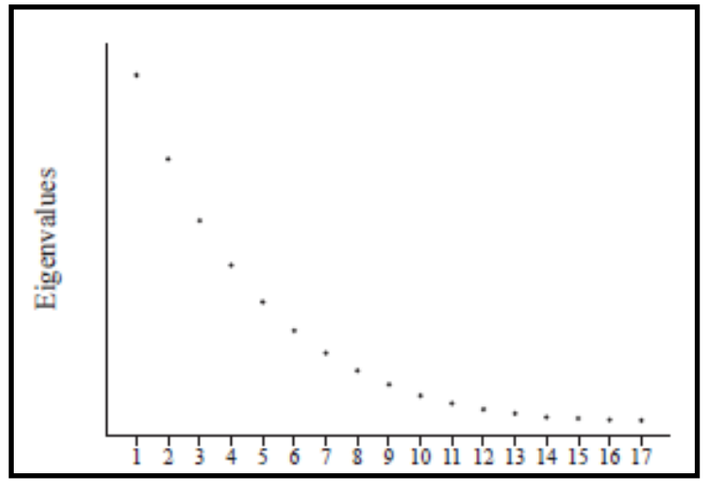


fig 16

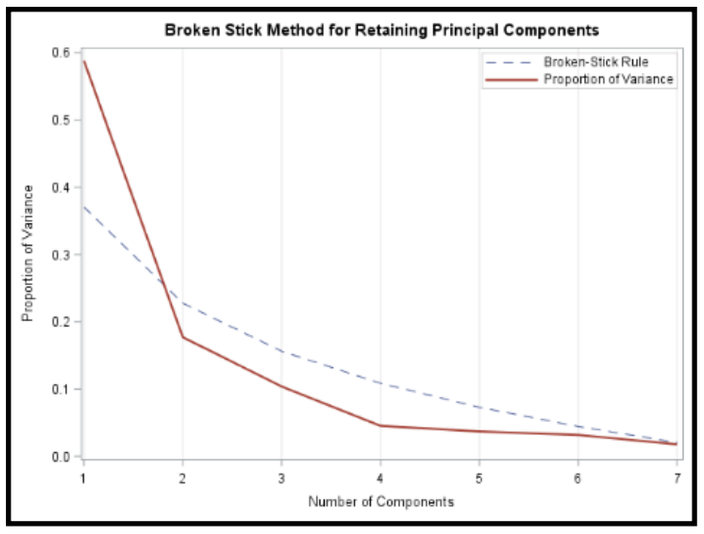


fig 17

1.5 Technique to tune number of components, hyperparameter and kernel parameters for given dataset

Tuning number of principle components and hyperparameter Sig^2 for kernel PCA can be done by selecting a set of initial values for the two parameters and then increasing the values and checking their reconstruction errors. These will reach a point where the values would increase to an optimal combination where the reconstruction error is at a minimum, and the projected points fully represent the original dataset. Kernel/regularisation parameters can be automated by optimizing a cross-validation based model selection. The simplest thing to do is to minimise a continuous model selection criterion using the Nelder-Mead simplex method, which doesn't require gradient calculation and works well for sensible numbers of hyper-parameters.

2 Spectral clustering

2.1 How Spectral clustering works

Spectral clustering involves 3 steps:

1. Compute a similarity graph

2. Project the data onto a low-dimensional space

3. Create clusters

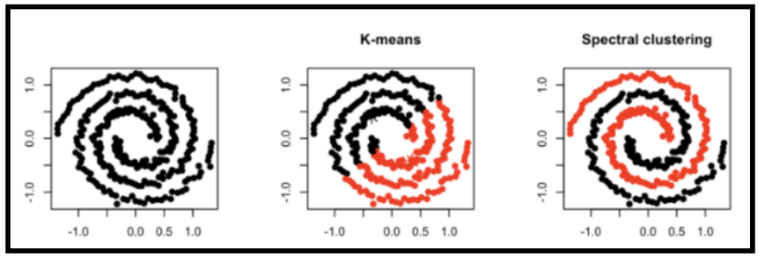


fig 18

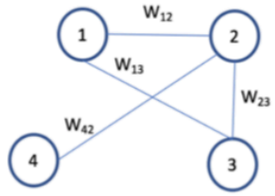
There are 2 broad approaches for clustering:

1. Compactness — Points that lie close to each other fall in the same cluster and are compact around the cluster center. The closeness can be measured by the distance between the observations. E.g.: K-Means Clustering

2. Connectivity — Points that are connected or immediately next to each other are put in the same cluster. Even if the distance between 2 points is less, if they are not connected, they are not clustered together. Spectral clustering is a technique that follows this approach. The difference between the 2 can easily be shown in Figure 5.

In spectral clustering, the data points are treated as nodes of a graph. Thus, clustering is treated as a graph partitioning problem. The nodes are then mapped to a low-dimensional space that can be easily segregated to form clusters. An important point to note is that no assumption is made about the shape/form of the clusters.

Step 1 — Compute a similarity graph:
We first create an undirected graph G = (V, E) with vertex set V = {v1, v2, …, vn} = 1, 2, …, n observations in the data. This can be represented by an adjacency matrix which has the similarity between each vertex as its elements .Thus, when we create an adjacency matrix for any of these graphs, Aij ~ 1 when the points are close and Aij → 0 if the points are far apart.
Consider a graph with nodes 1 to 4, weights (or similarity) wij and its adjacency matrix as illustrated in Figure 19.



	1	2	3	4
1	0	w_{12}	w_{13}	0
2	w_{12}	0	w_{23}	w_{24}
3	w_{13}	w_{23}	0	0
4	0	w_{24}	0	0

L: Graph, R: n x n symmetric adjacency matrix

fig 19

Step 2 — Project the data onto a low-dimensional space:
As we can see in Figure 19, data points in the same cluster may also be far away–even farther away than points in different clusters. Our goal then is to transform the space so that when the 2 points are close, they are always in same cluster, and when they are far apart, they are in different clusters. We need to project our observations into a low-dimensional space. For this, we compute the Graph Laplacian (Figure 20), which is just another matrix representation of a graph and can be useful in finding interesting properties of a graph.

The whole purpose of computing the Graph Laplacian L was to find eigenvalues and eigenvectors for it, in order to embed the data points into a low-dimensional space. So now, we can go ahead and find eigenvalues. We know that:

$$L\lambda = \lambda v$$

where v is the eigenvector of L corresponding to eigenvalue λ.
Thus we get eigenvalues $\{\lambda_1, \lambda_2, \dots, \lambda_n\}$ where $0 = \lambda_1 \leq \lambda_2 \leq \dots \leq \lambda_n$ and eigenvectors $\{v_1, v_2, \dots, v_n\}$.

1. If L has eigenvalue 0 with k different eigenvectors, such that $0 = \lambda_1 = \lambda_2 = \dots = \lambda_k$, graph G has k connected components.

2. If G is connected, i.e. $0=\lambda_1$ and $\lambda_2>0$, λ_2 is the algebraic connectivity of G. Thus, greater λ_2, greater the connectivity.

Spectrum of the Laplacian
$0 = \lambda_1 \leq \lambda_2 \leq \dots \leq \lambda_n$

Let us consider a numerical example as shown in Figure 21; We then compute eigenvalues and eigenvectors for L.

$$L = D - A,$$
where A is the adjacency matrix and D is the degree matrix such that

$$d_i = \sum_{\{j|(i,j)\epsilon E\}} w_{ij} \qquad Thus, L_{ij} = \begin{cases} d_i & if\ i = j \\ -w_{ij} & if\ (i,j) \epsilon E \\ 0 & if\ (i,j) \notin E \end{cases}$$

Computing Graph Laplacian

$d_1 = w_{12} + w_{13}$
$d_2 = w_{12} + w_{23} + w_{24}$
$d_3 = w_{12} + w_{23}$
$d_4 = w_{24}$

$L =$

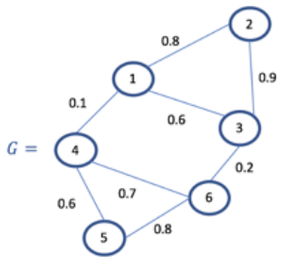
	1	2	3	4
1	d_1	$-w_{12}$	$-w_{13}$	0
2	$-w_{12}$	d_2	$-w_{23}$	$-w_{24}$
3	$-w_{13}$	$-w_{23}$	d_3	0
4	0	$-w_{24}$	0	d_4

Graph Laplacian for our example above

fig 20

Step 3: Create clusters (Figure 22):
For this step, we use the eigenvector corresponding to the 2nd eigenvalue to assign values to each node. On calculating, the 2nd eigenvalue is 0.189 and the corresponding eigenvector v2 = [0.41, 0.44, 0.37, -0.4, -0.45, -0.37].
To get bipartite clustering (2 distinct clusters), we first assign each element of v2 to the nodes such that {node1:0.41 , node2:0.44 , … node6: -0.37}. We then split the nodes such that all nodes with value > 0 are in one cluster, and all other nodes are in the other cluster. Thus, in this case, we get nodes 1, 2 & 3 in one cluster, and 4, 5 & 6 in the 2nd cluster.
It is important to note that the 2nd eigenvalue indicates how tightly connected the nodes are in the graph. For good, clean partitioning, lower the 2nd eigenvalue, better the clusters. For k clusters, we have to modify our Laplacian to normalize it as shown in Figure 23.

To summarize, we compute the laplacian matrix for every node or point in the graph. This gives us corresponding coordinates of smallest eigenvectors, so that every node in graph is represented by a set of coordinates. Then run k-means on these points to identify k clusters.

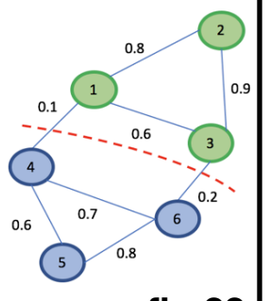


$G =$ $L =$

	1	2	3	4	5	6
1	1.5	-0.8	-0.6	-0.1	0	0
2	-0.8	1.7	-0.9	0	0	0
3	-0.6	-0.9	1.7	0	0	-0.2
4	-0.1	0	0	1.4	-0.6	-0.7
5	0	0	0	-0.6	1.4	-0.8
6	0	0	-0.2	-0.7	-0.8	1.7

fig 21

	v2
1	0.41
2	0.44
3	0.37
4	-0.40
5	-0.45
6	-0.37



Eigenvector v2 gives us bipartite clustering.

fig 22

1. For k clusters, compute the first k eigenvectors $\{v_1, v_2, \dots, v_k\}$.

2. Stack the vectors vertically to form a matrix with the vectors as columns.

3. Represent every node by the corresponding row of this new matrix. These rows form the feature vectors of the nodes.

4. Use K-Means Clustering to now cluster these points into k clusters $\{C_1, C_2, \dots, C_k\}$.

$L_{norm} = D^{-1/2}LD^{-1/2} =$

	1	2	3	4	5	6
1	1.0	-0.5	-0.4	-0.1	0	0
2	-0.5	1.0	-0.5	0	0	0
3	-0.4	-0.5	1.0	0	0	-0.1
4	-0.1	0	0	1.0	-0.4	-0.5
5	0	0	0	-0.4	1.0	-0.5
6	0	0	-0.1	-0.5	-0.5	1.0

Normalized Laplacian — Ng, Jordan, Weiss

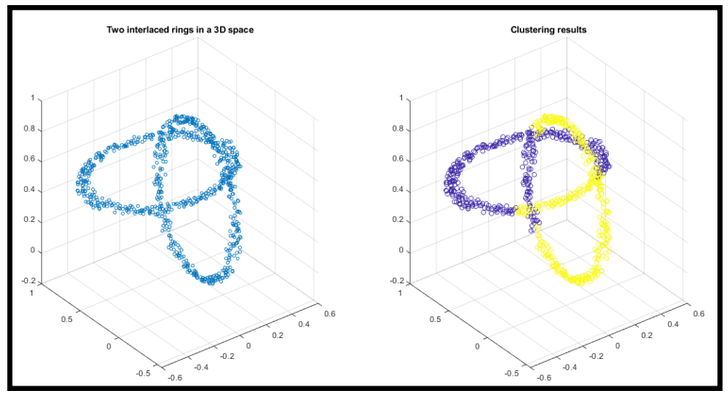
fig 23

2.2 Differences between Spectral clustering and classification

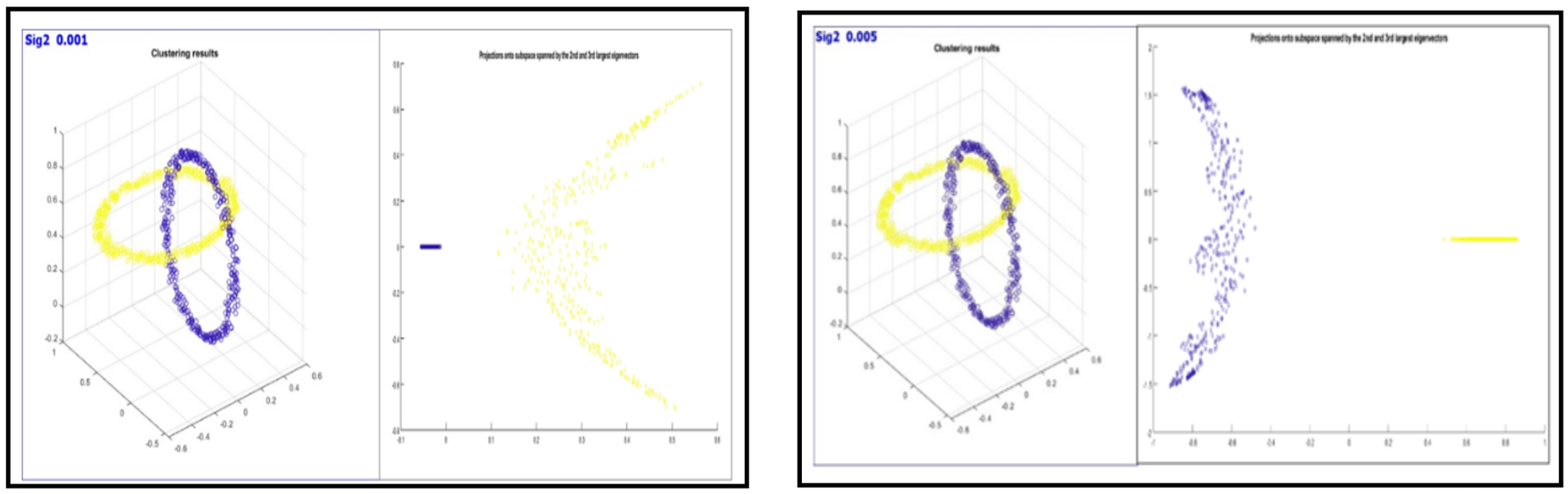
Spectral clustering is an unsupervised learning technique and does not require labels for clustering datapoints. The grouping is such that points that are connected or immediately next to each other are put in the same cluster. Even if the distance between 2 points is less, if they are not connected, they are not clustered together. Classification on the other hand, is supervised learning and requires a predefined set of labels to train the model to predict the category datapoint belongs to.

2.3 Influence of the sig2 on clustering results

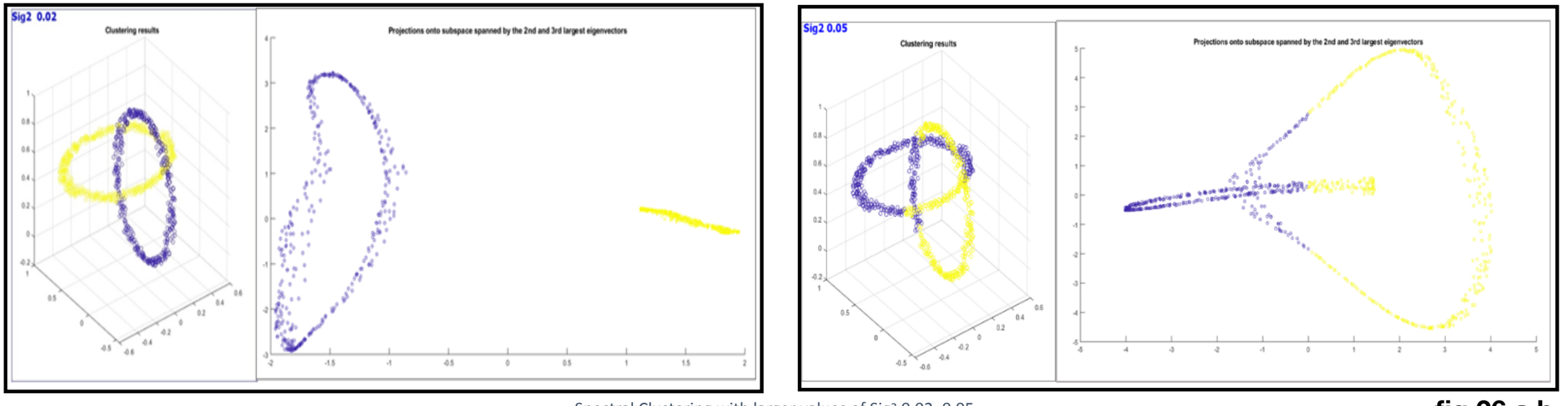
Figure 24 shows results of spectral clustering using sig^2 0.05. Most of the datapoints in two rings i.e. the two clusters, are separated correctly with a small overlap (error). Using different sig^2 0.001, 0.005, 0.01 and 0.02, shows (Figure.25) that the smaller sig^2, the more accurate partition of datapoints is into clusters. As sig^2 increases there comes a point that the projections of clusters get very close and some datapoints are put in wrong cluster (Figure.26). Sig^2 is the measurement of similarity in data based on distances or affinity between points. If all points of one cluster of are separated from all points of another by a distance that is significantly higher than given value of Sig^2, then the spectral clustering use this as a cut.
Small sigma values lead to very low eigen values which tend to make very small clusters containing outliers, and a very big cluster with all other points.



Spectral Clustering results with sig^2 0.05 **fig 24**



Spectral Clustering with smaller values of sig^2 (0.001, 0.005) **fig 25 a,b**



Spectral Clustering with larger values of Sig^2 0.02, 0.05 **fig 26 a,b**

3 Fixed-size LS-SVM

So far we have used the LS-SVM by providing it with a training set and getting a set of α's and a b through the training procedure. Unlike standard SVMs, the LS-SVMs lack sparsity such that the training set, and also the set of α's and b have to be retained for the classification phase. This may not seem like much of a problem in smaller datasets, however, when dealing with larger datasets this could lead to massive problems in terms of memory usage.

To overcome this limitation, methods like the Nystrom method are employed. This method aims to approximate the kernel matrix for the whole dataset by using only a subset of its entries. This way only a small fraction of the whole dataset has to be retained, and this new approximate of the kernel matrix can be used during the training and classification phases.

3.1 Primal vs Dual Solution

The specification of the estimation problem at the primal level is done by formulating a constrained optimization problem, where the model is expressed in terms of a feature map. The optimal model representation is obtained together with the solution from the conditions for optimality. At the dual level (problem in the Lagrange multipliers) the model is expressed in terms of a positive definite kernel function. For given tuning parameters the problem is convex. Through the choice of an appropriate loss function one obtains a sparse representation in SVMs.

In case the feature map is finite dimensional and explicitly known one has the choice between solving the primal or the dual problem (for the Gaussian kernel on the other hand one can only solve the dual). Consider e.g. the case of a linear parametric regression model $\hat{y} = w^T x + b$ with $w \in R^d$. The dual representation of the linear model is $\hat{y} = \sum^N{}_{i=1} \alpha_i x_i^T X + b$ with $\alpha \in R^N$. One distinguishes between the following cases then:

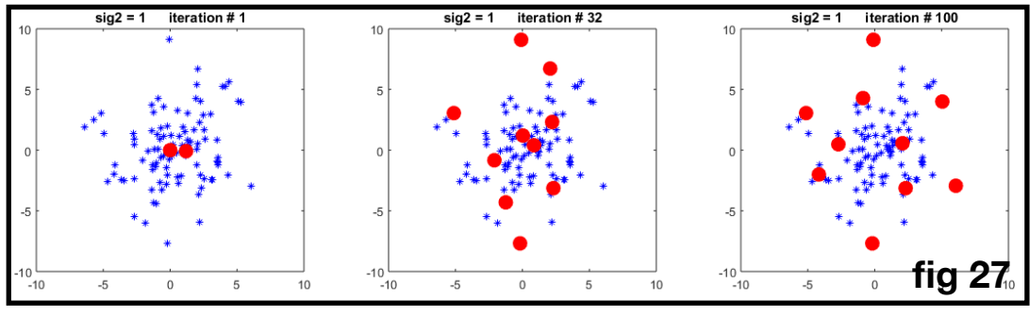
- Case d small, N large: solving the primal problem in $w \in R$ d is more convenient.
- Case d large, N small: solving the dual problem in $\alpha \in R$ N is more convenient.

For Fixed size LSSVM: regularization terms are considered from the beginning in the primal formulations. These models are also easier to extend to a wider class of problems in supervised and unsupervised learning than standard SVMs.

3.2 Effect of sig2 on the resulting fixed-size subset of datapoints

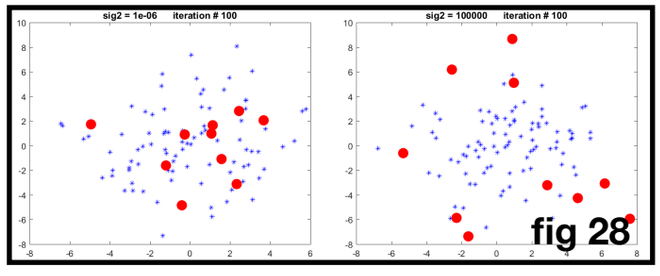
The results from the Nystrom method can be highly influenced by the choice of the vectors to be used to construct the kernel approximation. The example script used in this experiment has two distinct operations. The first is the addition of new vectors to the optimized subset, and the second is the replacement or substitution of the vectors in the optimized subset such that the entropy is maximized. The script also plots the selected vectors among the whole dataset for each iteration.

Figure 27 shows the results at different intervals of this selection process. The selected subset of vectors are represented by the big red dots. At the first iteration, the algorithm selects 2 vectors as a part of its optimized subset. By the 32nd iteration all 10 optimal vectors for our subset are found. However, there's still many more iterations to go. So in the upcoming iterations a series of replacements is made from the selected vectors until the final iteration where we now have our final optimal subset of vectors.



Selection procedure results plotted at different intervals.

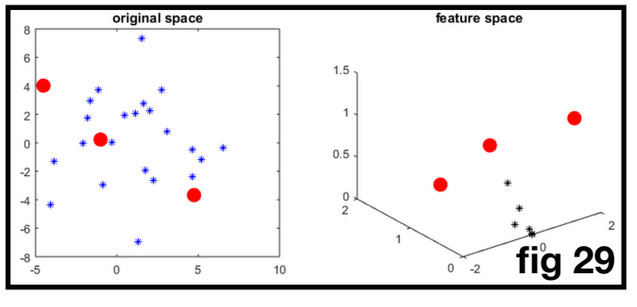
By changing the sig2 value to a much smaller value we can see that the algorithm selects the 10 initial optimal vectors sooner and the proceeding iterations have little substitutions made, resulting in a denser subset. On the other hand, very large Sig2 will cause more substitutions to me made than the additions. Sometimes the total number of iterations (that are set to 100 right now) are not enough to fully populate the optimal 10 vectors in the subset due to the slow growth rate. Overall, this results in a slower expansion of the subset but a higher sparsity for the subset. Figure 28 shows how the small sig^2 value creates a denser subset, while the larger sig^2 value outputs a sparser subset.



Result of the final iteration with sig2 at 0.000001 and 100000

3.3 Subset to which the algorithm converge

Referring to the selected optimal subset of vectors (section 1.3.1) to represent the kernel, we now see how the algorithm converges to achieve maximum variance when mapped onto the feature space. The results in the previous step show a direct relation between the entropy and variance in the data, thus it can be concluded that the vectors at farther sides have the highest entropy. Figure 29 shows a set of selected vectors in the original space mapped to the feature space. The algorithm converges to a set of vectors with maximum variance when mapped onto the feature space.



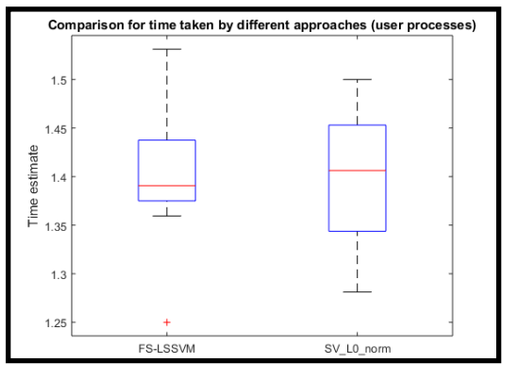
Reconstruction of the feature space map

3.3 Achieving sparser solutions results of Fixed-size LS-SVM Vs `approximation

We used the Nystrom method to approximate the kernel matrix using a fixed subset of its vectors. This approach greatly reduces computational expense, albeit at the cost of losing some accuracy. To overcome this, we further suppress the number of support vectors being used and consequently reduce the computational cost even more by using L_0 norm.

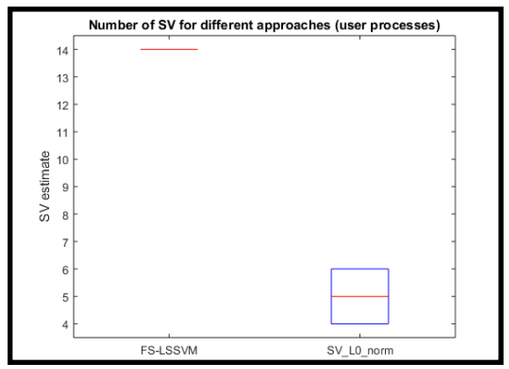
To do this we randomize the data 10 times and select support vectors from that shuffled data. The SVM is then trained on these support vectors and the model is tested on a predefined test data. We compare these results from the fixed-size LS-SVM on the same dataset in respects of testing error, number of support vectors, and the time taken.

Figure 30 shows a comparison of time taken by the two approaches. The mean time taken by FS-LSSVM is slightly lower than the L_0 norm, but the overall difference between the two is very low.



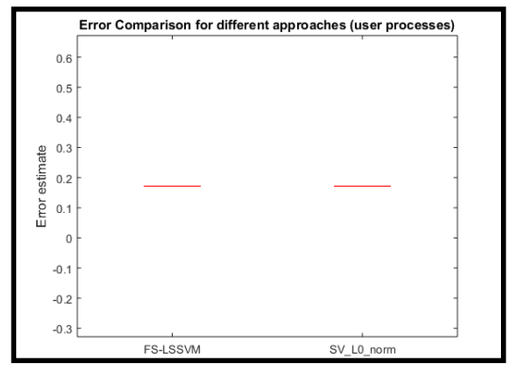
Comparison of Time taken by FS-LSSVM and SV-L_0-Norm **fig 30**

Figure 31 shows a comparison of the number of support vectors that are selected by the two approaches. FS-LSSVM takes 14 number of support vectors for this example while the L_0 norm only takes about 5.



Comparison of number of Support Vectors selected by FS-LSSVM and SV-L0-Norm **fig 31**

Figure 32 shows the error comparison for the two approaches. Both FS-LSSVM and L_0 norm have the exact same error of 0.2.



Comparison of Error estimate of FS-LSSVM and SV-L0-Norm **fig 32**

Considering all these comparisons, we can conclude that L0 norm performs much better than FS-LSSVM, because it uses a significantly lesser number of support vectors, all the while maintaining the same error rate and almost equal computation time as the FS-LSSVM. Thus, L0 norm achieves sparser solutions.

Homework Problems

1 Handwritten Digit Denoising

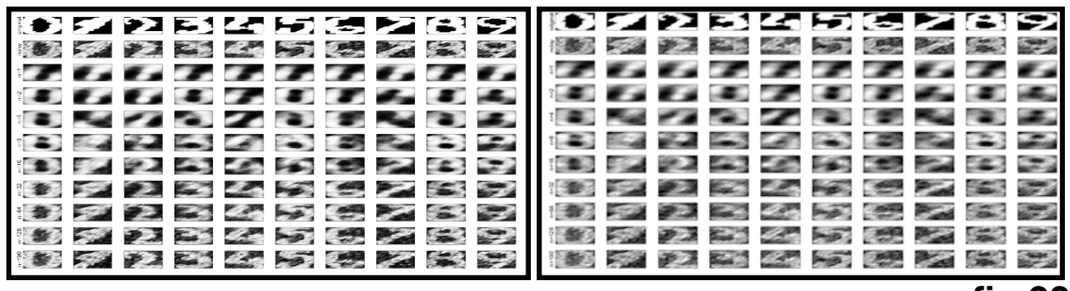
1.1 Effect of denoising factor

To compare performance of kernel PCA and linear PCA, iwe're using a dataset of binary images of handwritten numerals (0-9) extracted from a collection of Dutch utility maps. Gaussian noise with factor 0.3 is added to the dataset and then reconstruction of this data is performed by kernel PCA with different number of principle components (figure 33).

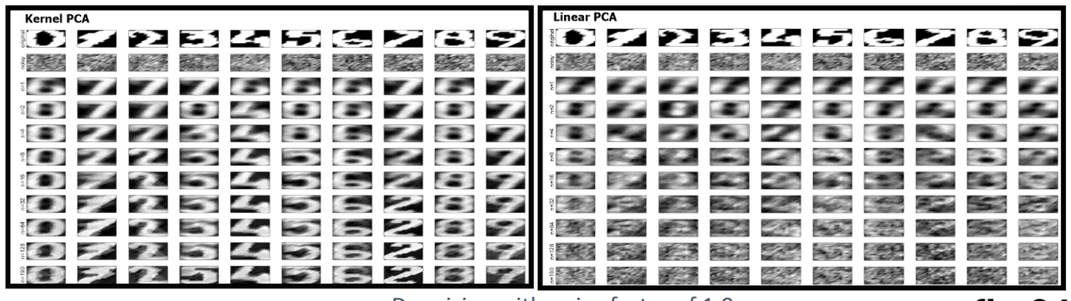
The first row of digits in figure 34 represents the original dataset, while second row shows the noisy dataset of digits.
The subsequent rows represent the reconstruction of these digits with increasing 'n' number of principal components.
With n=1 the reconstruction removes almost all noise, but it is bit blurred.
As the value of n is increased, the reconstruction keeps improving in terms of clearer images.
Improvements stop after 128 principal components.
All digits are correctly except digit 7 and 1: the first is wrongly classified as 2 from n =1, the second is wrongly classified as 9 from n =16 . The overall performance is efficient in terms of removing the noise and projecting the original digits in reduced dimensionality.

The same task is run with linear PCA on the same dataset to compare the two methods.
At n=190, all digits are accurately reconstructed by the linear method (even digit 7 which was reconstructed wrong by KPCA), and increasing the noise factor from 0.3 to 1.0 (figure 14) the the denoising effect of the kernel PCA improves at the expenses of the classification accuracies.

Overall, the kernel PCA outperforms the linear PCA in denoising the reconstructions.
Linear PCA removes some noise with lesser number of principle components but fails to do denoise with higher number.



KPCA vs Linear PCA in reconstructing noisy data **fig 33**



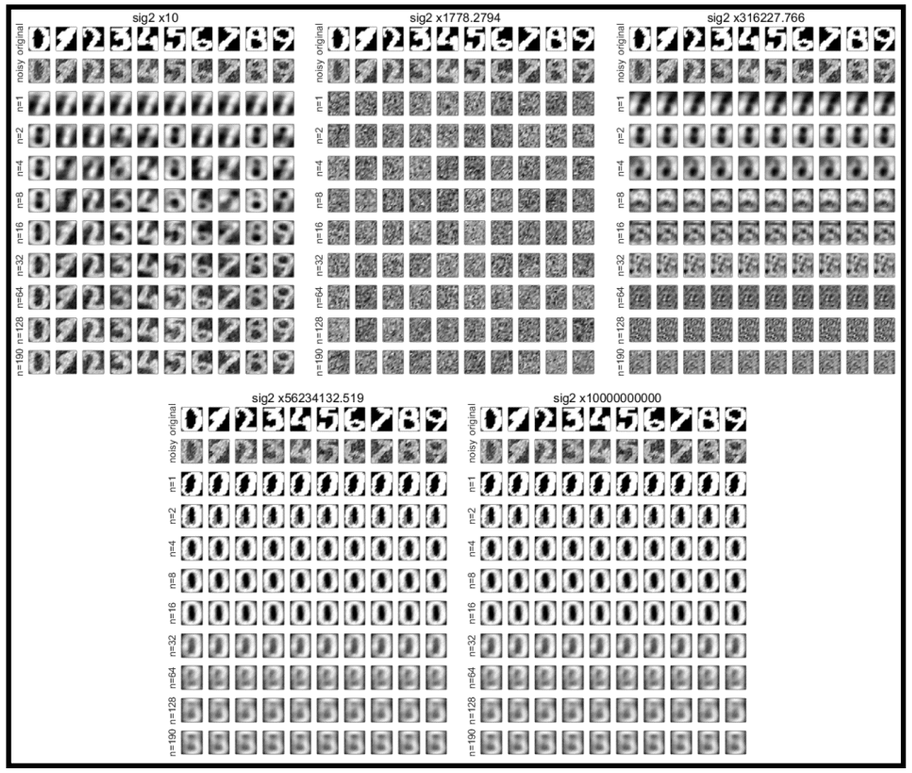
Denoising with noise factor of 1.0 **fig 34**

1.2 Effect of increasing and decreasing sigma parameter

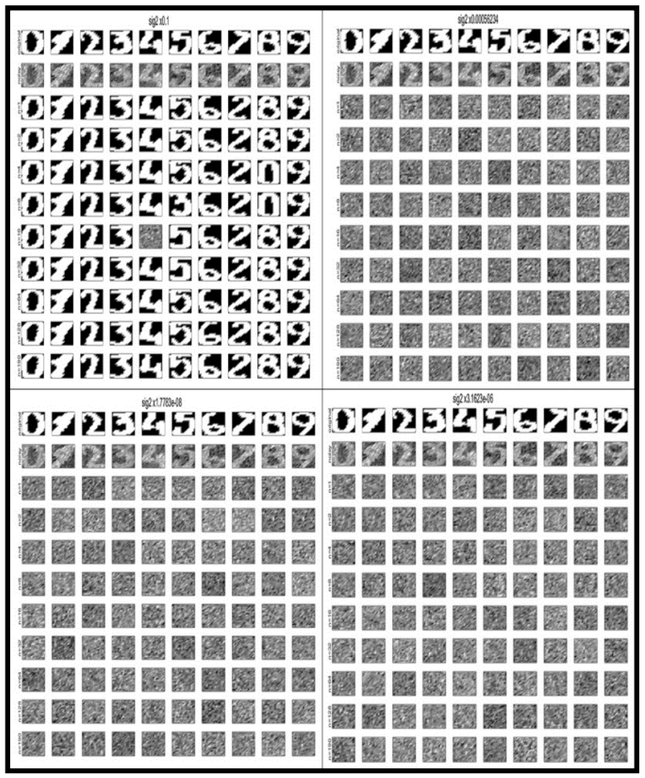
The results of increasing the sigma factor parameter for equispaced values in logarithmic scale can be seen in figure 35. With increase in logarithmic scale the digit reconstructions get noisier. At a very high value of Sig2 and few principle components used, all digits get wrongly classified as zeros or eights but with surprisingly little to no levels of noise.

Figure 36 show results of using small values of Sig2 with different number of principle components.
With Sig2 =0.1, we get noiseless reconstructions with only few misclassifications if the number of principle components is kept small;
With Sig2 <= 0.0005 we get noise images regardless the number of principle components.

In conclusion, to get optimal reconstructions with lesser noise and higher accuracy Sig2 values should not be changed drastically on a logarithmic scale.



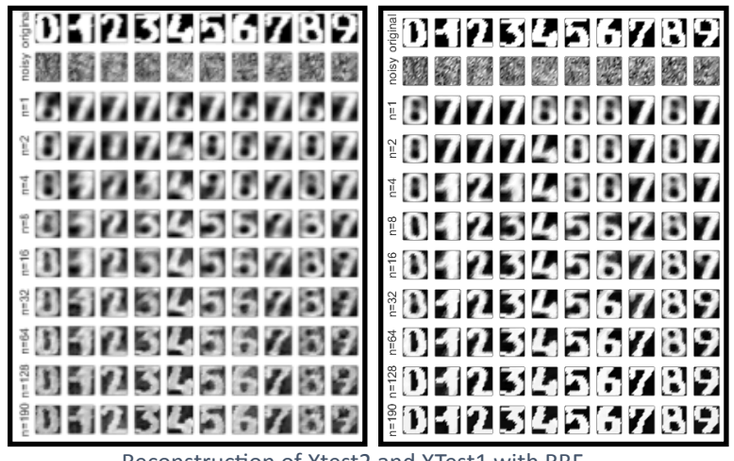
sig2 logarithmically scaled up **fig 35**



sig2 logarithmically scaled down **fig 36**

1.3 Reconstructing original XTest2 digits & investigating training (XTest) and validation sets (XTest1, XTest2) reconstruction error

The results of reconstructing the digits of Xtest2 are shown in figure 37a.
The reconstructions with fewer number of components are blurred (but little noisy) and almost all the digits are classified wrong.
As the number of components increase, the digits become less blurry and the accuracy of digit classification improves (figure 37b).
At n =32, we get maximum accuracy classification.
We use different values of Sig2 to find minimal reconstruction error on the validation sets XTest1 and XTest2.
We get an optimal value for validation set Xtest1 at Sig^2=18.
We use different various values of sig2 to check reconstruction errors on the second validation set Xtest1 and to find an optimal value for sig2. This reconstruction is shown in figure 37b.
As seen in the figure, the reconstructions were completely denoised, even though the noise-factor was set at 1.
There are some classification errors when low number of components are selected, but with higher number of components, the classification is perfect as well.



Reconstruction of Xtest2 and XTest1 with RBF kernel at noise and sig2 1.0 **fig 37**

2 Fixed-size LS-SVM

2.1 Shuttle (statlog) dataset

The Shuttle dataset contains 58,000 examples of 9 dimensions each i.e. it has 9 attributes, all of which are numerical. Approximately 80% of the data belongs to class 1. As this is a very large dataset, getting a good approximation of its support vectors can be helpful in making the training and classification phases faster. Hence, we examine classification results from Fixed-size LS-SVM and L_0 norm method.

The parameter k is a constant factor that is used to determine maximum the number of support vectors to be used for training and classification. Therefore, number of components = k * size of the dataset. We set the value of k at 1, so that we would get maximum 241 support vectors. A small value of k is necessary to train and use the model for classification in a reasonable amount of time.

Figure 38 shows the time comparison of the two approaches, FS-LSSVM and L_0 norm. Both approaches use about the same amount of time of almost 135 seconds each.

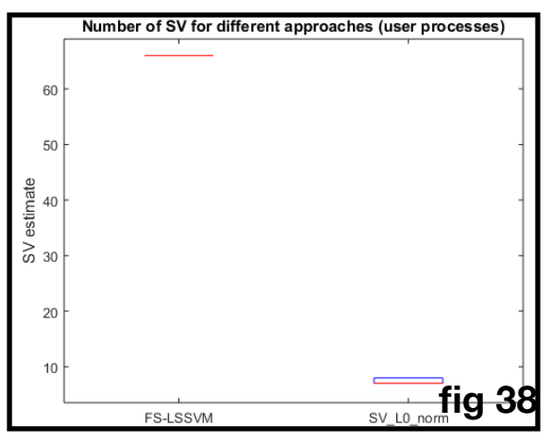


fig 38

Figure 39 shows the comparison of the number of components selected by each approach. The number of components selected by L_0 norm are about 8, FS-LSSVM selects about 68 components which is significantly higher than L_0 norm.

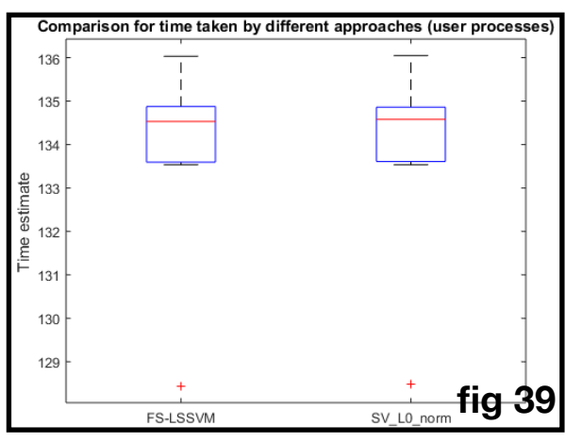


fig 39

Figure 40 shows the error comparison of these two approaches. Again, as we would expect the error rate of both the approaches is also almost exactly the same at 0.2145455.

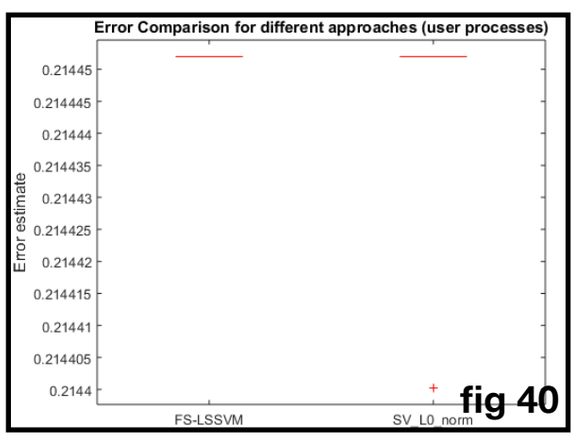


fig 40

Given these results, we can see how we can benefit from the L_0 norm approach when dealing from with a larger dataset as it offers the same amount of accuracy while using a much smaller number of components.

3 California dataset

The California dataset is a regression dataset which contains information on the California housing market from the 1990 census. It has 20640 cases with 9 variable each. We compare the FS-LSSVM and the L_0 norm approaches on this dataset. As this is considerably not a very large dataset, we set the value of k at 6 giving us a maximum of 862 support vectors.

Figure 41 shows the time comparison of the two approaches. Both these approaches take the same about of time with an average time of 67 seconds, a lowest time of 64 seconds and a highest time of up to 76.5 seconds.

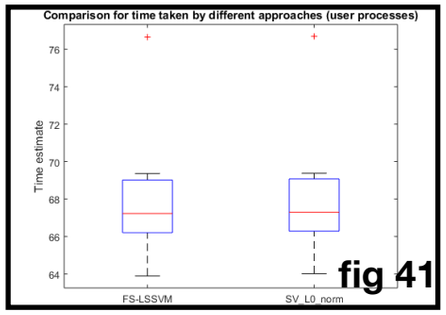


fig 41

Figure 42 shows the number of components selected by both the approaches. The FS-LSSVM selects only 43 components, while the L_0 norm selects an average of about 7 components with going as low as selecting only 1 component to a high to selecting 43 components.

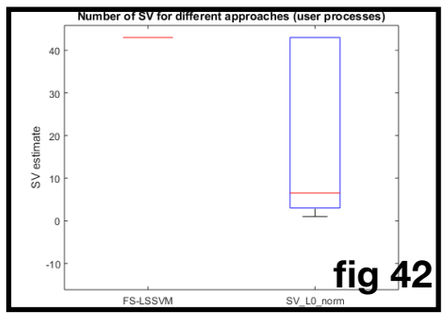


fig 42

Figure 43 shows the error rates of the two approaches. FS-LSSVM gives an error rate of a little over 0.35. Whereas, the L0 norm has an average error rate of a little above 0.4. Its range of error is at a minimum at 0.36 and going as high as up to 0.9 which is an extremely large amount of error.

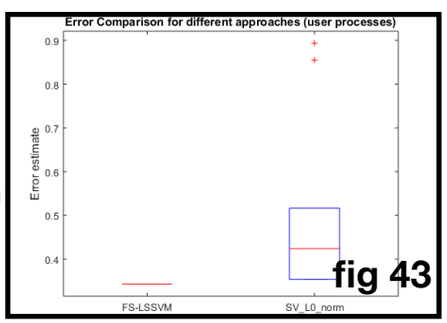


fig 43

These results show that the L_0 norm may not exactly be the best approach in all cases, as selecting very few components can result in a massive loss of accuracy.