Stockholms universitet

Expanding Density Peak Clustering Algorithm Using Gaussian Kernel and its Application on Insurance Data

Jie Wen

Masteruppsats i försäkringsmatematik
Master Thesis in Actuarial Mathematics

Masteruppsats 2020:11
Försäkringsmatematik
September 2020

www.math.su.se

Matematisk statistik
Matematiska institutionen
Stockholms universitet
106 91 Stockholm

Mathematical Statistics
Stockholm University
Master Thesis **2020:11**
http://www.math.su.se

Expanding Density Peak Clustering Algorithm Using Gaussian Kernel and its Application on Insurance Data

Jie Wen*

September 2020

Abstract

The purpose of study is to construct the Gaussian kernel density peak clustering (GKDPC) algorithm, as an alternative approach to achieve the task of clustering data. The GKDPC algorithm uses Gaussian kernel to define the density estimator, and attach it to the density peak clustering (DPC) algorithm to automatically obtain the correct number of clusters while locating the positions of the cluster centers. The demonstrations of the GKDPC algorithm have showed the superiority on free to parameters and robustness aspects over other algorithm counterparts. The applica- tion on the insurance data has also showed that the GKDPC has great potentials to be used in real-world industries.

*Postal address: Mathematical Statistics, Stockholm University, SE-106 91, Sweden. E-mail: wjroomnew@gmail.com. Supervisor: Chun-Biu Li.

Acknowledgements

I would first like to thank my supervisor Chun-biu Li, I could never made this far without his guidance, advises and encouragements. It was the countless times of communication with him throughout the process that enlightened me. Most importantly, I am really appreciate his forgiveness when I failed my duty and was about to give up.

I would also like to thank Professor Filip Lindskog for providing me the guidance and resources on the aspect of actuary. And lastly I would like to thank Mr. Jan-Olov Persson for providing me the administrative supports.

Contents

1 Introduction

Machine learning, as a subset of artificial intelligence, has been applied widely in countless industries these days. The idea of machine learning is to program algorithms for computers to solve tasks without using explicit instructions. Its attempt is to provide us with objective solutions to real world problems. There are two major aspects of machine learning, the supervised learning and the unsupervised learning. The concept of supervised learning is to use a set of input variables to predict a set of output variables (which are referred as labels). Substantially, the objective of supervised learning is to find the set of functions that link up the input-output relationship. The concept of unsupervised learning on the other hand, no labels are provided. The main focus under this aspect is therefore to find the inner relationship within the input variables.

There are two major directions within the category of unsupervised learning, dimension reduction and clustering. Dimension reduction focuses on reducing the number of variables according to considerations such as their importance. For clustering, the objective is to group observations so that the observations within in the same group are more similar than observations from other groups. This study will employ some dimension reduction technique, such as the principal component analysis (PCA), but the main focus is on the aspect of clustering algorithm and its applications.

The concept of clustering also has many alternatives. Generally, clustering alternatives can be classified into the following ideas: combinatorial algorithms, mixture modelling and mode seeking [1, p. 507]. Combinational algorithms work directly on the observed data and consider clustering problems as combinational optimization problems. On the mixture modelling aspect, objects are assumed to follow some distributions, and the aim is to infer the density distributions and its parameters. Lastly, the mode seeker is a non-parametric version of the mixture modelling idea. In this case the detailed information of the density distributions is not of interest, only the density modes are emphasised. The modes can be defined by some unique features of the observations, for example the observations with higher densities.

In this thesis, we shall investigate in depth the branch of density based mode seeking clustering algorithms. Density based methods, in contrary to the most popular optimization clustering algorithms, are commonly known for its robustness in complex environment, for example when the observations are irregularly distributed. Of course, there are different ways to construct the local density estimator and locate the density modes. One of the newest methods on this aspect was introduced by Alex Rodriguez and Alessandro Laio from the paper "*Clustering by fast search and find of density peaks*" [2] in 2014, and often referred as the density peak clustering (DPC) algorithm.

Nevertheless, the DPC algorithm has its own weaknesses. An obvious issue

is the structure of the kernel, the simple cut-off kernel proposed in the original work of Rodriguez and Laio is not free of parameters and the density estimator function is not smooth. A algorithm which is not free of parameters indicates users have to manually select the parameter values. In practice, this means users have to be experienced and have external information on the data in order to select the correct parameter values. Thus, the dependency on manually selected parameters is an obvious disadvantage to the algorithm and greatly limits its performance. On the other hand, the non-smooth density estimator function can only produce integer values. As a consequence, the chance of having identical density estimations on multiple observations is largely increased, and this would cause problematic clustering results. In order to overcome these difficulties, we would like to replace the DPC density function by the famous Gaussian kernel, and use a data-driven algorithm to automatically adjust the kernel parameter.

A typical usage of the clustering technique would be the topic of "risk classification" in actuarial science. The fundamental purpose of risk classification, explained by Robert J. Finger, "*individuals who are expected to have the same costs are grouped together. The actuary then calculates a price for the group and assumes that the price is applicable to all members of the group.*[4]". For the principle of actuary science, policy owners are hardly to be considered homogeneous, and people believe variables such as age, region and behaviour of insured person or insured objects to have significant impacts to the risks in life and non-life insurance respectively. The aim for risk classification is therefore to classify policies into different risk groups according to their characteristics, so that different prices could be set for different risk groups.

Transitional risk classification methods are based on regression analysis combined with some subjective decisions making. For instance, the rating factors are often subjectively selected, and the performance of the selected factors are only evaluated through the generalized linear model (GLM) results. In contrary, unsupervised techniques could provide us a data-driven alternative for the task of risk classification. In this study, we shall also investigate the application of our clustering algorithm to real world insurance data.

2 Methods and Theories

The main objective of this study is to construct and demonstrate the DPC algorithm based on the Gaussian kernel density estimator, and we call this algorithm Gaussian kernel DPC (GKDPC). To prepare this, we need to first explain and demonstrate the original DPC algorithm. Then, we shall discuss the dissimilarity and kernel function details under the conception of DPC.

In addition, the k-means algorithm will also be introduced as a comparison to benchmark our results.

2.1 The k-means clustering

The k-means clustering algorithm is probably the most popular algorithm among the clustering categories. The k-means algorithm is well known for its balance between effectiveness, robustness and computational difficulties. Due to its popularity and usefulness, it will be a good benchmark to compare and evaluate our GKDPC algorithm.

The fundamental idea behind the k-means algorithm is that, provided with hyperparameter k as the number of clusters, a within-cluster index is defined as the sum of squares over the Euclidean dissimilarities between observations and cluster centers. The task is to find the positions of center points, along with the partition of observations into k groups, which minimizes the overall within-cluster index.

For m dimensional quantitative observations $x_1, x_2, ..., x_n \in X$, the dissimilarities between observation x_i and x_j in Euclidean distances is defined as

$$d_{i,j} = \sum_{l=1}^{m} (x_{i,l} - x_{j,l})^2 = \| x_{i,l} - x_{j,l} \|^2 . \tag{1}$$

In k-means algorithm, we presume that there are k clusters. Every observation is assigned to one of the clusters. Each cluster contains one distinct center, and it should be representing the average position of its cluster members. In this scenario, we set center points $M_1, ..., M_k$ for cluster $1, ..., k$, and assume x_i is assigned to cluster k_i, represented by the function $C : C(x_i) = k_i$. Thus, the within cluster sum of square $W(C)$ is defined as

$$W(C) = \sum_{l=1}^{k} \sum_{i=1}^{n} I_{i,l} d_{i,M_l}, \tag{2}$$

where $I_{i,l}$ is an indicator function: $I_{i,l} = 1$ if $l = k_i$ and $I_{i,l} = 0$ otherwise. The objective of k-means is to find the mean points $M_1, ..., M_k$ and the partition C that minimize $W(C)$. The following algorithm is constructed to approach the objective.

Algorithm 1 The k-means algorithm

1. Randomly pick k center points.

2. Assign each point to the nearest center point according to the Euclidean dissimilarity.

$$C(x_i) = \underset{1 \leq l \leq k}{\operatorname{argmin}} \parallel x_i - M_l \parallel^2 \tag{3}$$

3. Calculate the means for each assigned cluster, if the means are different from the previous centers, then replace the previous centers with the means.

4. Iterate step 2 and 3 until the centers no longer changes.

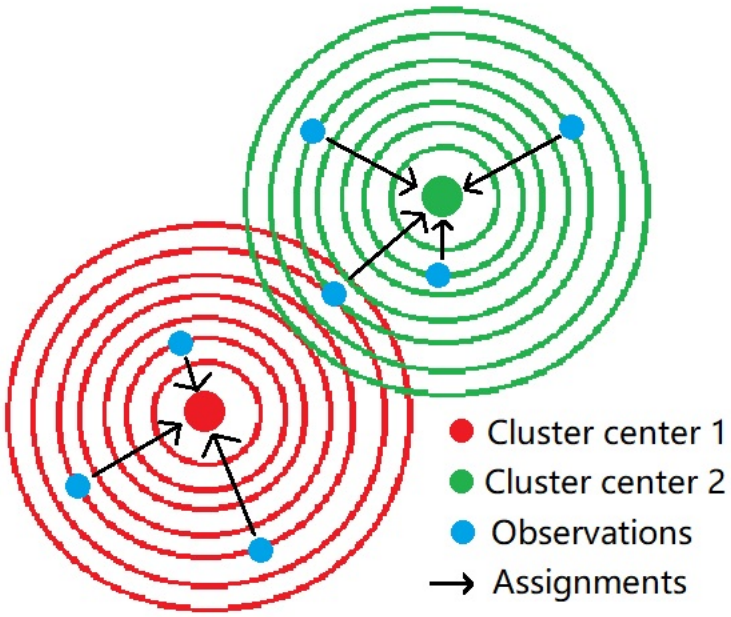
The convergence of mean vector is guaranteed as iteration proceeds because of two reasons: first, the ways of cluster partitioning is finite; second, the replacement of mean vector only occurs when the present partition C_t produces smaller $W(C_t)$ than the previous $W(C_{t-1})$, this indicates $W(C_t)$ must be a monotone decreasing function of the iteration number t. However, the convergence may lead to a local minimum, and different starting centers would lead to different clustering results.

More importantly, the k-means algorithm depends heavily on the Euclidean dissimilarities between points and centers. This could lead to many potential disadvantages. The first problem is about the Euclidean metric, this means that this algorithm can only handle numerical inputs. The second problem is that the k-means algorithm is not valid for non-spherical observations.

The incapability to handle non-spherical observations for k-means algorithm is a well known disadvantage. The assignments of k-means algorithms can be interpreted as drawing rings around the center points from inside to outside. In this setting, an observation is assigned to the cluster which it is positioned in the inner ring. For example, if an observation positions at the inner ring of cluster A and the outer ring of cluster B, we would assign this observation to belong in cluster A. This clustering procedure is illustrated in figure 1. Since those rings are spherically shaped, the k-means is excellent at handling with

spherically distributed observations, and of course it will be problematic when facing non-spherical observations.

Figure 1: The assignment of observations in k-means clustering.



The structure of a center point combine with an Euclidean radius is very common among clustering analysis, and we will constantly reviewing it in the later sections. One of the major purpose of this study is to overcome the difficulties mentioned in above, and we will achieve it by introducing the density peak clustering in the next subsection.

2.2 The density peak clustering (DPC)

The DPC algorithm is based on the assumption that cluster centers are the high-density points and surrounded by points with lower densities, while the distance between cluster centers should be significantly large. To capture the observations which contain both features, we define two quantities for each point x_i: ρ_i which represents the local density of x_i and δ_i which represents the min-

imum distance from x_i to a point with higher density than x_i.

In the original formulation [2, equation 1], the local density of point x_i is defined as

$$\rho_i = \sum_{j=1}^{n} \chi(d_{i,j} - d_c), \tag{4}$$

where $d_{i,j}$ is the Euclidean distance between point x_i and point x_j; d_c is a predetermined threshold distance, it is a hyperparameter of the DPC algorithm; and $\chi(x)$ is an indicator function that $\chi(x) = 1$ if $x < 0$ and $\chi(x) = 0$ otherwise.

ρ_i is basically the total number of points within the m-dim hypersphere formed by the center point x_i and the radius d_c. More neighbour points within the threshold distance indicates the density of the target point is higher. The objective of the DPC algorithm is to locate the points with higher densities, since high density points are likely to be cluster centers. However, high density points are often concentrated, meaning high density points are most likely to be neighbours. Thus, another restriction is added, called distance to higher density points, later refer as the distance estimator, for the purpose of eliminating neighbouring high-density points.

In the original formulation [2, equation 2], the distance estimator of point x_i is defined as

$$\delta_i = \min_{j:\rho_j > \rho_i} (d_{i,j}), \tag{5}$$

if point x_i has the highest density, we set δ_i=max($d_{i,j}$).

The distance estimator establishes a vector of each points' closest distance towards the point which have the higher density to the target point, except the point with the maximum density. Under this concept, if there are two neighbouring points both having high density estimations, then the one with lower density will have a very little distance estimation. The strategy is that, only the points with both high density estimation and large distance estimations should be selected as cluster centers. An index can be established via the product between the density estimators and the distance estimators, let us call it the product index, $PI_i = \rho_i \delta_i$ for $i \in (1, ..., n)$. Only the points with both high density estimations and large distance estimations will have large product index values. Hence, the cluster centers can be located through sorting the index from low to high and pick the points that have obviously large product index

value. If we plot the sorted index, such points should be located on the top-right corner of the plot. The sorted index plots often referred as the "decision graph".

After locating the DPC cluster centers, the rest of observations are assigned to the nearest observation which has higher density than the target one,

$$C(x_i) = \underset{j:\rho_j > \rho_i}{\operatorname{argmin}}(d_{i,j}). \tag{6}$$

Eventually all of the observations will be assigned into a cluster containing one of the cluster centers.

This method of assigning points is significantly different from the k-means counterparts. The clear advantage of the DPC assigning method is that, it is no longer restricted to spherically clustered observations. Thus, the DPC algorithm is more robust on complicated environments.

To summarize DPC algorithm

Algorithm 2 The density peak algorithm

1. Calculate the dissimilarity matrix for $d_{i,j}, i \in 1, ..., n, j \in 1, ..., n$ according to the Euclidean metric.

2. For every point x_i, estimate the points' density according to $\rho_i = \sum_{j=1}^{n} \chi(d_{i,j} - d_c)$, the distance $\delta_i = \min_{j:\rho_j > \rho_i} (d_{i,j})$ and multiply them to produce the product index PI_i.

3. Plot the sorted PI to obtain the decision graph. Determine the cluster centers through observing the decision graph and picking the obviously large points.

4. Assign rest of the points to the nearest point with higher density $C(x_i) = \underset{j:\rho_j > \rho_i}{\operatorname{argmin}}(d_{i,j})$.

2.3 The selection of dissimilarity measure

Dissimilarity is a key term in clustering algorithms. As what it sounds, the aim of dissimilarity is to evaluate the dissimilarity of two targets. Dissimilarities

are usually defined by a metric, where metric is a function on a set that unusually satisfies four properties: non-negativity ($d_{i,j} \geq 0$), identity of indiscernibles ($x_i = x_j$ if and only if $d_{i,j} = 0$), symmetry ($d_{i,j} = d_{j,i}$) and triangle inequality ($d_{i,k} \leq d_{i,j} + d_{j,k}$). The most common dissimilarity measure is the Euclidean distance between two points.

$$d_{i,j} = (\sum_{l=1}^{m} |x_{i,l} - x_{j,l}|^2)^{\frac{1}{2}}, \tag{7}$$

where $x_{i,l}$ represents point x_i on the l^{th} dimension. Euclidean distance is one special case of the more general Minkowski distance which is defined as

$$d_{i,j} = (\sum_{l=1}^{m} |x_{i,l} - x_{j,l}|^p)^{\frac{1}{p}}. \tag{8}$$

When p=2, Minkowski is the Euclidean distance. Other notable Minkowski distances includes Manhattan distance when p=1. The Manhattan distance can be a useful alternative when we are handling ordered categorical data. Another distance definition worth mentioning is the Hamming distance. The Hamming distance between two strings (points as sequences of characters) is defined by the number of positions which the two points in the corresponding position are not equal. Let us assume two points x_i and x_j in the form of sequence of characters, the hamming distance is defined as

$$d_{i,j} = x_i \oplus x_j, \tag{9}$$

where $\oplus$ is the exclusive disjunction operation between two strings. The output of the exclusive disjunction operation is 1 when inputs differ and 0 otherwise. The Hamming distance is the better alternative for the spaces based on nominal variables, such as gender and geographical location. Detailed explanations on the application of Manhattan and Hamming distance is provided in section 4.

More alternative of determining the dissimilarity measure is to add some extra monotone function on top of the Minkowski distance. Commonly used functions include the exponential function $\exp(d_{i,j})$ and logarithm function $\ln(d_{i,j})$. For instance, if the density algorithm is based on the inverse dissimilarity (the similarity), and the similarity is defined as some function of $\frac{1}{d_{i,j}}$. If $d_{i,j} = 0$ the similarity would become infinite, thus causes problems in further calculations. In this case, taking the exponential function over the dissimilarities could

avoid such problems. More importantly, taking the exponential of dissimilarities would let the algorithm distributes more weights on the nearby points than the faraway points. The Gaussian kernel demonstrates in the next subsection is in fact a form of exponential dissimilarity function.

Another issue we have to keep in mind is that in multi-dimensional spaces, the difference in the spread of observations on each dimensions would also have impact on the dissimilarity measure. For example, when a set of points with 2 dimensions, the points on the first dimension are distributed between 0 and 1, and the points on the second dimension are distributed between 0 and 1000. In this situation, it is likely that for many metric based dissimilarity, the effect of the second dimension will overwhelm the effect of the first dimension. This indicates that we should not focus on the absolute scale on the axis but rather the relative differences among the axis.

A procedure to offset the effect of imbalanced dissimilarities is always required in practical cases. Such procedures are mostly based on means of a weighted average (convex combination)

$$\begin{aligned} D(x_i, x_j) &= \sum_{l=1}^{m} w_l d(x_{i,l}, x_{j,l}); \\ \sum_{l=1}^{m} w_l &= 1. \end{aligned} \tag{10}$$

Here $D(x_i, x_j)$ is the balanced dissimilarity between observation x_i and x_j; for $l \in (1, ..., m)$, $d(x_{i,l}, x_{j,l})$ is the dissimilarity between x_i and x_j on the l^{th} dimension; w_l is the weight of dissimilarity assigned to the l^{th} dimension.

If the dissimilarities are not balanced in the first place, setting all weights to the same value, for example $w_k = 1$ for all $k \in (1, ..., m)$ does not give all dimension the correct influence. Alternatively, the weights should be determined according to the observation spreadness, so that sparse dimensions receive lower weights, and tight dimensions receive higher weights. This way all dimensions can be aligned.

The derivation of dimension alignment is explained in the book "*The Elements of Statistical learning*" [1, p. 505]. Let us consider the overall average of $D(x_i, x_j)$

$$\bar{D} = \frac{1}{n^2} \sum_{i=1}^{n} \sum_{j=1}^{n} D(x_i, x_j) = \sum_{l=1}^{m} w_l \bar{d}_l, \tag{11}$$

with average dissimilarity on the l^{th} dimension

$$\bar{d}_l = \frac{1}{n^2} \sum_{i=1}^{n} \sum_{j=1}^{n} d(x_{i,l}, x_{j,l}). \tag{12}$$

Let us reverse the above equations. In order to give the dimension all equal influence, then $w_l \bar{d}_l$ should be same for all l, therefore $w_l \sim 1/\bar{d}_l$. This weight estimator is valid for any definition of dissimilarity in the form of equation 10. For instance, if the dissimilarity is defined by the Euclidean metric, $\bar{d}_l = \frac{1}{n^2} \sum_{i=1}^{n} \sum_{j=1}^{n} (x_{i,l} - x_{j,l})^2 = 2\text{var}(x_l)$.

To rebalance the dissimilarities is equivalent to rebalance the data set on different dimensions and then calculate the dissimilarities, e.g. $X_l = w_l x_l$, where X_l is the balanced data on dimension l, x_l is the original data on dimension l, w_l is the dimension weight based on the dissimilarity measure. If all dimensions have Euclidean based dissimilarities, then the rebalance procedure would be

$$X_l = w_l x_l = \frac{x_l}{2\text{var}(x_l)}. \tag{13}$$

Noticing that the standard score of the observation on dimension l is defined as

$$z_l = \frac{\bar{x} - x_l}{s}, \tag{14}$$

where s is $\sqrt{\text{var}(x_l)}$. This means equation 13 works similar to the non-centered standard score. Both rebalancing approaches are based on compressing the data scale according to the observation variance on each particular dimension. In other words, the standardization of data would also eliminate the issue of imbalanced dimensions when dissimilarities are defined by Euclidean metric. Thus, the standardization of data is a necessary step in Euclidean based clustering problems.

Lastly, I have to mention that mixed-type data is more often than pure numerical data in practice, hence the standardization is not the once and for all solution. Therefore, the re-weighting of the dominions must be based on the type of data and the corresponding ways of defining dissimilarities.

2.4 The kernel function of Gaussian kernel DPC algorithm

Besides the dissimilarity measures, the choice of the density estimation is another key issue of density based clustering algorithms. The density of an observation is a measurement on the number of observed points per some predefined unit space. The relevant density in regards of density based clustering is the local density of a target point x_i. This local density should provide a measurement that the level of concentration on the position of point x_i could be evaluated.

It should be noticed that the local density of a point is not the true probability distribution density (PDF). The curse of dimensionality leads to a fact that the true PDF of a specific point decrease in an exponential rate when the number of dimension increases. Detailed explanations on this phenomena can be found in the book "*The Elements of Statistical learning*" [1, Chapter 2.5]. As a consequence, the approach to estimate the true distribution density in practice is rather pointless, since the dimension size is usually not restricted. An ideal local density estimator should be simple, while only reflexes certain characteristics, such as the order of the points' true density.

The local density estimation usually take dissimilarity measure as input, and there are alternative methods to construct such functions. Two of the main strategies for the density estimations are the followings. The first is to count number of neighbouring points in a m-dim hypersphere restricted by the dissimilarity input, an example would be the DBSCAN. The second strategy is to sum up the dissimilarities for the closer neighbours, an example would be the k-nearest-neighbour (kNN) algorithm.

The DPC algorithm belongs to the first strategy. Equation 4 transforms dissimilarities associated with a target point into the local density of this point. In the concept of machine learning, such function is classified as a kernel function.

The DPC kernel function $\chi(d_{i,j} - d_c)$ has some obvious drawbacks. First, the function $\chi(d_{i,j} - d_c)$ is either 1 or 0, which is extremely non-smooth. Under this kernel function the density contribution of a neighbour point near $x_i + d_c$ and a point near x_i are both equal to 1, and it is of course not reasonable. Another drawback is the "one-size-fits-all" problem. The parameter d_c must be predetermined and it is employed to all observations. When a sparse cluster and a tight cluster appeared simultaneously in the space, the suitable d_c for the sparse cluster may not be suitable for the tight cluster, thus causing extra difficulty to the estimation of d_c.

In order to overcome the difficulties in the DPC kernel function, we would like to employ the Gaussian kernel instead. First of all, let us formulate the multivariate Gaussian PDF,

$$f(\mu, \Sigma) = \frac{1}{(2\pi)^{m/2}|\Sigma|^{1/2}} e^{-\frac{1}{2}(\mu-x)'\Sigma^{-1}(\mu-x)}, \tag{15}$$

where μ is the mean vector, m is the dimension and Σ is the covariance matrix. The term $(\mu-x)'\Sigma^{-1}(\mu-x)$ is so called squared Mahalanobis distance, and it is also squared Euclidean distance between point μ and x if Σ is a diagonal matrix.

If we build up the kernel function based on the multivariate Gaussian PDF, this would provide a smooth alternative for the kernel function. The intention of Gaussian kernel is to let nearby observations to enhance the density of a target observation according to the distances between them. The density estimator of a target observation x_i according to the Gaussian kernel can be formulated as

$$f(x_i, \Sigma) = \frac{1}{n(2\pi)^{m/2}|\Sigma|^{1/2}} \sum_{j=1}^{n} e^{-\frac{1}{2}(x_i-x_j)'\Sigma^{-1}(x_i-x_j)}, \tag{16}$$

where m is the dimension for the data set, $x_i - x_j$ is a m-sized vector representing the displacement of x_i and x_j on each dimension respectively. The values of m and $x_i - x_j$ are determined by the data if the data is in the form of $n * m$ matrix. The covariance matrix Σ on the other hand is a hyperparameter, so we have to decide a reasonable estimator for it.

It is easy to recognize equation 16 as the average of n multivariate Gaussian probability densities centered around x_i. The density contribution of a point x_j is inverse proportional to the linear transformation $(x_i - x_j)'\Sigma^{-1}(x_i - x_j)$. The key issue is to understand the effect of Σ towards the density estimation function. The geometric interpretation of a covariance matrix in general is that, it creates rings of m-dimensional ellipses centered around a target point x_i. Another point x_j, would receive a probability density according to the ring it is positioned. The exponential term in equation 16 will be closer to 1 if x_j lies in the inner rings, and closer to 0 if x_j lies in the outer rings. Since the linear transformation is taken places in an exponential function, this indicates that the density value quickly dies out when x_j move towards outer rings.

Basically, the covariance matrix controls the spread of observations, and geometrically the spreadness forms a m-dim ellipsoid. This ellipsoid has its m-dim volume and the directions of ellipsoid axis. This means that all covariance matrices contain two aspects, the "size" and the "orientation", representing the m-dimensional volume (formed by the magnitude of each ellipsoid axis) and the direction of ellipsoid axis respectively.

The separation of the size and the orientation of matrices can be achieved via eigendecomposition. Let us start with the definition of eigenvalue and eigenvectors. Assume a $m * m$ square matrix Σ,

$$\Sigma v = v\lambda, \tag{17}$$

where v is an eigenvector and λ is an eigenvalue. For $m * m$ matrices, there will be m eigenvectors each corresponds to an eigenvalue. If we list the eigenvector as column vectors of a matrix V, $V = (v'_1, v'_2, ..., v'_m)$ and set the diagonal matrix Λ, $\Lambda_{i,i} = (\lambda_1, \lambda_2, ..., \lambda_m)$ for $i \in (1, 2, ..., m)$. We could formulate equation 17 as

$$\begin{aligned} \Sigma V &= V\Lambda \\ \Sigma &= V\Lambda V^{-1}. \end{aligned} \tag{18}$$

The term $V\Lambda V^{-1}$ is the eigendecomposition of matrix Σ. Since Λ is a diagonal matrix, the above procedure is also called the diagonalization of Σ. The real spectral theorem states that, providing with a real symmetric matrix Σ, there must exist orthogonal matrix V and real diagonal matrix Λ, so that $\Sigma = V\Lambda V^{-1}$. V is orthogonal also means V^{-1}=V'. Moreover, if Λ is formulated from big to small, namely $\Lambda_{1,1} = \lambda_1$, $\Lambda_{2,2} = \lambda_2, ..., \Lambda_{m,m} = \lambda_m$ for $\lambda_1 \geq \lambda_2 \geq ... \geq \lambda_m$, the decomposition $V\Lambda V^{'}$ is unique.

The decomposed form $V\Lambda V^{'}$ of a matrix is essentially helpful to separate the size and the orientation. Geometrically, the eigenvalues λ_l represents the magnitude of the ellipsoid axis on the l^{th} dimension. The volume of m-dimensional ellipsoid is given by

$$V_m = \frac{\pi^{\frac{m}{2}}}{\Gamma(\frac{m}{2}+1)} \prod_{j=1}^{m} \lambda_j. \tag{19}$$

where $\Gamma(x)$ is the Gamma function. Therefore, the volume and shape is determined only by the matrix Λ. The linear transformation matrix V on the other hand represents the rotation of Σ to the basis of Λ, while preserving the volume.

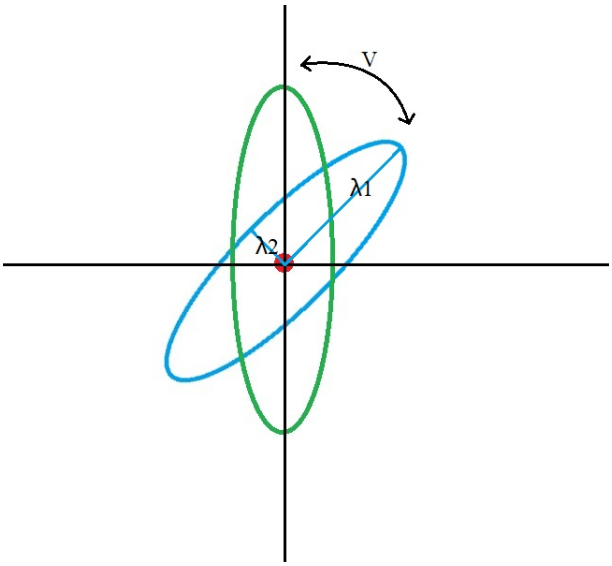


Figure 2: The geometric interpretation of eigendecomposition in 2 dimensional space. V: the rotation matrix; λ_1, λ_2: the eigenvalues.

It is easy to see that covariance matrices are real and symmetric, therefore eigendecomposition can always be employed to separate the size and the orientation. And thus the task of finding the ideal kernel distribution covariance can be separated to the task of determining ellipsoid volume, shape and orientation, respectively.

2.5 Parameter estimation for the Gaussian kernel algorithm

The simplest form for the Σ matrix would be the no-orientation and m-dim spherical shape. No-orientation indicates $V = I_m$, and the corresponding eigenvalues become the variance on each of the dimensions. And if we are constructing a m-dim spherical volume, then the variance on each dimension is equal to the same value σ^2, so the covariance matrix can be written in the form $\Sigma = \sigma^2 * I_m$. In this situation, the problem of estimating the whole covariance matrix has reduced to estimating a scalar which represents the radius of a m-sphere. In reality the knowledge on data are often limited, thus making extra assumptions on data clustering behavior would be rather unreasonable. In this sense, picking Σ in the form of $\sigma^2 * I_m$ would be a conservative choice for an arbitrary data set.

If we set $\Sigma = \sigma^2 * I_m$, Σ^{-1} is simply $\frac{1}{\sigma^2} * I_m$, as the result, equation 16 becomes

$$
\begin{aligned}
f(x_i, \Sigma) &= \frac{1}{n(2\pi)^{d/2}|\Sigma|^{1/2}} \sum_{j=1}^{n} e^{-\frac{1}{2}(x_i - x_j)'(\frac{1}{\sigma^2} * I_m)(x_i - x_j)} \\
&= \frac{1}{n(2\pi)^{d/2}|\Sigma|^{1/2}} \sum_{j=1}^{n} e^{-\frac{1}{2}\sum_{l=1}^{m}(\frac{x_{i,l} - x_{j,l}}{\sigma})^2}.
\end{aligned} \tag{20}
$$

The term $\sum_{l=1}^{m}(\frac{x_{i,l}-x_{j,l}}{\sigma})^2$ inside the exponential function is basically the squared Euclidean distance between $\frac{x_i}{\sigma}$ and $\frac{x_j}{\sigma}$. The inverse proportionality between the displacement $x_i - x_j$ and the parameter σ (The σ parameter is often called the "bandwidth" of Gaussian kernel.) is much more convenience in this regard. Increasing the value of σ would cause the drop on density at an exponential rate. In geometric perspective, the σ draws an spherical boundary between the neighbouring points which contribute densities and the far away points which contribute no densities to the target point. This boundary is similar to the radius threshold parameter d_c for the original DPC algorithm in equation 4. What differs from the DPC algorithm is that, the points which contributes density, are contributing smoothly according to the Euclidean distance between the neighbouring points and the target point.

At this stage, the problem has basically reduced to the estimation of scalar standard deviation for the Gaussian kernel distribution. In one dimensional situation, the relationship between standard deviation and Gaussian probability density can be simplified into the "three sigma rule". In short words, for one-dimensional normal distribution centered around μ, around 68 percent of cumulative densities are lying in between $(\mu-\sigma, \mu+\sigma)$ quantile; around 95 percent of cumulative densities are lying in between $(\mu - 2\sigma, \mu + 2\sigma)$ quantile; and around 99.7 percent of cumulative densities are lying in between $(\mu-3\sigma, \mu+3\sigma)$ quantile.

In 1-dim situations, the three sigma rule implies over 95 percent of cumulative distribution density are concentrated within the 1-dim sphere with center x_i and radius 2σ. This means that any point outside of this sphere would provide very limited amount of probability density to contribute the target density.

Remember our goal is to find the density estimator so that the densities of the data points can be ordered. The situation we must avoid is that all points receive almost the same amount of densities. The three sigma rule tells us that, 95 percent of the cumulative probabilities are concentrated in the region $(x_i - 2\sigma, x_i + 2\sigma)$, therefore σ would be too small for a target observation x_i when all neighbour points are outside of this region. On the other hand if σ is too large, this would means all other points are concentrated around the central peak and all provide maximum (which are identical) density contributions.

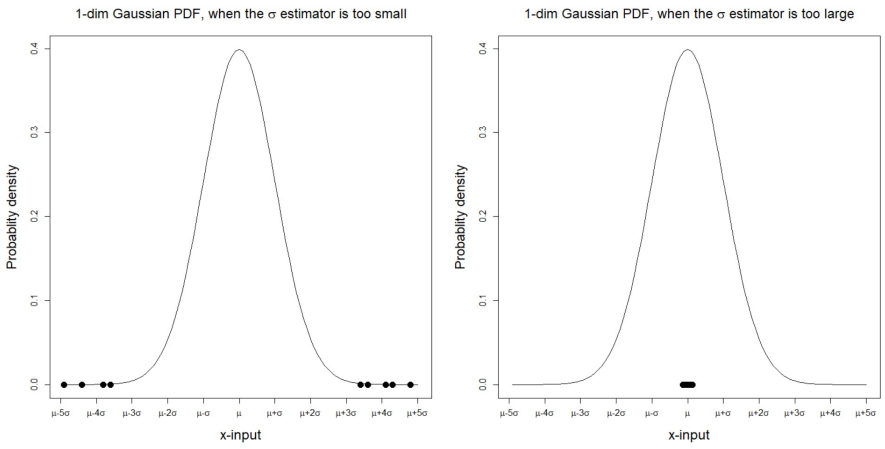


Figure 3: The impact of too small (left) and too large (right) sigma estimation. The x-axis is the domain. In our scenario, the target observation x_i is the center of the x-axis marked as μ. The neighbouring observations x_j are the black dots lying on the x-axis. The bell-shaped curves represent the probability densities of 1-dim Gaussian distribution. On the left plot, all neighbouring points will contribute 0 densities to the target point, because they are positioned outside of $(x_i - 2\sigma, x_i + 2\sigma)$. And on the right plot, all neighbouring points will contribute maximum densities (which is around 0.4 in this scenario) to the target point, because they are all positioned near center μ. If the parameter σ is either too large or too small, one of the above situations will be occurred on majority of the target observations, thus all targets will receive identical density estimations.

Figure 3 demonstrates the 1-dim Gaussian kernel when the σ is improperly sized. The curves are representing the bell-shaped Gaussian distribution function, and the dots representing neighbouring points around the target point positioned at μ. The left plot illustrates the situation when the σ is too small. In this case all neighbouring points are positioned outside of $(\mu - 2\sigma, \mu + 2\sigma)$ region, as the result all these points are contributing basically zero densities to the target point. The right plot illustrates the situation when the σ is too large. In this case all neighbouring points will be concentrated near the point μ, as a result all these points will provide almost maximum density contributions. The problem is, if all target points receive either average over 0 or average over maximum density contributions, the density estimations for all targets will become almost the same and therefore they are unable to be ordered.

A strategy to avoid the unsuitable σ estimations is that, we should keep σ as small as possible. At the same time we should not allow the σ to be so small, to an extent that no neighbouring observations are appearing in the $(x_i - 2\sigma, x_i + 2\sigma)$ region for majority of the targets observations. Based on

this strategy, we have constructed the 95-percent 1-nearest-neighbour (1-NN) σ estimator:

$$\begin{aligned} 2\hat{\sigma} &= \sigma_{1-NN} \\ \hat{\sigma} &= \sigma_{1-NN}/2, \end{aligned} \tag{21}$$

where σ_{1-NN} is defined as

$$\sigma_{1-NN} = (d_x : Pr(d_X \leq d_x) = 0.95), \tag{22}$$

where d_X represents the 1-NN Euclidean distances of all observations. The above estimator ensures that for 0.95 percent of observations, at least one neighbour points must be located in the $(x_i - 2\sigma, x_i + 2\sigma)$ region, which is 95-percent of the density quantile. Even one such point would provide the target observation a distinct kernel density estimation, so the observation density can be ordered. This estimation is very easy to obtain, we just need to calculate and sort the 1-NN distances for all observations, and pick the left quantile value which covers 95 percent of the 1-NN distance.

Let us continue towards the multivariate circumstances. The three sigma rule does not hold in multivariate situations, even if the covariance matrix is in the form of $\sigma * I_m$. In higher dimensions, the radius $2 * \sigma$ will no longer preserve the 95 percent quantile. Let us say $o * \sigma$ is correct radius which is able to preserve 95 percent quantile in m-dimension, and call "o" as the "sigma multiplier". Then, let us call the the m-dim hypersphere centered around x_i with radius $o * \sigma$ as "o-sigma-sphere". Consider a multivariate Gaussian distribution with covariance $\sigma * I_m$, the cumulative probability percentage quantiles of the o-sigma-spheres shrink when the dimension size increases. According to some basic Monte-Carlo simulations, the rough quantile levels for 1-to-4 sigma-sphere on dimensions 1-to-5 are listed in the table 1. Detailed information on how to obtain table 1 via simulations is provided in the appendix section.

Radius	1-dim	2-dim	3-dim	4-dim	5-dim
1σ	0.6806000	0.3927250	0.1971000	0.0906250	0.0385625
2σ	0.9541750	0.8631875	0.7371625	0.5918625	0.4500500
3σ	0.9972500	0.9893500	0.9705375	0.9394875	0.8907500
4σ	0.9999500	0.9997125	0.9988125	0.9970625	0.9931250

Table 1: Cumulative probability percentage quantiles on different dimensions. The rows represent the hypersphere radius in terms of $o * \sigma$, and the columns represent the dimension of data. The value in each cell represents the cumulative density quantile formed by the corresponding radius in the corresponding dimension.

Our goal for the 1-NN σ estimator is to preserve at least one neighbouring observation in the 95 percent multivariate Gaussian central quantile, namely the m-dim sphere centered around x_i with radius $o*\sigma$. As we can see from the table, in 2-dim and 3-dim cases the required radius distances are in between 2σ and 3σ. And for 4-dim and 5-dim cases, the required radius distances are in between 3σ and 4σ. Clearly, there is a increasing relationship between dimension size and the sigma multiplier which preserves the central density quantile. Let us define such relationship as a function $g(m, q)$, where m is the dimension, q is the percentage quantile we are aiming to preserve, and $g(m, q)$ is the sigma multiplier which makes the m-dim hypersphere with radius $g(m, q) * \sigma$ covers q percent of cumulative densities. Hence the 1-NN σ estimator for multi-dimensions can be defined as the 1-dimensional 1-NN estimator divided by the sigma multiplier $o = g(m, q)$, namely

$$\hat{\sigma}_m = (d_x : Pr(d_X \leq d_x) = 0.95)/g(m, 0.95). \tag{23}$$

The analytical solution for determining the function $g(m, q)$ can be summarized as the following: provide with a quantile value q, we have to find the value $o = g(m, q)$ so that the integral

$$q = \int_{-o\sigma}^{o\sigma} ... \int_{-o\sigma}^{o\sigma} \mathrm{N}(\mathbf{x}, \sigma^*)(x1)...d(xm) \tag{24}$$

holds, where $\mathrm{N}(\mathbf{x}, \sigma^*)$ is the multivariate Gaussian PDF with mean parameter $\mathbf{x} = (x1, ..., xm)$ and variance parameter $\sigma^* = \sigma * I_m$. Unfortunately I do not have a good solution for this integral at the moment. Instead, we could obtain the numerical solution of $g(m, q)$ using the Monte-Carlo simulation table 1.

In addition, the estimator σ_{1-NN} is only dependent on the observations, as a result the determinate $|\Sigma|$ in equation 16 is a constant. Since we are only interested in the ordering of density estimations, the constant multiplier $\frac{1}{(2\pi)^{d/2}|\Sigma|^{1/2}}$ can be eliminated. Thus, he final version of our 1-NN Gaussian kernel density estimator can be displayed as:

$$f(x_i, \hat{\sigma}_m) = \frac{\sum_{j=1}^{n} e^{-\frac{1}{2}\sum_{l=1}^{m}(\frac{x_{i,l}-x_{j,l}}{\hat{\sigma}_m})^2}}{n} = \frac{\sum_{j=1}^{n} K_{\hat{\sigma}_m}(x_i - x_j)}{n}, \qquad (25)$$

where the Gaussian kernel function $K(d_x)$ is basically the modified Gaussian PDF for dissimilarity d_x, and $K_\sigma(d_x) = K(\frac{d_x}{\sigma})$.

Let us call the DPC method with kernel function replaced by equation 25 as the Gaussian kernel DPC (GKDPC) algorithm, the algorithm summary is provided in the table below. The main body of the GKDPC algorithm is almost the same as original DPC, the only difference is that the kernel function in step 3 has been changed from equation 4 to equation 25.

Algorithm 3 The Gaussian kernel DPC

1. Calculate the dissimilarity matrix for $d_{i,j}, i \in 1, ..., n, j \in 1, ..., n$ according to the Euclidean metric.

2. Estimate the σ parameter according to the 95-percent 1-NN dissimilarity quantile, $\hat{\sigma_m} = (d_x : Pr(d_X \leq d_x) = 0.95)/g(m, 0.95)$.

3. For every point x_i, estimate the points' density according to $\rho_i = \frac{1}{n}\sum_{j=1}^{n} e^{-\frac{1}{2}\sum_{l=1}^{m}(\frac{x_{i,l}-x_{j,l}}{\hat{\sigma_m}})^2}$, the distance $\delta_i = \min_{j:\rho_j>\rho_i}(d_{i,j})$, and multiply them to produce the product index PI_i. Note: the term $\sum_{l=1}^{m}(\frac{x_{i,l}-x_{j,l}}{\hat{\sigma}_m})^2$ can be formulated as $\frac{d_{i,j}^2}{\hat{\sigma}_m^2}$, where $d_{i,j}$ is obtained in step 1.

4. Plot the sorted product index to obtain the decision graph. Determine the number of clusters and locate the cluster centers through observing the decision graph and picking the obviously large points.

5. Assign rest of the points to the nearest point with higher density, $C(x_i) = \underset{j:\rho_j>\rho_i}{\operatorname{argmin}}(d_{i,j})$.

3 The demonstrations of Gaussian kernel DPC

In the following sections we shall demonstrate the performance of the Gaussian kernel DPC under three different settings: 1. Generate an ideally formed spherical data, operate k-means, DPC, and Gaussian kernel DPC on this data and compare the results. 2. Generate an non-spherical data, operate k-means, DPC, and Gaussian kernel DPC to evaluate its advantage on arbitrary shaped observations. 3. Apply Gaussian kernel DPC on a real world insurance data to demonstrate its usefulness in practice.

3.1 Testing clustering algorithms on synthetic spherical data

For the first and second tasks, some reasonably synthetic data are required. The first test is to perform the DPC and Gaussian kernel DPC together with the k-means algorithm on spherically clustered data. The aim of adopting this setting is to use k-means as a benchmark to illustrate that the DPC based algorithms are able to perform just as good.

To achieve our goal, we decide to synthesize m-dimensional n-size data set, with space length $(0, length)$ on all dimensions. Among n observations, the data will be split to c clusters, each containing n_l observations, $\sum_{l=1}^{c} n_l = n$. For each cluster, the data is generated according to Gaussian distribution $N(\mu_{j,l}, \sigma_{j,l})$, $j = (1, ..., m)$ on each dimensions. The $\mu_{j,l}$ parameter locates the position of the true l^{th} cluster center on j-axis. The sigma parameter $\sigma_{j,l}$ determines the spread length of l^{th} cluster on the j-axis.

In this particular task, we set $m = 4$, $n = 300$, $length = 100$, $c = 3$, $n_c = (80, 100, 120)$. The parameters μ and σ are obtained through random number generators, so that our data clusters could be arbitrarily positioned and in arbitrary size. In addition, we have added two extra restrictions: the distance between μ should be significantly large, at least on more than 2 dimensions, so that clusters are not overlapping each others; the value of $\sigma_{j,l}$ for $j = 1, ..., 4$ should be more or less the same, so that overly stretched ellipses will not be synthesized. Outlier are not considered in this study, which means we will not generate any outlier in the synthetic data, and our clustering algorithms will not consider robustness against outliers.

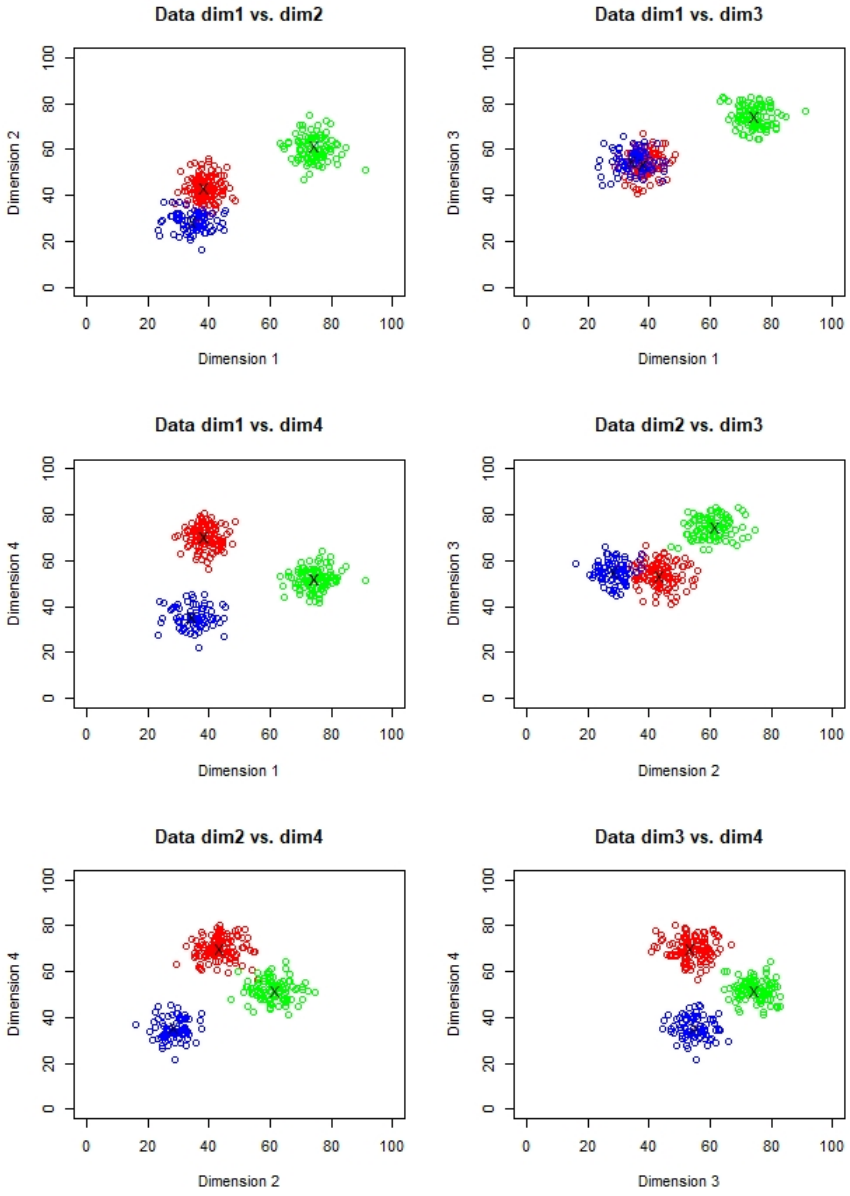


Figure 4: Synthetic spherical clusters on every pairwise dimensions. Each color represents a distinct cluster. The X-marks represent the true cluster centers.

Figure 4 illustrates the synthetic spherical clusters produced from the above procedure. The blue, red and green dots each representing a distinct cluster, and the X-marks represent the true cluster centers (the mean parameters of the

Gaussian generator.) It should be noticed that the blue and red clusters are overlapping in dimension 1 and dimension 3. This is actually good because an effective cluster algorithms should be able to detect cluster even on a subset of dimensions.

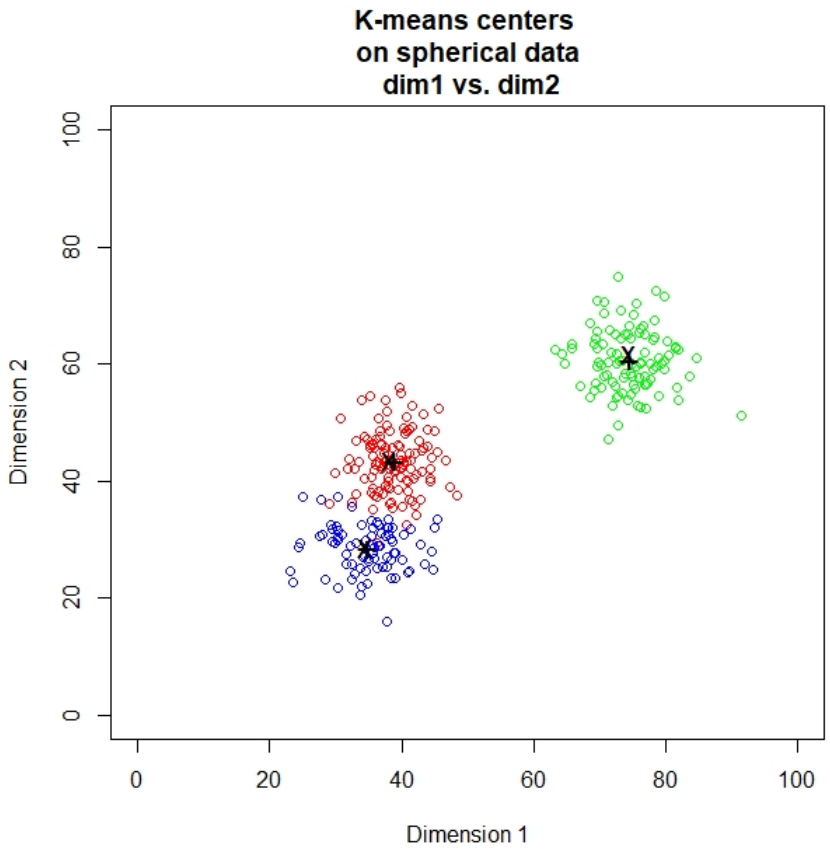


Figure 5: K-means clustering results on dimension 1 and 2: locations of the cluster centers. Each color represents a distinct cluster. The X-marks represent the true cluster centers. And '+' marks the estimated cluster centers.

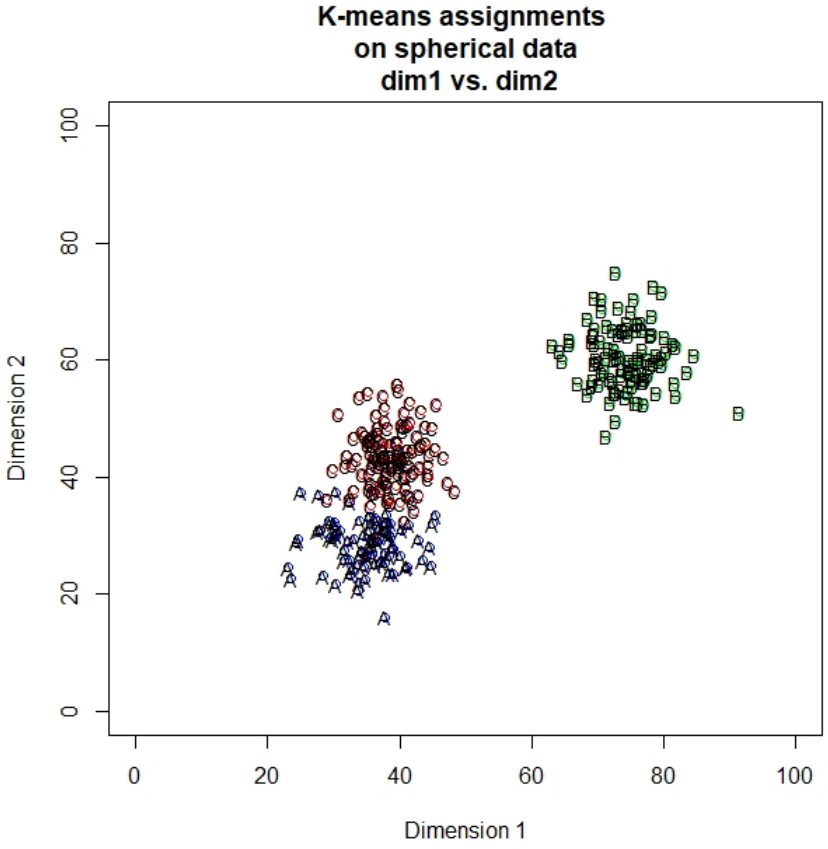


Figure 6: K-means clustering results on dimension 1 and 2: observation assignments. Each color represents a distinct cluster. The alphabets represents points' assignment according to the algorithm. The points with the same alphabets meaning they have assigned to the same cluster.

Since the number of clusters is known, we could directly operate the k-means algorithm with the true hyperparameter $k = 3$. Figure 5 and 6 shows the clustering result on dimension 1 and 2 obtained by the k-means algorithm. In figure 5, the estimated cluster centers are marked in '+', as we can see the calculated cluster centers are not far from the true cluster centers marked in 'X'. Figure 6 displays all of the points assigned to the 3 clusters centers, represented by alphabets A, B and C. Once again, we can see that each colored point are assigned to a distinct alphabet. Even the points in between the intersection of red and blue clusters are distinctively assigned to the correct alphabets, this is because the clusters are not overlaping in other dimensions. This indicates the k-means algorithm is working well in ideally distributed spherical data.

The DPC algorithm on the other hand, does not require the knowledge of number of clusters in the first place. The DPC algorithm has its own radius threshold parameter d_c. Too small d_c will causes the algorithm to over clustering (high density points will be appearing everywhere within the same true cluster, thus many cluster centers will be detected), and too large d_c will lead to low accuracy. Thus users have to manually search for the a suitable d_c value to be inserted. In this particular data, d_c=5 is obtained via manual trial and error. We will insert d_c=5, and the algorithm is able to automatically calculate the true center number. From the obvious large product index points in the decision graph, and tracking down the positions of those points, the cluster centers will be located.

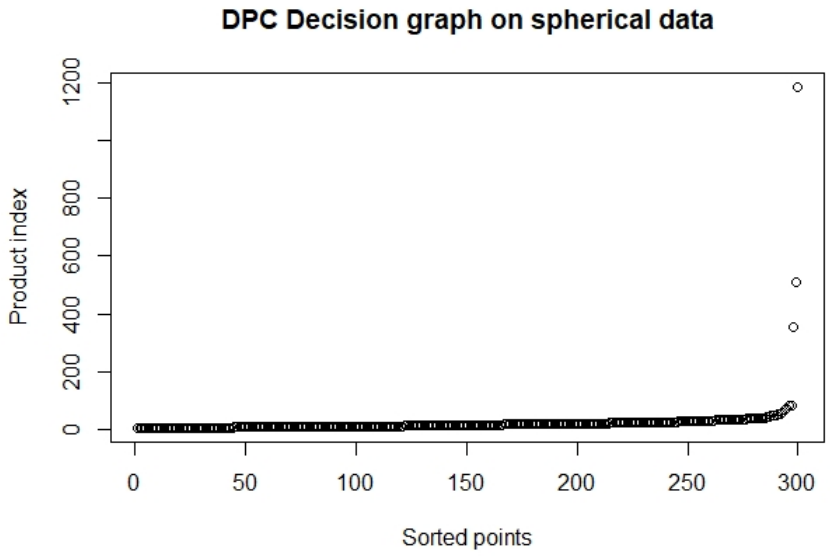


Figure 7: DPC Decision graph on spherical data

According to the decision graph, illustrated in figure 7, we can clearly observe there are 3 points locate in the top-right corner with obviously product index values. Thus, 3 would be the reasonable number of clusters to be considered.

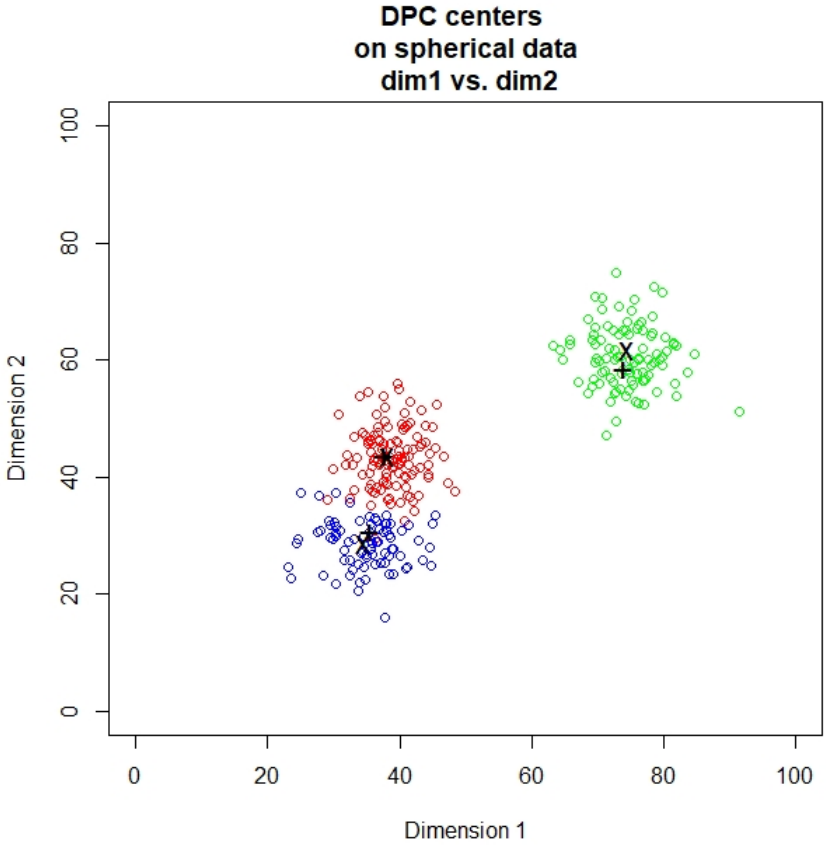


Figure 8: DPC clustering results on dimension 1 and 2: locations of the cluster centers. Each color represents a distinct cluster. The X-marks represent the true cluster centers. And '+' marks the estimated cluster centers.

Figure 8 and 9 illustrates DPC clustering result on spherical data on the first 2 dimensions. In figure 8, the estimated cluster centers are marked as '+' and the true cluster centers are marked as 'X'. This plot shows that, the DPC centers is to some extent close to the true center points, but more biased compare to the k-means center estimations. This is because DPC algorithms consider high-density points as centers, but the high density points are not necessarily the mean points of the cluster members. Thus, it is reasonable that average based algorithms such as k-means perform better in term of cluster center estimation.

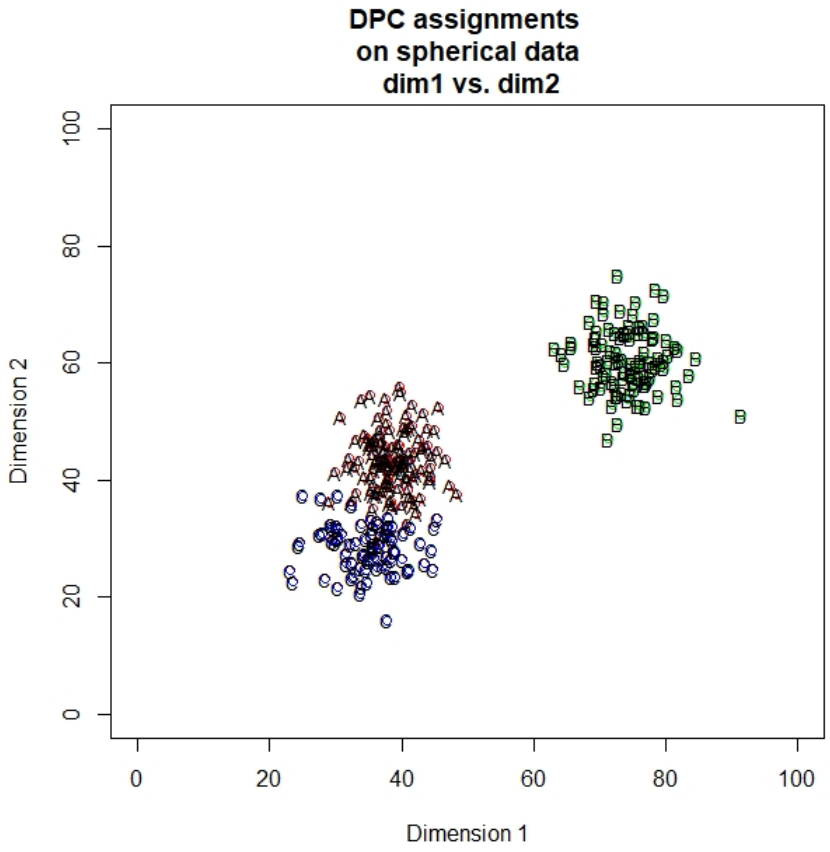


Figure 9: DPC clustering results on dimension 1 and 2: observation assignments. Each color represents a distinct cluster. The alphabets represents points' assignment according to the algorithm.

Figure 9 displays all of the points assignments according to DPC algorithm. Same as the result for k-means, all points are perfectly assigned to the correct cluster.

The last but most important algorithm to be tested is our GKDPC algorithm. In practice the Gaussian kernel DPC is a parameter-free while computation heavy variate of the DPC. Neither number of clusters nor radius threshold is required for the Gaussian kernel DPC. As we discussed in section 2, Gaussian DPC requires the bandwidth parameter σ, but it can be automatically obtained using methods like Scott's rule of thumb [5], 1-NN density quantile (equation 22), or other methods. In this particular demonstration, we will adopt the 1-NN density quantile method to obtain the bandwidth. Our data is 4-dimensional, thus the sigma multiplier function $g(m, 0.95)$ is equal to around 3.1 according

to table 1. The rest of the Gaussian kernel DPC procedures are the same as DPC.

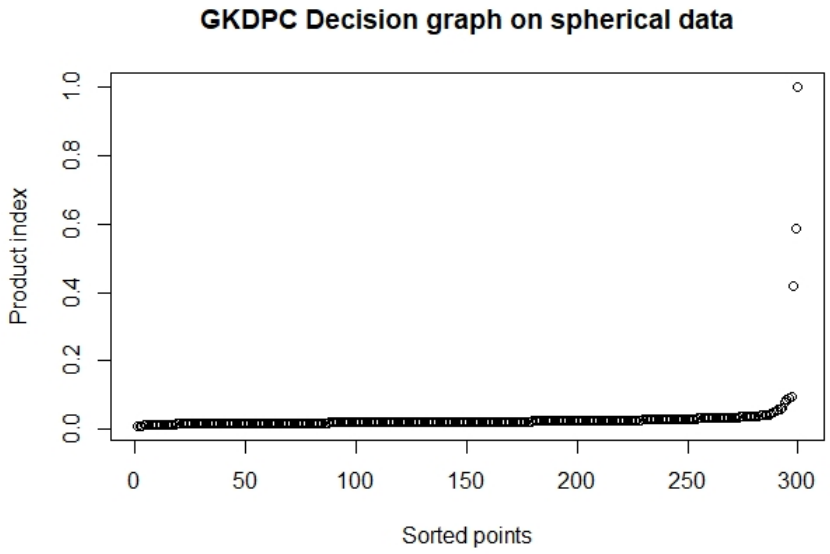


Figure 10: GKDPC Decision graph on spherical data

Figure 10 illustrates the decision graph from Gaussian kernel DPC algorithm operating on spherical data. Same as the decision graph from DPC, there are 3 obvious high product index values lying on the top-right corner of the plot. Thus it is safe to say the Gaussian DPC algorithm also states the correct number of cluster to be 3.

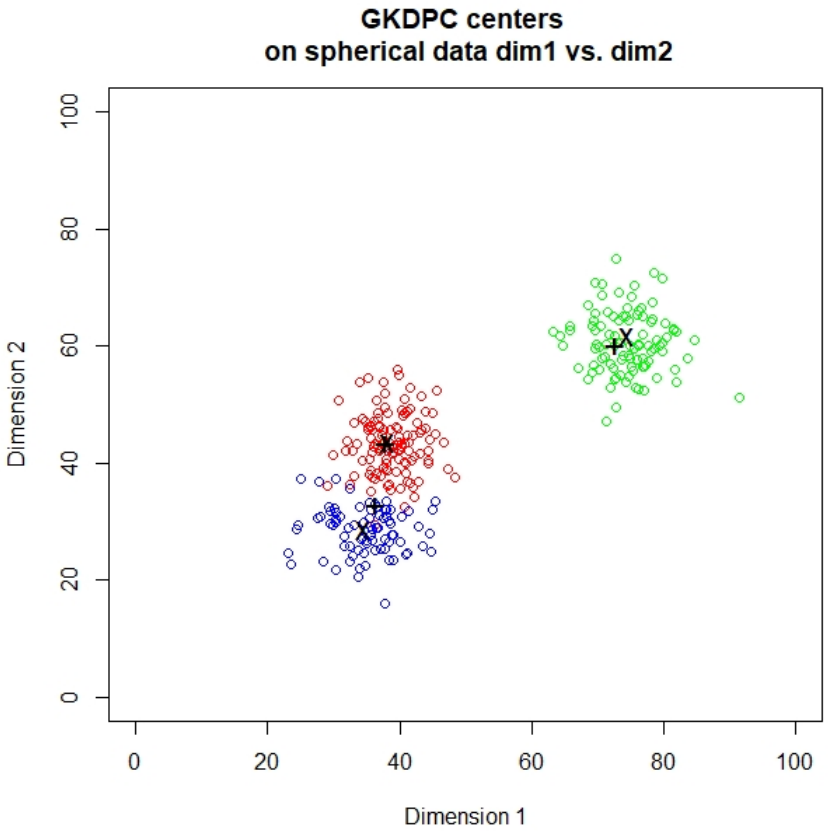


Figure 11: GKDPC clustering results on dimension 1 and 2: locations of the cluster centers. Each color represents a distinct cluster. The X-marks represent the true cluster centers. And '+' marks the estimated cluster centers.

Figure 11 and 12 illustrates Gaussian kernel DPC clustering result on spherical data on the first 2 dimensions. Again, figure 11 marks the Gaussian kernel DPC estimated cluster centers, and from the figure we can see that this cluster center has more bias compare to k-means, but has more or less the same bias as the original DPC centers.

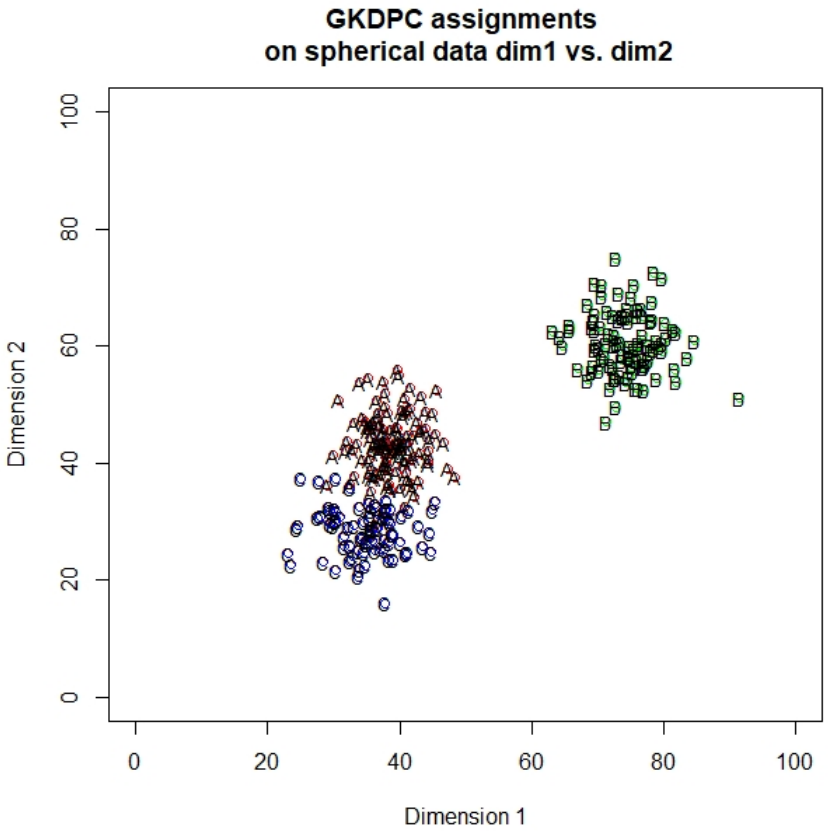


Figure 12: GKDPC clustering results on dimension 1 and 2: observation assignments. Each color represents a distinct cluster. The alphabets represents points' assignment according to the algorithm.

Figure 12 marks the assignments estimated by the Gaussian kernel DPC algorithm for the first two dimensions. Once again the result is exactly the same as the k-means, the DPC and the true clusters. From this demonstration, we could to some extent conclude that the DPC and Gaussian kernel DPC are able to produce good results just as the popular k-means algorithm on spherical data.

Drawing conclusion from observing figures is rather insubstantial. However, detailed comparison between unsupervised learning algorithms is a difficult task, a task deserves many stand alone studies. Generally, there are 3 validating categories for clustering algorithms, they are the internal criteria, the external criteria and the relative criteria[6, Section 4]. The basic idea of the relative criteria involves comparing clustering schemes and results between different clustering methods, or comparing the results from the same method but with different

parameters. In fact, the algorithm comparisons (using k-means as a benchmark to evaluate the performance DPC based algorithms) performed in this section is a form of validation based on relative criteria.

The internal criteria are the validation criteria that uses only the information within the data set, while the external criteria can be based on information outside of the data set. In practice, the internal criteria are much more useful than the external criteria, since internal criteria does not rely on any external knowledge over the data set. As a trade off the internal validations methods are often complicated and inaccurate.

Fortunately, we have full information about this synthetic data. Thus, the requirement of adopting an external criteria is fulfilled. A simple external validation tool called adjusted-Rand-index (ARI)[7] can be employed for the validation task. The main idea of ARI is to compare the similarity between two partitions. Given two partitions $P = (P_1, ..., P_p)$ and $Q = (P_1, ..., P_q)$ of a set S with n elements, and define $r_{i,j}$ as the number of elements in S that are assigned to both partitions P_i and Q_j, we will get a following contingency table:

	Q_1	Q_2	...	Q_q	Sums
P_1	$r_{1,1}$	$r_{1,2}$	...	$r_{1,q}$	a_1
P_2	$r_{2,1}$	$r_{2,2}$	...	$r_{2,q}$	a_2
...	...	...	...	...	...
P_p	$r_{p,1}$	$r_{p,2}$	...	$r_{p,q}$	a_p
Sums	b_1	b_2	...	b_q	n

Table 2: Table: Contingency table for adjusted-Rand-index

And the adjusted-Rand-index is defined as

$$\text{ARI} = \frac{\sum_{i,j} \binom{r_{i,j}}{2} - (\sum_i \binom{a_i}{2} \sum_j \binom{b_j}{2})/\binom{n}{2}}{(\sum_i \binom{a_i}{2} + \sum_j \binom{b_j}{2})/2 - (\sum_i \binom{a_i}{2} \sum_j \binom{b_j}{2})/\binom{n}{2}}, \tag{26}$$

for $i \in (1, ..., p)$ and $j \in (1, ..., q)$.

Detailed explanations of the adjusted-Rand-index is provided in the appendix section. If the two partitions are similar, the ARI value between the two partitions is closer to 1. Basically, we could use the adjusted Rand index to compare the clustering results and the true cluster groups, the clustering algorithm produces larger ARI value would be considered better algorithm model. In experimental circumstances, true cluster groups are often known. Thus ARI would be the perfect methods to evaluate the performances of different methods

or models in such circumstances.

Unsurprisingly, all k-means, DPC and Gaussian kernel DPC algorithms receive ARI=1, which means all of them produces perfect clustering assignments on the ideally generated spherical data experiment. Although the k-means algorithm is known for its efficiency in clutering spherically formed target clusters, this experiment has shown the DPC-based algorithms are also able to produce good results.

3.2 Testing clustering algorithms on synthetic non-spherical data

The aim of second test is to illustrate the advantage of DPC algorithms over k-means when dealing with non-spherical clusters. We decide to produce two high-dense concave shaped clusters enfolding each other in 2-dimensional space. In addition, an extra less-dense spherical cluster is added as a distraction.

Two concave shaped clusters are generated based on a quadratic equation. For y-axis, 301 points are generated on top of the quadratic equations $y_1 = (x_1 - 25)^2/7 + 10$ and other 301 points are generated on $y_2 = -y_1 + 70$. For x-axis, the domain on x_1 is $(10, 10.1, 10.2, ..., 40)$ and x_2 is $(25, 25.1, 25.2, ..., 55)$. After that, every point is shifted according to a Gaussian distribution on both y and x axis, with randomly chosen variance parameters.

Deploying random points around quadratic equation is not just because it creates non-spherical clusters. Another reason is that, the peak of the quadratic equation will provide the highest density among other areas. And the DPC based algorithms require such high density regions to locate the potential cluster centers.

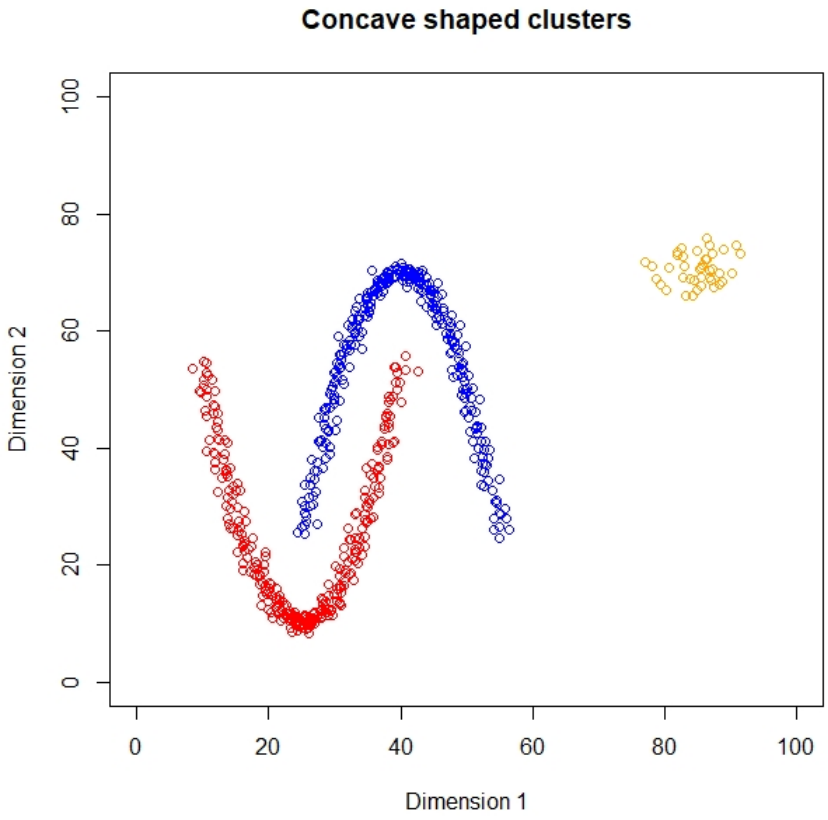


Figure 13: Synthetic concave shaped clusters. Each color represents a distinct cluster.

The synthetic concave clusters are showed in figure 13. Red dots represent points (x_1, y_1), blue dots represent points (x_2, y_2) and orange dots represent the additional spherical cluster formed by 40 bivariate Gaussian samples.

Our procedure is the same as before. We will employ k-means, DPC and Gaussian kernel DPC on this non-spherical data. Starting with the k-means, we know the correct number of clusters is 3, so we can directly insert the true hyperparameter and run the algorithm.

The k-means clustering result on the concave shaped data is illustrated in figure 14 and 15.

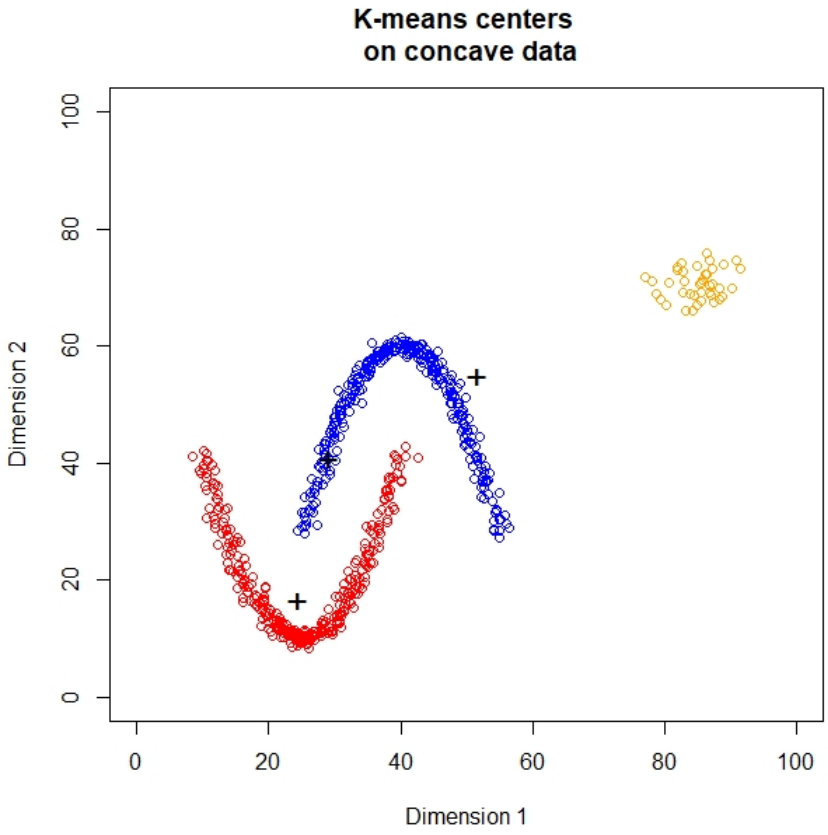


Figure 14: K-means clustering results for the concave data: locations of the cluster centers. Each color represents a distinct cluster. '+' marks the cluster centers estimated by the algorithm.

In figure 14, the estimated cluster centers are marked as '+'. There are no geometric based cluster centers for the non-spherical data. Nevertheless, the k-means algorithm produces 3 cluster center points, which makes no sense in this situation.

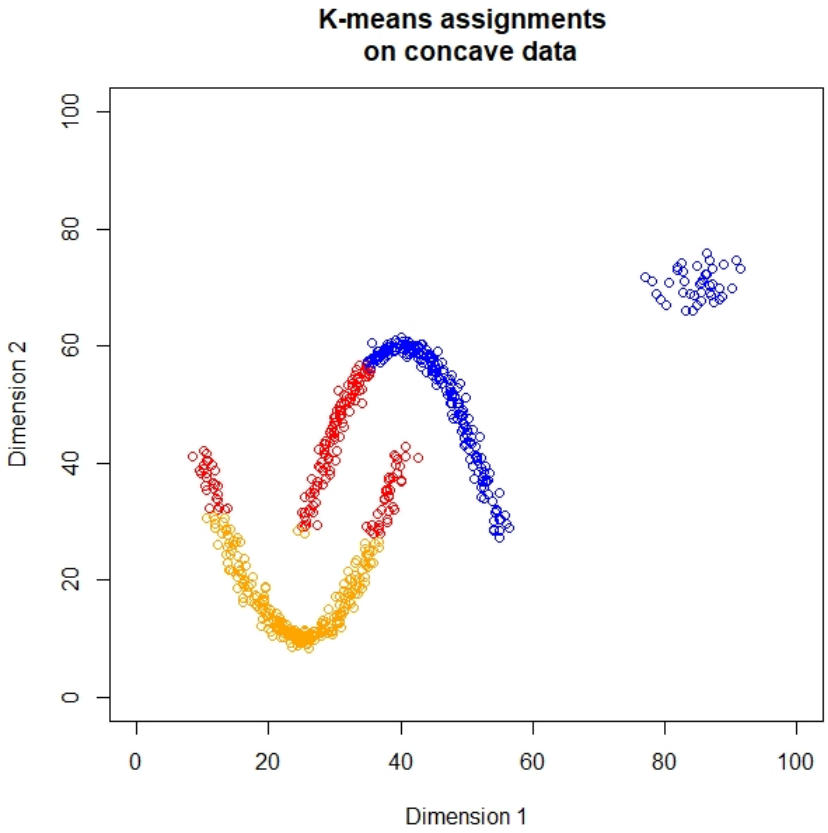


Figure 15: K-means clustering results for the concave data: observation assignments. Each color represents a distinct cluster assigned by the algorithm. The points with same color meaning they have assigned to the same cluster.

Figure 15 shows the point assignment according to k-means algorithm. Red, blue and orange each representing a distinct cluster. As we can see, each assigned cluster appears to be split into two or three pieces. Of course, this is due to the k-means property of only able to recognize spherically shaped data. Therefore, it is unsurprising to see that k-means algorithms forced two non-spherically shaped data into spherical clusters, which led to an unsatisfying result.

Next on the list is the original DPC algorithm. DPC is able to automatically calculate the best number of clusters, but it requires another hyperparameter the radius threshold to operate. In complicated situations, the selection of the threshold parameter is a difficult task. As we have mentioned, the threshold is a fixed parameter, but the data sparseness on different locations are not always

fixed. Therefore in a situation where there are multiple clusters separated in different sparseness, the DPC with an unsuitable threshold parameter will recognize the sparse clusters as part of dense neighbouring clusters.

In addition, the estimated density is defined as the number of points within the radius threshold, so it must be an integer value. The problem with an integer density value is that, there are possibilities that multiple points produce the same obviously large product index value, thus all of them will be selected as potential cluster centers. This phenomena will affect the clustering result in a negative way, that a clearly compact cluster will be recognized as multiple clusters.

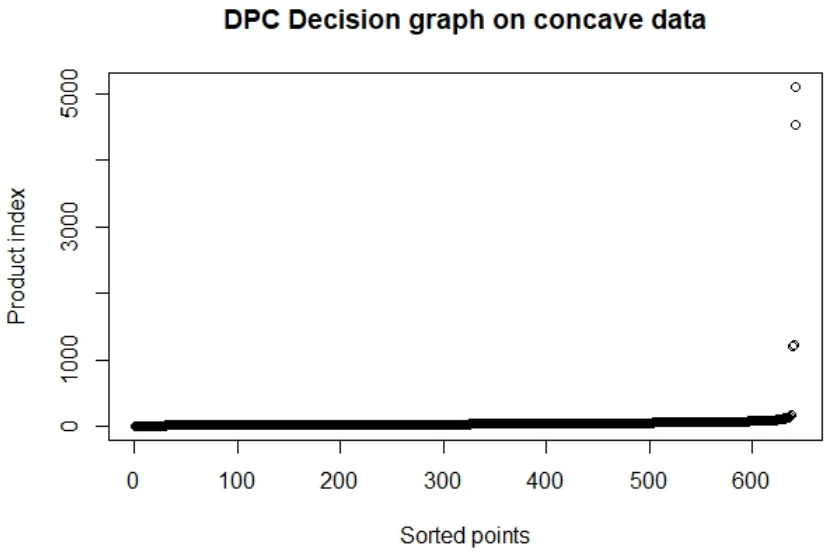


Figure 16: DPC Decision graph on concave data.

Via manual trial and error, we decided to pick the radius threshold $d_c = 5$. Figure 16 illustrates the DPC decision graph on the concave data, obtained using radius threshold $d_c = 5$. If we observe clearly, we would see there are actually 4 points significantly larger than rest of the points, since two of the points are overlapping each other in the third highest position. This is a bad sign, it tells us there is a high chance that two very similar points are both selected as potential cluster centers. Thus, a problematic clustering result should be expected.

The clustering result is illustrated in figure 17 and 18.

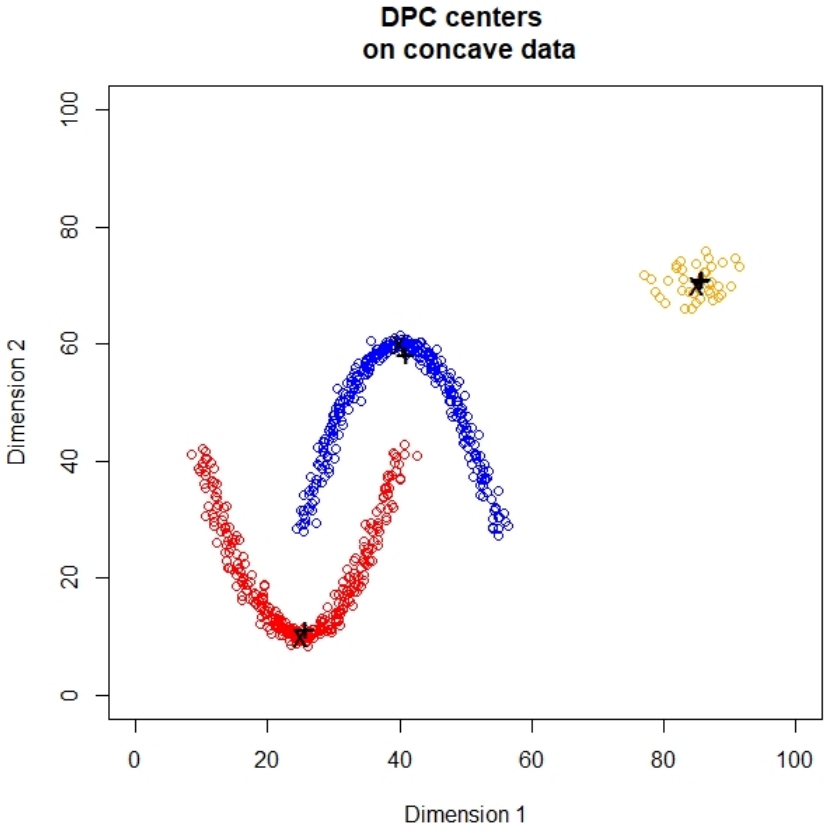


Figure 17: DPC clustering results for the concave data: locations of the cluster centers. 'X' marks the true density based cluster centers. '+' marks the cluster centers estimated by the algorithm.

There are no cluster centers for non-spherical data in the geometric sense, but cluster centers can be defined in the term of high density points. It is easy to realize the points with highest density is the peak of the quadratic equations, and let us consider such points as cluster centers. Figure 17 marks the estimated cluster centers as '+' and the density-based true cluster centers as 'X'. The DPC algorithm has located the correct centers for the concave shaped clusters. However, two estimated cluster centers has appeared in the spherical cluster lying in the top-right corner. The reason is simple, when $d_c = 5$, the highest density value of the spherical observations is shared by two points. The distance estimation for these two points are the distance between themselves and some points in the blue clusters, and these two distance estimations should be similar values. As a result both of these two points will receive similarly large values in the product index.

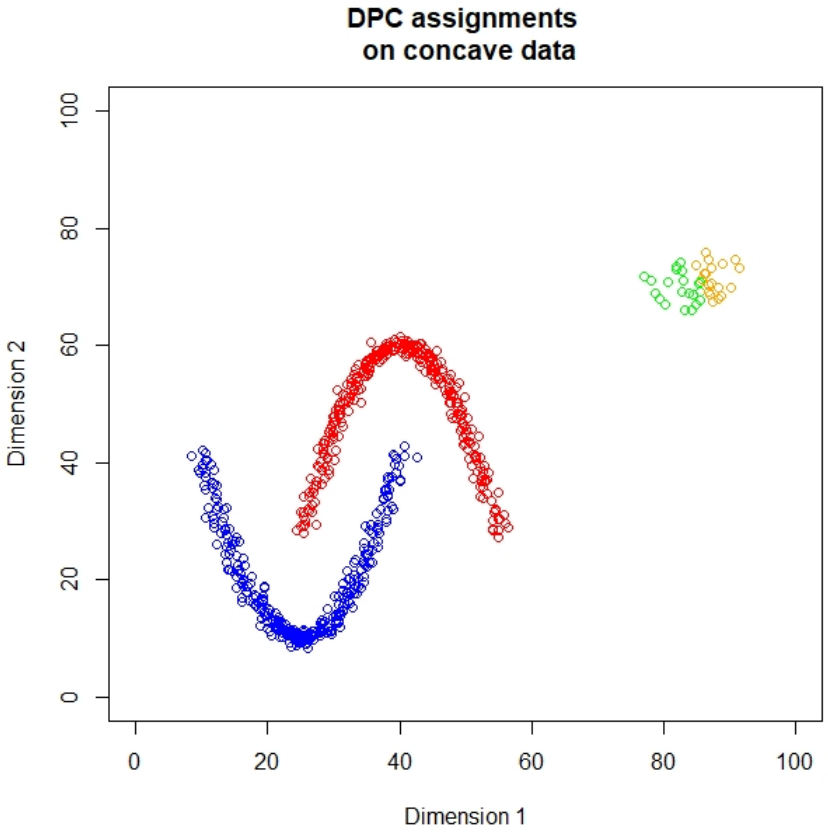


Figure 18: DPC clustering results for the concave data: observation assignments. Each color represents a distinct cluster assigned by the algorithm. The points with same color meaning they have assigned to the same cluster.

Figure 18 illustrate the DPC point assignments. The red and blue concave shaped clusters are assigned perfectly well. This indicates the DPC algorithm is much superior on non-spherical data compare to k-means. However, as we have expected, the spherical cluster on the top-right has been separated into orange and green clusters.

This particular test has demonstrated the advantage of DPC algorithm on the aspect of clustering non-spherical data. On the other hand it also pointed out the lack of robustness in complicated situations due to its "one-size-fits-all" characteristic. Let us proceed with Gaussian kernel DPC, and see if a smoother kernel function would overcome this difficulty. Gaussian kernel DPC does not

require any hyperparameter, we just need to obtain the sigma multiplier from table 1, which is around 2.5 for 2-dimensional data. Therefore we can directly run the program and obtain the result.

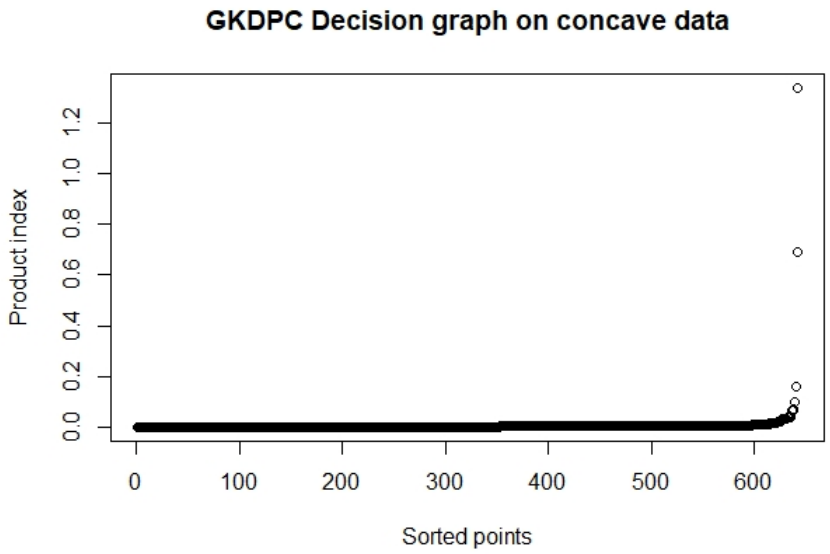


Figure 19: GKDPC Decision graph on concave data

Again, the first step is to obtain the decision graph, illustrated in figure 19. There are 2 points obviously larger than rest of the points. The third highest point is larger than rest of the points, but not in a significant level compared to the first and the second. It is debatable to consider the third point as a potential cluster center. In practice, more efforts should be put to evaluate if considering 3 cluster centers is a reasonable decision. Since we know that 3 is the true number of cluster centers, we will recognize the third highest point as one of the cluster center.

The Gaussian kernel DPC clustering results are illustrated in figure 20 and 21.

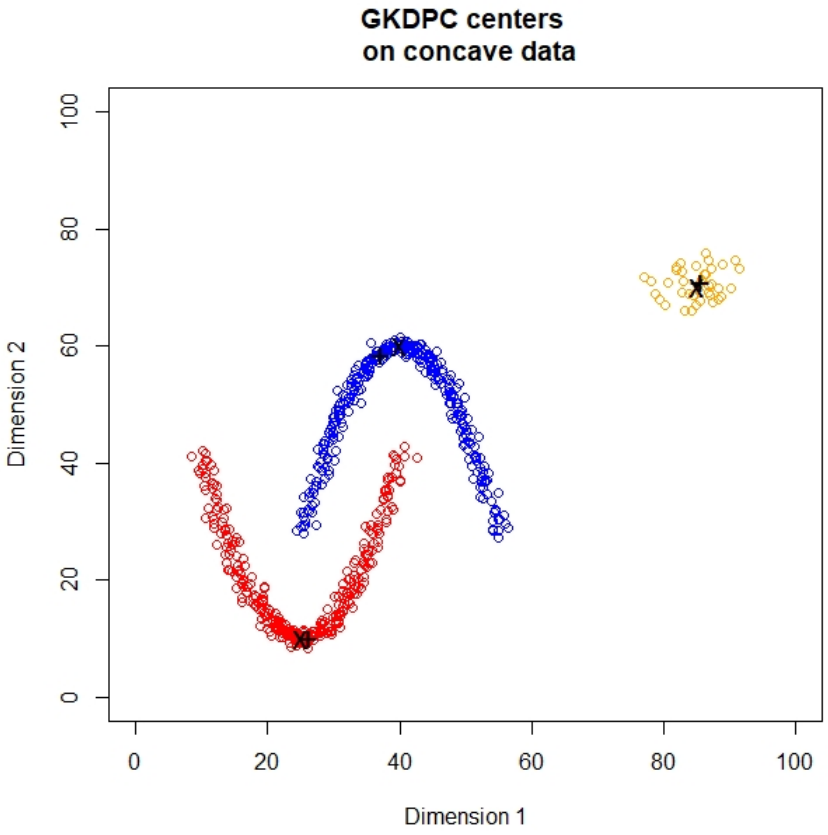


Figure 20: GKDPC clustering results for the concave data: locations of the cluster centers. 'X' marks the true density based cluster centers. '+' marks the cluster centers estimated by the algorithm.

Again, figure 20 marks the estimated cluster centers and the density-based true cluster centers. This time the algorithm has estimated a low bias cluster center on both concave clusters and spherical clusters.

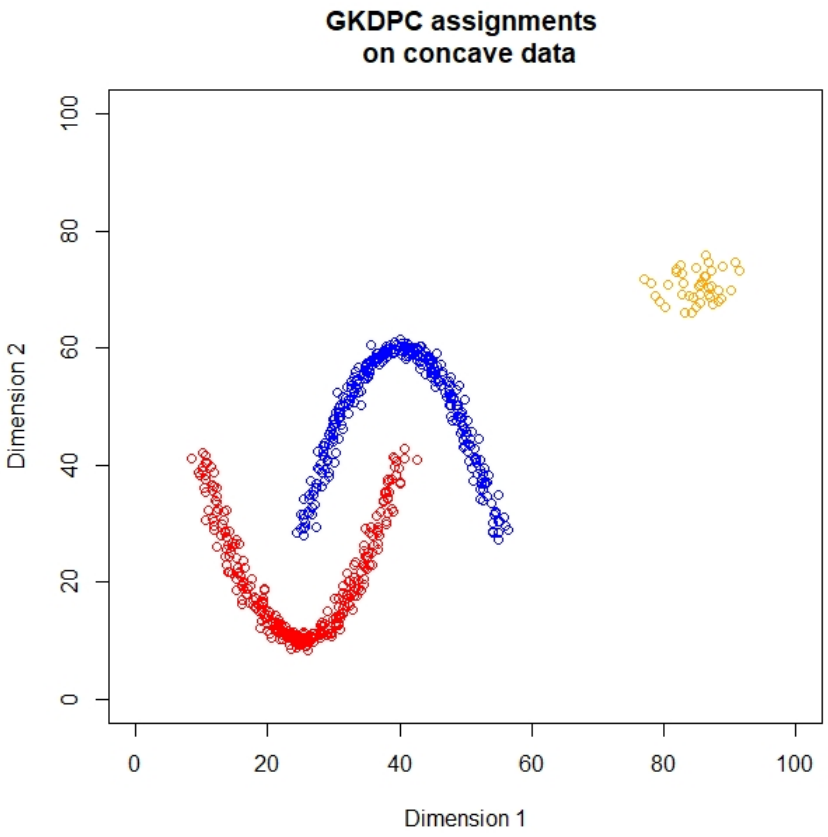


Figure 21: GKDPC clustering results for the concave data: observation assignments. Each color represents a distinct cluster assigned by the algorithm.

Figure 21 illustrates the point assignments according to Gaussian DPC. And here all points are perfectly assigned to the correct cluster. In conclusion, through this test, the Gaussian kernel DPC has proven its advantage on non-spherical data over the k-means algorithm and robustness against complicated environments over the original DPC.

Before ending this section, we could use the adjusted-Rand-index again to evaluate the clustering performances.

	ARI
k-means	0.478349
DPC	0.9960682
GKDPC	1

Table 3: The ARI value of each algorithms

According to table 3, the k-means algorithm provided the worst ARI value, this is expected since its poor performance on non-spherical data. The original DPC receives an ARI close to 1, not as good as GKDPC but good enough to be considered as a good result, although we can clearly observe the result is problematic from figure 18. This is because ARI tends to punish more on misplaced elements in large clusters, while the penalty of partition split in small clusters is not decisive.

Let us review the detail of the ARI comparison between the DPC clustering result and the true cluster groups. The partition of true cluster center is $(301, 301, 40)$, meaning the first 301 observations are assigned into cluster 1, the second 301 observations are assigned into cluster 2, and the last 40 observations are assigned into cluster 3. The partition of DPC clustering result is $(301, 301, 19, 21)$. Let us then set $(\sum_i \binom{a_i}{2} \sum_j \binom{b_j}{2})/\binom{n}{2} = E$, equation 26 for this particular scenario becomes

$$\frac{(2*\binom{301}{2}+\binom{19}{2}+\binom{21}{2})-E}{(2*\binom{301}{2})+(\binom{40}{2}+\binom{19}{2}+\binom{21}{2})/2-E}.$$

As we can see, the only different part between the numerator and denominator is $(\binom{19}{2} + \binom{21}{2}) = 381$ and $(\binom{40}{2} + \binom{19}{2} + \binom{21}{2})/2 = 580.5$, which are tiny values compare to rest of the identical parts $2 * \binom{301}{2} = 90300$ and $E = 40139.9$. This shows a limitation of ARI being not able to detect over-clustering in small clusters.

The Gaussian kernel DPC clustering result receives 1 in ARI, this simply means all points has assigned to the true cluster. In the end, the Gaussian kernel has provided the best results on all aspects.

4 Applying the Gaussian kernel DPC on real world insurance data

In the previous sections, we have showed the ability of the Gaussian kernel DPC when dealing with ideally synthetic data. The next challenge would be employing the algorithm on real world data. A situation to demonstrate clustering algorithms could be the risk classification problem in insurance industry.

It is reasonable that policy owners with different attributes would lead to different level of risks, which the risks will in long term becomes an amount of compensations. Therefore, to be able to accurately classify different customer groups is essentially important for insurance modelling. Provided with real world insurance data, containing information about policy owners and the compensation payments, the aim is to classify owners' risk, so that in further generalized linear model (GLM) modeling, people with high risk level would pay more premium, and vice versa.

In non-life insurance, a traditional risk classification modeling is demonstrated in "*Non-Life Insurance Pricing with Generalized Linear Models*" [3]. This book takes an example of motor insurance case. Basically, the actuaries have selected the relative attributes, including categorical attributes, numerical attributes and the reference attributes. The categorical attributes are formulated as tariff cells, then a GLM is operated on the numerical attributes to match the reference attributes on the corresponding tariff cells. The tariff cells produce similar GLM models will be merged. Eventually, when a new customer is willing to sign a policy, he/she will be classified into the cell according his/her attributes, and the premium will be calculated according to the GLM model of this cell.

A problem with the traditional risk classification is that, the selection and ranking of attributes are subjectively made. Also all decisions are reference based (GLM), which means that it requires external information. Hence, suitably designed, pure data-driven unsupervised machine learning algorithms would be able to provide some new perspectives to the risk classification problem.

4.1 Data explanation

The data used in this section is named "dataOhlsson", provided in "*Non-Life Insurance Pricing with Generalized Linear Models*"[3], and it can be obtained from the R-package "insuranceData". This data comes from the former Swedish insurance company Wasa, and concerns partial casco insurance, for motorcycles. It contains aggregated data on all insurance policies and claims during 1994-1998. The data contains 64548 observations, each represents the state of one policy. The states are separated into 9 attributes: owner's age, owner's gender,

geographic location (zone), vehicle class, vehicle age, bonus class, policy duration, number of claims and claim cost.

Owner's age are integer values, ranged in (0,99). Because the amount of intervals for the age variable are relatively large, this variable should be considered as continuous numerical variable. Owner's gender has only two options male (M) and female (K), it is a categorical variable. Zone has 7 levels, each representing a distinct region in Sweden, for instance "Zone1" represents the three largest cities in central Sweden. Zone is a categorical variable. Vehicle class represents the type of vehicle classified in EV ratio, EV=(Engine power in kW * 100)/(Vehicle weight in kg + 75), and then the EV ratio is classified into 7 levels, it should be classified as a categorical variable. Vehicle age are integer values, ranged in (0,99), this variable should also be considered a continuous numerical variable. Bonus class is a credibility evaluation of policy owner, it has 7 levels. An owner start with level 1, for each claim-free year the bonus class is increased by 1. And after the first claim the bonus class is decreased by 2. Bonus class is a categorical variable. Duration represents the length of policy years, measured in non-negative real numbers, and it is a continuous numerical variable. Number of claims occurred to the owner, measured in integer values, is a numerical variable. The claim cost is the amount of compensation paid to the owner, measured in non-negative real numbers, it is a continuous numerical variable.

Among the 9 variables, 5 of them are numerical variables, 4 of them are categorical variables. For the categorical variables, we could subdivide them into more detailed data-types. Needless to say, gender is a binary variable. Vehicle class and bonus class however, are unlike ordinary categorical variables. The levels for these two variables are ordered, for example, level 4 is closer to level 3 than level 1. This type of variables are called ordinal variables. And ordinal variable has another subset, the interval variable, which means numerical variable measured in interval scale. In our case, bonus class should be classified as an interval variable, because it is actually continuous time variable measured in yearly scale. It is unsure if we should classify vehicle class as an interval variable at this stage, because we do not know the EV ratio was distinguished by the intervals. For simplicity we will consider it as interval variable. The zone variable on the other hand is unlike ordinal variables, the levels in zone variable does not have an ordered relationship. This type of variables are the nominal variable.

Different variable type have different properties, and thus should be treated differently. As we have mentioned in section 2.3, the dissimilarity measurement should be defined differently for different variable types.

Variable type	Dissimilarity
Numerical	Euclidean
Interval	Euclidean
Ordinal	Manhattan
Nominal	Hamming
Binary	Hamming

Table 4: The reasonable dissimilarity measurement for each variable type

Table 4 provides the reasonable dissimilarity measurements for different variable types. It should be clear that Euclidean dissimilarity is suitable for numerical variable. For nominal and binary variables, there is no intrinsic ordering to the categories. If two points are not in the same category in on one nominal or binary variable, we only know they are different in this variable, but we do not know how much they are unlike each other. Comparing the dissimilarity between two points in nominal and binary space is equivalent to counting the number of mismatches between the two points. Thus we should use Hamming distance defined by equation 9 to be the reasonable dissimilarity on nominal and binary variables.

For ordinal variable, the differences between the levels are ordered, but the spaces between the levels are unknown. Applying Euclidean dissimilarity is equivalent to assuming the spaces in between the intervals are known to be equal and continuous, which is not reasonable at all. The meaning of applying Manhattan dissimilarity is that, we are willing to measure the dissimilarity based on the differences in levels, while not involving any presumptions on the interval space between each levels.

For interval variable, the differences between the levels are ordered, and the spaces between the levels are equal and continuous. An clear example would be a daily scaled time variable measured in years. In this case the presumptions for applying Euclidean dissimilarities are fulfilled, and this is the reason for recommending Euclidean dissimilarities for interval variable.

As we have discussed in section 2.4, the Gaussian kernel DPC is heavily dependent on the Euclidean dissimilarity. Therefore, forcing the algorithm to handle variables other than numerical and interval would cause problems, unless specific modifications are made. At the current stage the Gaussian kernel DPC algorithm is still unable to handle such a task. As a backup plan, we decide to only pick the female gender class and the first zone class (three large cities in central Sweden), and focus on the data within this particular cell which contains 1243 samples with only numerical and interval variables.

On top of the numerical variables, the claim cost and number of claims are usually the response variables (referred as labels in machine learning) in non-life

insurance pricing. The purpose of unsupervised learning is to extract information without involving any response variable, thus these two variables should also be excluded from the consideration. Eventually, there are 5 variables left to proceed with: Owner's age, vehicle class, vehicle age, bonus class and duration.

4.2 Exploratory analysis

If we are aiming to proceed with the clustering algorithm, the first step is to detect if there are clustering signatures in the data set. Therefore, exploratory analysis on the data set is required. This subsection will be mainly focused on the application of histograms and principal component analysis to gain additional knowledge on the clustering signatures of the data set.

An clock-shaped-upper-peak in the variable histogram indicates there are large number of points concentrated around this point for this variable. If there are signatures of clustering structures in overall data sets, then there should be clustering structures on some particular variables. The intention of plotting histograms is that, the clustering structures can be observed from histogram on each of the variables through frequency upper-peaks.

Figure 22: Histogram of owner's age and vehicle class

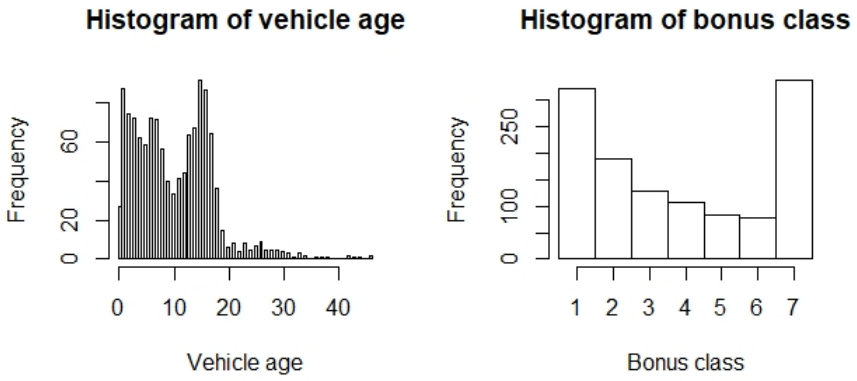


Figure 23: Histogram of vehicle age and bonus class

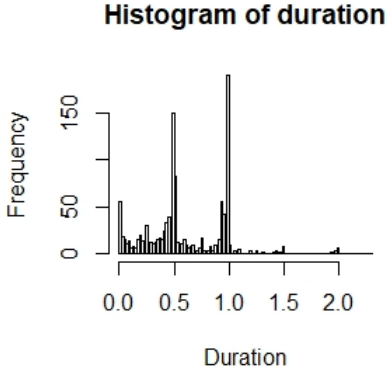


Figure 24: Histogram of duration

Figure 22, 23 and 24 illustrate the histograms for all 5 variables. According to the figure, vehicle age clearly has 2 upper-peaks, this indicates the potential to have 2 clusters. Owner's age has multiple unclear peaks, it could indicate more than 2 clusters. The bonus class variable have 2 peaks at level 1 and level 7, but the clustering behavior on interval variable is not as clear as numerical data. The duration variable has three peaks around 0, 0.5 and 1. This is due to the fact that owners tend to sign one year or half year policy.

To sum it up, the histogram does indicate some clustering structures on the data set. On top of the histograms, we will employ PCA to further investigate the clustering signatures. A short introduction about PCA is presented below.

Assume a $n * m$ centralized matrix X, the singular value decomposition (SVD) of X is be formulated as $\mathbf{X} = U\Sigma V'$, where U is the a $n * n$ orthogonal matrix, Σ is a $n * m$ non-negative rectangular diagonal matrix, and V is a $m * m$ orthogonal matrix. The SVD solution is unique if we choose the Σ so that its diagonal elements are in a descending order, $\sigma_{1,1} \geq \sigma_{2,2} \geq ... \geq \sigma_{m,m} \geq 0$. Based on this, we define principal components of X as the vector Y,

$$Y = U\Sigma' V' V = U\Sigma = XV. \tag{27}$$

The main concept of SVD is closely related to the eigendecomposition presented in section 2.4. The only practical difference is the target matrix (to be decomposed) of eigendecomposition must be a square matrix, while SVD does not have restrictions on target matrix.

Basically, PCA is defined as an orthogonal linear transformation which projects data set into a new basis, in the way that the projection causes the direction of the greatest variance is lying on the first coordinate (first principal component), direction of the second greatest variance on the second coordinate (second principal component), and so on. The purpose for employing the PCA is that, majority of the data information can be observed with first few components capturing most of the variances in the data.

In our case, observing clustering structures from multiple scatter plots is rather confusing. However, we could reduce the dimensions of our data set using PCA, then plot the first two components. Usually the first few components carries enough information and it can be used to represent the behavior of all original dimensions. If there appears multiple clusters on the first two principal components, then there will be a high possibility that the original data would also containing multiple clusters.

Before operating the PCA procedure, we should notice that there exist imbalance of variable scale. The ranges of the two age variables are (0,99), while the ranges of the interval class variables are (1,2,..,7). It is conspicuous that the variables are not in the same scale, in this case the age variables will dominate over class variables due to their large scales. Therefore, it is necessary to rebalance the weight of each dimension according to the definition of dissimilarity. As we have stated in section 2.3, if all of the variable dissimilarities are Euclidean metric based, the standardization (taking the standard score on each variable data) of data would be a good option.

Variables	PC1	PC2
Owner's age	-0.6218036	0.06696524
Vehicle class	0.1689204	-0.82548711
Vehicle age	0.4150577	-0.21994128
Bonus class	-0.5628520	-0.51278933
Duration	-0.3094364	0.05253215

Table 5: First two principal components of standardized insurance data

Table 5 displays the first 2 principal components of the standardized insurance data. Most coefficients in the first two principal components have relatively large absolute values, this means that no variable is dominating over others in term of their variances. However, the cumulative proportion of variance for the first 2 principal components is only 0.4963. In PCA, the term proportion of variance for component i is the amount of variance the i^{th} component accounts for in the data. And the term cumulative proportion between components i to j is the accumulated amount of explained variance between components i and j. The first 2 cumulative proportion equals to 0.4963 implies that these 2 principal components can only provide very limited (less than a half) information for all original variables.

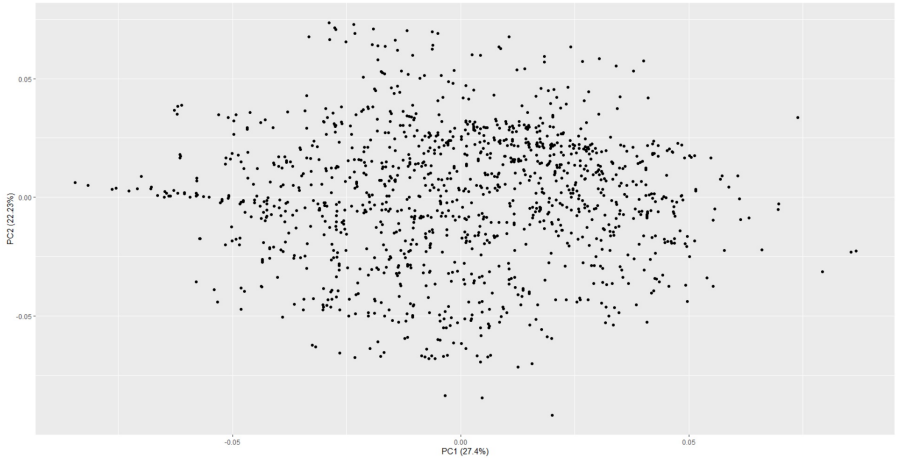


Figure 25: PCA scatter plot for arranged insurance data

Figure 25 is the scatter plot of the first 2 principal components. Unfortunately, no obvious multiple clustering structures can be directly observed from the figure. This means we should expecting only one cluster center to be found through any clustering method. This result is contradicting to the histogram analysis, since clear multiple cluster structures have appeared in histograms. I do not have any clue for this contradiction at the moment. On the other hand,

the first 2 component only contains 49.63 percent of the overall variance, which means that any conclusion drawn from this PCA result is rather weak. Therefore, the actual clustering result is still uncertain.

4.3 Clustering result on the insurance data

In this subsection the Gaussian kernel DPC will finally put to use. The standardization of data is employed. No outliers are assumed. The data set is 5-dimensional, according to table 1, the sigma multiplier defined in section 2.5 which fulfills 95 percent central cumulative probability quantile is in between 3 and 4. More detailed Monte-Carlo simulations would reveal that this value is around 3.4, and we will use this value to proceed with our GKDPC clustering algorithm. After inserting the data into the algorithm, the first step is to evaluate the decision graph.

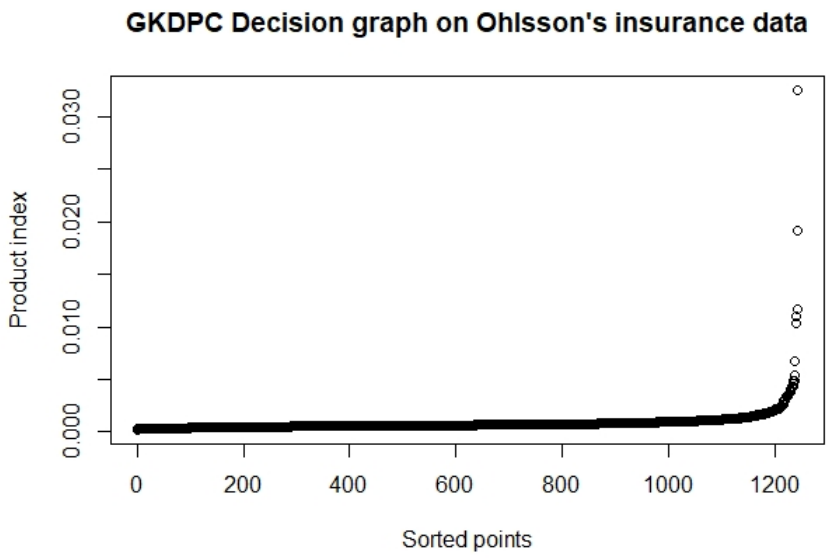


Figure 26: DPC Decision graph on insurance data

Figure 26 illustrates the decision graph of the arranged insurance data. According to the figure, There are 2 obviously large product index values and 3 relatively high product index values. These 5 points can be considered as potential center points, and thus we can confirm that 5 should be a reasonable number of clusters. Since many of the variables are interval, each of these 5 points are containing multiple observations (policies). Table 6 shows detailed

information of the clustering result.

Cluster centers	Owner's age	Vehicle class	Vehicle age	Bonus class	Duration	Assigned policies
Center1	29	3	16	1	0.975342	213
Center2	49	3	4	7	1.000000	243
Center3	43	3	6	2	0.498630	507
Center4	28	3	6	7	0.498630	92
Center5	26	5	15	1	0.504110	188

Table 6: Clustering result for insurance data. Column 1 displays the ranks of the cluster centers. Column 2-6 locate the position of the cluster centers. Column 7 shows the number of policies assigned to the corresponding cluster center.

The clustering result provided in table 6 is unsurprising. The frequency peaks showed in the histograms in figure 22, 23 and 24 have all appeared in this result, such as around 45 and 27 for Owner's age, around 15 and 5 for vehicle age, 3 for vehicle class, 1 and 7 for bonus class, 0.5 and 1 for duration.

The cluster with most policies assigned is actually center3, the third highest in the decision graph. More than 500 policies have assigned to the third cluster, which is around half of the total sample size. The cluster centers is to some extent representing the unique features of the policies which assigned to them. And this clustering results in some way says that the third cluster center contains the most common features for the overall population.

Furthermore, we can use heatmaps to illustrate interval type data in figures. Figure 27 illustrates owner's age and vehicle age data in a heatmap, the darkness of color represents the number of policies on the position according to the color-index. The cluster centers from table 6 are marked as '+' on top of the heatmap. We can see that the 5 cluster centers are representing 2 clusters in the top-left, 1 cluster in the bottom left and 2 clusters in the bottom right. Those 5 points are positioned in relatively dark regions, so the GKDPC clustering result matches our expectation.

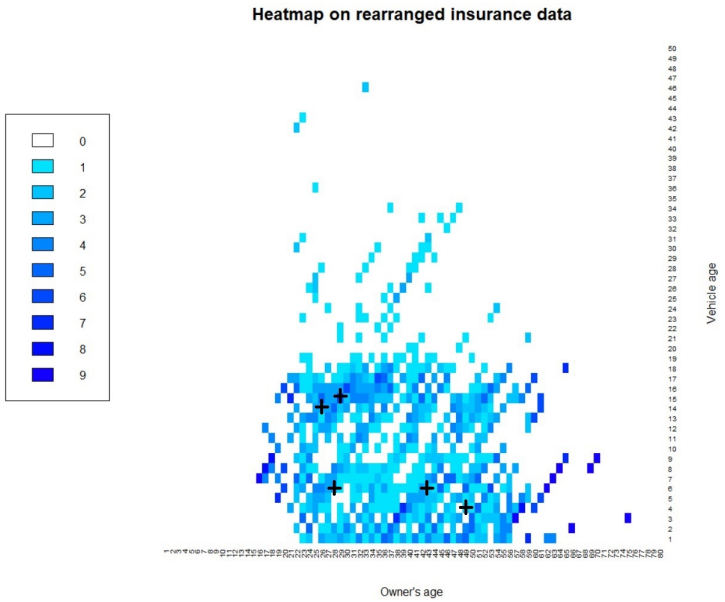


Figure 27: Heat map of age variables and cluster centers. Each position represents the number of policies with the corresponding age attributes. The number of policies is illustrated by the darkness according to the index in the left. '+' marks the cluster centers obtained using GKDPC algorithm.

Let us summarize the application of clustering algorithms on aspect of risk classification. The traditional method is to subjectively select relative attributes, and then merge the attributes which have similar GLM results. The concept of clustering algorithm on the other hand is the opposite. The clustering algorithm considers all observations in the first place, and then find a data-driven method to split them into reasonable risk groups. The usefulness of clustering algorithms are pretty straightforward: they are data-driven and relatively objective, therefore it could provide some unique perspectives which are difficult from those captured by human observations.

With the cluster centers obtained, the supervised techniques, such as GLM modelling with the reference variables as labels can be employed to gain the relationship between risk groups and the actual claim cost. Just as the traditional GLM method, the GLM modeling based on the clustered risk groups can also be used to predict the claim cost of new policies.

The validation of the clustered risk groups is a difficult task, since the requirements for the external criteria are most likely to be unavailable in practical scenarios. Nevertheless, the relative validation criteria can still be employed for this task. We could validate the performance of the clustered insurance risk groups by comparing it with the result of the traditional GLM model. The pur-

pose of risk classification is that, we would like to classify policies into groups so that the groups with higher risks (and therefore claim costs) are separated from the groups with lower risks. Thus, one comparison criteria can be set based on the sparseness of risk groups in terms of their claim costs or claim frequencies. Under this criteria, the methods which produce sparse risk groups should be considered better methods, vice versa. Another comparison criteria could be based on the prediction performances. Under this concept, the methods which predict closer to the actual claim costs (which the actual claim costs can be set by the cross-validation scheme), and less deviations to the average claim cost should be consider better methods.

The most important benefit of adopting the clustering algorithms in risk classification is that, it releases a potential to greatly expand the size of variables to be considered. With modern days technologies, more features can be extracted from each individual person. The traditional risk classification method is difficult do handle increasing number of variables. For instance, adding just an extra categorical variable indicates a large increase in the number of tariff cells for the traditional GLM modelling, thus the computational complexity will be greatly boosted. The clustering algorithms on the contrary, are more efficient when handling data sets with large amount of variables. Of course, the current GKDPC is still unable to consider many categorical variable types. As long as the limitation of mixed-type data is overcame for the GKDPC, there is no doubt the clustering algorithm would provide great values to the insurance industries. It would be really exciting to see such data-driven techniques to be applied even more in industries in the future.

5 Discussions

In this section, we shall discuss the topics that are unresolved in the main content of this article, or have the potential to be further investigated in future studies.

5.1 Density-based outlier detection

We have not included any outliers throughout this study. Of course this does not mean density based algorithms cannot deal with it. In fact, many density based algorithms are well-known for the ability to effectively and robustly detecting outliers, such as the DBSCAN algorithm. The advantage for density based algorithms on outlier detection is simple: no-matter which density estimation method we use, the points with significantly lower densities are mostly likely to be the outliers.

The only problem for the density-based algorithm in this aspect is that, most of the algorithms are not parameter-free, meaning that we still need to have some knowledge on the data in order to determine the suitable parameter. For instance DBSCAN requires the threshold parameter d_c. Of course a unsuitable hyperprameters would cause problematic outlier detection results.

As we have demonstrated, the Gaussian kernel DPC with σ_{1-NN} hyperparameter is free of parameter to some extent. This means the Gaussian kernel density could provide us with a parameter-free outlier detection alternative.

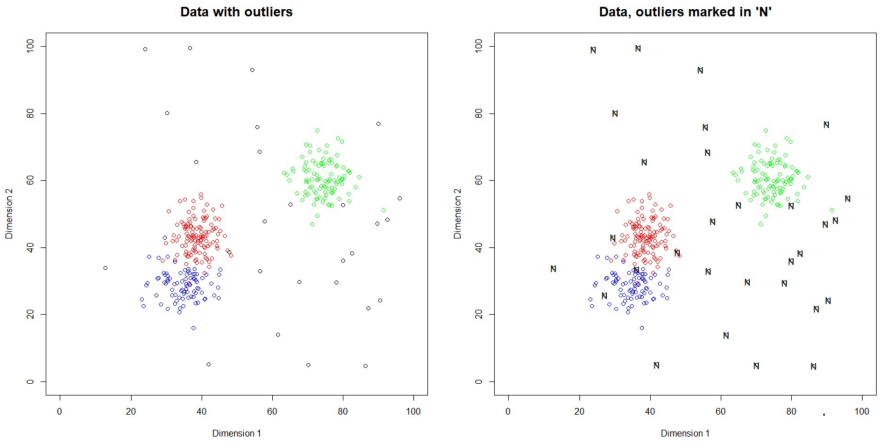


Figure 28: GKDPC based outlier detection. Left: Data with outliers added. Right: outliers detected using the Gaussian kernel density estimation. Colored points represents clustered observations and black points represents randomly generated outliers. 'N' marks the detected outliers.

Let us demonstrate a scenario by adding some random noises on the same synthetic spherical data in section 3.1. Figure 28 illustrates the data set with 30 extra randomly generated outliers (left) and the outliers detected (marked in 'N') by locating the obviously low Gaussian kernel density points (right). According to the figure, all outliers have been detected. Provided with extra modifications, the Gaussian kernel density-based algorithm has the potential to become a reliable outlier detection algorithm.

Gaussian kernel density estimator with 95 quantile 1-NN bandwidth $\hat{\sigma}_m$ performs badly in noisy data. This because in noisy circumstances, the 1-NN distances for outliers are also cosidered, and those distances are much larger than the 1-NN distances for clustered points. Therefore, if we include the outliers in our scenario, then equation 21 and 22 would likely to produce an overly large $\hat{\sigma}_m$ estimation. Hence the ideal clustering algorithm procedures should be first filtering out the outliers, and then operate the clustering algorithms. The density-based outlier detection scheme can be directly attached to the Gaussian kernel DPC algorithm or any clustering algorithms in this regard.

If the main purpose of applying outlier detection is to remove the outlier effects in the GKDPC parameter estimation, then we should not use the Gaussian kernel to define the density estimator in the first place. In this scenario, other density-based outlier detection methods such as k-NN should be applied instead of Gaussian kernel.

5.2 Limitations to the robustness and advanced data assignments

Density based algorithms are very robustness in general, but they still have their limitations. For instance, based on the concave synthesizer in section 3.2, let us move the two concave clusters toward each other, by redefining dependent variables $y_1 = ((x_1 - 25)^2)/5 + 10$ and $y_2 = -y_1 + 80$ while keeping the same independent variables. The DPC and GKDPC clustering results are displayed in figure 29.

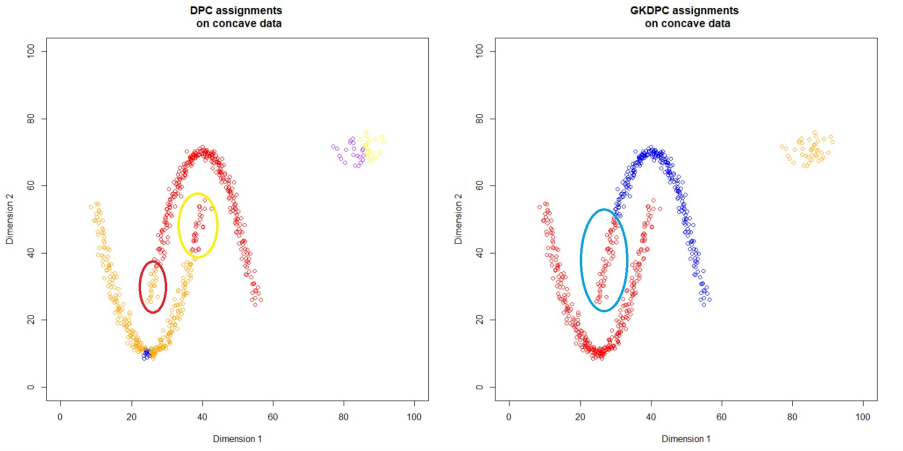


Figure 29: DPC (left) and GKDPC (right) clustering results on compressed concave clusters. Each color represent a distinct cluster. The circles mark the problematically assigned points.

From figure 29, we can see that large proportions of observations are misassigned (the points marked in circles). In order to understand the reason behind it, we should illustrate the clustering result in another alternative. According to equation 6, each point is assigned to the closest neighbour point with higher density estimations. If we link-up all these connections, this will form one spanning tree for every cluster. This way, we will be able to directly observe the problematic connections.

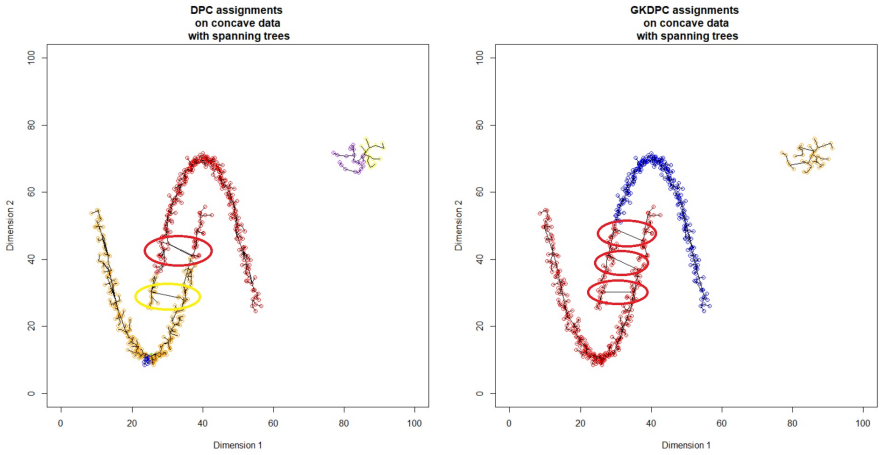


Figure 30: DPC (left) and GKDPC (right) clustering results on compressed concave clusters, with spanning trees marked as black lines. Each color represent a distinct cluster. The circles mark the problematic connections.

The spanning tree on DPC (left) and GKDPC (right) results are illustrated in figure 30. We could clearly see there are multiple mis-linked connections (marked in circles) for both results. This shows that equation 6 may cause problematic assignments in extreme situations, for instance when target clusters are closely twisted. Even if just two points are mis-connected, this would cause a problem so that many points will be mis-assigned eventually.

The solution to this limitation of robustness would be applying advanced assignment algorithms, and one of the directions would be the fuzzy-clustering. In the concept of fuzzy-clustering, the assignment of a point is presented as the probabilities of the target point belonging to each of the clusters. A related study on DPC fuzzy clustering can be found in the article "*Adaptive density peak clustering based on K-nearest neighbors with aggregating strategy*" [8]. This study used the k-NN approach to construct the fuzzy clustering for DPC. The result of this study shows that fuzzy clustering could be an alternative to enhance the robustness of DPC algorithm.

Another topic I would like to discuss is the robustness comparison between DPC and GKDPC algorithm. Although GKDPC outperforms original DPC in the concave shaped data according to figure 18 and 21. The decision graph figure 19 on the other hand shows for GKDPC algorithm, the third highest point is just barely higher than the rest of the points. It is to some extent questionable to state 3 cluster centers has been detected provided with this evidence.

One of the reason for original DPC having problematic clustering result (the sparse cluster has been splitted) in this scenario is because the "one-size-fits-

all" characteristic of the hyperparameter d_c. The GKDPC also suffers from the same problem, the Gaussian kernel bandwidth $\hat{\sigma}_m$ is also a universal parameter. A suggestion to overcome the "one-size-fits-all" problem is to customize the parameter according to local environments, so that each observation receives a unique parameter estimation.

5.3 Modifications for mixed-type data

In section 4, we have only picked the variables which can be measured in Euclidean dissimilarities. In real world circumstances, most of data are mixed-types. If a clustering algorithm is only suitable for numerical data, then the usage of the algorithm will be extremely limited. However, applying clustering algorithm on mixed-type data sets has been a difficult task, and many studies has been done about it. For instance, the k-means algorithm has its mixed-type variation called the k-prototype algorithm developed by Zhexue Huang. Detailed description can be found in his article "*Extensions to the k-Means Algorithm for Clustering Large Data Sets with Categorical Values*" [9]. The methodology for the k-prototype is that, we should split the data set into numerical part and categorical part. Let us say, dimension 1 to d_n are the numerical variables and dimension d_{n+1} to m are the categorical variables. For the numerical part, dissimilarities are presented in Euclidean distances. And for the categorical part, dissimilarities are presented as the number of mismatches (e.g. the Hamming distance). Then the two dissimilarities are combined according to

$$d_{tot}(x,y) = \sum_{d=1}^{d_n} d_{num}(x_d, y_d) + \gamma \sum_{d=d_{n+1}}^{m} d_{cat}(x_d, y_d), \tag{28}$$

where $d_{tot}(x,y)$ represents the total dissimilarity between point x and point y; $d_{num}(x_d, y_d)$ represents dissimilarity in numerical space, namely Euclidean distance; $d_{cat}(x_d, y_d)$ represents dissimilarity in categorical space, namely Hamming distance. γ is a special hyperparameter, it determines the weight which balances between the numerical variables and the categorical variables. When the total dissimilarity model is defined, then k-means algorithm and k-modes algorithm (A variation of k-mean, it focuses on minimizing dissimilarities around the mode-points instead of mean-points) are employed on numerical space and categorical space respectively.

The article "*Clustering Technique for Risk Classification and Prediction of Claim Costs in the Automobile Insurance Industry*" [10] has demonstrated an application of the k-prototype algorithm. They used an Australian automobile insurance data set, which is similar to the data used in this paper in section 4.

In their study, both numerical and categorical variables were considered, and the k-prototype algorithm has produced some fairly reasonable results.

Basically, the task for fitting the mixed-type data can be described as 3 problems. The first is to find the suitable dissimilarity measurements on different data space; the second is to determine some sensible ways to balance the spaces; the third is to construct a reasonable function, which takes the result from the first two problems as input, and processes reasonable clustering results. I have made some suggestions for future studies to modify our Gaussian kernel DPC. On the dissimilarity aspect, I would like to introducing Manhattan distance to measure the dissimilarity of ordinal variables, and introducing Hamming distance to measure the dissimilarity of nominal and binary variables.

On the weight rebalance aspect, a solution would be applying equation 12, where $\bar{d}_k$ is defined by the dissimilarity measure on the k^{th} dimension.

Lastly, we have already pointed out that the Gaussian kernel function is heavily dependent on Euclidean dissimilarities. Actually we could use other kernel functions for non-numerical or non-interval variables, as long as the kernel function catches the feature that closer points provide more contribution to the density of the target point. The term "closer" in this situation means smaller value in dissimilarity measure, which can be Euclidean, Manhattan, Hamming or any other reasonable metrics.

6 Conclusions

Training arbitrarily shaped data sets has been a difficult task for clustering algorithms. The density based algorithms are designed to overcome such difficulties, thus good clustering solutions can be obtained even in complicated environments. This study has developed the Gaussian kernel density peak clustering algorithm, a variation of the density peak clustering algorithm. The GKDPC algorithm adopts the multivariate Gaussian distribution function as the density kernel, and uses the data-based one-nearest-neighbour bandwidth as its parameter. The GKDPC algorithm has two obvious advantages compared to the other clustering algorithm counterparts: it is more robust in complicated environments and it is free of parameters.

This study has also provided two comparisons between GKDPC with the original DPC and k-means algorithms based on synthetic data sets. The spherical data experiment shows that the performance of the three algorithms are similarly good. The concave data experiment on the other hand shows that the GKDPC algorithm outperforms the other two algorithms. Moreover, a real world motor insurance data has been deployed to test the performance of the GKDPC algorithm, and the result is fairly reasonable, despite some of the limitations, such as being unable to cluster categorical variables.

Some future directions of the GKDPC include expanding its use in outlier detection, further increasing the algorithm robustness via fuzzy clustering and customizing parameters, and introducing Manhattan and Hamming distances to make the algorithm being able to cluster ordinal and nominal data.

Clustering algorithms could become helpful tools to many business industries due to their ability to easily handle large amount of variables. With reasonably designed algorithms, new machine learning techniques has the potential greatly improve the existing industrial models in the future.

References

[1] Trevo Hastie, Robert Tibshirani, Jerome Friedman.: *The Elements of Statistical learning*, Second edition. Springer, (2017)

[2] Alex Rodriguez, Alessandro Laio.: *Clustering by fast search and find of density peaks*, Science, Vol. 344, Issue 6191, page 1492-1496, (2014)

[3] Esbjörn Ohlsson, Björn Johansson.: *Non-Life Insurance Pricing with Generalized Linear Models*, Springer, (2010)

[4] Robert J. Finger.: *Foundations of Casualty Actuarial Science*, Chapter 6

[5] David W.Scott, Stephan R. Sain.: *Multi-dimensional density estimation*, Computer Science, (2004)

[6] Maria Halkidi, Yannis Batistakis, Michalis Vazirgiannis.: *On Clustering Validation Techniques*, Journal of Intelligent Information Systems, page 107–145, (2001)

[7] Lawrence Hubert, Phipps Arabic.: *Comparing Partitions*, Journal of Classification 2, page 193-218, (1985)

[8] Liu Yaohui, Ma Zhengming, Yu Fang.: *Adaptive density peak clustering based on K-nearest neighbors with aggregating strategy* Knowledge-Based Systems, Volume 133, Page 208-220, (2017)

[9] Zhexue Huang.: *Extensions to the k-Means Algorithm for Clustering Large Data Sets with Categorical Values*, Data Mining and Knowledge Discovery 2, page 283–304, (1998)

[10] Ai Cheo Yeo, Kate A. Smith, Robert J. Willis, and Malcolm Brooks.: *Clustering Technique for Risk Classification and Prediction of Claim Costs in the Automobile Insurance Industry*, International Journal of Intelligent Systems in Accounting, Finance & Management, page 39–50, (2001)

A Obtaining the numeric value of sigma multiplier using Monte-Carlo simulations

Table 1 was employed to obtain the required σ estimators in section 2.5. For columns $i = 1, ..., m$, rows $j = 1, ..., o$, the cell $C_{i,j}$ in table 1 represents the value of percentage quantile within the $j * \sigma$ radius in i-dim hypersphere.

The numerical solution for table 1 cells can be easily achieved via Monte-Carlo simulations. Let us generate n samples, each sample is generated according to the multivariate Gaussian distribution $N(\mathbf{0}, Var)$, where $\mathbf{0}$ is a vector of zeros with length i, and $Var = v * I_i$, v is an arbitrary value greater than 0, I_i is a size i identity matrix. After that, we should define an indicator function χ: $\chi(x_k, j) = 1$ if $d_{x_k,\mathbf{0}} - v * j < 0$ and $\chi(x_k, j) = 0$ otherwise, where $d_{a,b}$ is the Euclidean dissimilarity between a and b (equation 1). The total number of observations located within the j-sigma-sphere for k in $1, ..., n$ is simply $\sum_{k=1}^{n} \chi(x_k, j)$. And the percentage of observations located within the j-sigma-sphere is therefore $\frac{\sum_{k=1}^{n} \chi(x_k,j)}{n}$.

Repeat $C_{i,j} = \frac{\sum_{k=1}^{n} \chi(x_k,j)}{n}$ for data generated over dimension $i = 1, ..., m$ and sigma multiplier $j = 1, ..., o$, we will eventually obtain table 1. Moreover, we could repeat the above process multiple times, and take the average value on each cell to make the numerical solution to be more stable.

This numerical approach is able to produce a relatively good solution in lower dimensions. However, once the dimension size increases, the number of samples n needed to keep the solution stable will be dramatically increased. Of course, this is due to the curse of dimensionality, and it implies that this numerical approach will no longer be accurate in high-dimensional situations.

The method to obtain the sigma multiplier in order to fulfill the desired quantile is also not hard. For example, if we want to find the sigma multiplier that fulfils the 95 percent quantile in 5-dim. We can see from table 1 that this value is in between 3 and 4. Then we just need to generate n multivariate Gaussian samples in 5-dimensional space, insert $j = (3, 3.1, ..., 3.9, 4)$ in the $C_{i,j}$ function, and pick the j that produces the result which is closest to 0.95.

B Detailed explanations on the adjusted-Rand-index

The ARI is an extension of the Rand-index. Let us preserve the setting of adjusted-Rand-index in section 3.1, the Rand-index is simply defined as

$$R = \frac{a+b}{a+b+c+d}, \tag{29}$$

where a represents the number of pairwise elements in S that are in the same subset in P and same subset in Q; b represents the number of pairwise elements in S that are in different subset in P and different subset in Q; c represents the number of pairwise elements in S that are in the same subset in P but different subset in Q; d represents the number of pairwise elements in S that are in the different subset in P but same subset in Q. The sum $a+b+c+d$ is equal to all pairwise combinations of S, which is $\binom{n}{2}$.

The Rand-index provides an $(0, 1)$ index for two partitions of the same set. The Rand-index is close to 1 indicates the partitions are similar. And the Rand-index is close to 0 indicates the partitions are unlike. When the two partitions are exactly the same, the Rand-index is equal to 1.

Let us demonstrate a short example for the Rand-index. Assume a set $S = (1, 2, 3, 4, 5, 6)$, and two partitions $P = (P1, P2)$ where $P1 = (1, 2, 3, 4)$, $P2 = (5, 6)$ and $Q = (Q1, Q2, Q3)$ where $Q1 = (1, 2)$, $Q2 = (3, 4, 5)$, $Q3 = (6)$. According to the definition of Rand-index, the pairwise elements $(1, 2)$ and $(3, 4)$ are belonging to a; the pairwise elements $(1, 5), (1, 6), (2, 5), (2, 6), (3, 6), (4, 6)$ are belonging to b; the pairwise elements $(1, 3), (1, 4), (2, 3), (2, 4), (5, 6)$ are belonging to c; lastly $(3, 5)$ and $(4, 5)$ are belonging to d. The total number of pairwise combinations is $\binom{6}{2} = 15$, thus the Rand-index of partition P and Q is $\frac{a+b}{15} = \frac{2+6}{15} = 0.53$.

The adjusted-Rand-index is an extension of Rand-index, it adjusted the original Rank-index as the form (index - expected index)/(max index - expected index). Hubert and Arabie have showed the following in "*Comparing partitions*" [7]: 1. The Rand-index definition equation 29 can be simplified to a constant linear transformation of $\sum_{i,j}\binom{r_{i,j}}{2}$ for $i \in (1, ..., p)$ and $j \in (1, ..., q)$. 2. The expected index, $E[\sum_{i,j}\binom{r_{i,j}}{2}]$, is equal to $(\sum_i \binom{a_i}{2}\sum_j \binom{b_j}{2})/\binom{n}{2}$. 3. The ratio (index - expected index)/(max index - expected index) can be eventually formulated as equation 26.

The purpose behind such adjustment is that, the randomness within the partitions' structure is affecting validation accuracy, and therefore it is needed to subtract the expected randomness in order to remove this effect. The ARI is 1 when the partitions are identical, which is same as the original Rand-index. ARI can be negative, but negative ARI have no substantive use. If ARI=0, under the concept of external clustering validation, it means the clustering result is equivalent to a randomly assigned partition with the same structure.

Let us once again demonstrate the procedure of ARI via the same example

for Rand-index. The contingency table for the example is displayed in table 7.

	Q_1	Q_2	Q_3	Sums
P_1	2	2	0	4
P_2	0	1	1	2
Sums	2	3	1	6

Table 7: Table: Contingency table of the example for Rand-index

The index is $2 * \binom{0}{2} + 2 * \binom{1}{2} + 2 * \binom{2}{2} = 2$; the expected index is $(\binom{4}{2} + \binom{2}{2}) * (\binom{2}{2} + \binom{3}{2} + \binom{1}{2}))/\binom{6}{2} = 1.87$; the max index is $(\binom{4}{2} + \binom{2}{2} + \binom{2}{2} + \binom{3}{2} + \binom{1}{2}))/2 = 5.5$. Eventually, the ARI for partitions P and Q is $\frac{2-1.87}{5.5-1.87} = 0.037$.