Senior Thesis in Mathematics

Depth Quantile Functions in Unsupervised Learning

Author:
Guy Thampakkul

Advisor:
Dr. Gabriel Chandler

Submitted to Pomona College in Partial Fulfillment
of the Degree of Bachelor of Arts

May 4, 2023

Abstract

Many machine learning and statistical methods rely on distance as a metric for similarity and dissimilarity. However, due to the curse of dimensionality and data sparsity in high dimensional ambient space, distance metrics are not always the best distinguishing measure. Depth Quantile Functions (DQF) map data points to real-valued single variable functions on the $[0, 1]$ interval using ideas of statistical depth. DQFs have been shown to be effective and interpretable as part of algorithms for multiscale geometric feature extraction for high-dimensional and non-Euclidean data with applications in both supervised (classification) and unsupervised (anomaly detection) learning settings. The novelty of this thesis extending the uses of Depth Quantile Functions to clustering. The algorithm outlined and implemented is an user-interactive clustering technique.

Contents

Chapter 1

Machine Learning

1.1 Artificial Intelligence and Machine Learning

Broadly, artificial intelligence (AI) refers to systems or machines, often called agents, that mimic human intelligence to perform tasks and can iteratively improve themselves based on information, often in the form of data. The ideal characteristic of AI is its ability to take actions that have the best chance of achieving a specific goal given a particular environment. Examples of AI include robots, conversational agents such as Alexa and Siri, and autonomous vehicles.

Machine learning (ML) is a sub-field of AI devoted to building models that uses data to improve performance on specific tasks. Often, ML uses algorithms and statistical models to analyze and draw inference from patterns in data. Two big sub-classes of machine learning include supervised learning and unsupervised learning. Examples of machine learning include facial recognition, classification, and search engine suggestions.

1.2 Supervised Learning

Supervised learning models learn function from input-output pair data known as training data. It aims to map new inputs to outputs as accurately as possible. Uses of supervised learning models include classification (labeling unlabeled data) such as object recognition and categorization. Examples of supervised learning models include logistic regression models, support vector machines (SVMs), and k-nearest neighbors. Labeling can be separated into two main categories: classification and regression. In classification, labeling refers to assigning a class to unclassified test data. For example, given a picture of a pet, an ideal model would accurately label it *dog*, *cat*, or *others*. In regression, labeling refers to assigning a value to an unknown dependent numeric variable in test data. For example, given the location, area, and number of bedrooms, predict the price of a house.

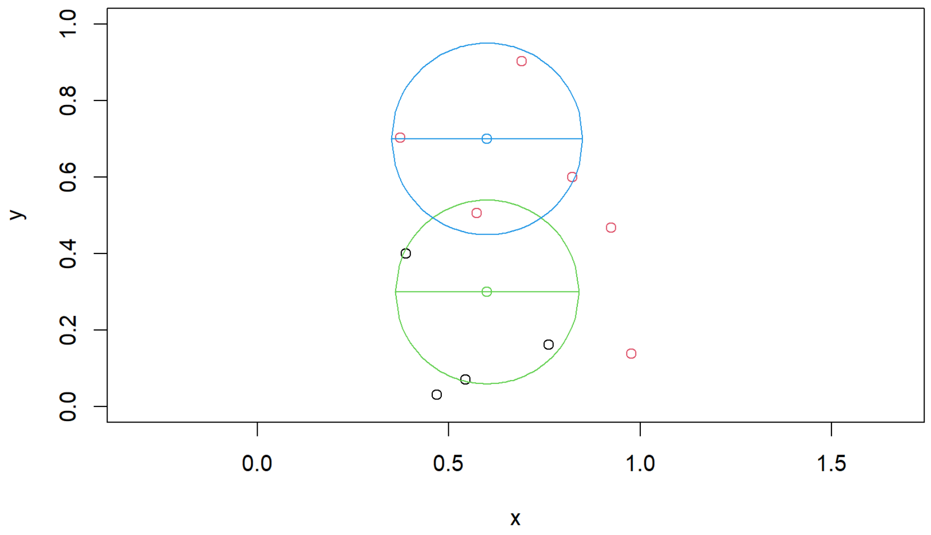
1.2.1 k-Nearest Neighbors (knn)

This section explains the work of Padraig Cunningham and Sarah Jane Delany [Cunningham and Delany, 2021] and Computational Statistics class notes of Jo Hardin (Pomona College) [Hardin, 2023].

k-Nearest Neighbor is one of the most straightforward machine learning techniques. The algorithm underlying the nearest neighbors method is predicting a value for test data based on features of their nearest neighbors in the training data set. In classification, this means labeling test data based on the majority label of their k nearest neighbors. In regression, this means predicting response variable for test data based on the average or weighted average of their k nearest neighbors. In both cases, k is a hyper-parameter, tuned to an optimal value in model training. Usually, neighbors are determined by distance between data points.

Algorithm 1 k-Nearest Neighbors

```
1. Choose a distance metric (e.g. Euclidean distance, correlation[1], etc.)
2. Compute the distances from each point in the test set to each point
   in the training set.
3. Consider a point in the test set. Find the $k$ closest points in the
   training set to the test observation using the distances computed in
   step 2.
4. For classification, find the majority class of the $k$ closest points and
   predict that class label to the test observation. For regression, predict
   the average the response variable of the $k$ closest points to the test
   observation.
```



4-Nearest Neighbor Classification in a 2D Feature Space

As an example, in the above figure, there are two classes, the black and red class, each with five data points. The green and blue data points are unlabeled. According to a 4-nearest neighbor model, the green data point will be classified as black as three out of its four nearest neighbors are black. The blue data point will be classified as red as all four of its four nearest neighbors are red. There are two steps in labeling unlabeled data in k-NN models. First is determining the data point's k nearest neighbors, and second is labeling the data either through majority voting or distance weighted voting.

More formally, suppose we have a training data set of n data points, $\mathbf{X} = \mathbf{x_1}, \mathbf{x_2}, ..., \mathbf{x_n}$. Each $\mathbf{x_i}$ is a d dimensional vector, i.e. each data point has d continuous features[2]. First, to weight all features equally, all features are normalized by z-scaling[3]. Each $\mathbf{x_i} \in \mathbf{X}$ is labeled with a label $y \in \mathbf{Y}$, which will be discrete in the classification setting and continuous in the regression setting. Our objective is to classify an unlabeled data point, $\mathbf{q}$. For each $\mathbf{x_i} \in \mathbf{X}$, the distance between $\mathbf{q}$ and $\mathbf{x_i}$ is calculated as follows:

[2]Categorical features are converted to multiple numeric binary features through one-hot encoding. Each categorical value is converted into a new column and assigned a binary value of 1 or 0. The column that corresponds to the categorical feature is assigned 1 and the rest are assigned 0.

[3]Without normalization or scaling, features with large magnitude will dominate the determination of nearest neighbors.

$$distance(\mathbf{q}, \mathbf{x_i}) = \sum_{j=1}^{d} w_j |\mathbf{q_j} - \mathbf{x_{ij}}|. \tag{1.1}$$

This is a summation over all normalized features where w_j is the weight of each feature (how important the feature is) and d is the total number of features. The k nearest neighbors are determined based on the distance metric described.

There are many ways to label $\mathbf{q}$. The most straightforward method is to weight all k neighbors equally. This means labeling $\mathbf{q}$ as the majority class of its neighbors in the classification setting and as the average of its neighbors' response feature in the regression setting. However, it may be more appropriate to assign more weight to the closer neighbors in labeling. In the classification setting, we can think about it like a voting system where the vote of the closer neighbors counts for more.

In the classification setting, each category gets assigned points and the test data point is labeled the category with the most points. We define points as follows:

$$Points(c_i) = \sum_{j=1}^{k} \frac{1}{distance(\mathbf{q}, \mathbf{x_j})^p} 1(c_i, y_j). \tag{1.2}$$

where c_i is each category label and $1(c_i, y_j)$ is 1 if c_i, the class, and y_j, the j^{th} data point's label is equivalent and 0 otherwise. p is an additional hyper-parameter which adjusts the influence of distance. A typical value for p is 1. Higher p means the influence of further neighbors are further reduced[4].

Formally,

$$label(\mathbf{q}) = \arg\max_{c_i}(Points(c_i)). \tag{1.3}$$

In the regression setting, the label is simply a weighted average of the nearest neighbors' response variable. Formally,

$$regression(\mathbf{q}) = \frac{1}{k} \sum_{i=1}^{k} \frac{y_j}{distance(\mathbf{q}, \mathbf{x_j})^p} \tag{1.4}$$

where y_i is the response variable value for the i^{th} neighbor.

[4]an alternative to using inverse distance to reduce influence of further neighbors is using an exponential function, $e^{distance(\mathbf{q},\mathbf{x_j})}$

Strengths of k-Nearest Neighbors

- Easy implementation for both categorical and continuous data sets.
- Intuitive and easy to understand.
- It can be used for any number of categories (as opposed to just binary classification).
- Any distance metric can be used.
- No training period. More test data can be added to strengthen model and model would not have to be "re-trained."
- Model is non-parametric and therefore these are no distributional assumptions on the data.
- Good model for imputing missing data as a data processing step.

Shortcomings of k-Nearest Neighbors

- One class can dominate if it has a large majority and inappropriate value of k is chosen.
- Euclidean distance is dominated by scale.
- Euclidean distance breaks down in higher dimensions (may not be appropriate for data sets with a large number of explanatory variables).
- Computationally intensive for large training data sets.
- Output does not provide information on which explanatory variables are informative.
- Sensitive to noise and outliers.

1.3 Unsupervised Learning

This section follows Jo Hardin's (Pomona College) Computational Statistics class notes [Hardin, 2023].

Unsupervised learning models are not given any training data but rather learn patterns from unlabeled data. A use of unsupervised learning models is to cluster data into categories in settings where the categories are unclear to us, humans, and therefore we cannot give the models pre-labeled data. Examples of unsupervised models include k-means clustering and hierarchical clustering. Unsupervised learning methods can be useful for describing the data and similarities of observations in the same group, as well as to find new ways to group similar observations. Clustering creates groups of observations in unsupervised settings. Another use of unsupervised learning for anomaly detection, where models are trained to look for normal behavior using an unlabeled data set. For each test point in our data, our model categorizes it as a normal data points or an anomaly. Broadly, this is done by looking at whether its features are consistent with the majority of data.

Many unsupervised learning algorithms use distance-based metrics as a measure of similarity. Mathematically, for two vectors $\mathbf{x}, \mathbf{y} \in \mathbb{R}^d$, the distance function, $d(\mathbf{x}, \mathbf{y})$ must satisfy the following properties:

1. $d(\mathbf{x}, \mathbf{y}) \geq 0$
2. $d(\mathbf{x}, \mathbf{y}) = d(\mathbf{y}, \mathbf{x})$
3. $d(\mathbf{x}, \mathbf{y}) = 0 \iff \mathbf{x} = \mathbf{y}$
4. $d(\mathbf{x}, \mathbf{y}) \leq d(\mathbf{x}, \mathbf{z}) + d(\mathbf{z}, \mathbf{y})\ \forall \mathbf{z} \in \mathbb{R}^d$

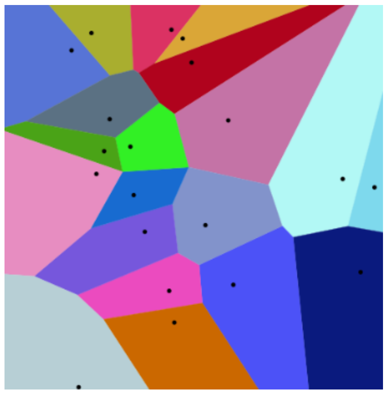
One of the most common distance metric used in machine learning is Euclidean distance:

$$d_E(\mathbf{x}, \mathbf{y}) = \sqrt{\sum_{i=1}^{d} (x_i - y_i)^2} \tag{1.5}$$

d_E satisfies all the distance metric requirements above. One obvious strength of Euclidean distance is it directly measures what is commonly understood as how far away items are from each other, which is an intuitive way of measuring similarity. However, Euclidean distance is not scale invariant; it only measures magnitude differences and not pattern differences. Due to these properties, it is sensitive to outliers.

k-means Clustering

k-means clustering is an unsupervised learning algorithm that aims to partition n observations into k clusters where eac observation belongs to the cluster with the nearest cluster center, the mean of all data points in the cluster. These mean cluster centers partition the data space into Voronoi cells. Voronoi diagrams partitions a space into regions closest to their centers. In k-means clustering, each mean cluster center has a Voronoi cell all all points in each cell is assigned to the cluster.



20 centers and their Voronoi cells

k-means clustering minimizes within-cluster variances by minimizing squared Euclidean distances. Mathematically, it is designed to find a partition of the observations such that the following objective function is minimized.

$$\arg \min_{C_1,\ldots,C_k} \left\{ \sum_{k=1}^{K} 2 \sum_{i \in C_k} \sum_{j=1}^{p} (x_{ij} - \overline{x}_{kj})^2 \right\} \tag{1.6}$$

k-means algorithm is guaranteed to converge to a local minimum of the objective function. If a point is in a different center's Voronoi cell, reassigning the point to a new cluster will lower the objective function. Means minimize squared differences, therefore taking a new mean will lower the objective function. With finitely many points, the algorithm will converge in a finite number of steps.

An upside of k-means clustering is there is no hierarchical structure[6] as points can move from one cluster to another. A major weakness in k-means clustering is that, with random assignment of initial clusters, the algorithm is non-deterministic. This also means that the algorithm may not lead to the optimal solution[7]. Other shortcomings include the user having to pre-define

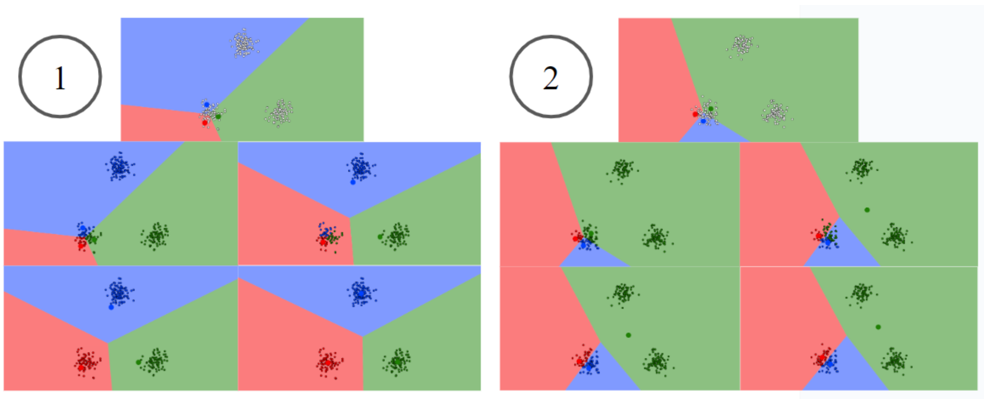
[6]Hierarchical Clustering algorithm section below

[7]The algorithm always converges to a local minimum, but not always to the global minimum

Algorithm 2 k-means Clustering

```
1. (Preprocessing) Scale/normalize each feature[5].
2. Pick a $k$ value for the number of desired clusters.
3. Randomly assign a number, from 1 to $k$, each of the observations
   which will serve as the initial cluster assignments for the observations.
4. Iterate until the cluster assignments converge.
    • For each of the $k$ clusters, compute the cluster center by taking
      the mean of all the data points in the cluster.
    • Reassign clusters to each point to the new cluster centers' Voronoi
      cells.
```

the number of clusters, k, as well as the algorithm's reliance on Euclidean distance.



k-means is non-deterministic.

In the figure above, we run the $k = 3$-mean algorithm twice with slightly different starting cluster centers and arrive at different solutions. The first arriving at the optimal solution, but the second arriving at a local minimum. The non-determinism of k-means clustering can be combated by starting the cluster centers at all $\binom{n}{k}$ combinations of k points and taking the cluster that minimizes the variances. However, that is extremely computationally intensive.

Hierarchical Clustering

This section was inspired by the book, *The Elements of Statistical Learning: Data Mining, Inference, and Prediction* [Hastie et al., 2009].

Hierarchical clustering is also known as hierarchical cluster analysis or HCA. It aims to build a hierarchy of cluster or sets of nested clusters, often organized in a dendrogram[8]. There are two main types of hierarchical clustering: agglomerative and divisive.

Alglomerative hierarchical clustering is a "bottom-up" approach where each observation starts in its own clusters. Clusters are merged as we move up the hierarchy until they are all in one cluster; clusters that are "closer" to each other based on a similarity (or disimilarity) metric. Divisive hypercritical clustering is a "top-down" approach where all observations start in one cluster. Clusters are split up as we move down the tree until all observations are in its own cluster.

Algorithm 3 Agglomerative Hierarchical Clustering

```
1. (Preprocessing) Scale the data
2. All observations start in its own cluster.
3. Compute all $\binom{n}{2}$ pair-wise dissimilarities (with a measure such as
   Euclidean distance).
4. For $i = n, n-1, \ldots, 2$:
     • Examine all pairwise inter-cluster dissimilarities among the $i$ clus-
       ters and identify the pairs of clusters that are least dissimilar.
     • Merge the two clusters; the dissimilarity between these two clus-
       ters indicates the height in the dendrogram at which the fusion
       should be placed.
     • Compute the new pairwise inter-cluster dissimilarities among the
       $i-1$ remaining clusters.
```

In order to decide which clusters should be merged or where a cluster should be split, a measure of dissimilarity between clusters of observations is required.

Some examples of distance measures (that are not necessarily mathematical metrics) include [rdo,]:

[8]A dendrogram is a diagram representing a tree

- Euclidean distance: Refer to Equation 1.6.
- Maximum distance: the maximum distance between two components of the data points

$$\max |x_i - y_i| \tag{1.7}$$

- Manhattan distance: absolute distance between two vectors

$$\sum_{i=1}^{d} |x_i - y_i| \tag{1.8}$$

- Canberra distance

$$\sum_{i} |x_i - y_i|/(|x_i| + |y_i|) \tag{1.9}$$

  However, terms with zero numerator and denominator are omitted from the sum and treated as if the values were missing.

- Minkowski distance: the p norm

$$\sqrt[p]{\sum_{i=1}^{d} (x_i - y_i)^p} \tag{1.10}$$

In addition to the metric, the user must specify a linkage criterion which informs how dissimilarity scores between clusters of points are calculated.

Without loss of generality, let A be cluster 1 and B be cluster 2. The three more common linkage types are:

- Single linkage: Distance between clusters is the distance between the closest pair of individuals from the two clusters.

$$\min_{a \in A, b \in B} d(a, b) \tag{1.11}$$

- Complete linkage: Distance between clusters is the distance between the farther pair of individuals from the two clusters.

$$\max_{a \in A, b \in B} d(a,b) \tag{1.12}$$

- Average linkage: Distance between clusters is the average distance between pairs of observations across the two clusters.

$$\frac{1}{|A| \cdot |B|} \sum_{a \in A} \sum_{b \in B} d(a,b) \tag{1.13}$$

- Centroid linkage

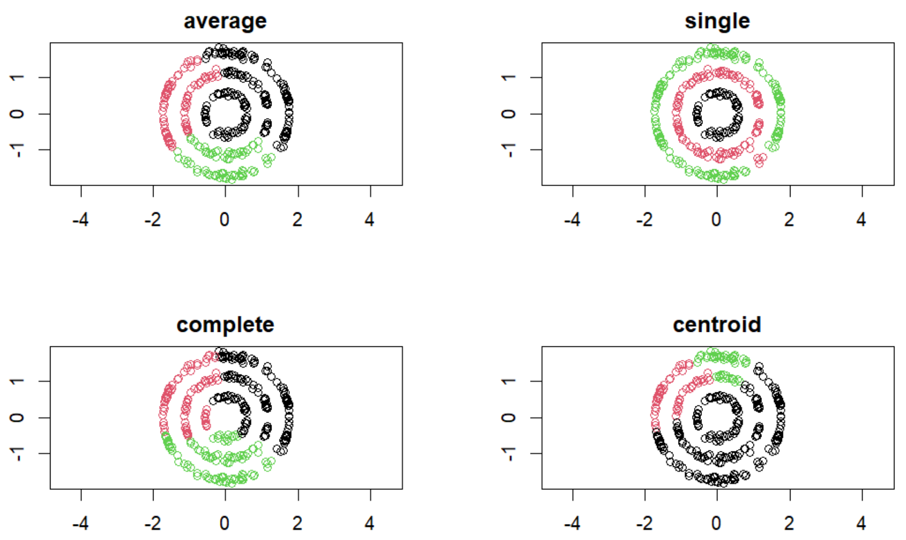
$$\|\mu_A - \mu_B\|^2, \text{ where } \mu_A \text{ and } \mu_B \text{ are the centroids}^9 \text{ of } A, B \tag{1.14}$$

- Hausdorff linkage

$$\max_{x \in A \cup B} \min_{y \in A \cup B} d(x,y) \tag{1.15}$$

Other types of linkages include weighted average linkage, median linkage, minimum increase of sum of squares linkage, Hausdorff linkage, etc.

The dissimilarity metric and type of linkage greatly affects the results of clustering.



Hierarchical Clustering with Euclidean distance metric and four different types of linkages

In figure 1.3, we see that hierarchical clustering with different types of linkages results in different clustering.

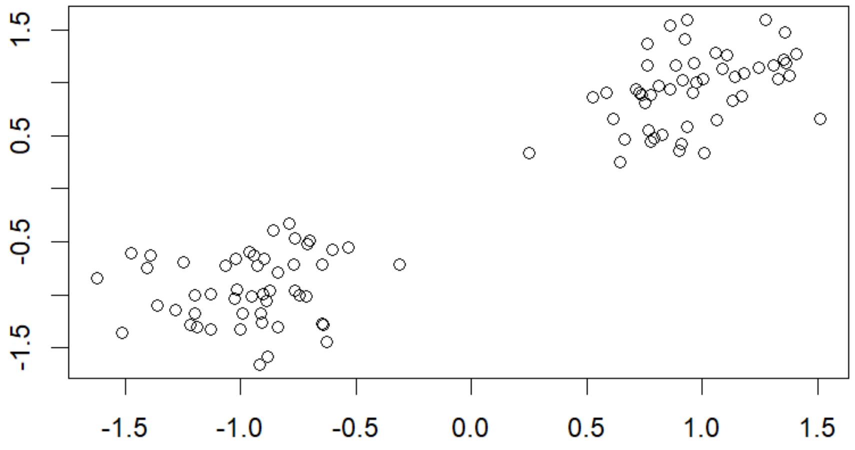
1.4 Distance in High Dimensions

This section follows the work written by James Thorn [Thorn, 2021].

Distance metrics mention in the previous sections such as Euclidean, Manhanttan, and Minkowsky, to name a few, lose their meaning in high dimensions and become unreliable similarity/disimilarity metrics. This is often refered to as the *curse of dimensionality*. In machine learning, this contradicts our intuition that more features in our data equates to better models. However, this is not the case. In fact, dimensionality reduction as a data preprocessing step can drastically alter distance-based model accuracy.

As the number of dimensions increase, data becomes more sparse. Each dimension increases the "volume" of the ambient space resulting in greater distance between data points and higher chance of differentiation. More data is often required to combat data sparsity and within group differentiation as well as maintain high accuracy and generalizability. Moreover, additional features that do not have value or differentiable qualities adds unnecessary noise to the data. As the number of dimensions increases, the average distance between two points in the data set increases and the relative difference in distances between two points decreases, leading to distances getting "blurred" and losing their meaning.

1.4.1 Simulation and Clustering Example

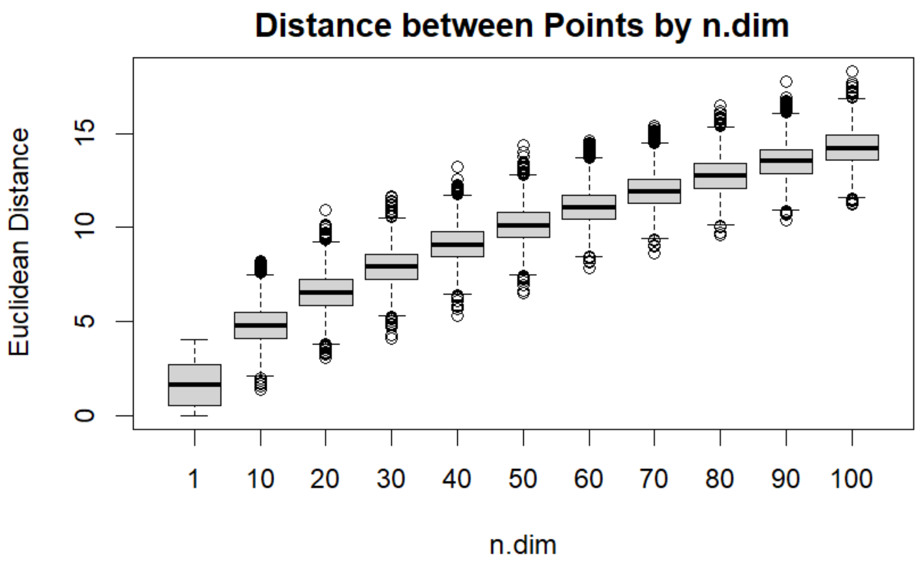


Bimodal Gaussian Distribution in Two Dimensions

We start with a bimodal Gaussian dataset ($n = 100$) in two dimensions as shown in figure 1.4.1. Each feature of the data set is simulated from two random normal distributions. The first feature of the first $n = 50$ data points were simulated from a normal distribution of mean 0 and standard deviation 1 and while that of the second $n = 50$ data points were simulated from a normal distribution of mean 6 and standard deviation 1. Data was z-scaled. We raise the dimensionality of the data through simulating addition of noisy features by adding a vector of random values generated from the standard normal. These features do not contain any information.

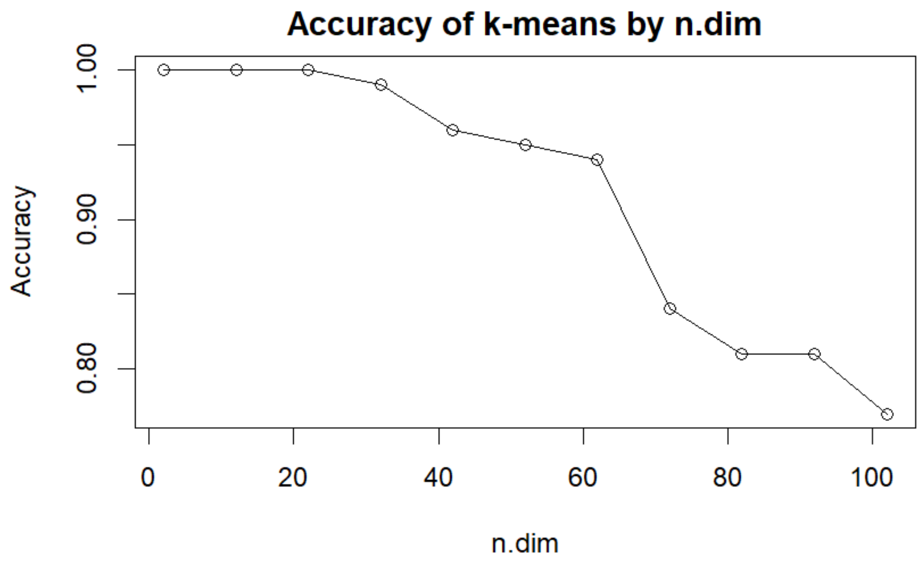
Mean and Standard Deviation of Distances between Points in High Dimensions

n.dim	Mean	SD	SD/Mean
2	1.64	1.15	0.70
12	4.79	1.01	0.21
22	6.55	1.02	0.16
32	7.94	0.99	0.12
42	9.11	0.99	0.11
52	10.15	0.99	0.10
62	11.09	0.98	0.09
72	11.96	0.95	0.08
82	12.77	0.97	0.08
92	13.53	0.96	0.07
102	14.25	0.96	0.07



Boxplots of Distances between Data points in High Dimensions

From the table and boxplots 1.4.1, we observe that the average distance between points increases as the number of dimension increases. However, the standard deviation remains approximately the same. This leads in a decrease in ratio of standard deviation to average distance, which results in differentiation between data points being more difficult. We show this in figure 1.4.1 below where the accuracy of the k-means clustering algorithm decreases as the number of dimensions increase.



Accuracy of k-means clustering in High Dimensions

1.5 On the Terms Supervised and Unsupervised Learning

This section was inspired and motivated by Gabriel Chandler (Pomona College).

towardsdatascience.com publishes that "the objective of clustering is to find different groups within the elements in the data. To do so, clustering algorithms find the *structure* in the data so that elements of the same cluster (or group) are *more similar* to each other than to those from different clusters" [Roman, 2019]. On the other hand, "Anomaly detection is the process of identifying *unexpected items* or events in data sets which *differ* from *the norm*" [Li, 2019].

The reason supervised learning and unsupervised learning are coined their respective terms is because supervised learning methods are *supervised* by labeled data, while unsupervised techniques are given unlabeled data as input.

However, from the descriptions of clustering and anomaly detection, unsupervised learning tasks, their objectives are unclear and somewhat ill-defined. In supervised settings, with labeled data, there are clear metrics for measuring similarity and dissimilarity. The algorithm's objective is clear: look for ways data points with the same label are similar and look for ways data points with different labels are different. However, in unsupervised learning, algorithms are tasked with finding *structure* and *the norm* in the data with little to no guide of how to do so.

Therefore, an argument can be made that supervised learning, while given labeled data, requires no human *supervision* as the task is clear. Conversely, because unsupervised learning tasks are often ill-defined and not constraint by labeled data, it can often be beneficial to understand why and how the algorithms arrive to their notion of *structure* and *norm.* In other words, it feels like unsupervised learning necessitates *supervision.*

Let's proceed with an analogy akin to this. Supervised learning is if a student were tasked with learning a topic in Mathematics and were given a textbook along with clear learning objectives while unsupervised learning is if a student were tasked with doing research. In the first case, the student is given a text and clear objectives akin to labeled data in supervised learning. In the latter case, the student is simply given a topic and is expected to learn by themselves. This is similar to unsupervised learning algorithms simply being given a data set with no context and tasked to makes sense of it. Thus, does it not feel like supervised learning and the student given the textbook requires less *supervision* while unsupervised learning and the student doing research require more *supervision* and constant feedback and check-ins? To me, it certainly does.

Interpretable Machine Learning

A black box is defined as a complex piece of equipment where the contents are mysterious to the user. Black box models are machine learning models that give you a result or reach a decision without explaining or showing how they did so. A classic example of a black box machine learning tool is Neural Networks.

Neural networks are extremely power machine learning technique shown to be more effective than classical machine learning algorithms in traditional machine learning tasks such as classification. However there is little to no information of how a trained model came to its conclusion. Training methods involves looking for optimal hyper-parameters rather than contributing human knowledge or intuition about the task. Naturally, this results in a tool that largely has no human involvement in the decision process as well as no contribution to human knowledge. This has led to the growth of theoretical machine learning research where mathematicians and scientists attempt to better understand the behavior of black box machine learning models as well as Bayesian models with the goal of being able to inject prior knowledge into algorithms' training process.

Interpretable machine learning models give us information and/or insight about why new data may have been sorted the way they did and can include human involvement in decision processes.

Chapter 2

Notions of Depth

2.1 Statistical Depth

This section follows the works of Dutta and Dyckerhoff [Dutta et al., 2011, Dyckerhoff and Mozharovskyi, 2016].

Data depth are methods of ordering multivariate data according to their centrality in a data cloud. Data depth has application in many fields of statistics such as multivariate data analysis, statistical quality control, classification, multivariate risk management, and robust linear programming.

2.1.1 Depth

Given a data cloud $\mathbf{X} = (\mathbf{x}_1, \mathbf{x}_2, ...\mathbf{x}_n)$ with data points $\mathbf{x_i} \in \mathbb{R}^d$, a statistical depth function assigns to an arbitrary point $\mathbf{p} \in \mathbb{R}^d$ its degree of centrality, $f(\mathbf{p}|\mathbf{X}) \in [0, 0.5]$, where 0.5 is the center.

2.1.2 Half-spaces

Half-space is a term in geometry. A half-space is either of the two parts that a hyperplane divides an affine space.

An affine space is a geometric structure similar to Euclidean space with no point of origin. Thus, it is more general than Euclidean space. No vector in an affine space has a fixed origin, therefore no vector can be uniquely associated to a point. Any vector space may be viewed as an affine space by disregarding the notion of a zero vector.

In 2 dimensions, a half-space is either of the two parts which a line divides the 2-dimensional Euclidean space. In 3 dimensions, a half-space is either of the two parts which a plane divides the 3-dimensional Euclidean space. More generally, in n dimensions, a half-space is either of the two parts which an $n-1$-dimensional hyperplane divides the n-dimensional Euclidean space.

Points that are not incident to the hyperplane are partitioned into two convex sets. Any subspace connecting a point in one half-space to a point in the other half-space must intersect the hyperplane. An affine space is convex if given any two points in the subset, the subset contains the whole line segment that joins them.

A half-space can either be open or closed. An open half-space is either of the two open sets produced by the subtraction of a hyperplane from the affine space. A closed half-space is the union of an open half-space and the hyperplane that defines it.

A half-space can be specified by a linear inequality, derived from the linear equation of the defining hyperplane. A strict linear inequality specifies an open half-space. A non-strict linear inequality specifies a closed half-space.

Open Half-space:

$$a_1x_1 + a_2x_2 + ... + a_nx_n > b \text{ or } a_1x_1 + a_2x_2 + ... + a_nx_n < b$$

Closed Half-space:

$$a_1x_1 + a_2x_2 + ... + a_nx_n \geq b \text{ or } a_1x_1 + a_2x_2 + ... + a_nx_n \leq b$$

2.1.3 Tukey's Half-space Depth

This section follows the work of Tukey [Tukey, 1975].

Tukey's depth (also known as half-space depth) is a measure of depth of a point in a fixed set of point. Given a set of points $\mathbf{X}$ in an d-dimensional space, a point $\mathbf{x}$ has Tukey depth k/n where k is the smallest number of points in any closed half-space that contains p and n is the number of points in the entire set. Tukey depth has applications in computational statistics and geometry. For example, we can measure Tukey depth of a data point relative to the data set.

Formally, the *Tukey half-space depth* of a point $\mathbf{p} \in \mathbb{R}^d$ with respect to n data points $\mathbf{x_1}, \mathbf{x_2}, ..., \mathbf{x_n} \in \mathbf{R}^d$ is defined by

$$\text{HD}(p|\mathbf{x_1}, \mathbf{x_2}, ..., \mathbf{x_n}) = \tfrac{1}{n} \inf_{p \neq 0} i|p$$

Tukey half-space depth is a commonly used depth function for multivariate data. One measure of center for multivariate data is the Tukey median, the multivariate median associated with the half-space depth.

Tukey median of a point set is a point maximizing the Tukey depth. A center point is a point of depth at least $\frac{n}{d+1}$. A Tukey median must be a center point, but not every center point is a Tukey median. In 2-dimensions, the median is the point with half-space depth 0.5.

Depth functions are often used in analyzing multivariate data as they measure the centrality of a point x with respect to a data set. They also measures probability distributions which help to define an ordering of the data points.

2.1.4 Calculations of Tukey Depth

This section follows the work of Dyckerhoff [Dyckerhoff and Mozharovskyi, 2016].

In a d-dimensional Euclidean space, a Tukey median (and therefore a center point) may be constructed in $O(n^{d-1}+nlogn)$ time. In a 2-dimensional Euclidean plane, a center point may be constructed in linear time.

Naive Method

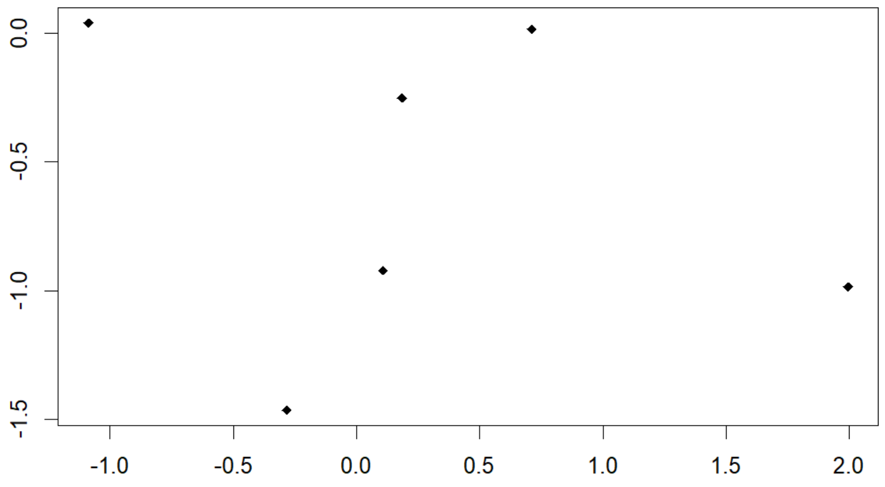
The most straight-forward, naive method to calculate Tukey half-space depth for a point, $\mathbf{x} \in \mathbf{X}$, is to start by considering all possible half-spaces with unique set of points, where the hyperplane separator contains $\mathbf{x}$. Then, determine the half-space with the lowest number of data points. The Tukey half-space depth is k/n where k is the number of data points in the said half-space, and n is the number of points in $\mathbf{X}$.

In one dimension, only one pair of half-space containing unique set of points associated with any one particular point exists. Thus, half-space depth can be calculated in $O(1)$ time. In two dimensions, $n-1$ unique pairs of set of points within half-spaces exists. These are created by linear separators with point p and $n-1$ other points. Thus, half-space depth can be calculated in $O(n-1) = O(n)$ time. In three dimensions, $\binom{n-1}{2}$ unique pairs of set of points within half-spaces exists. These are created by planar separators with point p and all $\binom{n-1}{2}$ combinations of 2 other points.

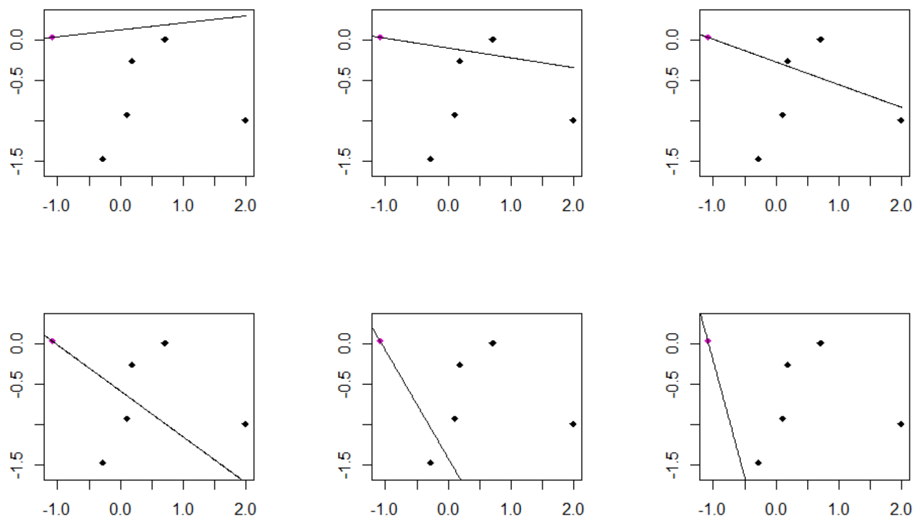
Complexity of Tukey half-space depth calculations explode as dimension of space that data points live in approaches infinity.

Sample Calculations

Let $\mathbf{X} = (\mathbf{x_1}, \mathbf{x_2}, \ldots, \mathbf{x_6})$, depicted in the figure 2.1.4 be our data set.

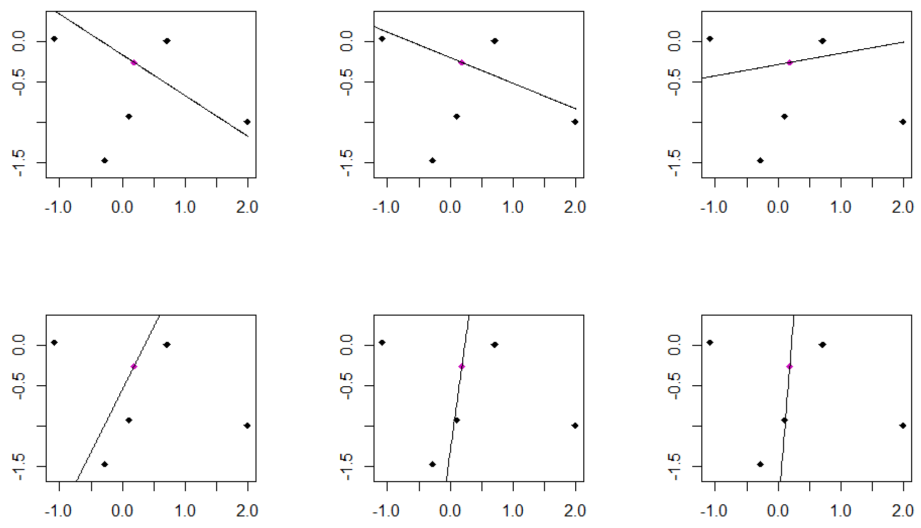


Sample Depth Calculation Data



Point 1 Sample Tukey Half-space Depth Calculation. $\mathrm{HD}(\mathbf{x_1}|\mathbf{X}) = \frac{1}{6}$

When calculating the half-space depth of point $\mathbf{x_1}$, the half-space with the minimum number of points is the top left picture in figure 2.1.4. Therefore, $\mathrm{HD}(\mathbf{x_1}|\mathbf{X}) = \frac{1}{6}$.

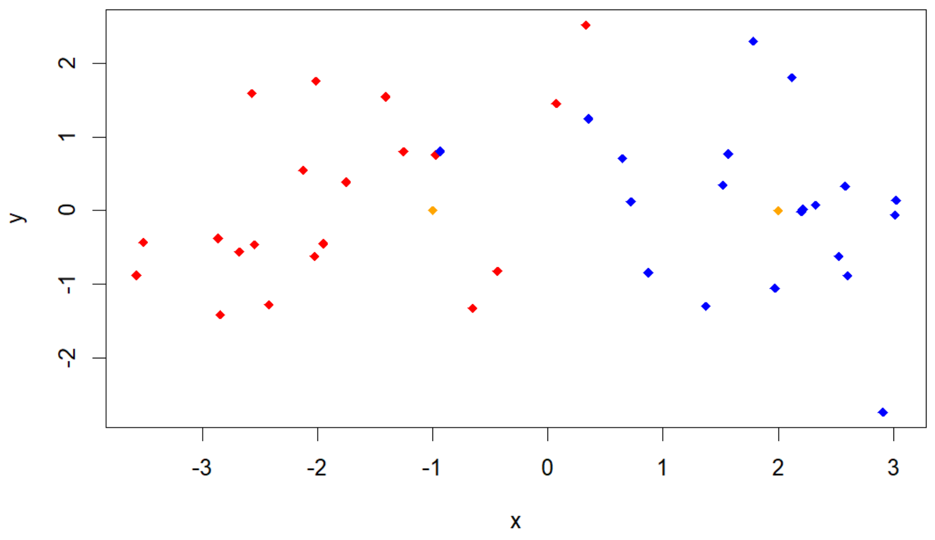


Point 2 Sample Tukey Half-space Depth Calculation. $\text{HD}(\mathbf{x_1}|\mathbf{X}) = \frac{1}{3}$

When calculating the half-space depth of point $\mathbf{x_2}$, the half-space with the minimum number of points is the bottom left picture in figure 2.1.4. Therefore, $\text{HD}(\mathbf{x_2}|\mathbf{X}) = \frac{2}{6} = \frac{1}{3}$.

2.2 Depth in Machine Learning

This section introduces statistical depth as a measure in machine learning.



Statistical Depth in Supervised Learning

In figure 2.2, we can see that the depth of the orange point on the left is higher relative to the red data than its half-space depth relative to the blue data. Intuitively, the left orange data point is deeper in the red data than it is in the blue data. Therefore, we would label that point red. Conversely, the right orange point has a higher depth relative to the blue data than that relative to the red data. Thus, it would be classified and labeled as blue.

Chapter 3

Depth Quantile Functions

3.1 Depth Quantile Functions (DQFs)

This chapter explains the works of Gabriel Chandler and Wolfgang Polonik [Chandler and Polonik, 2021, Chandler and Polonik, 2022]

Depth quantile functions is the basis for a method for extracting multi-scale geometric features from a data cloud. Each pair of data points are mapped into a real-valued feature function defined on the unit interval known as *quantile functions*. By averaging subsets of these functions, each data point is mapped on a single function that contains information about it relative to the data set called *depth quantile functions*. This methodology extracts multi-scale information about the shape of the underlying distribution based on a sample of independent and identically distributed data of size n from a Hilbert space. The approach is grounded on the idea of a distribution of local depth values of a given point.

Each feature function that corresponds to each pair of data point encode certain information about the geometry and shape of the point cloud. These geometric properties are reflected in the functions resulting in potential for machine learning algorithms that rely on them to be intuitive and interpretable. In addition, because all pairs of data points are mapped to single-variable functions on the unit interval regardless of their dimension in ambient space, these functions can be plotted and used as a visualization tool contributing to user intuition and overall interpretability.

3.1.1 Construction of DQFs

Assume that the data lie in $\mathbb{R}^d$. For each point $x \in \mathbb{R}^d$, a distribution of depth values of x is constructed. We define an anchor point in terms of two data points; this results in a matrix of feature functions where each row of the matrix corresponds to a depth distribution of a pair of points, one of which is $x \in \mathbb{R}^d$. The distributions of depths are calculated by randomly selecting subsets of $\mathbb{R}^d$ containing x and finding the depths of x within the subsets; specifically, right spherical cones are used as subsets.

By constructions, these depth distributions provide information about density in the area surrounding x with small quantiles, while large quantiles contain information about the global depth. We can think about this as looking at each data point through a traffic cone starting at the point (so we only see the point itself) and slowly walking backwards, recording how many other points we see along the way. Our first recordings contain information about density around the point, while our last recordings contain information about global depth. Intermediate scales tell us density as we move further from the point.

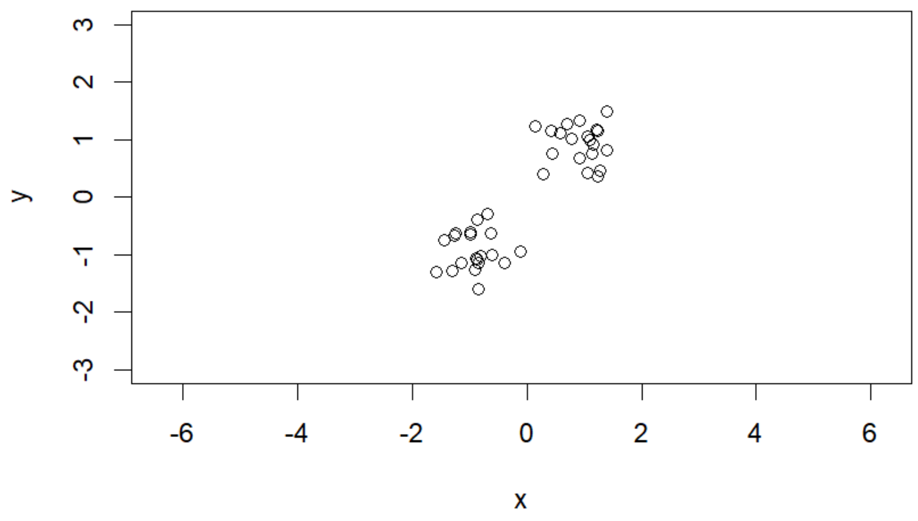
Sample Calculation

In this subsection, the calculation of a single quantile function[1] is shown.

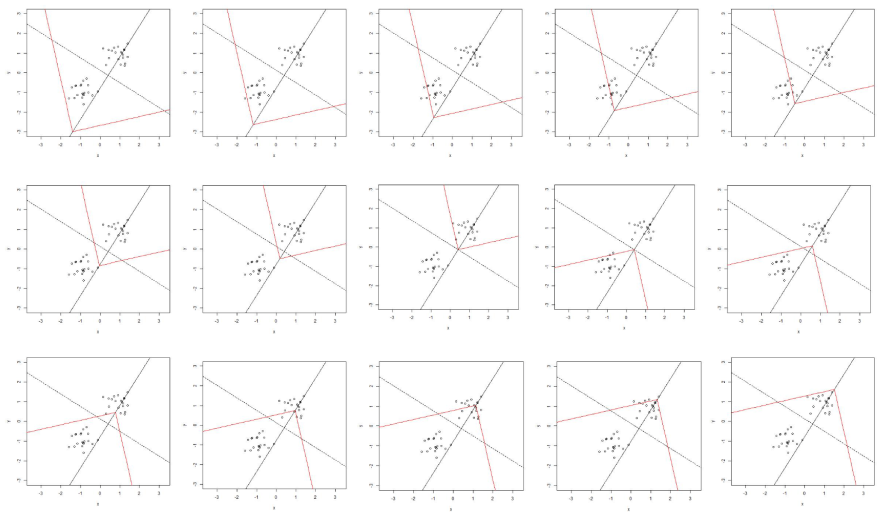
Algorithm 4 General Algorithm for Calculating Depth Quantile Functions

```
1. Measure “Conal” depths of the midpoint between x and all other
   points $\mathbf{x_i}$ at 100 positions along the line that intersects each pair of
   points. This results in $(n - 1)$ 100 depth values. (See figure 3.1.1
   below for example).
2. Sort the 100 depth values. Repeat for all $n - 1$ 100-depth vectors
   called quantile functions.
3. Average $n - 1$ quantile vectors into a depth quantile function.
4. Repeat for all $n$ points, which results in n depth quantile functions,
   one for each point.
   Applies to $nD, n > 2$ space as well.
```

[1]Each pair of data point maps to a single quantile function. Quantile functions are averaged to result in a single depth quantile function corresponding to each data point



Sample Data Set for Quantile Function Calculation



Quantile Function Calculation Cones in 2 Dimensions

Figure 3.1.1 shows a subset of cone tips on the line that intersects the pair of points of interest. For each cone tip, the corresponding depth is the minimum number of data points within the cone in the two sides of the perpendicular line through the midpoint of the line intersection. This can be extrapolated to higher dimensions, however visualizations of depth and quantile function computations are not possible.

3.1.2 Depth Quantile Functions Calculation Algorithm

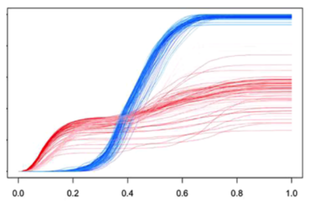
1. Instantiate *pairs* list containing all pairs of points (or random subset of pairs of points to reduce computation)
2. Calculate gram matrix for the data
   (a) z-scale the data
   (b) Select a kernel, $k(x, y)$
   (c) Compute the symmetric gram matrix, G, where $(G_{i,j} = k(x_i, x_j))$
3. Compute *splits* vector, the position of cone tips on the lines between pairs of data points (based on distribution of cone tips such as normal, uniform, etc. and g.scale).
4. Instantiate depthity vector (0 vector of length length(*splits*))
5. Instantiate *norm.k*2, *error.k*, *k.to.mid* vectors (all 0 vectors of length equal to number of data points)
6. Instantiate *depth* matrix (0 matrix with number of rows equal to length(*pairs*) and number of columns equal to length(*splits*))
7. Instantiate *qfs* (quantile functions) matrix (0 matrix with number of rows equal to length(*pairs*) and 100 columns)
8. For all pairs of points in *pairs* or index p:
   (a) Set i to index of first point in pair; set j to index of second point in pair
   (b) For all data points, k:
      i. Set $norm.k2[k] = G[k,k] + \frac{1}{4} * (G[i,i] + G[j,j]) + \frac{1}{2} * G[i,j] - G[k,i] - G[k,j]$
      ii. Set $k.to.mid[k] = G[k,i] - G[k,j] + \frac{1}{2} \cdot \frac{G[j,j]-G[i,i]}{\sqrt{G[i,i]+G[j,j]-2G[i,j]}}$
      iii. Set $error.k[k] = \sqrt{|norm.k2[k] - k.to.mid[k]^2|}$

```
    (c) For $c$ in $1 : length(splits)$
        i. Set $s < -splits[c]$
        ii. Instantiate $good$ vector (1 vector of length equal to number of data points)
        iii. $good[k.to.mid/s > 1] < -0$ (remove points on other side of the cone tip)
        iv. Set $d.to.tip$ to $|k.to.mid - s|$
        v. $good < -good * (|atan(error.k/d.to.tip| < angle/360 * 2 * pi)$ (set points outside of cones to 0)
        vi. $good < -good * (1 - 2 * (sign(k.to.mid) == sign(s)))$ (Set points on one side of cone to -1 and the other to 1)
        vii. $depthity[c] < -min(sum(good1 == -1), sum(good1 == 1))$
    (d) Set $qfs1[p,] < -quantile(depth, seq(0, 1, length = 100), na.rm = TRUE)$
9. Instantiate $dqf$ matrix (0 matrix with number of rows equal to number of data points and 100 columns)
10. For $i$ in $1 : n.obs$: set $dqf[i,]$ to element-wise average from vectors in $qfs$ that considers data point $i$)
11. Return $dqf$
```

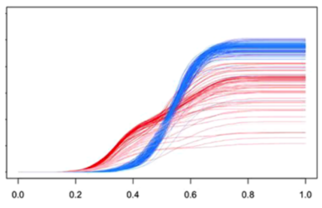
3.1.3 Applications

Classification

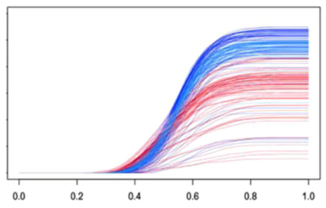
We explore the the use of depth quantile functions in two classic classification tasks: Classification of the Iris and Wine data sets. Fisher's Iris data consists of 3 classes, each with 50 observations each, all living in $\mathbb{R}^4$.



(a) Iris Data: Setosa vs. Versicolor



(b) Wine Data: Class 1 vs. Class 3



(c) Wine Data: Class 2 vs. Class 3

Average depth quantile functions for the classic Iris and Wine Data Sets [Chandler and Polonik, 2021]

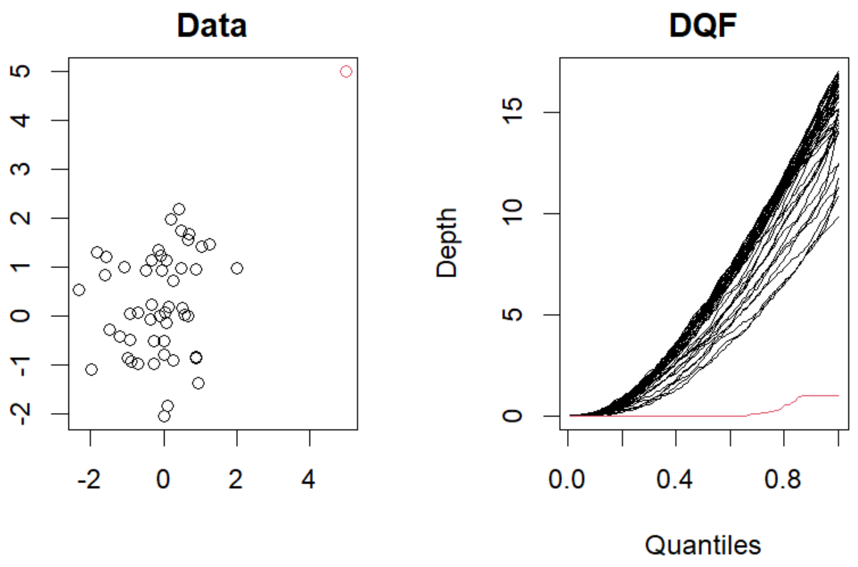
In figure 3.1.3, the red and pink functions are within group comparisons while the blue and purple between group. For Iris data, a correct classification rate of 97.33% was achieved through the leave-one-out performance evaluation method. In contrast, a k-nearest neighbors (k-NN) classifier ($k = 13$) achieves a rate of 96.67% on the z-transformed data. The left-most plot in figure 3.1.3 shows that the two point clouds not only occupy different space in ambient space, but also have relatively different geometry as well.

In all three plots in figure 3.1.3, we gain intuition on both geometry and relative positions of the classes of Iris and Wines.

Anomaly Detection

Simulated Data

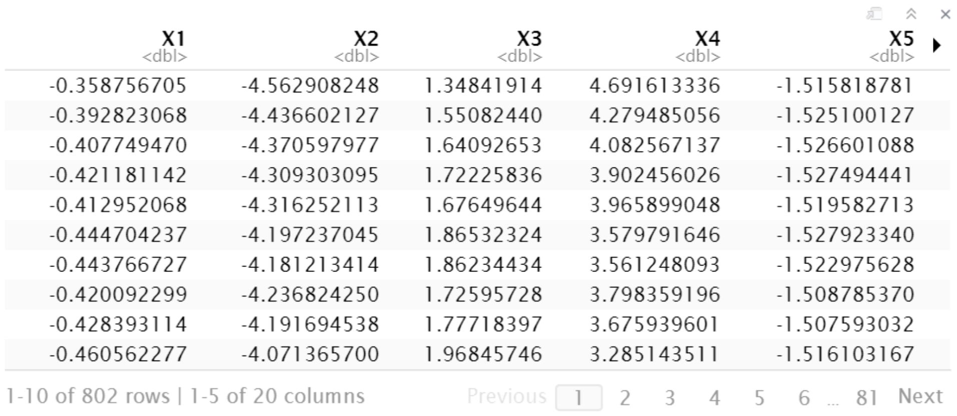
First, we look at the simplest of examples. We attempt to differentiate the point $(5, 5)$ from an $n = 50$ data set of points randomly generated from a two dimensional standard normal as shown in figure **??**.



Simple Simulated Anomaly Detection Data Set

In the DQF plot above, we can clearly see that the depth quantile function of the outlier point is different that those within the cluster. We see that it has low (zero) depth locally and some depth as the cone tips arrive at the cluster. On the other hand, the depth quantile functions of all other points within the cluster is non-zero even at low quantiles, suggesting that they are, in fact, in the cluster.

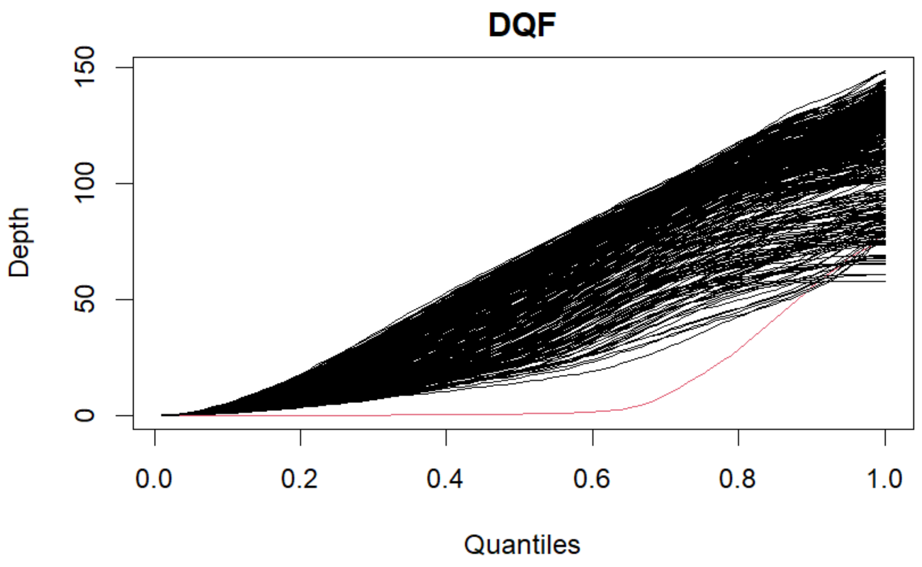
Next, we look for outliers in a 20-dimensional data set with 802 observations.



X1 <dbl>	X2 <dbl>	X3 <dbl>	X4 <dbl>	X5 <dbl>
-0.358756705	-4.562908248	1.34841914	4.691613336	-1.515818781
-0.392823068	-4.436602127	1.55082440	4.279485056	-1.525100127
-0.407749470	-4.370597977	1.64092653	4.082567137	-1.526601088
-0.421181142	-4.309303095	1.72225836	3.902456026	-1.527494441
-0.412952068	-4.316252113	1.67649644	3.965899048	-1.519582713
-0.444704237	-4.197237045	1.86532324	3.579791646	-1.527923340
-0.443766727	-4.181213414	1.86234434	3.561248093	-1.522975628
-0.420092299	-4.236824250	1.72595728	3.798359196	-1.508785370
-0.428393114	-4.191694538	1.77718397	3.675939601	-1.507593032
-0.460562277	-4.071365700	1.96845746	3.285143511	-1.516103167

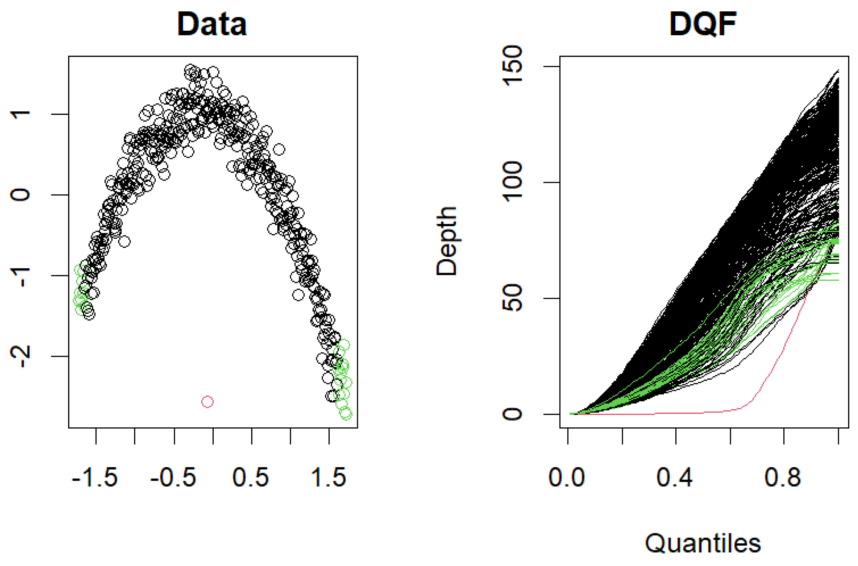
1-10 of 802 rows | 1-5 of 20 columns   Previous 1 2 3 4 5 6 ... 81 Next

High dimensional Anomaly Detection Data Set



DQFs of High dimensional Anomaly Detection Data Set

In figure 3.2, we detect one outlier. This is what we expected as the data set above is, in actuality, data simulated from a parabola with one point living off of the manifold as shown in figure 3.1.3, but matrix-transformed into 20 dimensions.

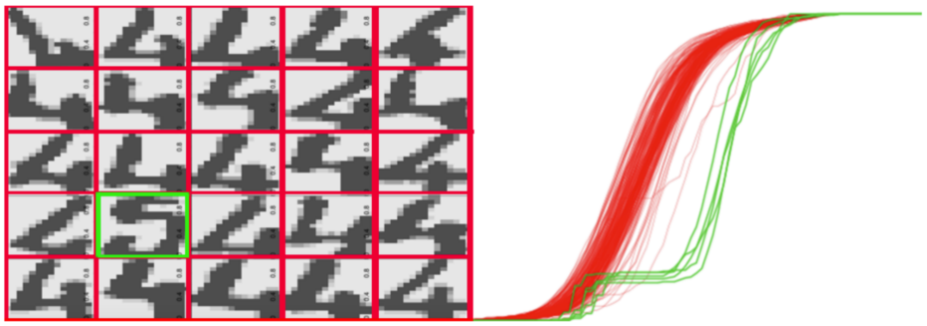


DQFs of High dimensional Anomaly Detection Data Set

To gain further intuition of the interpretation of depth quantile functions, we look at the green points in the data that lie on the edge of the parabola. Their depth quantile functions are also colored green in the DQF plot. We see that those functions are very similar in shape as the black functions that correspond to other data points on the parabola (at least in the [0,0.5] interval). However, all the green functions are mostly in the lower half in terms depth at all quantiles. This is because our cone tips move away from the midpoints in both directions and in one direction, there are no data points. Therefore, when the depth quantile functions corresponding to the pairs of points that involve edge points are averaged, the result is relatively low depth compared to averages of pairs of points that are "deep" in the data set (not on the edge). Another characteristic to note of the DQFs of the green edge data points is the leveling off of the functions as the quantiles approaches 1. This property follows from the same reason described.

Real Data

Now, we consider a real data set, the *multiple features* data set. The data set consists of $d = 649$ features of handwritten digits, with 200 observation for each digit. This data set was subsetted into one of size $n = 205$ with 200 "4" digits and the first 5 "5" digits.



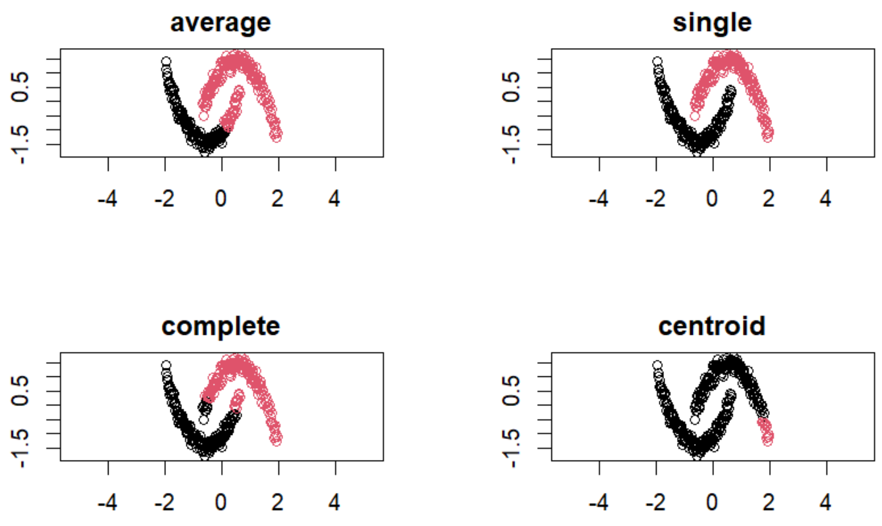
[Chandler and Polonik, 2022]

We can see in figure 3.2 that the depth quantile functions corresponding the the "5" digits are clearly differentiable from the functions corresponding to the "4" digits. When looking at the plot, identifying the "5" digits as outliers is fairly straightforward.

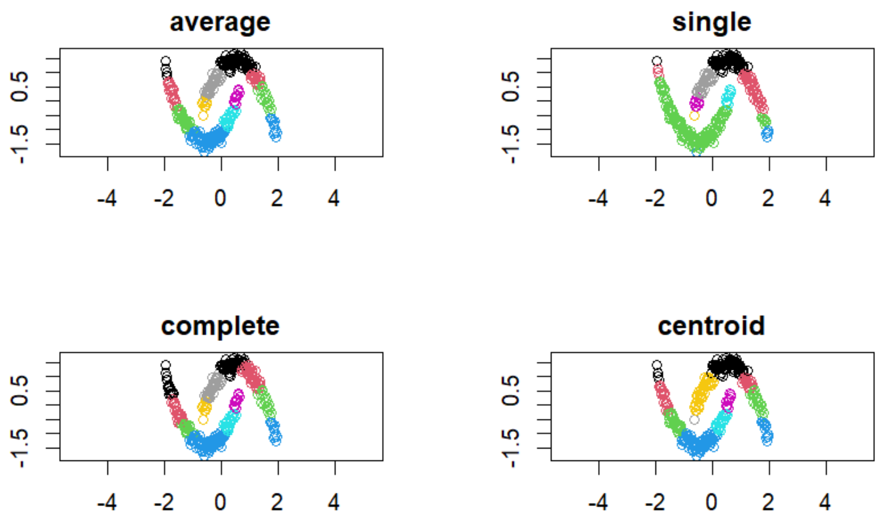
3.2 DQFs for Clustering

Previously, depth quantile functions are shown to be effective in classification and anomaly detection tasks. This section outlines a novel interactive clustering algorithm by adapting the DQF-anomaly detection algorithm. As described in Chapter 1, many clustering algorithms that rely on distance-based metrics perform poorly on high dimensional and geometrically irregular data. In figure 1.3, we see differing results of different clustering algorithms, many of which are incorrect.

The idea behind the DQF Clustering algorithm is to first use a standard clustering algorithm, such as hierarchical clustering, to perform clustering on a data set. However, we greatly overshoot the number of clusters, then rely on the DQF-anomaly algorithm algorithm to combine clusters. In clustering, the two possible errors an algorithm can make are to assign data points that belong in different clusters to the same cluster or to assign data points that belong in the same cluster to different clusters. By overshooting our desired number of clusters, we relax our expectations for standard clustering algorithms. Now, assigning data points that belong in the same cluster different clusters is no longer incorrect. The only error is assigning data points that belong to different clusters the same cluster. Increasing the number of clusters significantly reduces settings where standard clustering algorithms would make this error.



Only one type of hierarchical clustering technique results in a error-free clustering.



By increasing the number of clusters, all types of hierarchical clustering technique results in error-free clustering.

With more clusters than desired, we need to combine some of these clusters. We turn to the DQF-anomaly algorithm to determine whether each pair of clusters should be combined starting with the closest clusters. Here, we defined the distance between two clusters as the distance between the two closest points across the clusters. We then run the DQF-anomaly algorithm on the closest point from the second clusters with all the data points in the first cluster and run the algorithm again on the closet points from the first cluster with all the data points from the second cluster. Next, we determine whether the points outside the clusters are anomalies to each other by examining the two resulting DQF plots. If the two points are not anomalies, we combine the two clusters. We repeat this process until all the closest points are anomalies to all the other clusters.

The challenge to this algorithm is the computational intensity of the DQF-anomaly algorithm. In order to combine clusters with the DQF-anomaly algorithm, fast calculations of depth quantile functions of subsets of the data is critical. This results in an additional novelty presented in this paper: the DQF-subset algorithm. The DQF-subset algorithm allows us to calculate depth quantile functions for any subset of the data while only going through the computationally intensive parts of the DQF-anomaly algorithm once.

3.2.1 DQF Subset

Extracting Information from Pairs of Points

The computationally intensive aspect of depth quantile functions calculation is the computation of the gram matrix and, even more so, the calculation of where individual data points are relative to the ($length(splits)$) cone tips resulting from each pair of points in the data set. The original `dqf.anomaly` stores these relative positions in *good* vectors indexed by data point where 0 indicates that the data point is outside the cones tips (and therefore not considered in depth calculation) and 1/-1 to indicate which side of the midpoint the data points within the cones are on[2]. `dqf.subset` organizes a list of *ret.good* matrices corresponding to each pair of points. Each row of a *ret.good* matrix corresponds to a *good* vector that contains information on each data point's relative position to a cone tip. This collection of *good* vectors results in extraction of depth quantile functions of subsets of the data without going through the computationally intensive procedure of recalculating each data point's relative to the $length(splits)$ cone tips for all pairs of points. With this information, extracting depth quantile functions for each pair of points is the same as in `dqf.anomaly`, but with *ret.good* matrices corresponding to pairs of points including unwanted points removed. For the *ret.good* matrices that remain, the indices of unwanted data points in the *good* vectors are removed.

Algorithm for `dqf.subset`

1. Instantiate *pairs* list containing all pairs of points (or random subset of pairs of points to reduce computation)
2. Calculate gram matrix for the data
    (a) z-scale the data
    (b) Select a kernel, $k(x, y)$
    (c) Compute the symmetric gram matrix, G, where ($G_{i,j} = k(x_i, x_j)$)
3. Compute *splits* vector, the position of cone tips on the lines between pairs of data points (based on distribution of cone tips such as normal,

[2]Depth is then computed by taking the minimum between the number of '1s' and '-1's in the *good* vectors

uniform, etc. and g.scale).

4. Instantiate $norm.k2$, $error.k$, $k.to.mid$ vectors (all 0 vectors of length equal to number of data points)
5. **(new)** Instantiate $ret.pairs$ matrix (0 matrix with number of rows equal to nrow($pairs$) and 2 columns)
6. **(new)** Instantiate $k.to.mids$ matrix (0 matrix with number of rows equal to nrow($pairs$) and number of columns equal to number of data points)
7. **(new)** Instantiate $ret.gs$ matrix (0 matrix with number of rows equal to length($splits$) and number of columns equal to number of data points)
8. **(new)** Instantiate $ret.goods$ list (an empty list)
9. For all pairs of points in $pairs$ or index p:
    (a) Set i to index of first point in pair; set j to index of second point in pair
    (b) **(new)** Set p^{th} row of $ret.pairs$ matrix to $[i, j]$.
    (c) For all data points, k:
        i. Set $norm.k2[k] = G[k,k] + \frac{1}{4} * (G[i,i] + G[j,j]) + \frac{1}{2} * G[i,j] - G[k,i] - G[k,j]$
        ii. Set $k.to.mid[k] = G[k,i] - G[k,j] + \frac{1}{2} \cdot \frac{G[j,j]-G[i,i]}{\sqrt{G[i,i]+G[j,j]-2G[i,j]}}$
        iii. Set $error.k[k] = \sqrt{|norm.k2[k] - k.to.mid[k]^2|}$
    (d) **(new)** Set p^{th} row of $k.to.mids$ matrix to $k.to.mid$ vector
    (e) For c in $1 : length(splits)$
        i. Set $s < -splits[c]$
        ii. Instantiate $good$ vector (1 vector of length equal to number of data points)
        iii. $good[k.to.mid/s > 1] < -0$ (remove points on other side of the cone tip)
        iv. Set $d.to.tip$ to $|k.to.mid - s|$
        v. $good < -good * (|atan(error.k/d.to.tip| < angle/360 * 2 * pi)$ (set points outside of cones to 0)

vi. $good < -good * (1 - 2 * (sign(k.to.mid) == sign(s)))$ (Set points on one side of cone to -1 and the other to 1)

vii. **(new)** Set c^{th} row of $ret.gs$ matrix to the $good$ vector

10. Return $ret.pairs$, $ret.goods$, $k.to.mids$, $splits$

3.2.2 Algorithm for `dqf.clustering`

Inputs: `data`, `dqf.s` (object returned from `dqf.subset`), n.clusters (desired number of initial clusters), and all other inputs to `dqf.anomaly`

1. Scale the data

2. Compute initial clusters with a standard clustering algorithm (like hierarchical clustering). Store the cluster assignments in the *clusters* vector (length equal to number of data points)

3. Compute $inter.dists$ matrix, an $n.clusters \times n.clusters$ inter-cluster distance matrix where the distance between cluster i and cluster j is the distance between the minimum distance between a points from cluster i and a point from cluster j. Set $inter.dists[i, i]$ to ∞.

4. Set $closest.pts$ matrix, $ann.clusters \times n.clusters$ matrix where $M_{i,j}$ is the index of the point (within data set) in cluster j that is closest to cluster i.

5. Instantiate $combined.clusters$ list, a list of vectors containing which clusters have been combined ($CC[i]$ contains clusters that have been combined with cluster i[3].)

6. Instantiate $max.dist$, $max.dist = \max(inter.dists) + 1 < \infty$.

7. Calculate $dqf.subset$.

8. Determine combination of clusters

   (a) Use $inter.dists$ matrix to determine the two closest clusters. WLOG, call them C_1 and C_2.

[3]If clusters 1,4,7 have been combined, then $CC[1] = CC[4] = CC[7] = [1, 4, 7]$

(b) Use *closest.pts* matrix to determine the points from clusters 1 and 2 closest to C_2 and C_1 respectively. Call them, pt_1 and pt_2.

(c) Use *dqf.subset* to extract two sets depth quantile functions: First, the DQFs of points in C_1 and pt_2. Then, the DQFs of points in C_2 and pt_1.

(d) Determine whether those two clusters should be combined by examining the two the DQF plots.

- If yes, update *combined.clusters* list accordingly. Set $inter.dists[i, j]$ to $\forall i, j \in combined.clusters[i]$. We do not need to look to combine any of the combined clusters anymore.
- If not, set $inter.dists[i, j]$ to $\forall i, j \in combined.clusters[i]$. We do not need to look to combine any of the combined clusters anymore.
- To delay the decision[4], set $inter.dists[i, j]$ to $max.dist$ $\forall i \in combined.clusters[i], j \in combined.clusters[j]$ to revisit combining these clusters after all other pairs of non-combined clusters have been visited.

9. Modify *clusters* according to *combined.clusters* list.

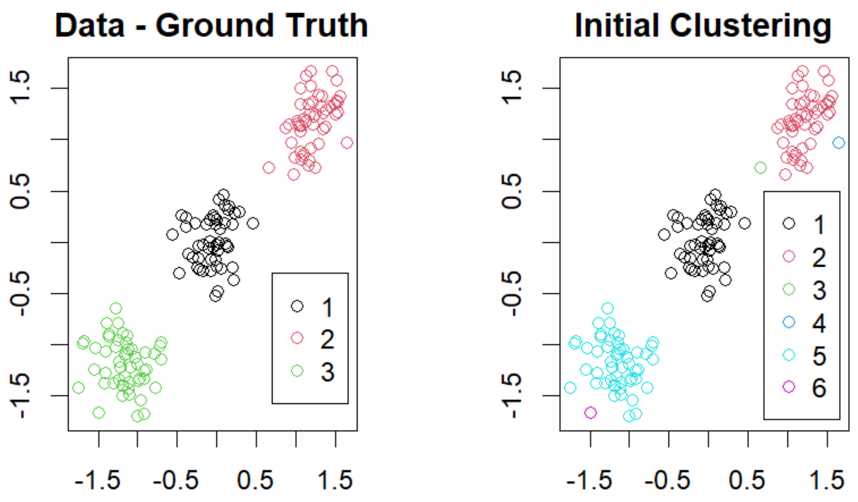
10. Return *clusters*.

3.2.3 Simulated Examples

Three Clusters

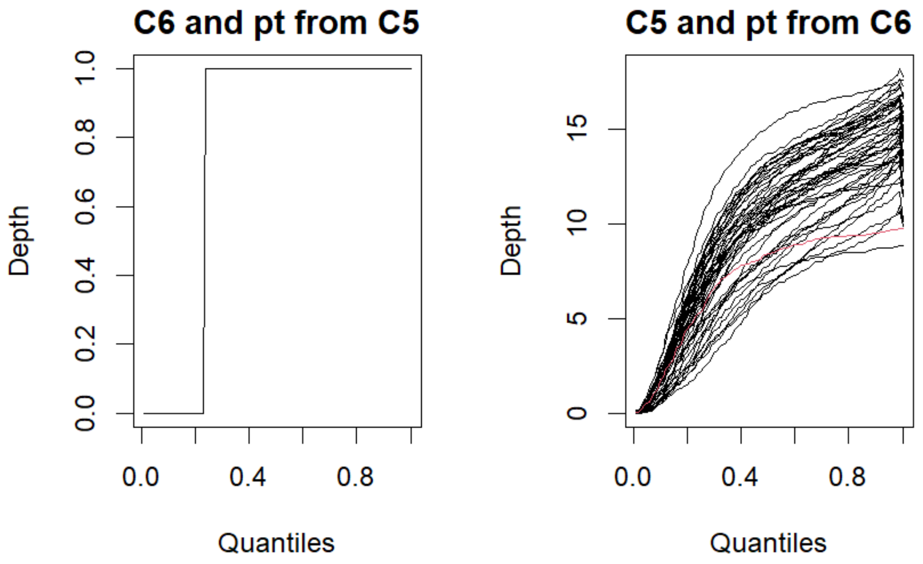
We start with a simple three cluster tri-model two dimensional Gaussian distribution ($n = 150$) data set as shown in figure 3.2.3. The first 50 data points were generated randomly from two standard normal distributions (one for each feature). The second 50 data points were generated randomly from two normal distributions each centered at -5 with standard deviation 1. The third 50 data points were generated from two normal distributions each centered at 5 with standard deviation 1. Both features in the data were then z-scaled. We overshoot the number of clusters from 3 clusters to 6 clusters.

[4]we may want to delay cluster combination decision to look to combine other clusters first and include more data in future iterations of DQF calculations if the current plots are ambiguous.



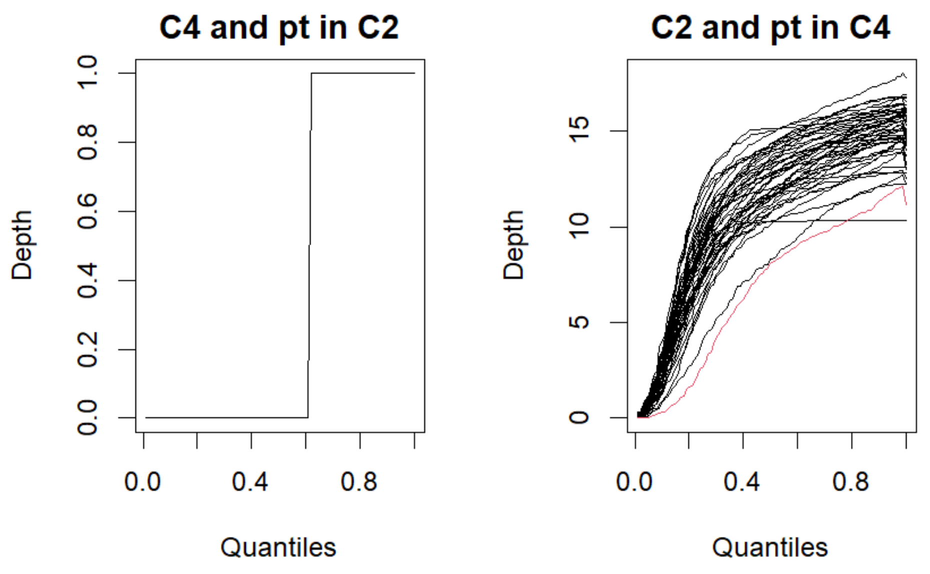
Simple Tri-modal Simulated Data Set for DQF-Clustering

Next, we begin to combined clusters. Cluster(s) [5] and cluster(s) [6] are the closest by Euclidean distance (0.295). Therefore, we look at the DQFs of data points in cluster 5 and the closest points from cluster 6 and the DQFs of data points in cluster 6 and the closest points from cluster 5 as shown in figure 3.2.3. We choose to combine these clusters as the red line, the depth quantile function of the point from cluster 5, in the right DQF plot is well within that of any other point in cluster 6. The left DQF plot contains a single function because cluster 5 only contains one data point. Therefore, there was only one pair of points and the maximum depth is 1.



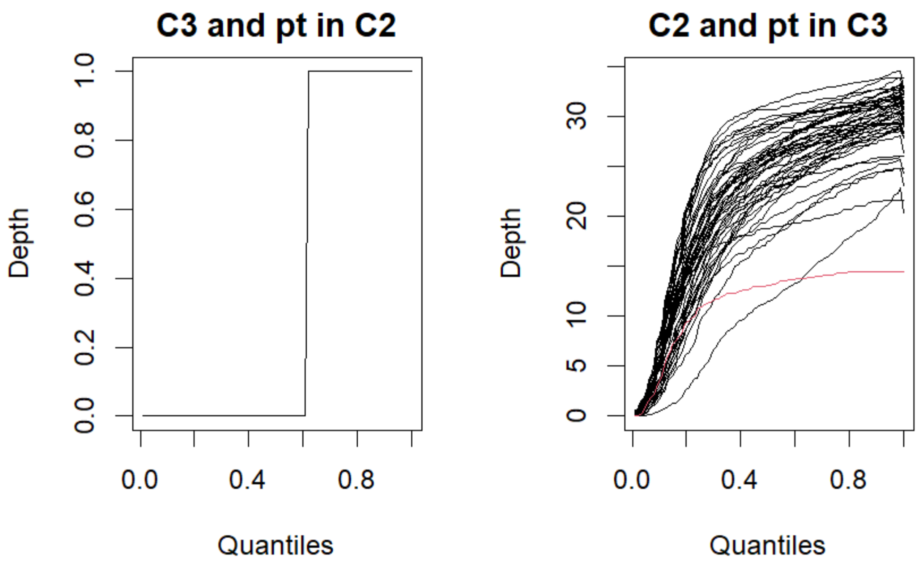
Depth Quantile Functions for Clusters 5 and 6

After combining clusters 5 and 6, the two closest clusters by Euclidean distance (0.310) are clusters 2 and 4. The left DQF plot is similar to the left DQF plot for clusters 5 and 6, which is uninformative. Therefore, we direct our attention to the DQF plot on the right, which displays the depth quantile functions for the point in cluster 4 compared to the points in cluster 2. This is an interesting case where the function that corresponds to our point of interest is not clearly within the bounds of the majority of the functions of the point in cluster 2. However, because the function has the same shape as the function of a point that exists in cluster 2, we elect to assign the point in cluster 4 as not an outlier compared to those in cluster 2. In fact, if we see a function that has a similar shape as some points in the cluster, even if the function is not within the bounds of the majority of the other functions, we want to combine the clusters as that point is most likely an point on the edge of the manifold. Looking back at the initial clustering in figure 3.2.3, we see that the point in cluster 4 is a point on the edge of cluster 2. With that said, it is not unreasonable to conclude that the point in cluster 4 is an outlier in the data set. Hence, although the correct decision in this case is to combine clusters 2 and 4, it is not unreasonable to decide to not combine them. Alternatively, we could delay the decision to combine the cluster.



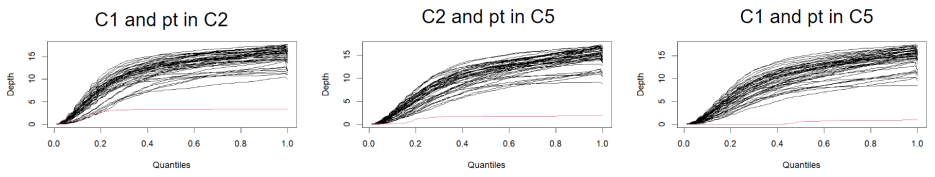
Depth Quantile Functions for Clusters 2 and 4

After combining clusters 2 and 4, the two closest clusters by Euclidean distance (0.315) are clusters 2 and 3. Based on the right DQF plot, we elect to combine the two clusters. Again, we perform anomaly ddetection on the point in cluster 3 compared to cluster 2. Here, we observe another instance of an edge point where the point has good local depth in the early quantiles and poor global depth in the latter quantiles because there are no data points on one side of the point.

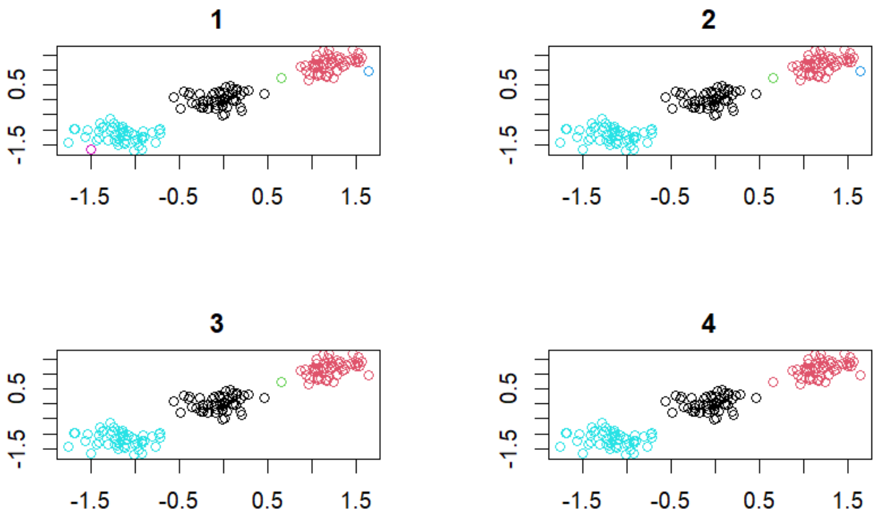


Depth Quantile Functions for Clusters 2 and 3

Now, three clusters remain, and when we look at the remaining DQF plots, it is clear we do not combine any of the three clusters. Six plots were produced, but only three were displayed as the inverse plots look very similar.



Final Combination of Clusters



Evolution of Clustering after Clusters Combinations

Granted, this is an extremely simple simulated data set that any standard clustering algorithm would do well on. However, it works!

Chapter 4

Code

4.1 Appendix

Appendix contains R functions relevant to the paper.

4.1.1 Code from DQFAnomaly R Package

sd.w

```
# Computes the windsorized standard deviation, called by dqf.outlier
# Inputs
- k:  (integer) number of observations at each extreme to alter
- x: (vector) numeric data values

sd.w <- function(x, k) {
  k <- floor(k)
  if (k == 0) {  #corresponds to non-robust adaptive DQF
    return(sd(x))
  } else {
    x <- sort(x)
    n <- length(x)
    x[1:k] <- x[k]
    x[(n-k+1):n] <- x[n-k+1]
    return(sd(x))
  }
}
```

subsamp.dqf

```
# Subsample pairs of points (random is observations are scrambled)
# called by dqf.outlier, computes random subset of pairs

subsamp.dqf <- function(n.obs, subsample) { subset of pairs
  pairs <- c()
  subsample <- floor(subsample/2)*2
  for (i in 1:n.obs) {
    for (j in (i+1):(i+subsample/2)) {
      pairs <- rbind(pairs, c(i,j*(j <= n.obs) + (j-n.obs)*(j > n.obs)))
    }
  }
  return(pairs)
}
```

dqf.outlier

```
# kernelized version of depthity

  # inputs:
  #   data (matrix or data frame) - a data matrix of explanatory variables
  #   kernel - of form "linear", "rbf" or "poly", or a user defined function
  #   g.scale (scalar) - scales the base distribution G
  #   angle (numeric vector of length 3)- angles of cone from midline,
              must live between 0 and 90
  #   p1 - first parameter for kernel
  #   p2 - second parameter for kernel
  #   n.splits (integer) - the number of split points at which the
              DQF is computed
  #   subsample (integer)- the number of random pairs for each observation
  #   z.scale (logical) - should the data be z-scaled first
  #   k.w (integer) - the number of points altered in the windsorized
              standard deviation
  #   adaptive - if TRUE, uses windsorized standard deviation to
              scale base distribution
  #   G - base distribution:   "norm" or "unif"
  ##
  # Output:
  #   angle  - vector of angles used, same as inputted
  #   dqf1, dqf2, dqf3 - matrices of depth quantile functions,
              rows are observations

dqf.outlier <- function(data = NULL, gram.mat = NULL, g.scale=2,
        angle=c(30,45,60), kernel="linear", p1=1, p2=0, n.splits=100,
        subsample=50, z.scale=TRUE, k.w=3, adaptive=TRUE, G="norm") {
  if (G=="norm") {
    param1 <- 0; param2 <- 1 }
  if (G=="unif") {
    param1 <- -1; param2 <- 1 }
  if (is.null(data) & is.null(gram.mat))
    stop("Either a data set or Gram matrix must be provided")
  if (min(angle) <= 0 | max(angle) >= 90)
    stop("Angles must be between 0 and 90")
  if (is.null(data))
    n.obs <- nrow(gram.mat)
  if (is.null(gram.mat))
    n.obs <- nrow(data)
  scram <- sample(n.obs)
  pairs <- subsamp.dqf(n.obs, subsample)
  if (is.null(gram.mat)) {
    if (z.scale==TRUE)
      data <- apply(data, 2, scale) #z-scale data
    if (is.function(kernel)==TRUE)
      kern <- kernel
    if (kernel == "linear") {
      kern <- function(x,y)
        return(sum(x*y))
    }
    if (kernel == "rbf") {
      kern <- function(x,y)
        return(exp(-sum((x-y)^2)/p1))
    }
    if (kernel == "poly") {
      kern <- function(x,y)
        return((sum(x*y)+p2)^p1)
    }
    data <- data[scram,]
    gram <- matrix(0,n.obs, n.obs)
    for (i in 1:n.obs) {
      for (j in i:n.obs) {
        gram[i,j] <- kern(data[i,], data[j,])
        gram[j,i] <- gram[i,j]
      }
    }
  } else {
    if (diff(dim(gram.mat))!=0)
      stop("Gram matrix must be square")
    if (isSymmetric(gram.mat) == FALSE)
```

```
    stop("Gram matrix must be symmetric")
  gram <- gram.mat
  gram <- gram[scram,]
  gram <- gram[,scram]
}
splits <- get(paste("q", G, sep=""))((1:n.splits)/(n.splits+1),param1,
      param2) * g.scale
depthity1 <- depthity2 <- depthity3 <- rep(0,length(splits))
norm.k2 <- error.k <- k.to.mid <- rep(0, n.obs)
dep1 <- dep2 <- dep3 <- matrix(0, nrow=nrow(pairs), ncol=n.splits)
qfs1 <- qfs2 <- qfs3 <- matrix(0, nrow=nrow(pairs), ncol=100)
for (i.subs in 1:nrow(pairs)) {
  i <- pairs[i.subs,1];  j <- pairs[i.subs,2]
  for (k in 1:n.obs) {
    norm.k2[k] <- gram[k,k] + 1/4*(gram[i,i]+gram[j,j]) +
          1/2*gram[i,j]-gram[k,i]-gram[k,j]
    k.to.mid[k] <- (gram[k,i]-gram[k,j]+1/2*(gram[j,j]-gram[i,i]))
          /sqrt(gram[i,i]+gram[j,j]-2*gram[i,j])
    error.k[k] <- sqrt(abs(norm.k2[k] - k.to.mid[k]^2))
  }
  for (c in 1:length(splits)) {
    good <- rep(1, n.obs)
    s <- splits[c] * (sd.w(k.to.mid, k.w)*adaptive + (adaptive==FALSE))
    good[k.to.mid/s > 1] <- 0  #points on other side of cone tip removed
    d.to.tip <- abs(k.to.mid - s)
    good1 <- good * (abs(atan(error.k / d.to.tip)) < (angle[1]/360*2*pi))
          #points outside of cone removed
    good1 <- good1 * (1 - 2*(sign(k.to.mid)==sign(s)))
          #which side of midpoint are they on
    depthity1[c] <- min(c(sum(good1==-1), sum(good1==1)))
    good2 <- good * (abs(atan(error.k / d.to.tip)) < (angle[2]/360*2*pi))
          #points outside of cone removed
    good2 <- good2 * (1 - 2*(sign(k.to.mid)==sign(s)))
          #which side of midpoint are they on
    depthity2[c] <- min(c(sum(good2==-1), sum(good2==1)))
    good3 <- good * (abs(atan(error.k / d.to.tip)) < (angle[3]/360*2*pi))
          #points outside of cone removed
    good3 <- good3 * (1 - 2*(sign(k.to.mid)==sign(s)))
```

```
        #which side of midpoint are they on
      depthity3[c] <- min(c(sum(good3==-1), sum(good3==1)))
    }
    qfs1[i.subs,] <- quantile(depthity1, seq(0,1,length=100), na.rm=TRUE)
    qfs2[i.subs,] <- quantile(depthity2, seq(0,1,length=100), na.rm=TRUE)
    qfs3[i.subs,] <- quantile(depthity3, seq(0,1,length=100), na.rm=TRUE)
  }
  dqf1 <- dqf2 <- dqf3 <- matrix(0,n.obs, 100)
  for (i in 1:n.obs) {
    dqf1[i,] <- apply(qfs1[which(pairs[,1]==i | pairs[,2]==i),],2,
            mean, na.rm=TRUE)
    dqf2[i,] <- apply(qfs2[which(pairs[,1]==i | pairs[,2]==i),],2,
            mean, na.rm=TRUE)
    dqf3[i,] <- apply(qfs3[which(pairs[,1]==i | pairs[,2]==i),],2,
            mean, na.rm=TRUE)
  }
  dqf1 <- dqf1[order(scram),]  #map back to original indicies
  dqf2 <- dqf2[order(scram),]
  dqf3 <- dqf3[order(scram),]

  return(list(angle=angle, dqf1=dqf1, dqf2=dqf2, dqf3=dqf3))
}
```

4.1.2 Plotting Functions

plot.dqf()

```
# plots depth quantile functions
    # inputs
    #   dqf: depth quantiles function dataframe
    #   labels: color vector for dqfs
    #   xlab, ylab, main: same inputs as plot


plot.dqf <- function(dqf,labels=NULL,xlab='',ylab='',main=''){
  x <- seq(.01,1,.01)

  if(nrow(dqf)==1){ # only one function
```

```
    plot(x,dqf,t='l')
  }
  else{
    n.functions <- nrow(dqf)
    if(is.null(labels)) labels <- rep(1,n.functions)

    plot(x,dqf[1,],t='l',ylim=c(0,max(dqf)),col=labels[1],
            xlab=xlab,ylab=ylab,main=main)
    for(i in 2:n.functions){
      lines(x,dqf[i,],col=labels[i])
    }
  }
```

4.1.3 Subset DQFs Calculation Functions

dqf.subset

```
# returns objects required to calculate dqfs for subsets of data
    # inputs:
    #   same as dqf.outlier()

    # outputs:
    #   dqf1: DQFs (same as dqf.outliers() function)
    #   ret.pairs: Subsampled pairs of points
    #   ret.goods: List of good vectors corresponding to pairs of points
    #   k.to.mids: distance from projected data to midpoints vectors
    #           (used for adapativity).
    #   splits: cone tip distances from midpoints
    #           (used for quantile calculations on distributions)


dqf.subset <- function(data = NULL, gram.mat = NULL, g.scale=2,
        angle=c(45), kernel="linear", p1=1, p2=0, n.splits=100,
        subsample=50, z.scale=TRUE, k.w=3, adaptive=TRUE, G="norm") {
  if (G=="norm") {
    param1 <- 0; param2 <- 1 }
  if (G=="unif") {
    param1 <- -1; param2 <- 1 }
```

```
if (is.null(data) & is.null(gram.mat))
  stop("Either a data set or Gram matrix must be provided")
if (min(angle) <= 0 | max(angle) >= 90)
  stop("Angles must be between 0 and 90")
if (is.null(data))
  n.obs <- nrow(gram.mat)
if (is.null(gram.mat))
  n.obs <- nrow(data)

pairs <- subsamp.dqf(n.obs, subsample)
if (is.null(gram.mat)) {
  if (z.scale==TRUE)
    data <- apply(data, 2, scale) #z-scale data
  if (is.function(kernel)==TRUE)
    kern <- kernel
  if (kernel == "linear") {
    kern <- function(x,y)
      return(sum(x*y))
  }
  if (kernel == "rbf") {
    kern <- function(x,y)
      return(exp(-sum((x-y)^2)/p1))
  }
  if (kernel == "poly") {
    kern <- function(x,y)
      return((sum(x*y)+p2)^p1)
  }
  #data <- data[scram,]
  gram <- matrix(0,n.obs, n.obs)
  for (i in 1:n.obs) {
    for (j in i:n.obs) {
      gram[i,j] <- kern(data[i,], data[j,])
      gram[j,i] <- gram[i,j]
    }
  }
} else {
  if (diff(dim(gram.mat))!=0)
    stop("Gram matrix must be square")
  if (isSymmetric(gram.mat) == FALSE)
    stop("Gram matrix must be symmetric")
  gram <- gram.mat
  #gram <- gram[scram,]
  #gram <- gram[,scram]
}
splits <- get(paste("q", G, sep=""))((1:n.splits)/(n.splits+1),param1,
      param2) * g.scale
depthity1 <- rep(0,length(splits))
norm.k2 <- error.k <- k.to.mid <- rep(0, n.obs)
dep1 <- matrix(0, nrow=nrow(pairs), ncol=n.splits)
qfs1 <- matrix(0, nrow=nrow(pairs), ncol=100)

ret.pairs <- matrix(0,nrow=nrow(pairs),ncol=2)
k.to.mids <- matrix(ncol = n.obs, nrow = nrow(pairs))

ret.gs <- matrix(ncol = n.obs, nrow = n.splits)
      # skeleton for goods data.frame
ret.goods <- list()

for (i.subs in 1:nrow(pairs)) {
  i <- pairs[i.subs,1];  j <- pairs[i.subs,2]

  #ret.pairs[i.subs,] <- c(scram[i],scram[j])
  ret.pairs[i.subs,] <- c(i,j)

  for (k in 1:n.obs) {
    norm.k2[k] <- gram[k,k] + 1/4*(gram[i,i]+gram[j,j]) +
          1/2*gram[i,j]-gram[k,i]-gram[k,j]
    k.to.mid[k] <- (gram[k,i]-gram[k,j]+1/2*(gram[j,j]-gram[i,i]))
          /sqrt(gram[i,i]+gram[j,j]-2*gram[i,j]) # return this
    error.k[k] <- sqrt(abs(norm.k2[k] - k.to.mid[k]^2))
  }

  k.to.mids[i.subs,] <- k.to.mid

  for (c in 1:length(splits)) {
    good <- rep(1, n.obs)
```

```
      s <- splits[c] * (sd.w(k.to.mid, k.w)*adaptive + (adaptive==FALSE))
      good[k.to.mid/s > 1] <- 0  #points on other side of cone tip removed
      d.to.tip <- abs(k.to.mid - s)
      good1 <- good * (abs(atan(error.k / d.to.tip)) < (angle[1]/360*2*pi))
            #points outside of cone removed
      good1 <- good1 * (1 - 2*(sign(k.to.mid)==sign(s)))
            # which side of midpoint are they on
      ret.gs[c,] <- good1
      depthity1[c] <- min(c(sum(good1==-1), sum(good1==1)))
    }
    ret.goods[[i.subs]] <- ret.gs
    qfs1[i.subs,] <- quantile(depthity1, seq(0,1,length=100), na.rm=TRUE)
  }
  dqf1 <- matrix(0,n.obs, 100)
  for (i in 1:n.obs) {
    dqf1[i,] <- apply(qfs1[which(pairs[,1]==i | pairs[,2]==i),],2,
            mean, na.rm=TRUE)
  }

  # reorder
  # dqf1 <- dqf1[unscram,]  #map back to original indicies

  return(list(dqf1=dqf1,ret.pairs=ret.pairs,ret.goods=ret.goods,
        k.to.mids=k.to.mids,splits=splits))
}
```

extract.depths

```
# subsets depths of desired subset of points
    # inputs:
    #   dqf.s object outputted from dqf.subset() function
    #   subset: indices of desired subset of points
    # outputs:
    #   depths: depths dataframe
    #   subset.pairs: subset of pairs corresponding to rows
    #           in the depths dataframe
extract.depths <- function(dqf,subset){
  n.subset <- length(subset)
  depthity1 <- rep(0,100)

  indices <- which(dqf$ret.pairs[,1] %in% subset
        & dqf$ret.pairs[,2] %in% subset)
  subset.pairs <- dqf$ret.pairs[indices,]

  depths <- matrix(0, nrow=length(indices), ncol=100)

  depths.count <- 1
  for(j in indices){
    gs <- dqf$ret.goods[[j]][,subset]
    for(i in 1:nrow(gs)){
      g <- gs[i,]
      depthity1[i] <- min(c(sum(g==-1), sum(g==1)))
    }
    depths[depths.count,] <- depthity1
    depths.count <- depths.count+1
  }

  return(list(depths=depths,subset.pairs=subset.pairs))
}
```

calculcate.qfs()

```
# calculates piece-wise quantile functions on a
        distribution based on splits vector

    # inputs:
    #   depths dataframe: output from the extract.depths() function
    #   k2mid: output from the dqf.subset() function
    #   splits: output from the dqf.subset() functions
    # outputs:
    #   qfs: Quantile functions dataframe
    #           corresponds to subset.pairs (output from extract.depths())


calculate.qfs <- function(depths,k2mid,splits){
  qfs <- matrix(0,nrow=nrow(depths),ncol=length(splits))

  for(i in 1:nrow(depths)){
    qf <- rep(0,100)
    d <- depths[i,]
    k.sd <- sd.w(k2mid[i,],3)

    q.prev <- 1
    min.s <- 1; max.s <- -1
    for(j in 0:max(d)){
      s <- splits[which(d==j)]
      min.s <- min(c(s,min.s)); max.s <- max(c(s,max.s))
      q <- ceiling((pnorm(max.s,sd=k.sd)-pnorm(min.s,sd=k.sd))*100)

      if(q>q.prev & q <= 100){
        qf[q.prev:q] <- j
        q.prev <- q+1
      }

    }
    qfs[i,] <- qf

  }
  return(qfs)
}
```

calculate.dqfs()

```
# calculates depth quantile functions from quantile functions
    # inputs:
    #   qfs: quantile functions dataframe from calculate.qfs()
    #   pairs: subset.pairs from extract.depths()
    #   subset: subset: indices of desired subset of points
    #           (intput to extract.depths)
    #   n.obs: total number of observations in original dataset

    # outputs:
```

```
    #   depth quantile functions of subsetted data


calculate.dqfs <- function(qfs,pairs,subset,n.obs){

  if(length(pairs)==2) return(qfs)

  dqfs <- matrix(0,n.obs, 100)
  for (i in 1:n.obs) {
    dqfs[i,] <- apply(qfs[which(pairs[,1]==i | pairs[,2]==i),],2,
            mean, na.rm=TRUE)
  }
  return(dqfs[subset,])
}
```

4.1.4 Clustering-Related Functions

row.col()

```
# calculates row and column number given 1D dataframe index

    # inputs:
    #   dataframe
    #   1D index number
    # outputs:
    #   row: row number
    #   col: column number


row.col <- function(dataframe,index){
  n.row <- nrow(dataframe); n.col <- ncol(dataframe)

  row <- index%%n.row
  if(row==0){ row <- n.row; col <- index/n.row}
  else{ col <- ceiling(index/n.row) }

  return(list(row=row,col=col))

}
```

initial.cluster

```
# calculates initial clusters using hierarchical clustering

    # inputs:
    #   data
    #   n.clusters: number of clusters desired
    # outputs:
    #   clusters: vector of clucster numbers corresponding to
    #           data points
    #   inter.dists: matrix of distances between closest points
    #           between clusters i and j
    #   closest.pts: matrix of indices of point in cluster j
    #           closest to cluster i


initial.cluster <- function(data,n.clusters){
  n.obs <- nrow(data)
  distance_mat <- dist(data, method = 'euclidean')
  Hierar_cl <- hclust(distance_mat, method = 'single')
  clusters <- cutree(Hierar_cl, k = n.clusters)

  # calculate minimum distance between clusters
  dist.mat <- as.matrix(distance_mat)
  clus.indices <- list()
  for(i in 1:n.clusters){
    clus.indices[[i]] <- which(clusters==i)
  }
# calculate min distance from cluster i to j by:
  # calculate min of distances between point i_1 to point js
    # calculate min of those

  inter.dists <- matrix(0,nrow=n.clusters,ncol=n.clusters)
        # distance between closest points of clusters
  closest.pts <- matrix(0,nrow=n.clusters,ncol=n.clusters)
        # M(i,j) is index of closest point in cluster j to cluster i
  for(i in 1:n.clusters){
    for(j in i:n.clusters){
      if(i==j){inter.dists[i,j] <- Inf}
      else{
        inter.dists[i,j] <- inter.dists[j,i]
                <- min(dist.mat[clus.indices[[i]],clus.indices[[j]]])

        min.index <- which(dist.mat==inter.dists[i,j])[1]
        rc <- row.col(dist.mat,min.index)
        row <- rc$row; col <- rc$col

        if(row %in% clus.indices[[i]]){ closest.pts[j,i] <- row;
                closest.pts[i,j] <- col }
        else{ closest.pts[i,j] <- row; closest.pts[j,i] <- col }

        }
    }
  }

  return(list(clusters=clusters,inter.dists=inter.dists,
        closest.pts=closest.pts))
}
```

max.dist()

```
# returns maximum distance between two clusters excluding
        infinity values

    # inputs:
    #   inter.dists: inter cluster distances matrix
    # outputs:
    #   maximum value excluding infinities
```

combine.clusters()

```
# merge lists of clusters

    # inputs:
    #   combined.clusters: list of already combined clusters
    #   c1: cluster 1 to combine
    #   c2: cluster 2 to combine
    # outputs:
    #   combined.clusters: new updated combined.cluster list
```

all.indices()

```
# returns indices of of original data points in clusters of interest

    # inputs:
    #   clusters: original cluster vector
    #   cluster.group: vector of clusters of interest
    # output:
    #   indices: vector of indices
```

closest.pt.clusters()

```
# calculates which cluster the closest point is in

    # inputs:
    #   inter.dists: inter cluster distance matrix
    #   cg1: cluster group 1
    #   cg2: cluster group 2
    # outputs:
    #   source cluster and destination cluster indices
    #       with closest points
    #       (to access closest.pt dataframe and isolate closest point)

closest.pt.clusters <- function(inter.dists, cg1, cg2){
  n.cg1 <- length(c(cg1))
  n.cg2 <- length(c(cg2))
  m <- min(inter.dists[cg1,cg2])
  min.index <- which(inter.dists==m)[1]

  rc <- row.col(inter.dists,min.index)
  row <- rc$row; col <- rc$col

  return(list(c1=row,c2=col))

}
```

compile.clusters()

```
# returns final cluster vector from combined.clusters list

    # inputs:
    #   clusters: original clusters vector (from initial.cluster())
    #   combined.clusters: list of combinaed clusters
    # outputs:
    #   final.clusters: final clusters vector
```

4.1.5 Prompt and Print Functions

combine.prompt

```
# prompts user whether to combine clusters

    # inputs:
    #   row: cluster 1
    #   col: cluster 2
    #   inter.dists: inter cluster distances matrix
    #   combined.clusters: list of combined clusters
    # outputs:
    #   none (print)
```

cluster.string()

```
# returns string with all associated combined.clusters clusters

    # inputs:
    #   cluster: clusters
    # output:
    #   ret: string in the form of {c1,c2,...}

cluster.string <- function(cluster){

  ret <- '{'

  for(i in 1:length(cluster)){
    if(i==length(cluster)){ret
```

```
            <- paste(ret,as.character(cluster[i]),sep='')}
    else{ret <- paste(ret, as.character(cluster[i]),' ',sep='')}
  }

  ret <- paste(ret,'}',sep='')

  return(ret)
}
```

4.1.6 Final Interactive dqf.clustering Functions

```
# final interactive dqf.clustering function

dqf.clustering <- function(data = NULL,dqf.s=NULL,n.clusters=20,
        gram.mat = NULL, g.scale=2, angle=c(45), kernel="linear",
        p1=1, p2=0, n.splits=100, subsample=50, z.scale=TRUE, k.w=3,
        adaptive=TRUE, G="norm"){

  # cannot input both data and calculated dqf.s
  if(!is.null(data) & !is.null(dqf.s)){
    print("Invalid Input. Function can only take one of `data` or
            `dqf.s` as input.")
    return()
  }

  data <- scale(data)

  # set n.obs - to number of observations
  if (is.null(data)) n.obs <- nrow(gram.mat)
  if (is.null(gram.mat)) n.obs <- nrow(data)

  subsample <- n.obs # calculate all sub-samples

  # initial clustering
  ic <- initial.cluster(data,n.clusters)
  clusters <- ic$clusters
  inter.dists <- ic$inter.dists
  closest.pts <- ic$closest.pts
```

```
combined.clusters <- list()
for(i in 1:n.clusters) combined.clusters[[i]] <- c(i)

# calculate dqf.subset
if(is.null(dqf.s)){
  print("Calculating ‘dqf.subset‘...")
  dqf.s <- dqf.subset(data = data, gram.mat = gram.mat, g.scale=g.scale,
          angle=c(45), kernel=kernel, p1=p1, p2=p2, n.splits=n.splits,
          subsample=subsample, z.scale=z.scale, k.w=k.w,
          adaptive=adaptive, G=G)
  print("‘dqf.subset‘ calculations complete.")
}

print("We will now begin combining clusters.")
print("You may type exit at any time and the function will return
          ‘dqf.subset‘ and current clusters.")
print("Press Enter to continue")
s = readline()
while(s != ’exit’){
  move.on <- FALSE

  if(min(inter.dists)==Inf){
    final.clusters <- compile.clusters(clusters, combined.clusters)
    return(list(dqf.s=dqf.s,final.clusters=final.clusters))
  }
  mi <- which.min(inter.dists) # mi is short for min.index
  rc <- row.col(inter.dists,mi)
  row <- rc$row; col <- rc$col

  # Prepare dqfs
  cpc <- closest.pt.clusters(inter.dists,combined.clusters[[row]],
          combined.clusters[[col]])
  pt2.index <- closest.pts[cpc$c1,cpc$c2]
          # index of closest point in cluster group 2 to cg1
  pt1.index <- closest.pts[cpc$c2,cpc$c1]
          # index of closest point in cluster group 1 to cg2
```

```
subset1 <- c(all.indices(clusters,combined.clusters[[row]]),pt2.index)
ed1 <- extract.depths(dqf.s,subset1)
depths1 <- ed1$depths; subset1.pairs <- ed1$subset.pairs
qfs1 <- calculate.qfs(depths=depths1,dqf.s$k.to.mids,dqf.s$splits)
dqfs1 <- calculate.dqfs(qfs1,subset1.pairs,subset1,n.obs)
labels1 <- rep(1,length(subset1)); labels1[which(subset1==pt2.index)] <- 2

subset2 <- c(all.indices(clusters,combined.clusters[[col]]),pt1.index)
ed2 <- extract.depths(dqf.s,subset2)
depths2 <- ed2$depths; subset2.pairs <- ed2$subset.pairs
qfs2 <- calculate.qfs(depths=depths2,dqf.s$k.to.mids,dqf.s$splits)
dqfs2 <- calculate.dqfs(qfs2,subset2.pairs,subset2,n.obs)
labels2 <- rep(1,length(subset2)); labels2[which(subset2==pt1.index)] <- 2

combine.prompt(row,col,inter.dists,combined.clusters)

s <- readline()
while(!move.on){
  if(s == "Yes"){
    print(glue("Clusters {cluster.string(combined.clusters[[row]])}
            and {cluster.string(combined.clusters[[col]])} were combined."))
    combined.clusters <- combine.clusters(combined.clusters,row,col)
    inter.dists[combined.clusters[[row]],combined.clusters[[col]]] <- Inf
            # don’t need to consider combining them anymore
    inter.dists[combined.clusters[[col]],combined.clusters[[row]]] <- Inf
    move.on <- TRUE
  }else if(s == "No"){
    # print("Are you sure? If yes, combinning clusters
            {cluster.string(combined.clusters[[row]])} and
            {cluster.string(combined.clusters[[col]])} will not
            be revisited.")
    # s <- readline()
    inter.dists[combined.clusters[[row]],combined.clusters[[col]]] <- Inf
    inter.dists[combined.clusters[[col]],combined.clusters[[row]]] <- Inf
    move.on <- TRUE
  }else if(s == "Later"){
    print("Later")
```

```
        s <- readline()
      }else if(s == row){
        plot.dqf(dqfs1,labels1)
        combine.prompt(row,col,inter.dists,combined.clusters)
        s <- readline()
      }else if(s == col){
        plot.dqf(dqfs2,labels2)
        combine.prompt(row,col,inter.dists,combined.clusters)
        s <- readline()
      } else if(s == 'exit'){
        move.on = TRUE
      } else{
        print("Invalid input")
        combine.prompt(row,col,inter.dists,combined.clusters)
        s <- readline()
      }
    }

  }

  final.clusters <- compile.clusters(clusters, combined.clusters)

  return(list(dqf.s=dqf.s,final.clusters=final.clusters))

}
```

Acknowledgements

Firstly, I would like to thank Professor Gabe Chandler (Pomona College) for advising me throughout the thesis process and for all the wonderful conversations we have had in our weekly meetings. I would also like to thank Professor Jo Hardin (Pomona College) for inspiring me to pursue an undergraduate degree in mathematics with a focus in statistics and all the professors in the Math department who have reinforced my passion in it. Last, but not least, I thank my family and friends who have always believed in and supported me. Thank you.

Bibliography

[rdo,] Dist: Distance matrix computation.

[Chandler and Polonik, 2021] Chandler, G. and Polonik, W. (2021). Multi-scale geometric feature extraction for high-dimensional and non-euclidean data with applications. *The Annals of Statistics*, 49(2):988–1010.

[Chandler and Polonik, 2022] Chandler, G. and Polonik, W. (2022). Anti-modes and graphical anomaly exploration via depth quantile functions. *arXiv preprint arXiv:2201.06682*.

[Cunningham and Delany, 2021] Cunningham, P. and Delany, S. J. (2021). k-nearest neighbour classifiers-a tutorial. *ACM computing surveys (CSUR)*, 54(6):1–25.

[Dutta et al., 2011] Dutta, S., Ghosh, A. K., and Chaudhuri, P. (2011). Some intriguing properties of tukey's half-space depth. *Bernoulli*, 17(4):1420–1434.

[Dyckerhoff and Mozharovskyi, 2016] Dyckerhoff, R. and Mozharovskyi, P. (2016). Exact computation of the halfspace depth. *Computational Statistics & Data Analysis*, 98:19–30.

[Hardin, 2023] Hardin, J. (2023). Class notes.

[Hastie et al., 2009] Hastie, T., Tibshirani, R., Friedman, J. H., and Friedman, J. H. (2009). *The elements of statistical learning: data mining, inference, and prediction*, volume 2. Springer.

[Li, 2019] Li, S. (2019). Anomaly detection for dummies.

[Roman, 2019] Roman, V. (2019). Unsupervised machine learning: Clustering analysis.

[Thorn, 2021] Thorn, J. (2021). The surprising behaviour of distance metrics in high dimensions.

[Tukey, 1975] Tukey, J. W. (1975). Mathematics and the picturing of data. In *Proceedings of the International Congress of Mathematicians (Vancouver, B. C., 1974), Vol. 2*, pages 523–531. Canad. Math. Congress, Montreal, Que.