An Exploration of the DBSCAN Algorithm

Stat 666 Term Project

Matthew Morgan, Jace Ritchie

Abstract

Density-based spatial clustering of applications with noise (DBSCAN) is a popular clustering algorithm. In this report, simulations prove the efficacy of DBSCAN over k-means as dimensionality scales over data generated with non-overlapping means, as well as the shared inefficacy of both methods when the data mean vectors overlap significantly. DBSCAN is also used to cluster a dataset of player/course performances from the 2014 season of the Professional Golf Association.

1 Introduction

In the field of unsupervised learning, algorithms are used to analyze and cluster unlabeled datasets. Each algorithm uses a specific clustering process in an effort to discover patterns and/or groupings in the data that might not be as evident to the human eye. The patterns and/or groupings in the data found by these algorithms are used in a wide range of applications.

One clustering algorithm that has become more popular is the density-based spatial clustering of applications with noise (DBSCAN) algorithm. DBSCAN is a popular clustering algorithm that can discover clusters of arbitrary shape and size while also identifying noise points. Due to its robustness and scalability, DBSCAN is often chosen for clustering tasks in various domains such as image processing, natural language processing, and social network analysis. This report aims to provide an in-depth exploration of the DBSCAN algorithm, including its core concepts, implementation, advantages, and limitations.

Specifically, this exploration of the DBSCAN algorithm includes a simulation study which compares the performance of the DBSCAN algorithm to the commonly used k-means algorithm. The simulation study focuses on evaluating the two algorithms with respect to their multidimensional scaling ability as well as their ability to cluster data generated from various settings. Additional exploration of the DBSCAN algorithm includes using it with a real-world dataset from the 2014 season of the Professional Golf Association to demonstrate its effectiveness and applicability. The insights gained from this exploration of the DBSCAN algorithm can be useful to practitioners and researchers interested in clustering techniques and can serve as a guide for selecting appropriate clustering methods for different types of data.

2 Methodology

2.1 Density-Based Spatial Clustering of Applications with Noise (DBSCAN)

DBSCAN is a clustering methodology based on the distance between points rather than a pre-allocated number of clusters as defined by Ester et al. (1996). In contrast to many popular clustering algorithms, instead of minimizing within-group variance and maximizing across-group variance with respect to a defined number of clusters, DBSCAN finds a network of points that are a given distance away from each other, ϵ. Clusters are defined when there is a group of points of at least a minimum size ($MinPts$) within ϵ of one of the points. From there, clusters are expanded by adding points within ϵ to the clusters. Thus, instead of finding a predetermined number of clusters, the number of clusters that DBSCAN finds is determined by the number of clusters found to be of at least size $MinPts$ with a density directly tied to ϵ. Furthermore, DBSCAN does not require that every point belong to a cluster. As a result, there may be some points that are simply labelled as "noise".

Algorithm 1 DBSCAN

Object: Given a set of data points $\{\mathbf{x}^{(1)}, \dots \mathbf{x}^{(m)}\}$ where $\mathbf{x}^{(i)} \in \mathbb{R}^n$, identify clusters with some minimum number of points that are a specified distance away from each other.

Procedure:

1: Set ϵ and $MinPts$ to be the specified distance and minimum number of points necessary to make a cluster, respectively.
2: Choose a random, unclustered point, p, that hasn't been selected yet. If such a point doesn't exist, go to Step 7.
3: Identify all t points that are closer than ϵ to p.
4: If p is a cluster or $t > MinPts$, then p and its t nearest neighbors form a cluster. Otherwise, go to Step 2.
5: If $t = 0$, go to Step 2.
6: Define p to be p and its t nearest neighbors. Go to Step 3.
7: Define all unclustered points as noise.

As a rule of thumb, $MinPts$ is generally set to twice the number of variables being considered (Sander et al., 1998). Additionally, ϵ is set to be the inflection point on a $MinPts - 1$ nearest neighbors distance plot, as shown in Figure 1 (Schubert et al., 2017). These heuristic methods will be used throughout the remainder of the report to find useful parameters for DBSCAN.

2.2 k-means

The other clustering algorithm considered in this project was k-means clustering (Rencher and Christensen, 2012). This method was considered as it is one of the most commonly used procedures in unsupervised learning. Given the prevalent use of the k-means clustering method, it was used as a baseline to which the DBSCAN clustering method could be compared. The details of the k-means algorithm are the following:

Algorithm 2 k-means

Object: Given a set of data points $\{\mathbf{x}^{(1)}, \dots \mathbf{x}^{(m)}\}$ where $\mathbf{x}^{(i)} \in \mathbb{R}^n$, predict k centroids and a label $c^{(i)}$ for each data point.

Procedure:

1: Initialize cluster centroids $\boldsymbol{\mu}_1, \boldsymbol{\mu}_2, \dots, \boldsymbol{\mu}_k \in \mathbb{R}^n$ randomly.
2: Assign each data point, $i \in 1, \dots, m$, to closest cluster centroid (using Euclidean distance).

$$c^{(i)} = \arg\min_j \|\mathbf{x}^{(i)} - \boldsymbol{\mu}_j\|^2$$

- Recalculate each of the cluster centroids, $j \in 1, \dots, k$.

$$\boldsymbol{\mu}_j = \frac{\sum_{i=1}^m 1\{c^{(i)} = j\}\mathbf{x}^{(i)}}{\sum_{i=1}^m 1\{c^{(i)} = j\}}$$

3: Repeat Step 2 until no more reassignments occur.

One interesting aspect of the k-means algorithm is that the number of clusters into which data points will be labeled must be chosen *a priori*. Therefore, there must be some sort of intuition beforehand about the underlying grouping of the data points when implementing the k-means algorithm.

3 Simulation Study

3.1 Setup

As a means of evaluating the performance of the DBSCAN algorithm and the k-means algorithm, a simulation study was conducted where the "true" underlying grouping of objects was set *a priori*. The simulation was conducted using 10,000 iterations. The data for the four groups were generated from four multivariate normal distributions, each with 100 data points, equal covariance matrices

$$\underset{p_s \times p_s}{\Sigma} = \begin{bmatrix} 5 & .5 & \dots & .5 \\ .5 & 5 & \dots & .5 \\ \vdots & \vdots & & \vdots \\ .5 & .5 & \dots & 5 \end{bmatrix},$$

and distinct $p_s \times 1$ mean vectors $\boldsymbol{\mu}_{1s}$, $\boldsymbol{\mu}_{2s}$, $\boldsymbol{\mu}_{3s}$, and $\boldsymbol{\mu}_{4s}$ for $s \in 1, \dots, 4$. Additionally, the data was standardized after being generated.

- The first set of mean vectors used to generate the data were of length $p_1 = 4$ with:
  - $\boldsymbol{\mu}_{11} = (5, 6, 7, 8)$, a baseline mean vector
  - $\boldsymbol{\mu}_{21} = (-5, -6, -7, -8)$, a negatively collinear mean vector
  - $\boldsymbol{\mu}_{31} = (15, 18, 21, 24)$, a positively collinear mean vector
  - $\boldsymbol{\mu}_{41} = (0, 0, 0, 0)$, a non-collinear mean vector
- The second set of mean vectors used to generate the data were of length $p_2 = 4$ with:
  - $\boldsymbol{\mu}_{12} = (5, 6, 7, 8)$, a baseline mean vector
  - $\boldsymbol{\mu}_{22} = (3, 4, 5, 6)$, an overlapping collinear mean vector
  - $\boldsymbol{\mu}_{32} = (7, 8, 9, 10)$, an overlapping collinear mean vector
  - $\boldsymbol{\mu}_{42} = (0, 0, 0, 0)$, a non-overlapping mean vector
- The third set of mean vectors used to generate the data were of length $p_3 = 8$ with:
  - $\boldsymbol{\mu}_{13} = (5, 6, 7, 8, 9, 10, 11, 12)$, a baseline mean vector
  - $\boldsymbol{\mu}_{23} = (-5, -6, -7, -8, -9, -10, -11, -12)$, a negatively collinear mean vector
  - $\boldsymbol{\mu}_{33} = (15, 18, 21, 24, 27, 30, 33, 36)$, a positively collinear mean vector
  - $\boldsymbol{\mu}_{43} = (0, 0, 0, 0, 0, 0, 0, 0)$, a non-collinear mean vector
- The fourth set of mean vectors used to generate the data were of length $p_4 = 12$ with:
  - $\boldsymbol{\mu}_{14} = (5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16)$, a baseline mean vector
  - $\boldsymbol{\mu}_{24} = (-5, -6, -7, -8, -9, -10, -11, -12, -13, -14, -15, -16)$, a negatively collinear mean vector
  - $\boldsymbol{\mu}_{34} = (15, 18, 21, 24, 27, 30, 33, 36, 39, 42, 45, 48)$, a positively collinear mean vector
  - $\boldsymbol{\mu}_{44} = (0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0)$, a non-collinear mean vector

The simulation study was conducted using mean vectors of increasing length to examine the multidimensional scaling ability of the two algorithms. Then, to assess the ability of the two clustering algorithms to group data generated from different mean vector settings, two sets of mean vectors of length 4 were used to generate the data with one set being non-overlapping and the other set overlapping.

For each simulation study setting s, values of $MinPts$ and ϵ were chosen for the DBSCAN algorithm according to the heuristics given in Section 2.1. The k-nearest neighbor distance plots used to select the values of ϵ for each simulation study setting are shown in Figure 1.

Within each iteration of the simulation study, after the data was generated and standardized, the DBSCAN algorithm was then implemented to cluster the data. Then, due to the fact that the DBSCAN algorithm automatically identifies the number of clusters in the data without prior knowledge, it was first checked whether the DBSCAN algorithm found the correct number of "true" underlying clusters.

Next, if the DBSCAN algorithm had found the correct number of "true" underlying clusters, the percent of the data points left unclustered was recorded. Then, on the set of data points that the DBSCAN algorithm

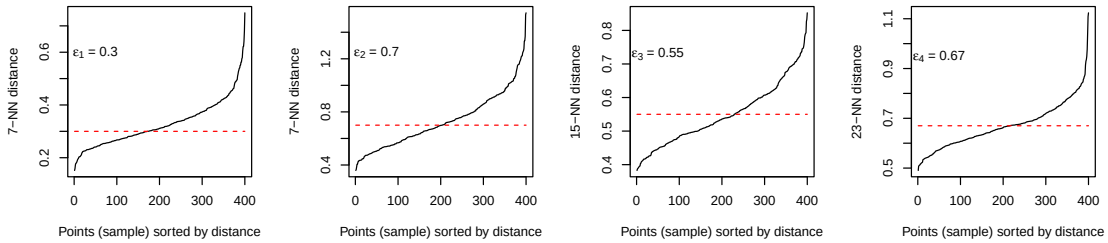

Figure 1: k-nearest neighbor distance plots for $s \in \{1, 2, 3, 4\}$ and their respective values of ϵ.

had clustered, k-means was utilized to cluster that same set of data points into 4 clusters as a comparison. Finally, the accuracy of group identification, as measured by the proportion of data points assigned to the correct group was recorded to evaluate the performance of the two clustering algorithms.

3.2 Results

Table 1: Simulation Study Results

s	Found Groups	Percent Noise	DBSCAN Accuracy	k-means Accuracy
1	0.9113	0.2067	0.8991	0.7228
2	0.0037	0.0008	0.0012	0.0017
3	1.0000	0.0523	1.0000	0.8802
4	1.0000	0.0225	1.0000	0.7892

An examination of Table 1 reveals that in the settings that used non-overlapping mean vectors ($s \in \{1, 3, 4\}$), there were four key results. The first result to note is that the DBSCAN algorithm was able to find the "true" underlying number of groups a significant majority or all of the time. The second result to note is that the DBSCAN algorithm classified less data points as noise as the dimensionality of the mean vectors increased. This phenomenon makes sense as the mean vectors were increasing in complexity as the dimensionality increased, which allowed for greater values of ϵ. This in turn caused the DBSCAN algorithm to classify fewer data points as noise. The third result to note is that the DBSCAN algorithm improved in its classification accuracy of the data points as the dimensionality of the mean vectors increased. This trend did not hold for the k-means algorithm. This behavior in the DBSCAN algorithm is most likely a result of the fact that the mean vectors were less likely to overlap as the dimensionality increased. The final key result to mention is that the DBSCAN algorithm more accurately classified the data points than the k-means algorithm did in all three settings.

A further examination of Table 1 reveals that in the setting that used heavily overlapping mean vectors ($s = 2$), both the DBSCAN algorithm and the k-means algorithm performed poorly. Specifically, the DBSCAN algorithm rarely found the correct number of "true" underlying groups and was widely inaccurate in its classification of the data points. It should also be noted that the k-means algorithm was just as inaccurate in its classification of the data points.

Figure 2 provides a visual representation of the clusters found by the DBSCAN algorithm in the settings of non-overlapping and overlapping mean vectors ($s \in \{1, 2\}$). An examination of Figure 2 further highlights the inefficiency of the DBSCAN algorithm to find distinct clusters in the setting where overlapping mean vectors were used to generate the data as it cannot separate out the groups. Given this inefficiency of the DBSCAN algorithm, it gives the indication that the DBSCAN algorithm relies on separation to be able to identify distinct clusters.

Overall, the simulation study conducted to evaluate the performances of the DBSCAN algorithm and the k-means algorithm revealed that the DBSCAN algorithm is robust to multidimensional scaling and excels

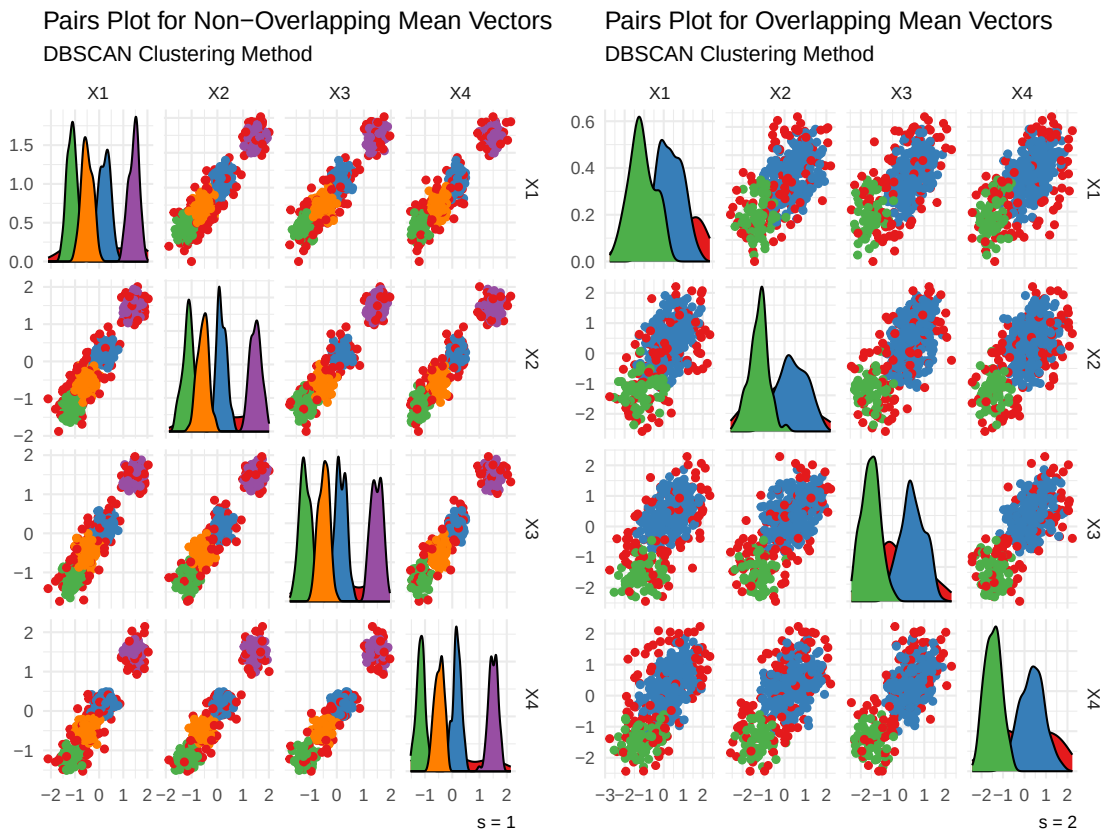

Figure 2: Pairs plots for one iteration of the simulation study for the first two settings, $s \in \{1, 2\}$. The red points represent the data points that the DBSCAN algorithm classified as noise.

at clustering when sufficient separation exists. In the settings where mean vectors with a non-overlapping structure were used to generate the data, the DBSCAN algorithm consistently outperformed the k-means algorithm. Finally, the simulation study revealed a shared weakness between the DBSCAN and k-means algorithms that both methods perform poorly when the data was generated from heavily overlapping mean vectors.

4 Application

4.1 Dataset

The real-world dataset that was used with the DBSCAN algorithm comes from the 2014 season of the Professional Golf Association (PGA). It included information such as the player name, the course being played on, the slope of the course, the players' strokes gained (SG) in several categories, and the score on a common par of 75 (CP). There were 1,086 observations, with 22 different players and 18 different courses. Every course played by a given athlete was played four times by that athlete. The columns used for clustering were SG while putting, SG from tee to green, SG on approach, SG around the green, CP, course slope, player's average driving distances, and the number of hits in regulation onto the green.

4.2 Results

After finding the appropriate values of $MinPts$ and ϵ using the heuristic methods of Section 2.1 to be 16 and 1.4 respectively, the DBSCAN algorithm was used to cluster the data. Using DBSCAN on the data after standardization, 16 clusters were identified with a median group size of 41.5 observations.

Table 2: Group attributes of clusters identified by DBSCAN for the golf data

Cluster	Noise	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
Observations	182	49	40	44	343	43	34	26	16	40	35	24	26	45	47	48	44
Players	19	1	1	1	9	1	1	1	1	1	1	1	2	1	1	2	1
Courses	18	13	11	13	18	11	11	9	5	13	12	7	9	12	14	13	12

After examining the contents of each cluster in Table 2, it was evident that the majority of the clusters (with the exception of cluster 4) were composed of a single player across several courses.

It is significant that three or four observations of a course were included in the same cluster for the same player, as this indicated that the player performed similarly on each of the courses. Overall these results were encouraging, since they implied that the DBSCAN algorithm found defined groups not included in the provided data, such as course and player. More specifically, with cluster 4, given that it contained roughly a third of the player/course combinations which were fairly similar, this cluster could be interpreted as an "average player on an average course" group. Finally, the 182 noise points identified by the DBSCAN algorithm can be thought of as outliers representing an instance where a player's performance on a particular course was distinct from all other performances in the dataset in some way.

5 Discussion

As shown in the simulations, the DBSCAN algorithm is quite competent at identifying collinear, but distinct groups even as the dimensionality of the data increases. It has some failings when identifying groups with overlapping means, but seemed to have similar failings to the k-means algorithm in those situations. Additionally, when applied to a real-world dataset, it seemed to identify known groups and find relationships that implied similarity between group members. Therefore, there seems to be practical use for this algorithm in both simulated and real-world scenarios.

After conducting this analysis, if all points are desired to be clustered, one proposal would be to use the centroids of the clusters identified by the DBSCAN algorithm as the initial cluster centroids in the k-means algorithm.

While this analysis explored some aspects of the DBSCAN algorithm, it is suggested that there be further exploration to continue to evaluate the robustness of this algorithm. Given that the simulation study in this analysis generated "true" underlying groups of equal size from normal data with identical covariance matrices, future work should include simulation studies where the data was generated from other settings. Generating data from non-normal distributions, generating data with groups of unequal size, and generating data with dissimilar covariance matrices are all data generation methods that should be considered in future simulation studies to further explore the abilities of the DBSCAN algorithm.

References

Ester, M., Kriegel, H.-P., Sander, J., Xu, X., et al. (1996). A density-based algorithm for discovering clusters in large spatial databases with noise. In *kdd*, volume 96, pages 226–231.

Rencher, A. C. and Christensen, W. F. (2012). *Cluster Analysis*, page 532–539. Wiley.

Sander, J., Ester, M., Kriegel, H.-P., and Xu, X. (1998). Density-based clustering in spatial databases: The algorithm gdbscan and its applications. *Data mining and knowledge discovery*, 2:169–194.

Schubert, E., Sander, J., Ester, M., Kriegel, H. P., and Xu, X. (2017). Dbscan revisited, revisited: Why and how you should (still) use dbscan. *ACM Trans. Database Syst.*, 42(3).

Appendix A: Analysis Code

```r
# Reading in necessary packages
library(dbscan)
library(dplyr)
library(ggplot2)
library(grid)
library(gridExtra)

######################
## Simulation Study ##
######################

###########
## Setup ##
###########

truth <- c(rep(1, 100), rep(2, 100), rep(3, 100), rep(4, 100))

# Collinear mean vectors
mean_vecs1 <- list(
  c(5:8),
  c(-5:-8),
  c(seq(from = 15, to = 24, by = 3)),
  c(rep(0, 4))
)

# Overlapping mean vectors
mean_vecs2 <- list(
  c(5:8),
  c(3:6),
  c(7:10),
  c(rep(0, 4))
)

mean_vecs3 <- list(
  c(5:12),
  c(-5:-12),
  c(seq(from = 15, to = 36, by = 3)),
  c(rep(0, 8))
)

mean_vecs4 <- list(
  c(5:16),
  c(-5:-16),
  c(seq(from = 15, to = 48, by = 3)),
  c(rep(0, 12))
)

# Covaraince matrices for each setting of mean vectors
cov_mat1 <- matrix(.5, nrow = 4, ncol = 4)
cov_mat1[row(cov_mat1) == col(cov_mat1)] = 5

cov_mat2 <- matrix(.5, nrow = 4, ncol = 4)
cov_mat2[row(cov_mat2) == col(cov_mat2)] = 5

cov_mat3 <- matrix(.5, nrow = 8, ncol = 8)
cov_mat3[row(cov_mat3) == col(cov_mat3)] = 5

cov_mat4 <- matrix(.5, nrow = 12, ncol = 12)
cov_mat4[row(cov_mat4) == col(cov_mat4)] = 5

n_vec <- rep(100, 4)

# par(mfrow = c(1, 4), oma = c(0, 0, 2, 0))

set.seed(16)
```

```r

##############
# Setting 1 #
##############

# Generate data
X_g1 <- mvtnorm::rmvnorm(n_vec[1], mean_vecs1[[1]], cov_mat1)
X_g2 <- mvtnorm::rmvnorm(n_vec[2], mean_vecs1[[2]], cov_mat1)
X_g3 <- mvtnorm::rmvnorm(n_vec[3], mean_vecs1[[3]], cov_mat1)
X_g4 <- mvtnorm::rmvnorm(n_vec[4], mean_vecs1[[4]], cov_mat1)

# Standardize data
X <- rbind(X_g1, X_g2, X_g3, X_g4)
X <- scale(X)

# Pick eps for DBSCAN
kNNdistplot(X, minPts = 8)
lines(x = 1:400, y = rep(.3, 400), lty = 'dashed', col = "red")
text(x = 50, y = .6, expression(paste(epsilon[1], " = 0.3")))

# Use dbscan to cluster data
db_res1 <- dbscan(X, eps = .3, minPts = 8)

pairs1 <- GGally::ggpairs(
  data.frame(X),
  columns = 1:4,
  mapping = aes(color = as.factor(db_res1$cluster + 1L)),
  upper = list(continuous = "points")
) +
  scale_color_brewer(palette = "Set1") +
  scale_fill_brewer(palette = "Set1") +
  theme_minimal() +
  labs(
    title = "Pairs Plot for Collinear Mean Vectors",
    subtitle = "DBSCAN Clustering Method",
    caption = expression(paste(s, " = 1"))
  )

##############
# Setting 2 #
##############

# Generate data
X_g1 <- mvtnorm::rmvnorm(n_vec[1], mean_vecs2[[1]], cov_mat2)
X_g2 <- mvtnorm::rmvnorm(n_vec[2], mean_vecs2[[2]], cov_mat2)
X_g3 <- mvtnorm::rmvnorm(n_vec[3], mean_vecs2[[3]], cov_mat2)
X_g4 <- mvtnorm::rmvnorm(n_vec[4], mean_vecs2[[4]], cov_mat2)

# Standardize data
X <- rbind(X_g1, X_g2, X_g3, X_g4)
X <- scale(X)

# Pick eps for DBSCAN
kNNdistplot(X, minPts = 8)
lines(x = 1:400, y = rep(.7, 400), lty = 'dashed', col = "red")
text(x = 50, y = .9, expression(paste(epsilon[2], " = 0.7")))

# Use dbscan to cluster data
db_res2 <- dbscan(X, eps = .7, minPts = 8)

pairs2 <- GGally::ggpairs(
  data.frame(X),
  columns = 1:4,
  mapping = aes(color = as.factor(db_res2$cluster + 1L)),
  upper = list(continuous = "points")
) +
  scale_color_brewer(palette = "Set1") +
  scale_fill_brewer(palette = "Set1") +
```

```
  theme_minimal() +
  labs(
    title = "Pairs Plot for Overlapping Mean Vectors",
    subtitle = "DBSCAN Clustering Method",
    caption = expression(paste(s, " = 2"))
  )

g1 <- grid.grabExpr(print(pairs1))
g2 <- grid.grabExpr(print(pairs2))
grid.arrange(g1, g2, widths = c(0.5, 0.5))

##############
# Setting 3 #
##############

# Generate data
X_g1 <- mvtnorm::rmvnorm(n_vec[1], mean_vecs3[[1]], cov_mat3)
X_g2 <- mvtnorm::rmvnorm(n_vec[2], mean_vecs3[[2]], cov_mat3)
X_g3 <- mvtnorm::rmvnorm(n_vec[3], mean_vecs3[[3]], cov_mat3)
X_g4 <- mvtnorm::rmvnorm(n_vec[4], mean_vecs3[[4]], cov_mat3)

# Standardize data
X <- rbind(X_g1, X_g2, X_g3, X_g4)
X <- scale(X)

# Pick eps for DBSCAN
kNNdistplot(X, minPts = 16)
lines(x = 1:400, y = rep(.55, 400), lty = 'dashed', col = "red")
text(x = 50, y = .8, expression(paste(epsilon[3], " = 0.55")))

##############
# Setting 4 #
##############

# Generate data
X_g1 <- mvtnorm::rmvnorm(n_vec[1], mean_vecs4[[1]], cov_mat4)
X_g2 <- mvtnorm::rmvnorm(n_vec[2], mean_vecs4[[2]], cov_mat4)
X_g3 <- mvtnorm::rmvnorm(n_vec[3], mean_vecs4[[3]], cov_mat4)
X_g4 <- mvtnorm::rmvnorm(n_vec[4], mean_vecs4[[4]], cov_mat4)

# Standardize data
X <- rbind(X_g1, X_g2, X_g3, X_g4)
X <- scale(X)

# Pick eps for DBSCAN
kNNdistplot(X, minPts = 24)
lines(x = 1:400, y = rep(.67, 400), lty = 'dashed', col = "red")
text(x = 50, y = .95, expression(paste(epsilon[4], " = 0.67")))

# Add a main title to the layout
mtext(
  "Figure 1: kNN Dist Plots with Varying minPts",
  outer = TRUE,
  cex = 1.5,
  line = 0,
  font = 2
)

# Add a subtitle to the layout
mtext(
  expression(paste("Red dashed line represents ", epsilon)),
  outer = TRUE,
  cex = 1,
  line = -2
)

#make into a function so we can change by mean vectors
#do simulations three times, once for mean vectors of length 4,5, and 6 respectively
```

```r
set.seed(16)
do_sim <- function(mean_vecs, n_vec, cov_mat, eps) {

  sims <- 10000
  n_groups <- 4
  found_groups <- logical(sims)
  db_acc <- numeric(sims)
  kmean_acc <- numeric(sims)
  db_rand <- numeric(sims)
  kmean_rand <- numeric(sims)
  perc_noise <- numeric(sims)

  for (i in 1:sims) {

    # Generate data
    X_g1 <- mvtnorm::rmvnorm(n_vec[1], mean_vecs[[1]], cov_mat)
    X_g2 <- mvtnorm::rmvnorm(n_vec[2], mean_vecs[[2]], cov_mat)
    X_g3 <- mvtnorm::rmvnorm(n_vec[3], mean_vecs[[3]], cov_mat)
    X_g4 <- mvtnorm::rmvnorm(n_vec[4], mean_vecs[[4]], cov_mat)
    X <- rbind(X_g1, X_g2, X_g3, X_g4)

    # Standardize data
    X <- scale(X)

    # Use dbscan to cluster data
    db_res <- dbscan::dbscan(X, eps = eps, minPts = 2*length(mean_vecs[[1]]))$cluster

    # Check if dbscan found 4 groups
    if (max(db_res) != n_groups) {
      found_groups[i] <- FALSE
      next
    }

    # How many point did dbscan classify as noise?
    perc_noise[i] <- sum(db_res == 0) / length(db_res)
    found_groups[i] <- TRUE

    # If 4 group were found by dbscan, compare it to k-means
    k_res1 <- kmeans(X[db_res != 0,], n_groups)$cluster
    k_res2 <- k_res1
    u_k <- unique(k_res1)
    k_res2[k_res1 == u_k[1]] <- 1
    k_res2[k_res1 == u_k[2]] <- 2
    k_res2[k_res1 == u_k[3]] <- 3
    k_res2[k_res1 == u_k[4]] <- 4

    # Check if groups were identified correctly bewteen the two methods
    db_red <- db_res[db_res != 0]
    db_acc[i] <- sum(db_red == truth[db_res != 0]) / length(db_red)
    kmean_acc[i] <- sum(k_res2 == truth[db_res != 0]) / length(db_red)
    # db_rand[i] <- fossil::adj.rand.index(db_red, truth[db_res != 0])
    # kmean_rand[i] <- fossil::adj.rand.index(k_res2, truth[db_res != 0])

  }

  # Return metrics
  return(
    list(
      perc_noise = perc_noise,
      found_groups = found_groups,
      db_acc = db_acc,
      db_rand = db_rand,
      kmean_acc = kmean_acc,
      kmean_rand = kmean_rand)
    )

}
```

```r
sim1 <- do_sim(mean_vecs1, n_vec, cov_mat1, .3)

sim2 <- do_sim(mean_vecs2, n_vec, cov_mat2, .7)

sim3 <- do_sim(mean_vecs3, n_vec, cov_mat3, .55)

sim4 <- do_sim(mean_vecs4, n_vec, cov_mat4, .67)

data.frame(
  setting = c(1:4),
  founds_groups = c(mean(sim1[['found_groups']]), mean(sim2[['found_groups']]), mean(sim3[['found_groups']]), mean(sim4[['found_groups']])),
  perc_noise = c(mean(sim1[['perc_noise']]), mean(sim2[['perc_noise']]), mean(sim3[['perc_noise']]), mean(sim4[['perc_noise']])),
  dbscan_acc  = c(mean(sim1[['db_acc']]), mean(sim2[['db_acc']]), mean(sim3[['db_acc']]), mean(sim4[['db_acc']])),
  kmeans_acc = c(mean(sim1[['kmean_acc']]), mean(sim2[['kmean_acc']]), mean(sim3[['kmean_acc']]), mean(sim4[['kmean_acc']]))
) %>%
  knitr::kable(
    format = "latex",
    digits = 4,
    col.names = c("$s$", "Found Groups", "Percent Noise", "DBSCAN Accuracy", "k-means Accuracy"),
    booktabs = TRUE
  )

##################
## Application ##
##################

golf_dat <- read.csv('2014GolfMod1.csv')
X <- golf_dat[, c(11:14, 24:27)]

# Standardize data
X <- scale(X)

# Pick eps for DBSCAN
kNNdistplot(X, minPts = 16)

db_res <- dbscan(X, eps = 1.4, minPts = 16)
db_res

pairs(X, col = db_res$cluster + 1L)

golf_red <- golf_dat[db_res$cluster == 1,]

golf_dat$cluster <- db_res$cluster
golf_dat  %>%
  group_by(cluster) %>%
  summarize(n_players = length(unique(Player.Name)), n_courses = length(unique(Course.Number))) %>%
  print(n = 29)
```