# Project Documentation

## Summary

The main contributions are based on sets of experiments structured in 3 sections. The synthetic data (section 3 and 4) experiments are intended to (1) highlight specific problems often reported to occur in cluster analysis and how topological manifold learning can solve these and (2) to show the subtleties of applying manifold learning as a tool to infer topological data structures. The real data experiments (section 4) are intend to substantiate the findings based on synthetic data on real world problems. Additional results are reported in the appendices 1 - 3.

***Important Note***
Some data files are too large to be uploaded to GitHub and need to be downloaded from https://syncandshare.lrz.de/getlink/fiSaZPAV3jzvGjAikmDQM3JS/ before `00_project-doc.Rmd` can be run!

Files to be placed in folder `data`:

- `embs_2D_layout_real_dat.RData`
- `embs_real_dat.RData`
- `sec4_real_dats.RData`

Files to be placed in folder `data/datasets`

- `mnist_train.csv`

## Technical aspects

This document gathers all necessary information for reproducibility. In the default setup, knitting the document takes a couple (~ 10) minutes. All relevant code will be part of the output, computation heavy experiments are – however – not executed. The corresponding code chunks are set to `eval=FALSE` and provide rough estimates for computation times for the individual experiments (obtained on a standard notebook with Linux Mint 19.2 Cinnamon, 1.90GHz × 4, 16 GB RAM, 8 cores). To run individual experiments the setup chunk needs to be run!

```
knitr::opts_chunk$set(
  message = FALSE,
  warning = FALSE
)

source("R/setup.R")
source("R/help_funs.R")
```

# Section 3

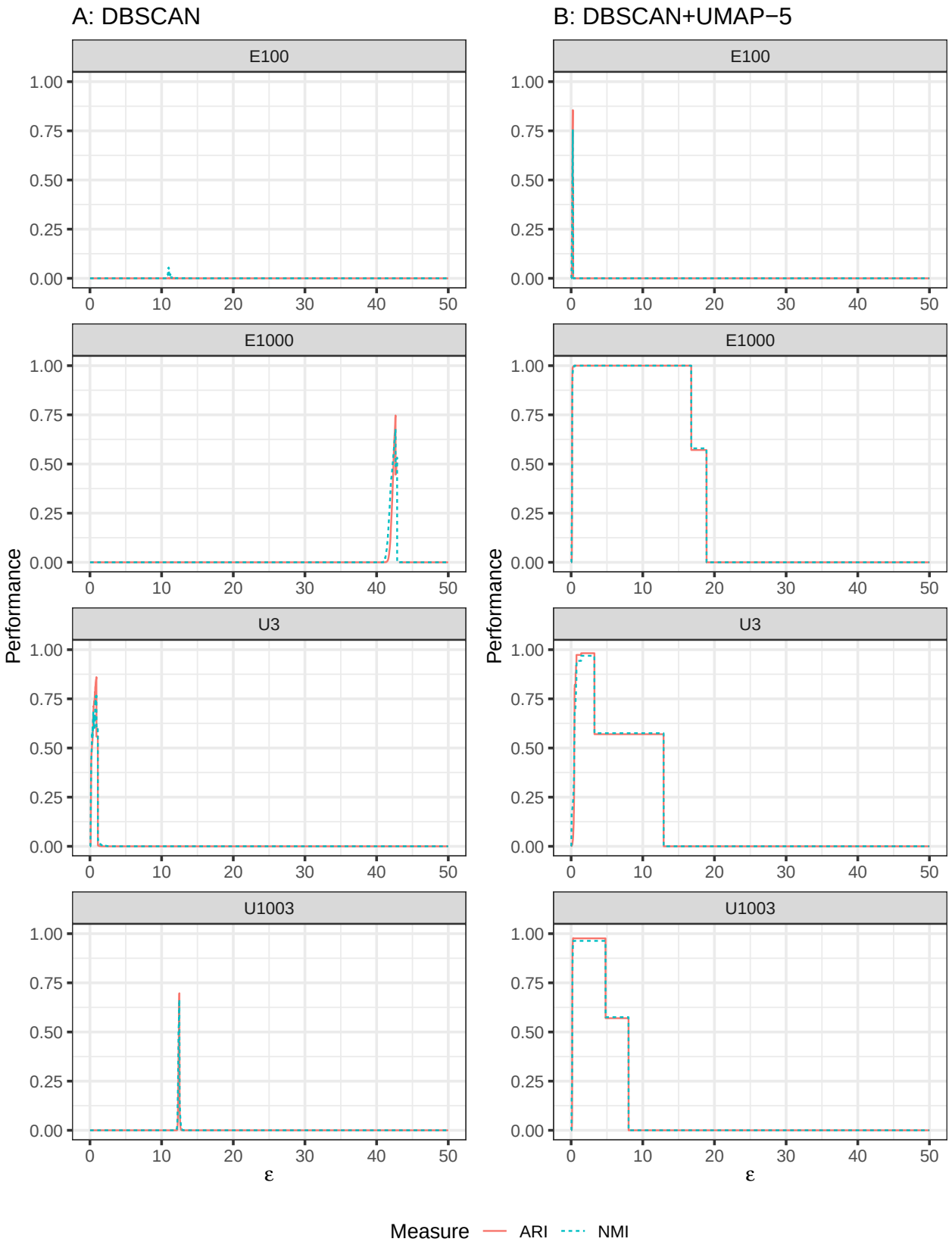
## 3.1 Experiments on the synthetic datasets $E_{100}$, $E_{1000}$, $U_3$ & $U_{1003}$.

```
# Number of clusters
n_g <- 3
n <- 500 # obs per clust
lbls <- rep(seq_len(3), each = n)

# UMAP
k <- 5 # umaps locality parameter
d <- 2 # dimensionality of embedding

# DBSCAN
eps_range <- seq(0.01, 50, by = 0.01)

# Data
equal_dens <-
  list(
    means = c(0, 0.5, 1),
    n = n,
    sigs = rep(1, n_g)
  )

unequal_dens <-
  list(
    p = 3,
    n = n,
    means = c(0, 3, 7),
    sigs = c(0.1, 1, 3)
  )

exp_settings <-
  list(
    e100 = c(
      list(p = 100),
      equal_dens
    ),
    e1000 = c(
      list(p = 1000),
      equal_dens
    ),
    u3 = unequal_dens,
    u1003 = unequal_dens
)
```

```
cluster_dat <- function(p, means, sigs, n) {
  cov_mats <- lapply(sigs, function(sig) diag(rep(sig, p * p), p, p))

  clusts <- mapply(function(mu, sig) MASS::mvrnorm(n = n, mu = rep(mu, p), Sigma = sig),
                   mu = means,
                   sig = cov_mats,
                   SIMPLIFY = FALSE)
```

```
  do.call(rbind, clusts)
}

set.seed(112)
exp_sec2_dat <- lapply(exp_settings, function(set) do.call(cluster_dat, set))
# set random states for UMAP embeddings reproducibility
set.seed(113)
rand_states <- sample(1:10000000, 4)

# add irrelevant features to create setting U_1000
exp_sec2_dat$u1003 <- cbind(exp_sec2_dat$u1003, matrix(runif(1500000), ncol = 1000))

# precomputing distances reduces computation time in dbscan if p is large
exp_sec2_dists <- lapply(exp_sec2_dat, dist)

exp_sec2_emb_dists <-
  mapply(function(dis, rand) {
           emb <- umap::umap(as.matrix(dis), random_state = rand,
                             n_neighbors = k, n_components = d, input = "dist")
           emb_dists <- dist(emb$layout)
           emb_dists
         },
         dis = exp_sec2_dists,
         rand = rand_states,
         SIMPLIFY = FALSE)
```

```
# approx. computation time for eps_range = seq(0.01, 50, by = 0.01): 2.5 h on 7 cores
if (Sys.info()[["sysname"]] == "Windows") {
  cores_to_use <- 1
} else {
  cores_to_use <- 7
}

exp_sec2_res <- parallel::mclapply(
  c(exp_sec2_dists, exp_sec2_emb_dists),
  function(dat) cluster_res(dat, eps_range = eps_range, lbls = lbls),
  mc.cores = cores_to_use
)

dt_sec2_res <- cbind(
  method = rep(c("DBSCAN", "UMAP+DBSCAN"), each = length(exp_settings) * length(eps_range)),
  setting = rep(names(exp_sec2_res), each = length(eps_range)),
  eps = rep(eps_range, length(exp_sec2_res)),
  as.data.table(do.call(rbind, exp_sec2_res))
)

dt_sec2_res_long <- melt(dt_sec2_res,
                         id.vars = c("method", "setting", "eps"),
                         measure.vars = c("ARI", "NMI"),
                         variable.name = "Measure",
                         value.name = "Performance")

# save(dt_sec2_res_long, file = "vignettes/data/res_sec2_long-range.RData")
```




```
# Optimal values
dt_sec2_res_long[method == "DBSCAN", max(Performance), by = c("nic_set", "Measure")]
```

```
##    nic_set Measure     V1
## 1:    E100     ARI 0.0025
## 2:   E1000     ARI 0.7469
## 3:      U3     ARI 0.8604
## 4:   U1003     ARI 0.6970
## 5:    E100     NMI 0.0539
## 6:   E1000     NMI 0.6780
## 7:      U3     NMI 0.7662
## 8:   U1003     NMI 0.6645
```

```
dt_temp <- dt_sec2_res_long
dt_temp[dt_temp[, .I[which.max(Performance)], by = c("method", "nic_set", "Measure")]$V1]
```

```
##          method setting   eps Measure Performance nic_set
##  1:      DBSCAN    e100 11.32     ARI      0.0025    E100
##  2:      DBSCAN   e1000 42.64     ARI      0.7469   E1000
##  3:      DBSCAN      u3  0.90     ARI      0.8604      U3
##  4:      DBSCAN   u1003 12.48     ARI      0.6970   U1003
##  5: UMAP+DBSCAN    e100  0.24     ARI      0.8554    E100
##  6: UMAP+DBSCAN   e1000  0.46     ARI      1.0000   E1000
##  7: UMAP+DBSCAN      u3  1.41     ARI      0.9821      U3
##  8: UMAP+DBSCAN   u1003  0.28     ARI      0.9772   U1003
##  9:      DBSCAN    e100 10.98     NMI      0.0539    E100
## 10:      DBSCAN   e1000 42.64     NMI      0.6780   E1000
## 11:      DBSCAN      u3  0.90     NMI      0.7662      U3
## 12:      DBSCAN   u1003 12.48     NMI      0.6645   U1003
## 13: UMAP+DBSCAN    e100  0.24     NMI      0.7568    E100
## 14: UMAP+DBSCAN   e1000  0.46     NMI      1.0000   E1000
## 15: UMAP+DBSCAN      u3  1.41     NMI      0.9689      U3
## 16: UMAP+DBSCAN   u1003  0.33     NMI      0.9634   U1003
```

### 3.2 Toy example

Experiments on the simply toy example.

```
toy_exp <- matrix(c(0, 0.6, 0.7, 1.3, 1.2, 1.5,
                    0.6, 0, 0.5, 0.75, 1.6, 1.3,
                    0.7, 0.5, 0, 1.4, 1.3, 1.1,
                    1.3, 0.75, 1.4, 0, 0.7, 0.75,
                    1.2, 1.6, 1.3, 0.7, 0, 0.75,
                    1.5, 1.3, 1.1, 0.75, 0.75, 0),
                  nrow = 6)

lbls <- c(1, 1, 1, 2, 2, 2)
ks <- c(k6 = 6, k3 = 3, k2 = 2)

# We use uwot::umap implementation here.
# Other than umap::umap, which is used in the other experiments,
# uwot::umap provides the high dim fuzzy graph, i.e. the topological
# representation (fgraph) as sparse matrix, however it is less well
# reproducible.
# - non-zero entries give the membership strength of th edge connecting
#   i to j (i.e. prob, similarity in tSNE)
# - can be used to compare the original distance matrix to the UMAP
```

```
#   induced distance metric
# - in uwot::umap default value for min_dist = 0.01 (in contrast to 0.1
#   in umap::umap). To make it consistent with the other experiments we
#   set it two umap::umap defaults (it does not change results)
# - "If you are only interested in the fuzzy input graph (e.g. for
#   clustering), setting 'n_epochs = 0' will avoid any further
#   sparsifying." (see uwot::umap help page)
min_dist <- 0.1
toy_dists <- as.dist(toy_exp)

# fuzzy graphs for different values of k (only fgraph --> n_epochs = 0)
l_embs_toy <- lapply(
  ks,
  function(k) uwot::umap(toy_dists, n_neighbors = k, ret_extra = "fgraph",
                         n_epochs = 0,
                         min_dist = min_dist)
)

lapply(l_embs_toy, function(emb) round(emb$fgraph, 2))
```

```
## $k6
## 6 x 6 sparse Matrix of class "dgCMatrix"
##
## [1,] .    1.00 0.95 0.29 0.53 0.25
## [2,] 1.00 .    1.00 0.90 0.19 0.30
## [3,] 0.95 1.00 .    0.24 0.45 0.58
## [4,] 0.29 0.90 0.24 .    1.00 1.00
## [5,] 0.53 0.19 0.45 1.00 .    1.00
## [6,] 0.25 0.30 0.58 1.00 1.00 .
##
## $k3
## 6 x 6 sparse Matrix of class "dgCMatrix"
##
## [1,] .    1.00 0.83 .    . .
## [2,] 1.00 .    1.00 0.58 . .
## [3,] 0.83 1.00 .    .    . .
## [4,] .    0.58 .    .    1 1
## [5,] .    .    .    1.00 . 1
## [6,] .    .    .    1.00 1 .
##
## $k2
## 6 x 6 sparse Matrix of class "dgCMatrix"
##
## [1,] . 1 . . . .
## [2,] 1 . 1 . . .
## [3,] . 1 . . . .
## [4,] . . . . 1 1
## [5,] . . . 1 . .
## [6,] . . . 1 . .
```

```
# dbscan clustering based on fgraphs

# turn similarities into dissimilarities via 1 - v_ij
l_fgraph_dis <- lapply(
```

```
  l_embs_toy,
  function(emb) {
    temp <- as.matrix(1 - round(emb$fgraph, 2))
    diag(temp) <- 0
    as.dist(temp)
  }
)

eps_range <- seq(0, 4, by = 0.01)

# compute dbscan clusters for different eps values
opt_eps <-
  lapply(
    l_fgraph_dis,
    function(fgraph) {
      sapply(
        eps_range,
        function(eps_val) {
          dbs <- dbscan(fgraph, eps = eps_val, minPts = 3)
          all(dbs$cluster == c(1, 1, 1, 2, 2, 2))
        }
      )
    }
)

# epsilon ranges yielding optimal cluster results
lapply(
  opt_eps,
  function(eps) range(eps_range[eps])
)
```

```
## $k6
## [1] 0.00 0.09
##
## $k3
## [1] 0.00 0.42
##
## $k2
## [1] 0.00 0.99
```

```
# The same as before but on the embedding coordinates instead of the fgraph only

# full grown embeddings (2D)
set.seed(3) # !!! see https://github.com/jlmelville/uwot, A Note on Reproducibility !!!

l_embs_toy_full <- lapply(
  ks,
  function(k) uwot::umap(toy_dists, n_neighbors = k, min_dist = min_dist,
                         # For reproducibility:
                         a = 1.8956, b = 0.8006, approx_pow = TRUE, init = "spca")
)

eps_range_full <- seq(0, 40, by = 0.01)
```

```r
# compute dbscan clusters for different eps values
opt_eps_full <-
  lapply(
    l_embs_toy_full,
    function(emb) {
      sapply(
        eps_range_full,
        function(eps_val) {
          dbs <- dbscan(emb, eps = eps_val, minPts = 2)
          all(dbs$cluster == c(1, 1, 1, 2, 2, 2))
        }
      )
    }
)

# epsilon ranges yielding perfect cluster results
lapply(
  opt_eps_full,
  function(eps) range(eps_range_full[eps])
)
```

```
## $k6
## [1] 0.77 0.97
##
## $k3
## [1] 0.69 1.81
##
## $k2
## [1] 0.57 7.12
```

```r
# !!! The results may not perfectly reproducible !!!
# See https://github.com/jlmelville/uwot, A Note on Reproducibility

reps <- 25

if (Sys.info()[["sysname"]] == "Windows") {
  cores_to_use <- 1
} else {
  cores_to_use <- 3
}

# computation time ~ 3 min
l_embs_toy_rep <- lapply(
  seq_len(reps),
  function(rep) {
   parallel::mclapply(
      ks,
      function(k) {
        set.seed(rep)
        uwot::umap(toy_dists, n_neighbors = k, min_dist = min_dist,
                   # For reproducibility:
                   a = 1.8956, b = 0.8006, approx_pow = TRUE, init = "spca")
      },
      mc.cores = cores_to_use
```

```
    )
  }
)

eps_range_full <- seq(0.01, 40, by = 0.01)
opt_eps_rep <-
  lapply(
    l_embs_toy_rep,
    function(embs) lapply(
      embs,
      function(emb) {
        sapply(
          eps_range_full,
          function(eps_val) {
            dbs <- dbscan(emb, eps = eps_val, minPts = 3)
            all(dbs$cluster == c(1, 1, 1, 2, 2, 2))
          }
        )
      }
    )
)

dt_ranges_k2 <- sapply(
  opt_eps_rep,
  function(eps) range(eps_range_full[eps$k2])
)

dt_ranges_k3 <- sapply(
  opt_eps_rep,
  function(eps) range(eps_range_full[eps$k3])
)

dt_ranges_k6 <- sapply(
  opt_eps_rep,
  function(eps) range(eps_range_full[eps$k6])
)

# dt_ranges_k2
apply(dt_ranges_k2, 1, mean)
```

```
## [1]  0.7292 20.7496
```

```
dt_ranges_k2[, which.min(dt_ranges_k2[2, ] - dt_ranges_k2[1, ])]
```

```
## [1] 0.6 3.7
```

```
# dt_ranges_k3
apply(dt_ranges_k3, 1, mean)
```

```
## [1] 0.6992 9.5608
```

```
dt_ranges_k3[, which.min(dt_ranges_k3[2, ] - dt_ranges_k3[1, ])]
```

```
## [1] 0.71 1.65
```

```
# dt_ranges_k6
apply(dt_ranges_k6, 1, mean)
```

```
## [1]  Inf -Inf
```

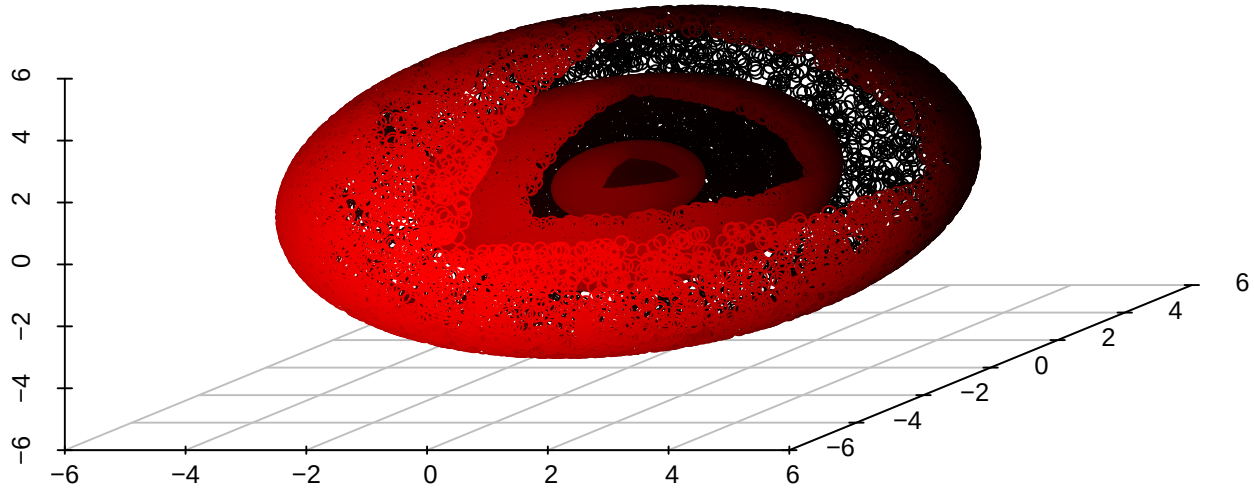
```
dt_ranges_k6[, which.min(dt_ranges_k6[2, ] - dt_ranges_k6[1, ])]
```

```
## [1]  Inf -Inf
```

# Section 4

### 4.1.1 Nested sphere example

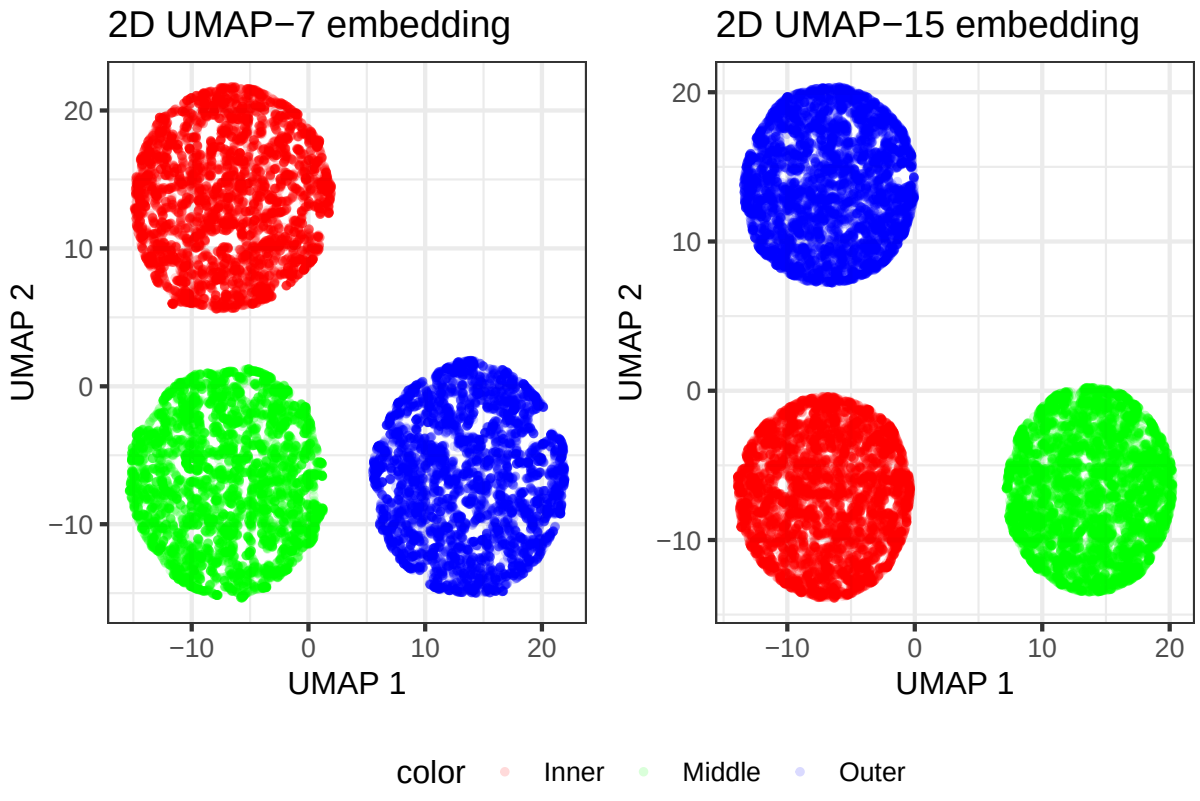
```
# data
sphere_dim <- 3
sphere_n <- 10000
radii <- c(5, 3, 1)
lbls_sphs <- factor(rep(1:3, each = sphere_n), labels = c("Inner", "Middle", "Outer"))

nested_spheres <- function(n, radii, d = 3) {
  sph_1 <- runif_on_sphere(n, d, r = radii[1])
  sph_2 <- runif_on_sphere(n, d, r = radii[2])
  sph_3 <- runif_on_sphere(n, d, r = radii[3])
  rbind(sph_1, sph_2, sph_3)
}

set.seed(3)
exp_sphs_dat <- nested_spheres(n = sphere_n, radii = radii, d = sphere_dim)
```

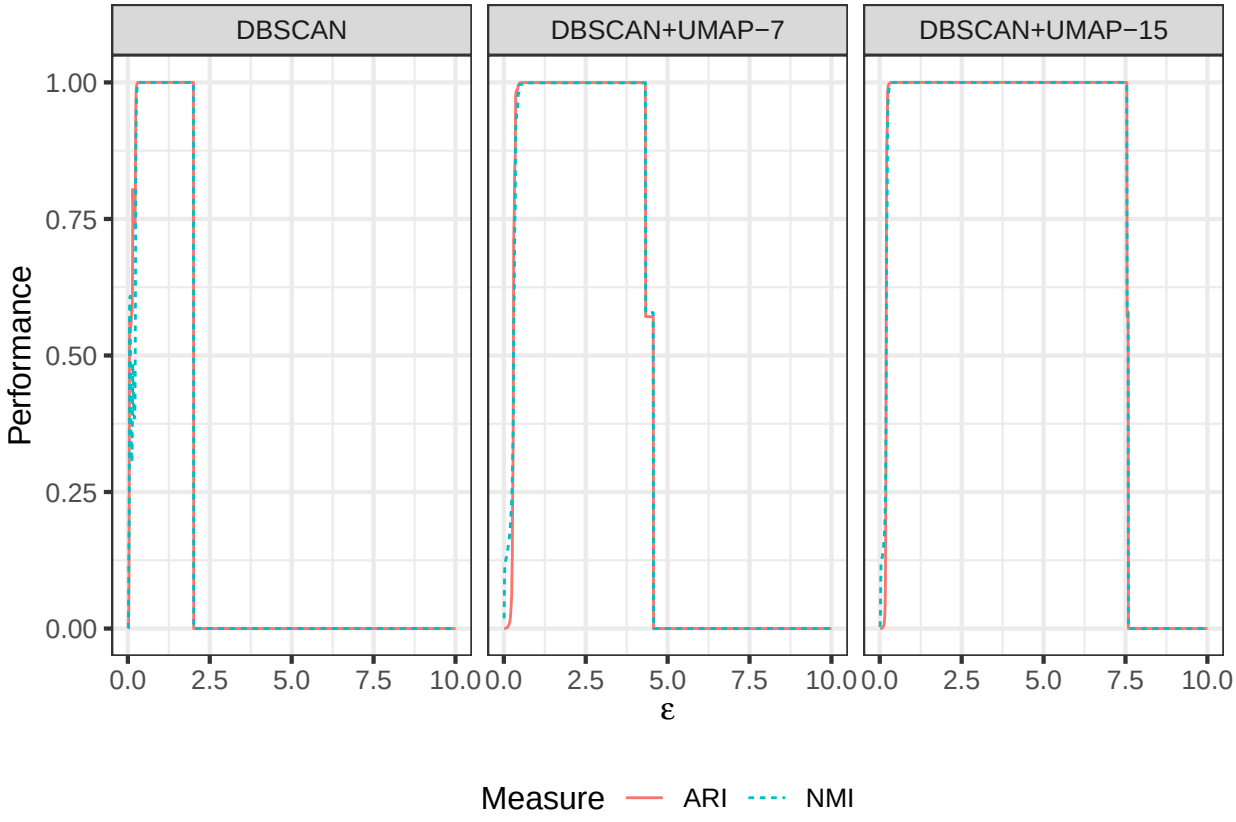



```
# experiments
set.seed(55)
# comp. time ~ 20 min
exp_sphs_emb7 <- umap::umap(exp_sphs_dat, n_neighbors = 7, n_components = 2)
exp_sphs_emb15 <- umap::umap(exp_sphs_dat, n_neighbors = 15, n_components = 2)
embs_sphs <- list(emb7 = exp_sphs_emb7, emb15 = exp_sphs_emb15)
# save(embs_sphs, file = "vignettes/data/embs_nested_sphs.RData")
```

```r
eps_range_sphs <- seq(0.01, 10, by = 0.01)

# ~ 4.1 h for n = 10000 & eps_range = seq(0.01, 10, by = 0.01)
cores_to_use <- 3
exp_sphs_res <- parallel::mclapply(
  list(exp_sphs_dat, exp_sphs_emb7$layout, exp_sphs_emb15$layout),
  function(dat) cluster_res(dat, eps_range = eps_range_sphs, lbls = lbls_sphs),
  mc.cores = cores_to_use
)

dt_sphs_res <- cbind(
  method = ordered(
    rep(c("DBSCAN", "DBSCAN+UMAP-7", "DBSCAN+UMAP-15"), each = length(eps_range_sphs)),
    levels = c("DBSCAN", "DBSCAN+UMAP-7", "DBSCAN+UMAP-15")),
  eps = rep(eps_range_sphs, 3),
  as.data.table(do.call(rbind, exp_sphs_res))
)

# save(dt_sphs_res, file = "vignettes/data/res_sec3_sphs.RData")
```

```
##            used  (Mb) gc trigger  (Mb) max used (Mb)
## Ncells  2213723 118.3    3911970 209.0  3911970  209
## Vcells 18818203 143.6   37547141 286.5 34467741  263
```

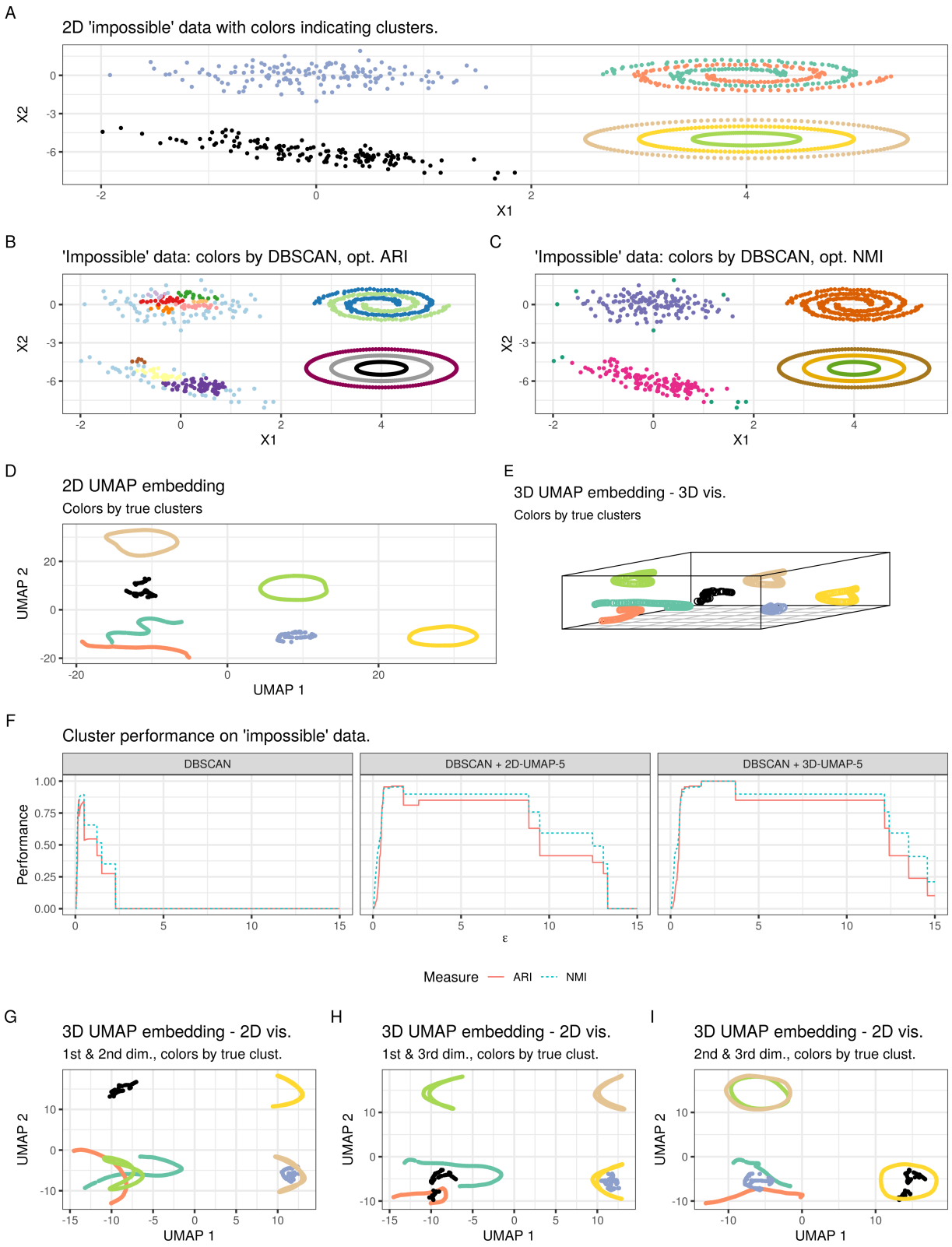





### 4.1.2 Impossible data

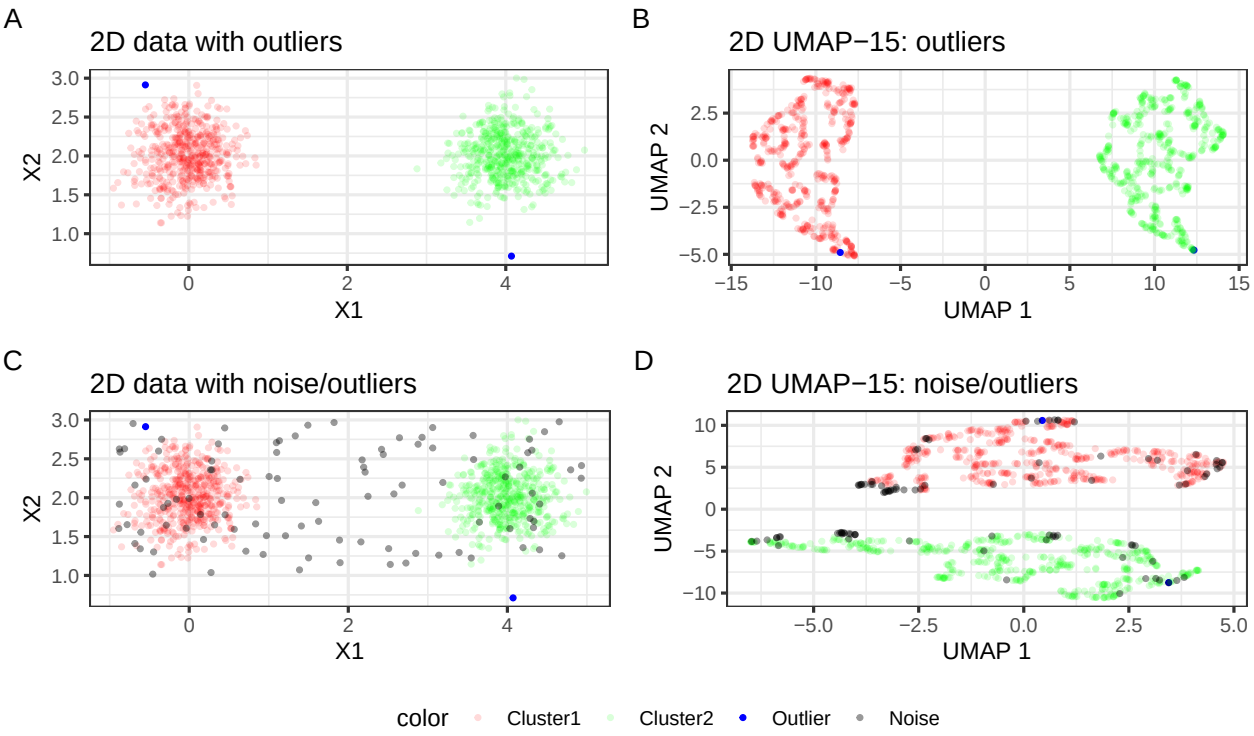
```
# Data

# construction of impossible data
set.seed(5)
sprls <- mlbench.spirals(300, 1.5, 0.03)
sprls$x[, 1] <- sprls$x[, 1] + 4
norm1 <- MASS::mvrnorm(n = 150, mu = c(0, 0), matrix(c(0.5, 0, 0, 0.5), nrow = 2))
norm2 <- MASS::mvrnorm(n = 150, mu = c(0, -6), matrix(c(0.5, -0.4, 0, 0.5), nrow = 2))

circleFun <- function(center = c(0,0), diameter = 1, npoints = 150){
  r = diameter / 2
  tt <- seq(0, 2 * pi, length.out = npoints)
  xx <- center[1] + r * cos(tt)
  yy <- center[2] + r * sin(tt)
  return(data.frame(x = xx, y = yy))
}

circles <- lapply(c(1, 2, 3), function(r) circleFun(c(4, -5), r))
imp_dat <- as.data.frame(rbind(sprls$x, norm1, norm2))
names(imp_dat) <- c("x", "y")
imp_dat <- do.call(rbind, c(list(imp_dat), circles))
imp_lbls <- factor(c(sprls$classes, rep(c(3, 4, 5, 6, 7), each = 150)))
```

```r
umap_2d <- umap::umap(imp_dat, n_neighbors = 5, n_components = 2)
umap_3d <- umap::umap(imp_dat, n_neighbors = 5, n_components = 3)

eps_range_imp <- seq(0.01, 15, by = 0.01)

# computation time ~ 1 min
cores_to_use <- 3
exp_res_imp <- parallel::mclapply(
  list(imp_dat, umap_2d$layout, umap_3d$layout),
  function(dat) cluster_res(dat, eps_range = eps_range_imp, lbls = imp_lbls),
  mc.cores = cores_to_use
)

opt_eps <- sapply(exp_res_imp, function(x) apply(x, 2, which.max))

opt_dbs_clusts <- lapply(opt_eps[, 1], function(x) dbscan(imp_dat, eps = eps_range_imp[x]))
opt_umap_clusts <- mapply(
  function(dat, eps) dbscan(dat, eps = eps_range_imp[eps]),
  eps = opt_eps[1, 2:3],
  dat = list(umap_2d$layout, umap_3d$layout)
)
```


A
2D 'impossible' data with colors indicating clusters.
X2
X1
B
'Impossible' data: colors by DBSCAN, opt. ARI
C
'Impossible' data: colors by DBSCAN, opt. NMI
D
2D UMAP embedding
Colors by true clusters
UMAP 2
UMAP 1
E
3D UMAP embedding - 3D vis.
Colors by true clusters
F
Cluster performance on 'impossible' data.
DBSCAN
DBSCAN + 2D-UMAP-5
DBSCAN + 3D-UMAP-5
Performance
ε
Measure ARI NMI
G
3D UMAP embedding - 2D vis.
1st & 2nd dim., colors by true clust.
H
3D UMAP embedding - 2D vis.
1st & 3rd dim., colors by true clust.
I
3D UMAP embedding - 2D vis.
2nd & 3rd dim., colors by true clust.

### 4.2 Outlier and noise points

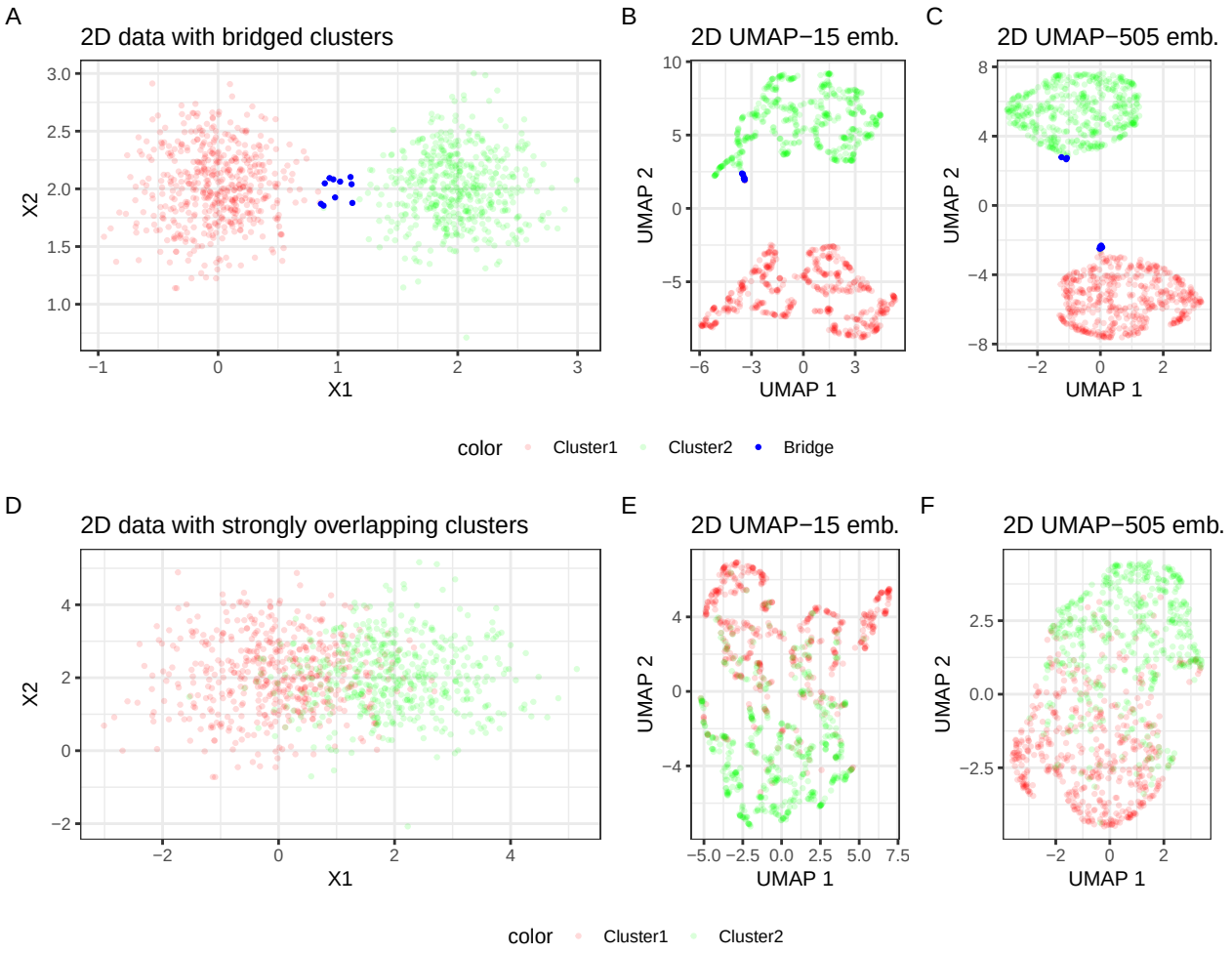
```r
# data setup

p <- 2
n <- 500
means <- list(c(0, 2), c(4, 2))
sig <- diag(rep(0.1, p * p), p, p)


# outlier

set.seed(2)
exp_outs_dat <- do.call(
  rbind,
  lapply(means, function(mu) mvrnorm(n = n, mu = mu, Sigma = sig))
)
exp_outs_emb <- umap::umap(exp_outs_dat, n_neighbors = 15, random_state = 2)

outs <- exp_outs_dat[, 2] < 1 | (exp_outs_dat[, 2] > 2.75 & exp_outs_dat[, 1] < 0)
lbls_outs <- rep(1:2, each = n)
lbls_outs[outs] <- 3
lbls_outs <- factor(lbls_outs, levels = c(1, 2, 3, 4),
                    labels = c("Cluster1", "Cluster2", "Outlier", "Noise"))


# noise

set.seed(4015)
noise <- cbind(runif(100, -1, 5),  runif(100, 1, 3))
exp_noisy_dat <- rbind(exp_outs_dat, noise)
lbls_noise <- c(lbls_outs, factor(rep("Noise", 100)))
exp_noise_emb <- umap::umap(exp_noisy_dat, n_neighbors = 15, random_state = 2)
```




### 4.3 Connected components

```
# data setup

p <- 2
n <- 500
means <- list(
  c(0, 2),
  c(2, 2)
)

# cc: bridged

sig_bridge <- diag(rep(0.1, p * p), p, p)

set.seed(2)
clusters <- do.call(
  rbind,
  lapply(means, function(mu) mvrnorm(n = n, mu = mu, Sigma = sig_bridge))
)
bridge <- cbind(runif(10, 0.85, 1.15), runif(10, 1.85, 2.15))
exp_bridge_dat <- rbind(clusters, bridge)

exp_bridge_embs <- lapply(
  c(k15 = 15, k505 = 505),
  function(k) umap::umap(exp_bridge_dat, n_neighbors = k, random_state = 331)
)

lbls_bridge <- factor(
```

```
  c(rep(1:2, each = n), rep(3, 10)),
  labels = c("Cluster1", "Cluster2", "Bridge")
)


# cc: overlap

sig_overlap <- diag(rep(1, p * p), p, p)
set.seed(2)
exp_overlap_dat <- do.call(
  rbind,
  lapply(means, function(mu) mvrnorm(n = n, mu = mu, Sigma = sig_overlap))
)

exp_overlap_embs <- lapply(
  c(k15 = 15, k505 = 505),
  function(k) umap::umap(exp_overlap_dat, n_neighbors = k, random_state = 332)
)

lbls_overlap <- factor(c(rep(1:2, each = n)), labels = c("Cluster1", "Cluster2"))
```



### 4.4 FCPS data

```
## settings

# UMAP
k <- 10
rand_state <- 10

# DBSCAN
eps_range_fcps <- seq(0.01, 20, by = 0.01)


## data

datsets <- c("Hepta", "Lsun", "Tetra", "Chainlink", "Atom",
             "EngyTime", "Target", "TwoDiamonds", "WingNut", "GolfBall")

exp_fcps_dats <- sapply(
  datsets,
  function(dat) {
     d <- read.table(paste0("data/FCPS/01FCPSdata/", dat, ".txt"))
     d <- d[, -1] # remove useless index column
     names(d) <- paste0("X", seq_len(ncol(d)))
     d
  }
)

lbls_fcps <- sapply(
  datsets,
  function(dat) {
    lbls <- read.table(paste0("data/FCPS/01FCPSdata/", dat, "_lbls.txt"))
    as.factor(lbls$V2)
  }
)


## embeddings

exp_fcps_embs <- lapply(
  exp_fcps_dats,
  function(dat) umap::umap(dat, n_neighbors = k, random_state = rand_state)
)


## clustering

# comp time: ~ 20 m
cores_to_use <- 5
exp_fcps_res <- parallel::mcmapply(
  function(dat, lbls) cluster_res(
    dat = dat,
    eps_range = eps_range_fcps,
    lbls = lbls_fcps[[lbls]]
```

```
  ),
  dat = c(exp_fcps_dats, lapply(exp_fcps_embs, function(emb) emb$layout)),
  lbls = rep(datsets, 2),
  SIMPLIFY = FALSE,
  mc.cores = cores_to_use
)

## results

dt_fcps_res <- cbind(
  method = factor(rep(c("DBSCAN", "DBSCAN+UMAP-10"),
                      each = length(names(exp_fcps_dats)) * length(eps_range_fcps))),
  data = rep(names(exp_fcps_res), each = length(eps_range_fcps)),
  eps = rep(eps_range_fcps, length(names(exp_fcps_res))),
  as.data.table(do.call(rbind, exp_fcps_res))
)

# save(dt_fcps_res, file = "vignettes/data/res_sec3_fcps.RData")
```

```
load("data/results/res_sec3_fcps.RData")

eps_threshold <- 0

dt_fcps_res[, .(maxARI = round(max(ARI), 2),
                maxNMI = round(max(NMI), 2)),
            by = c("data", "method")]
```

```
##            data         method maxARI maxNMI
##  1:       Hepta         DBSCAN   1.00   1.00
##  2:        Lsun         DBSCAN   1.00   1.00
##  3:       Tetra         DBSCAN   0.91   0.85
##  4:   Chainlink         DBSCAN   1.00   1.00
##  5:        Atom         DBSCAN   1.00   1.00
##  6:    EngyTime         DBSCAN   0.36   0.23
##  7:      Target         DBSCAN   1.00   0.97
##  8: TwoDiamonds         DBSCAN   0.95   0.85
##  9:     WingNut         DBSCAN   1.00   1.00
## 10:    GolfBall         DBSCAN   1.00    NaN
## 11:       Hepta DBSCAN+UMAP-10   1.00   1.00
## 12:        Lsun DBSCAN+UMAP-10   1.00   1.00
## 13:       Tetra DBSCAN+UMAP-10   0.99   0.99
## 14:   Chainlink DBSCAN+UMAP-10   1.00   1.00
## 15:        Atom DBSCAN+UMAP-10   1.00   1.00
## 16:    EngyTime DBSCAN+UMAP-10   0.29   0.26
## 17:      Target DBSCAN+UMAP-10   0.97   0.88
## 18: TwoDiamonds DBSCAN+UMAP-10   0.99   0.99
## 19:     WingNut DBSCAN+UMAP-10   1.00   1.00
## 20:    GolfBall DBSCAN+UMAP-10   1.00    NaN
```

```
dt_fcps_res[, .(eps_range_gr = range(eps[ARI > eps_threshold])),
            by = c("data", "method")]
```

```
##            data         method eps_range_gr
##  1:       Hepta         DBSCAN         0.04
##  2:       Hepta         DBSCAN         2.31
```

```
##  3:        Lsun         DBSCAN         0.09
##  4:        Lsun         DBSCAN         0.71
##  5:       Tetra         DBSCAN         0.24
##  6:       Tetra         DBSCAN         0.49
##  7:   Chainlink         DBSCAN         0.03
##  8:   Chainlink         DBSCAN         0.81
##  9:        Atom         DBSCAN         0.76
## 10:        Atom         DBSCAN        20.00
## 11:    EngyTime         DBSCAN         0.03
## 12:    EngyTime         DBSCAN         0.94
## 13:      Target         DBSCAN         0.04
## 14:      Target         DBSCAN         2.28
## 15: TwoDiamonds         DBSCAN         0.04
## 16: TwoDiamonds         DBSCAN         0.12
## 17:     WingNut         DBSCAN         0.05
## 18:     WingNut         DBSCAN         0.29
## 19:    GolfBall         DBSCAN         0.01
## 20:    GolfBall         DBSCAN        20.00
## 21:       Hepta DBSCAN+UMAP-10         0.08
## 22:       Hepta DBSCAN+UMAP-10        18.96
## 23:        Lsun DBSCAN+UMAP-10         0.06
## 24:        Lsun DBSCAN+UMAP-10        14.44
## 25:       Tetra DBSCAN+UMAP-10         0.06
## 26:       Tetra DBSCAN+UMAP-10         7.34
## 27:   Chainlink DBSCAN+UMAP-10         0.02
## 28:   Chainlink DBSCAN+UMAP-10         7.29
## 29:        Atom DBSCAN+UMAP-10         0.03
## 30:        Atom DBSCAN+UMAP-10        12.81
## 31:    EngyTime DBSCAN+UMAP-10         0.01
## 32:    EngyTime DBSCAN+UMAP-10         0.90
## 33:      Target DBSCAN+UMAP-10         0.03
## 34:      Target DBSCAN+UMAP-10        10.56
## 35: TwoDiamonds DBSCAN+UMAP-10         0.03
## 36: TwoDiamonds DBSCAN+UMAP-10         4.73
## 37:     WingNut DBSCAN+UMAP-10         0.03
## 38:     WingNut DBSCAN+UMAP-10         8.11
## 39:    GolfBall DBSCAN+UMAP-10         0.01
## 40:    GolfBall DBSCAN+UMAP-10        20.00
##            data         method eps_range_gr
```

# Section 5

## Real data experiments

```
load("data/datasets/coil20.RData")
wine <- read.csv("data/datasets/wine.csv")
pen <- read.csv("data/datasets/pendigits_csv.csv")
mnist_tr <- read.csv("data/datasets/mnist_train.csv")
mnist_te <- read.csv("data/datasets/mnist_test.csv")
mnist <- rbind(mnist_tr, mnist_te)
load_mnist() # Fashion mnist
fmnist_x <- rbind(train$x, test$x)
```

```r
fmnist_lbls <- as.factor(c(train$y, test$y))

fmnist_lbls_5 <- c(train$y, test$y)
fmnist_lbls_5[fmnist_lbls %in% c(0, 3)] <- 1
fmnist_lbls_5[fmnist_lbls == 1] <- 2
fmnist_lbls_5[fmnist_lbls %in% c(2, 4, 6)] <- 3
fmnist_lbls_5[fmnist_lbls == 8] <- 4
fmnist_lbls_5[fmnist_lbls %in% c(5, 7, 9)] <- 5
fmnist_lbls_5 <- as.factor(fmnist_lbls_5)

l_dats <- list(
  iris = list(
    dat_X = apply(iris[, -5], 2, scale, center = FALSE), # features on different scales
    dat_lbls = factor(iris[, 5], labels = 1:3),
    eps_range = seq(0.01, 100, by = 0.01),
    dat_name = "Iris"
  ),
  wine = list(
    dat_X = apply(as.matrix(wine[, -1]), 2, scale), # features on much different scales and units!
    dat_lbls = wine$Wine,
    eps_range = seq(0.01, 100, by = 0.01),
    dat_name = "Wine"
  ),
  coil = list(
    dat_X = coil20[, -ncol(coil20)],
    dat_lbls = as.numeric(coil20$Label),
    eps_range = seq(0.01, 30, by = 0.01),
    dat_nam = "COIL"
  ),
  pendigits = list(
    dat_X = as.matrix(pen[, -17]),
    dat_lbls = pen$class,
    eps_range = seq(0.01, 75, by = 0.01),
    dat_name = "pendigits"
  ),
  mnist = list(
    dat_X = mnist[, -1],
    dat_lbls = as.factor(mnist$label),
    eps_range =  seq(0.01, 3, by = 0.01),
    dat_name = "MNIST"
  ),
  fmnist = list(
    dat_X = fmnist_x,
    dat_lbls = fmnist_lbls,
    dat_lbls5 = fmnist_lbls_5,
    eps_range = seq(0.01, 3, by = 0.01),
    dat_name = "FMNIST"
  )
)

# save(l_dats, file = "vignettes/data/sec4_real_dats.RData")
```

### 4.1 DBSCAN directly applied to the data

```
load("data/sec4_real_dats.RData")

dat_selection <- c("iris", "wine", "coil", "pendigits", "mnist")
# Note: Computations time >> 24 h for all settings (depending on system)
# Only using the small datasets Iris, Wine, COIL: comp. time ~ 5 min
# dat_selection <- c("iris", "wine", "coil")

# FMNIST is excluded here and processed individually due to two label sets
# to reduce computation time
l_fmnist <- l_dats["fmnist"]
l_dats <- l_dats[dat_selection]

# coil has few observations but many features, precomputing distances saves some time
coil_x <- l_dats$coil$dat_X
l_dats$coil$dat_X <- dist(l_dats$coil$dat_X)

nic_names <- c("iris" = "Iris", "wine" = "Wine", "coil" = "COIL",
               "pendigits" = "Pendigits", "mnist" = "MNIST", "fmnist" = "FMNIST")


# clustering
# computation time: several hours (depending on system >> 24 h possible)
if (Sys.info()[["sysname"]] == "Windows") {
  cores_to_use <- 1
} else {
  cores_to_use <- 6
}

clust_res_real_dir <- parallel::mcmapply(
  function(dat) cluster_res(
    dat = dat$dat_X,
    eps_range = dat$eps_range,
    lbls = dat$dat_lbls
  ),
  dat = l_dats,
  SIMPLIFY = FALSE,
  mc.cores = cores_to_use
)

dt_res_real_d <- cbind(
  Data = unlist(
    lapply(
      names(clust_res_real_dir),
      function(nam) rep(nam, nrow(clust_res_real_dir[[nam]]))
  ),
  Method = "DBSCAN",
  eps = unlist(
    lapply(
      l_dats,
      function(dat) dat$eps_range)
  ),
```

```
  as.data.table(do.call(rbind, clust_res_real_dir))
)


dt_res_real_d$Data <- factor(
  dt_res_real_d$Data,
  labels = nic_names[levels(factor(dt_res_real_d$Data))]
)

# FMNIST
# computation time ~ 3h
clust_res_fmnist_dir <- sapply(
  l_fmnist$fmnist$eps_range,
  function(eps) dbscan::dbscan(l_fmnist$fmnist$dat_X, eps = eps)$cluster # minPts = 5 (default)
)

dt_res_fmnist_d <- lapply(
  l_fmnist$fmnist[2:3], # label sets: 10 and 5
  function(lbls) t(apply(clust_res_fmnist_dir, 2, function(res) performance(res, lbls)))
)

dt_res_fmnist_dir <- cbind(
  Data = factor(rep(c("FMNIST-10", "FMNIST-5"), each = length(l_fmnist$fmnist$eps_range))),
  Method = "DBSCAN",
  eps = rep(l_fmnist$fmnist$eps_range, 2),
  as.data.table(do.call(rbind, dt_res_fmnist_d))
)

dt_res_real_dir <- rbind(dt_res_real_d, dt_res_fmnist_dir)
dt_res_real_dir[, NMI := round(dt_res_real_dir$NMI, 3)
# save(dt_res_real_dir, file = "vignettes/data/res_real_dir.RData")
```

### 4.2 DBSCAN applied to UMAP embeddings

```
load("data/sec4_real_dats.RData")

dat_selection <- c("iris", "wine", "coil", "pendigits", "mnist")

# Note: Computations time >> 24 h for all settings (depending on system)
# Only using the small datasets Iris, Wine, COIL: comp. time ~ 5 min
dat_selection <- c("iris", "wine", "coil")

# FMNIST is excluded here and processed individually due to two label sets
# to reduce computation time
l_fmnist <- l_dats["fmnist"]
l_dats <- l_dats[dat_selection]

# DBSCAN on UMAP embeddings for different k
k <- c(5, 10, 15)
d <- 3
# random state for reproducibility
rand_state <- 256445188
```

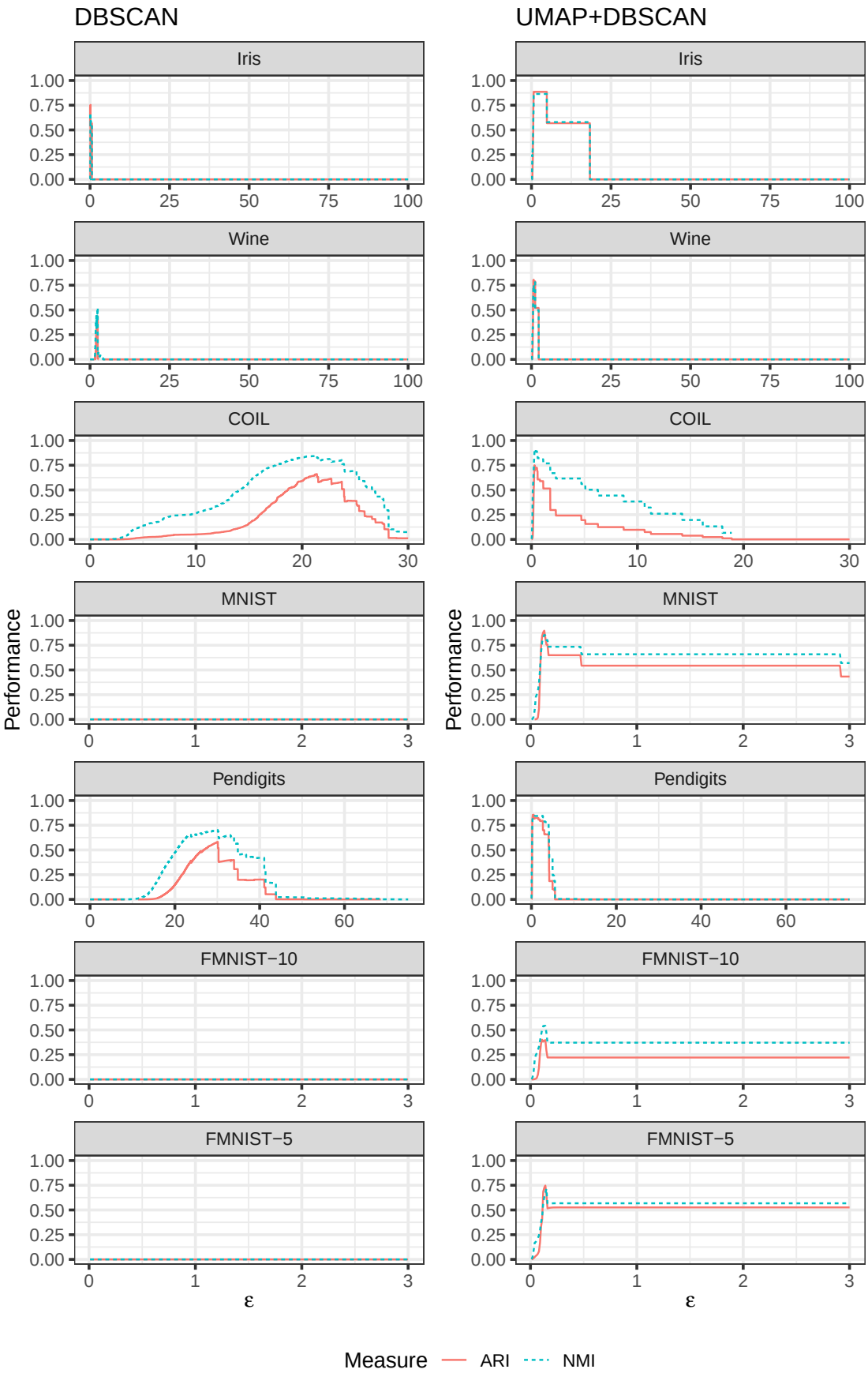
```r
emb_combs <-
  cbind.data.frame(
    dat = rep(names(l_dats), each = length(k)),
    k = rep(k, length(l_dats)),
    rs = rand_state
  )


# EMBEDDING
# comp time ~ 0.5 h
if (Sys.info()[["sysname"]] == "Windows") {
  cores_to_use <- 1
} else {
  cores_to_use <- 5
}

exp_real_embs <- mcmapply(
  function(dat, k, rs) umap::umap(
    as.matrix(l_dats[[dat]]$dat_X),
    n_neighbors = k,
    n_components = d,
    random_state = rs),
  dat = emb_combs$dat,
  k = emb_combs$k,
  rs = emb_combs$rs,
  SIMPLIFY = FALSE,
  mc.cores = cores_to_use
)
names(exp_real_embs) <- paste0(names(exp_real_embs), "_k", c(5, 10, 15))

exp_combs <-
  cbind.data.frame(
    setting = paste(emb_combs$dat, emb_combs$k, sep = "_"),
    dat = rep(names(l_dats), each = length(k))
  )

exp_layout <- lapply(exp_real_embs, function(emb) emb$layout)
names(exp_layout) <- exp_combs$setting

# CLUSTERING
# comp time several hours (depending on system >> 24 h possible)
if (Sys.info()[["sysname"]] == "Windows") {
  cores_to_use <- 1
} else {
  cores_to_use <- 3
}

exp_real_res <- parallel::mcmapply(
  function(set, dat) cluster_res(
    dat = exp_layout[[set]],
    eps_range = l_dats[[dat]]$eps_range,
    lbls = l_dats[[dat]]$dat_lbls
  ),
```

```
  set = exp_combs$setting,
  dat = exp_combs$dat,
  SIMPLIFY = FALSE,
  mc.cores = cores_to_use
)

eps_range_length <- sapply(l_dats, function(dat) length(dat$eps_range))

dt_res_real_emb <- cbind(
  Data = rep(dat_selection, length(k) * eps_range_length),
  Method = "UMAP+DBSCAN",
  eps = unname(unlist(
    sapply(l_dats[dat_selection],
           function(dat) rep(dat$eps_range, length(k)), simplify = FALSE))),
  as.data.table(do.call(rbind, exp_real_res)),
  k = rep(rep(k, length(dat_selection)), rep(eps_range_length, each = length(k)))
)

nic_names <- c("iris" = "Iris", "wine" = "Wine", "coil" = "COIL",
               "pendigits" = "Pendigits", "mnist" = "MNIST", "fmnist" = "FMNIST")

dt_res_real_emb$Data <- factor(
  dt_res_real_emb$Data,
  labels = nic_names[levels(factor(dt_res_real_emb$Data))]
)


# FMNIST

# embeddings
# comp time ~ 0.5 h
if (Sys.info()[["sysname"]] == "Windows") {
  cores_to_use <- 1
} else {
  cores_to_use <- 3
}

exp_fm_embs <- mclapply(
  c(5, 10, 15),
  function(k) umap::umap(
    as.matrix(l_fmnist$fmnist$dat_X),
    n_neighbors = k,
    n_components = 3,
    random_state = 256445188),
  mc.cores = cores_to_use
)

# clustering
# ~ comp time ~ 3 h
if (Sys.info()[["sysname"]] == "Windows") {
  cores_to_use <- 1
} else {
  cores_to_use <- 3
```

```
}

exp_fmnist_res <- parallel::mclapply(
  exp_fm_embs,
  function(dat) {
    clusts <- lapply(
      l_fmnist$fmnist$eps_range,
      function(eps) dbscan::dbscan(as.matrix(dat$layout), eps = eps)$cluster # minPts = 5 (default)
    )

    res_lbls5 <- vapply(
      clusts,
      function(clust) performance(clust, l_fmnist$fmnist$dat_lbls5),
      FUN.VALUE = numeric(2)
    )
    res_lbls10 <- vapply(
      clusts,
      function(clust) performance(clust, l_fmnist$fmnist$dat_lbls),
      FUN.VALUE = numeric(2)
    )

    rbind(res_lbls5, res_lbls10)
  },
  mc.cores = cores_to_use
)

names(exp_fmnist_res) <- k
dt_res_fm_emb <- lapply(
  exp_fmnist_res,
  function(res) {
    t_res <- t(res)
    dt_res <- as.data.table(t_res)
    dt_res
  }
)
dt_res_fm_emb <- rbindlist(dt_res_fm_emb, use.names = TRUE, idcol = TRUE)
dt_res_fm_emb <- rbind(dt_res_fm_emb[, c(1:3)], dt_res_fm_emb[, c(1, 4:5)])
dt_res_fm_emb[, ":="(Data = rep(c("FMNIST-5", "FMNIST-10"), each = 900),
                     Method = "UMAP+DBSCAN",
                     eps = rep(l_fmnist$fmnist$eps_range, 6),
                     k = .id)]
dt_res_fm_emb <- dt_res_fm_emb[, .id := NULL][, c(3:6, 1:2)]


# combine results
dt_res_real_emb <- rbind(dt_res_real_emb, dt_res_fm_emb)

# save(dt_res_real_emb, file = "vignettes/data/res_real_embs.RData")

dt_res_real_full <- rbind(dt_res_real_dir, dt_res_real_emb)

# save(dt_res_real_full, file = "vignettes/data/results/res_real_full.RData")
```

```
##         Data DBSCAN_NA_ARI DBSCAN_NA_NMI UMAP+DBSCAN_5_ARI UMAP+DBSCAN_5_NMI
```

```
##          Iris          0.75          0.67              0.70              0.75
##          Wine          0.44          0.52              0.81              0.77
## Pendigits          0.58          0.70              0.80              0.82
##          COIL          0.66          0.85              0.82              0.93
##         MNIST          0.00          0.00              0.69              0.70
## FMNIST-10          0.00          0.00              0.41              0.59
##  FMNIST-5          0.00          0.00              0.60              0.62
##    UMAP+DBSCAN_10_ARI UMAP+DBSCAN_10_NMI UMAP+DBSCAN_15_ARI UMAP+DBSCAN_15_NMI
## 1:               0.89               0.86               0.89               0.86
## 2:               0.81               0.78               0.80               0.79
## 3:               0.86               0.85               0.83               0.85
## 4:               0.75               0.91               0.70               0.88
## 5:               0.90               0.85               0.87               0.85
## 6:               0.40               0.54               0.38               0.54
## 7:               0.75               0.71               0.63               0.63
```

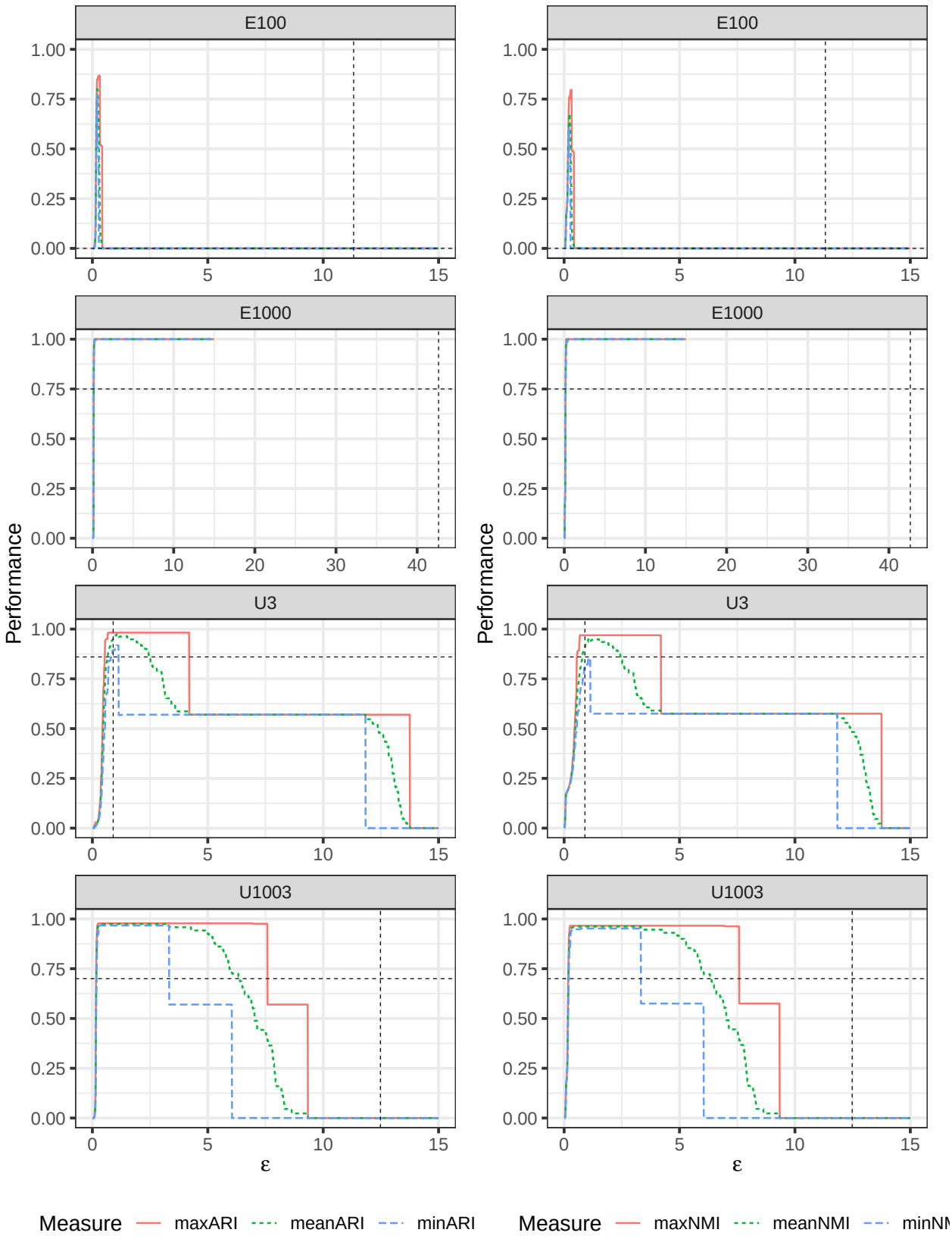

DBSCAN
UMAP+DBSCAN
Iris
Wine
COIL
MNIST
Pendigits
FMNIST−10
FMNIST−5
Performance
ε
Measure
ARI
NMI

# Appendix 1

## A1.1 Synthetic datasets $E_{100}$, $E_{1000}$, $U_3$ & $U_{1003}$

```
# embed each of the 4 settings 25 times to assess effect of SGD in constructing embedding
# layout on cluster results
reps <- 25
set.seed(110)
rand_states_rep <- sample(1:10000000, length(exp_settings) * reps)

rep_combs <- cbind.data.frame(
  rand_state = rand_states_rep,
  setting = rep(names(exp_sec2_dists), each = reps)
)

# ~ 1 min
if (Sys.info()[["sysname"]] == "Windows") {
  cores_to_use <- 1
} else {
  cores_to_use <- 4
}

exp_sec2_emb_rep_dists <-
  parallel::mcMap(
    function(set, rand) {
      emb <- umap::umap(
        as.matrix(exp_sec2_dists[[set]]),
        random_state = rand,
        n_neighbors = k,
        n_components = d,
        input = "dist")
      emb_dists <- dist(emb$layout)
      emb_dists
    },
    set = rep_combs$setting,
    rand = rep_combs$rand_state,
    mc.cores = cores_to_use
)

eps_range_rep <- seq(0.01, 15, by = 0.01)

# ~ 8 h with eps_range_rep = seq(0.01, 15, by = 0.01)
if (Sys.info()[["sysname"]] == "Windows") {
  cores_to_use <- 1
} else {
  cores_to_use <- 5
}

exp_sec2_rep_res <- parallel::mclapply(
  exp_sec2_emb_rep_dists,
  function(dat) cluster_res(dat, eps_range = eps_range_rep, lbls = lbls),
  mc.cores = cores_to_use
)
```

```r
dt_sec2_rep_res <- cbind(
  rep = rep(seq_len(reps), each = length(eps_range_rep) *  length(exp_settings)),
  setting = rep(rep_combs$setting, each = length(eps_range_rep)),
  eps = rep(eps_range_rep, length(exp_sec2_rep_res)),
  as.data.table(do.call(rbind, exp_sec2_rep_res))
)

# save(dt_sec2_rep_res, file = "vignettes/data/res_sec2_rep.RData")
```

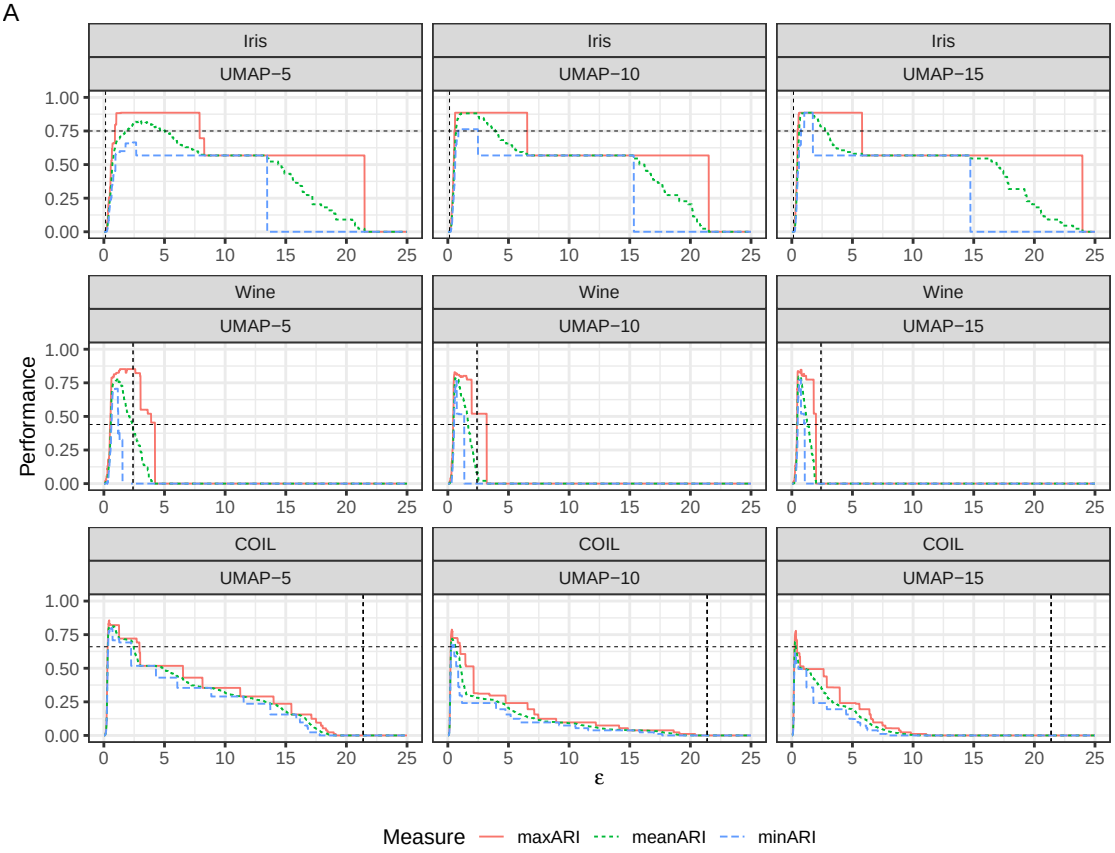

E100
E1000
U3
U1003
Performance
ε
Measure
maxARI
meanARI
minARI
maxNMI
meanNMI
minNMI

### A1.2 Real data

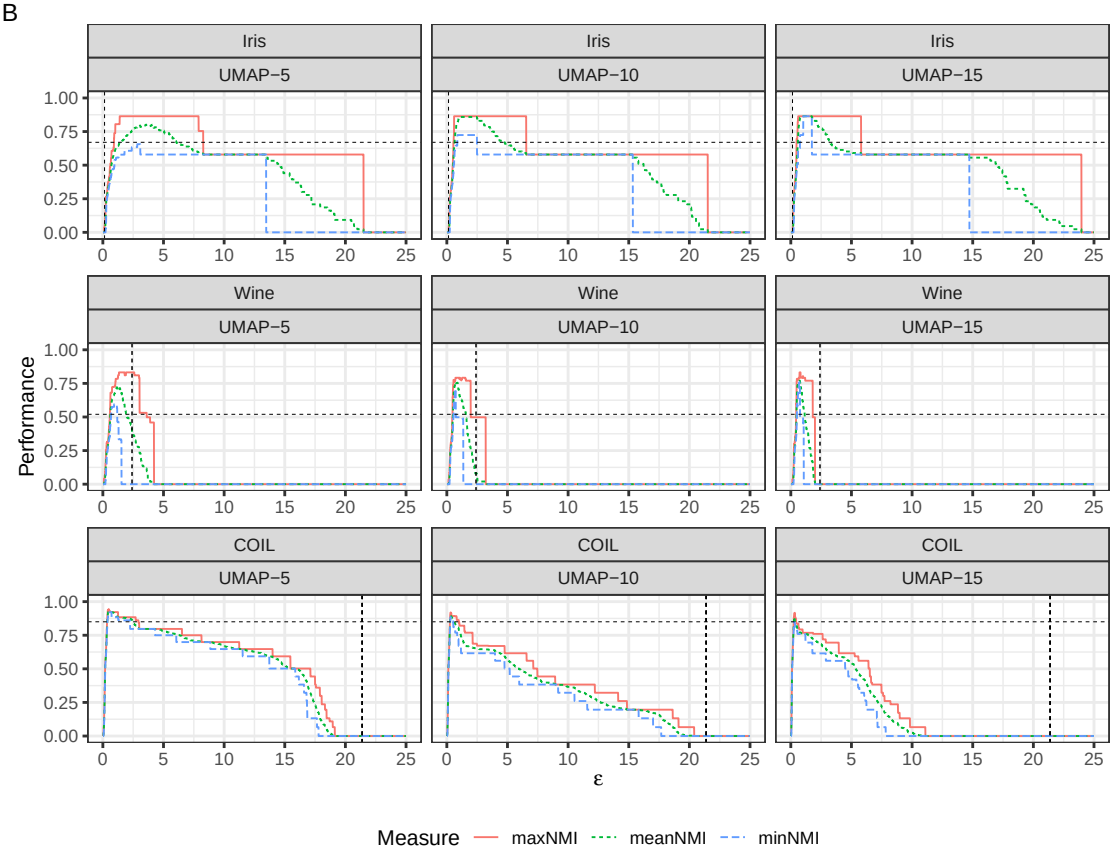
```
load("data/sec4_real_dats.RData")

## Setup
reps <- 25
dat <- c("iris", "wine", "coil")
l_dats <- l_dats[dat]

# computation time ~ 5 min
for (i in seq_along(l_dats)) {
  l_dats[[i]]$dat_X <- as.matrix(dist(l_dats[[i]]$dat_X))
}

## UMAP
k <- c(5, 10, 15)
rand_state <- 412745030 + 100000 * (1:(reps * length(dat) * length(k)))
d <- 3

## DBSCAN
eps_range <- seq(0.01, 25, by = 0.01)


# Embeddings
emb_combs <- expand.grid(seq_len(reps), k, dat) # rep(k, each = reps)
colnames(emb_combs) <- c("reps", "k", "dat")
emb_combs <- cbind(emb_combs,
                   rand_state = rand_state)

if (Sys.info()[["sysname"]] == "Windows") {
  cores_to_use <- 1
} else {
  cores_to_use <- 5
}

# computation time ~ 5 min
l_embs <- parallel::mcmapply(
  function(dat, k, rs) umap::umap(
    l_dats[[dat]]$dat_X,
    n_neighbors = k,
    n_components = d,
    random_state = rs,
    input = "dist"
  )$layout,
  dat = emb_combs$dat,
  k = emb_combs$k,
  rs = rand_state,
  SIMPLIFY = FALSE,
  mc.cores = cores_to_use
)

# Clusterings
clust_combs <- expand.grid(eps_range, seq_along(l_embs))
```

```r
colnames(clust_combs) <- c("eps_val", "embedding_iter")

# computation time ~ 1h
m_res <- parallel::mclapply(
  seq_len(nrow(clust_combs)),
  function(iter, ...) {
    eps_val <- clust_combs$eps_val[iter]

    emb_iter <- clust_combs$embedding_iter[iter]
    emb_layout <- l_embs[[emb_iter]]

    dat <- emb_combs$dat[[emb_iter]]
    dat_lbls <- l_dats[[dat]]$dat_lbls

    clustering <- dbscan(emb_layout, eps = eps_val)
    perfm <- performance(clustering$cluster, dat_lbls)
  },
  mc.cores = cores_to_use
)

dt_res_real_rep <- cbind(
  data = rep(emb_combs$dat, each = length(eps_range)),
  k = rep(emb_combs$k, each = length(eps_range)),
  rep = rep(emb_combs$reps, each = length(eps_range)),
  eps = clust_combs$eps_val,
  as.data.table(do.call(rbind, m_res))
)
# save(dt_res_real_rep, file = "vignettes/data/res_sec4_rep.RData")
```

A


Iris
UMAP−5
Iris
UMAP−10
Iris
UMAP−15
Wine
UMAP−5
Wine
UMAP−10
Wine
UMAP−15
COIL
UMAP−5
COIL
UMAP−10
COIL
UMAP−15
Performance
1.00
0.75
0.50
0.25
0.00
0
5
10
15
20
25
ε
Measure
maxARI
meanARI
minARI

B


Iris
UMAP−5
Iris
UMAP−10
Iris
UMAP−15
Wine
UMAP−5
Wine
UMAP−10
Wine
UMAP−15
COIL
UMAP−5
COIL
UMAP−10
COIL
UMAP−15
Performance
1.00
0.75
0.50
0.25
0.00
0
5
10
15
20
25
ε
Measure
maxNMI
meanNMI
minNMI

# Appendix 2

```
load("data/sec4_real_dats.RData")

## Setup
dat_selection <- c("iris", "wine", "coil", "pendigits")
l_dats <- l_dats[dat_selection]

# due the transformation d_ij = 1 - v_ij \in [0, 1], no eps values larger than 1 make sense.
# Since pendigits has many ob.s, reducing the eps range considerably reduces computation time.
# As the other datasets have lower number of obs, this is not as important.
l_dats$pendigits$eps_range <- seq(0.01, 1, by = 0.01)

min_dist <- 0.1 # note: not used in graph constr.
k <- c(5, 10, 15)

emb_combs <- expand.grid(k, dat_selection)
names(emb_combs) <- c("k", "data")

if (Sys.info()[["sysname"]] == "Windows") {
  cores_to_use <- 1
} else {
  cores_to_use <- 3
}

# fuzzy graphs for different values of k (only fgraph --> n_epochs = 0)
l_embs_fgr <- parallel::mcmapply(
  function(dat, k) uwot::umap(
    l_dats[[dat]]$dat_X,
    n_neighbors = k,
    ret_extra = "fgraph",
    n_epochs = 0,
    min_dist = min_dist),
  dat = emb_combs$data,
  k = emb_combs$k,
  SIMPLIFY = FALSE,
  mc.cores = cores_to_use
)

# turn similarities into dissimilarities via 1 - v_ij
l_emb_fgr_dis <- lapply(
  l_embs_fgr,
  function(emb) {
    temp <- as.matrix(1 - round(emb$fgraph, 5))
    diag(temp) <- 0
    as.dist(temp)
  }
)

exp_combs <-
  cbind.data.frame(
    setting = paste(emb_combs$dat, emb_combs$k, sep = "_"),
    dat = rep(names(l_dats), each = length(k))
```

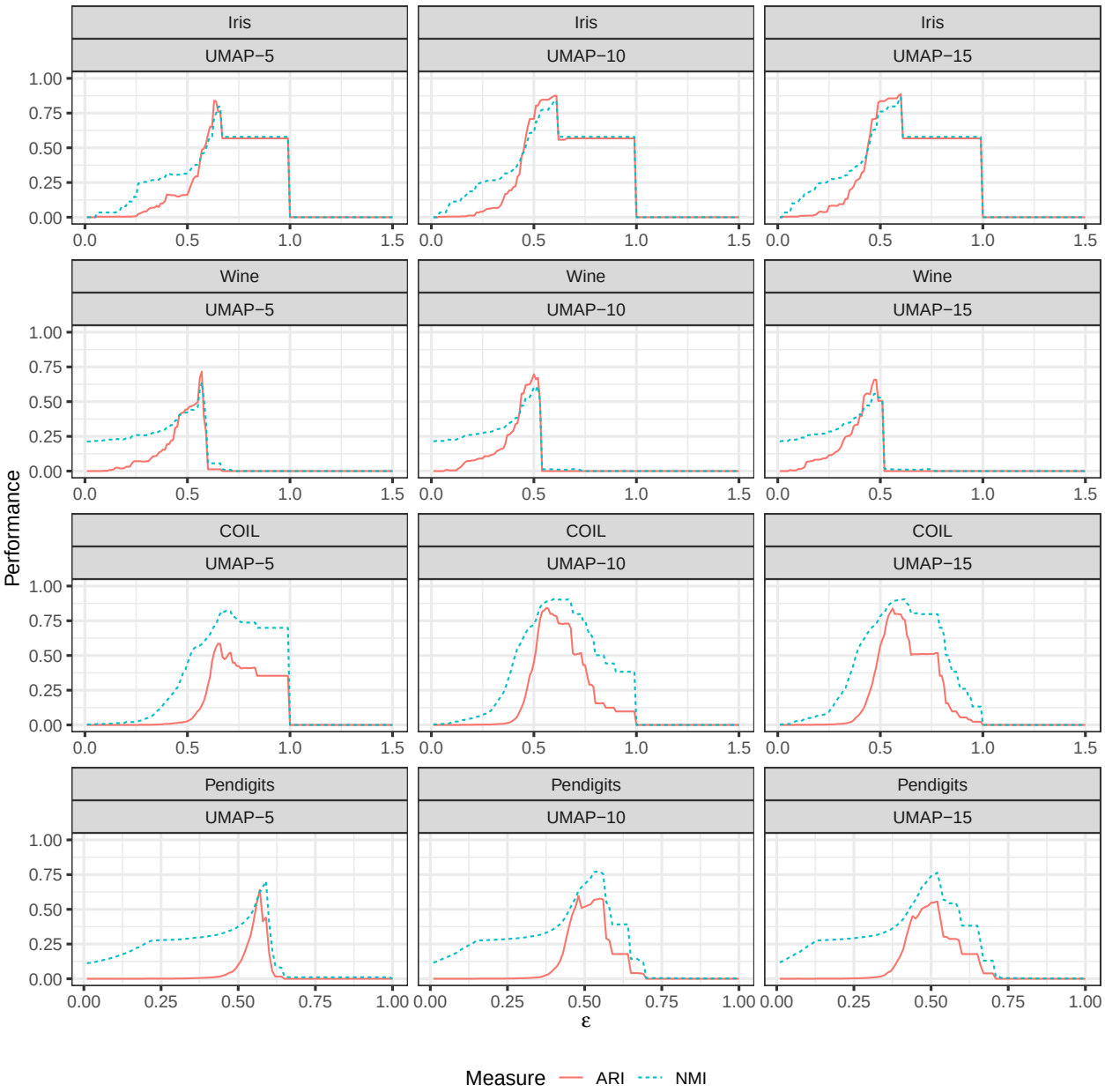
```
  )
names(l_emb_fgr_dis) <- exp_combs$setting


if (Sys.info()[["sysname"]] == "Windows") {
  cores_to_use <- 1
} else {
  cores_to_use <- 3
}

# computation time ~ 45 min
exp_real_fgr_res <- parallel::mcmapply(
  function(set, dat) cluster_res(
    dat = l_emb_fgr_dis[[set]],
    eps_range = l_dats[[dat]]$eps_range,
    lbls = l_dats[[dat]]$dat_lbls
  ),
  set = exp_combs$setting,
  dat = exp_combs$dat,
  SIMPLIFY = FALSE,
  mc.cores = cores_to_use
)

eps_range_length <- sapply(l_dats[dat_selection], function(dat) length(dat$eps_range))

dt_res_real_fgraph <- cbind(
  data = unlist(lapply(
    dat_selection,
    function(dat) rep(dat, length(k) * length(l_dats[[dat]]$eps_range)))
  ),
  k = rep(rep(k, length(dat_selection)), rep(eps_range_length, length(k))),
  eps = unname(unlist(
    lapply(l_dats[dat_selection],
           function(dat) rep(dat$eps_range, length(k))))),
  as.data.table(do.call(rbind, exp_real_fgr_res))
)

# save(dt_res_real_fgraph, file = "vignettes/data/res_app_fgraph.RData")
```

```
##    Measure        V1
## 1:     ARI 0.8856970
## 2:     NMI 0.8639757
##    Measure    V1
## 1:     ARI 0.875
## 2:     NMI 0.839
##
##
##  Iris
##
##  fgraph+embedding
##
##  Optimal values:
##  0.89 0.86
##  Optimal eps-range:
```

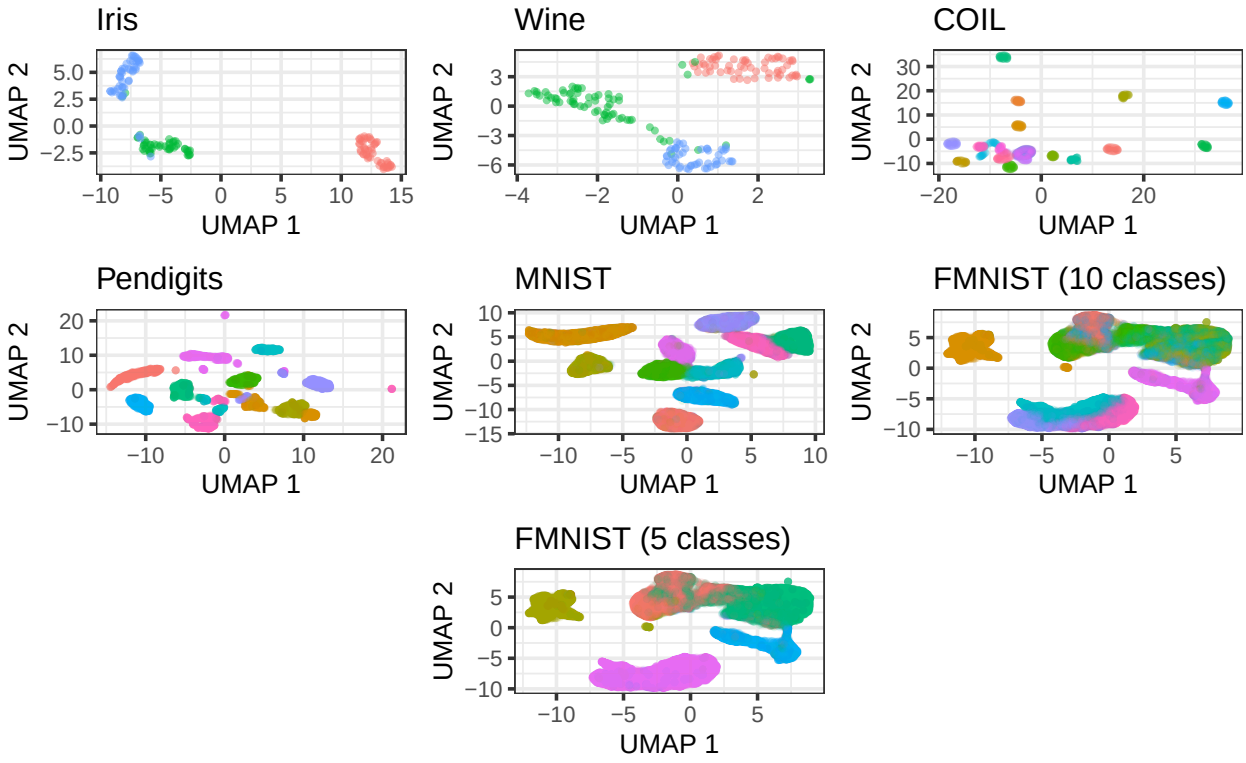
```
##  0.67 4.82
##  0.67 4.82
##
##  fgraph only
##
##  Optimal values:
##  0.88 0.84
##  Optimal eps-range:
##  0.6 0.61
##  0.6 0.61   Measure        V1
## 1:     ARI 0.8052042
## 2:     NMI 0.7807804
##    Measure    V1
## 1:     ARI 0.697
## 2:     NMI 0.612
##
##
##  Wine
##
##  fgraph+embedding
##
##  Optimal values:
##  0.81 0.78
##  Optimal eps-range:
##  0.64 0.69
##  1.11 1.16
##
##  fgraph only
##
##  Optimal values:
##  0.7 0.61
##  Optimal eps-range:
##  0.5 0.5
##  0.52 0.52
##
##
##  COIL
##
##  fgraph+embedding
##
##  Optimal eps-range (ARI/NMI > 0.6):
##  0.25 0.8
##  0.18 4.7
##
##  fgraph only
##
##  Optimal eps-range (ARI/NMI > 0.6):
##  0.52 0.68
##  0.45 0.79
##
##  Pendigits
##
##  fgraph+embedding
```

```
##
##  Optimal eps-range (ARI/NMI > 0.6):
##  0.17 4.04
##  0.16 4.13
##
##  fgraph only
##
##  Optimal eps-range (ARI/NMI > 0.6):
##  Inf -Inf
##  0.48 0.56
```

```
##         data UMAP-5_ARI UMAP-5_NMI UMAP-10_ARI UMAP-10_NMI UMAP-15_ARI
## 1:      Iris       0.84       0.80        0.88        0.84        0.89
## 2:      Wine       0.72       0.64        0.70        0.61        0.66
## 3:      COIL       0.59       0.82        0.84        0.91        0.84
## 4: Pendigits       0.64       0.70        0.60        0.77        0.56
##    UMAP-15_NMI
## 1:        0.86
## 2:        0.56
## 3:        0.90
## 4:        0.76
```


Iris
UMAP−5
Iris
UMAP−10
Iris
UMAP−15
Wine
UMAP−5
Wine
UMAP−10
Wine
UMAP−15
COIL
UMAP−5
COIL
UMAP−10
COIL
UMAP−15
Pendigits
UMAP−5
Pendigits
UMAP−10
Pendigits
UMAP−15
Performance
ε
Measure
ARI
NMI

# Appendix 3


Iris
Wine
COIL
Pendigits
MNIST
FMNIST (10 classes)
FMNIST (5 classes)
UMAP 1
UMAP 2