Comparsion of Dimensionality Reduction Techniques on RNA-seq Data

Ricards Marcinkevics (GitHub: i6092467)

11 January 2019

Introduction

With the advance of high-throughput sequencing technologies and their increasing availability, statistical bioinformatics researchers have to work with vast amounts of high-dimensional data. In many statistical and machine learning tasks, high dimensionality, *i.e.* large number of variables, or features, is challenging and undesirable. Thus, data dimensionality reduction techniques can be useful for various purposes; namely, for visualisation and exploratory analysis [1]; for feature selection and construction in predictive tasks, such as classification [2]; for data compression and summarisation etc.

Dimensionality reduction is a transformation of data into a lower dimensional space, which preserves (as much as possible) properties of the original data [3]. This problem is ill-posed and is solved by imposing some assumptions about the geometry of data [3]. Alternatively, dimensionality reduction can also be seen as a task of distance metric learning to compare data points in terms of the learnt similarity [4]. By now, there exists a rich taxonomy of non- and linear, un- and supervised data dimensionality reduction algorithms. For a comprehensive review of various techniques, we refer the reader to [3], [4], [5] and [6].

While discussions of statistical properties, assumptions and running time complexities of the algorithms are abundant in the literature, it would be interesting to investigate differences between algorithms in practice. In this report, we examine several dimensionality reduction approaches in a 'controlled' manner on simulated RNA-seq data. There are several goals to the experiments that we performed:

1. *To compare the ability of different algorithms to preserve biological signals (under various tuning parameter values) in RNA-seq datasets with two treatment groups and differential expression.* The ability to visualise differential expression and clustering is crucial in the exploratory analysis of RNA-seq data.
2. *To compare algorithms when applied to datasets with no differential expression.* Some techniques have several tuning parameters which affect the output of dimensionality reduction. The variability in the output might lead to visualising spurious differences and clusters.
3. *To compare the scalability of algorithms.* While some techniques are powerful, their application and benefit can be limited due to long running times.

The remainder of the report briefly introduces dimensionality reduction techniques that are compared; explains criteria that are used in the comparison of methods; describes the data generation procedure; and provides the results of experiments and their discussion. The R code that was developed for the project is given in line.

Dimensionality Reduction Techniques

In this section we briefly introduce commonly applied dimensionality reduction methods that are compared in the next sections. We describe assumptions and underlying models of these techniques. Generally, the methods can be distinguished into two main classes. *Linear* techniques assume that data are located in a linear subspace, whereas *non-linear* approaches do not make this strong assumption and, therefore, are able to learn more complex representations [3].

In the problem of dimensionality reduction we consider a set of n observations of d variables, given by $X = \begin{bmatrix} x_1 & x_2 & \cdots & x_n \end{bmatrix}$; note, that $X \in \mathbb{R}^{d \times n}$. In the setup of RNA-seq experiments, datasets consist of n

samples with d genes measured. The dimensionality reduction of X is given by transformed dataset $Y \in \mathbb{R}^{k \times n}$, where usually $k \ll d$ [3], [6]. Normally, Y is obtained by optimising some criterion which quantifies how well the properties of the initial data are preserved by the transformation.

Principal Component Analysis

Principal Component Analysis (PCA) is one of the most well-known linear dimensionality reduction techniques, used in various data analysis applications. Two different formulations of it were proposed by Pearson in 1901 and by Hotelling in 1933 [6]. PCA constructs new dimensions as linear combinations of the original variables to maximise the variance captured by these dimensions. Formally, PCA finds linear mapping M such that maximises $M^T Cov(X) M$, where $Cov(X)$ is the empirical covariance matrix computed from the data [3].

Multidimensional Scaling

Multidimensional scaling (MDS) is a technique which tries to find low-dimensional representation of the data $Y = \begin{bmatrix} y_1 & y_2 & \cdots & y_n \end{bmatrix}$ that preserves pairwise distances between points as much as possible [3]. There exist various criteria, referred to as "stress functions", which can be optimised to achieve this. For example, we could minimise the discrepancy between pairwise Euclidean distances in X and Y:

$$\sum_{1 \leq i,j \leq n} \left(||x_i - x_j||_2^2 - ||y_i - y_j||_2^2 \right)^2 .$$

Under known data point coordinates and the stress function given above, MDS is equivalent to PCA. However, note, that MDS is more general, since it allows reconstructing point coordinates when *only* the pairwise distances are given [6].

Isomap

As opposed to the methods explained before, Isomap accounts for distributions of points in neighbourhoods [3]. Isomap constructs a graph wherein vertices correspond to data points and are connected by edges between close neighbours. Consequently, geodesic distances between observations are computed as shortest path distances within the graph. Finally, a low-dimensional representation of the original data is found by running MDS on the matrix of pairwise geodesic distances. In general, Isomap is useful for datasets where points concentrate on a curved manifold. This method has several limitations, in particular, it does not perform adequately if the underlying manifold is non-convex or if there exist 'holes' in the manifold [3]. Moreover, the algorithm has several tuning parameters which can affect the quality of the dimensionality reduction.

t-distributed Stochastic Neighbour Embedding

t-distributed stochastic neighbour embedding (t-SNE) [7] is a method for data dimensionality reduction that tries to reflect both global and local structures of the data. It is especially suitable for datasets wherein points lie on *several* manifolds. t-SNE measures pairwise similarities between data points in high- and low-dimensional spaces. Similarity between points x_i and x_j is given by $p_{ij} = \frac{p_{j|i} + p_{i|j}}{2n}$, where

$$p_{j|i} = \frac{\exp\left(-||x_i - x_j||^2 / 2\sigma_i^2\right)}{\sum_{k \neq i} \exp\left(-||x_i - x_k||^2 / 2\sigma_i^2\right)}$$

with variance σ_i^2, the choice of which depends on the value of the parameter called "perplexity". On the other hand, similarity between y_i and y_j, the low-dimensional representations of x_i and x_j, is $q_{ij} = \frac{(1+||y_i - y_j||^2)^{-1}}{\sum_{k \neq l}(1+||y_k - y_l||^2)^{-1}}$. Note, that the latter similarity is derived from the Student t-distribution with one degree of freedom. Using gradient descent, t-SNE finds low-dimensional coordinates Y that minimise Kullback-Leibler divergence between similarities p_{ij} and q_{ij}:

$$\sum_i \sum_j p_{ij} \log \frac{p_{ij}}{q_{ij}}.$$

Diffusion Maps

Diffusion maps use Markov chains constructed based on the data [3]. This method considers a complete weighted graph $G = (V, E, W)$, where V are vertices corresponding to data points, $E = V \times V$ and $W : E \to \mathbb{R}_{>0}$ are edge weights. In particular, Gaussian kernel is used to compute the weights. Edge between points i and j has weight $W_{ij} = e^{-\frac{||x_i - x_j||^2}{2\sigma^2}}$. Graph G forms a Markov chain with states defined by vertices and possible transitions given by directed edges. Transition probability matrix P of this chain is given by $P_{ij} = \frac{W_{ij}}{\sum_k W_{ik}}$. Using P, probabilities for transitions between states after several steps can be computed. Let $P^{(t)}$ be the matrix with t-step transition probabilities. Diffusion maps build new variables for transformed data Y by performing the eigendecomposition of $P^{(t)}$. Note, that t is a tuning parameter, which has to be specified by the user. Point coordinates in new variables are given by products of eigenvectors and corresponding eigenvalues of $P^{(t)}$, discarding the largest eigenvalue: $\lambda_2 v_2, \lambda_3 v_3, ...\lambda_{k+1} v_{k+1}$. One can show that such representation Y tries to retain diffusion distances between data points. Intuitively, the diffusion distance is based on t-step transition probabilities between two points and, thus, considers many probable paths in graph G between the points. Such distance might be a more robust measure than, for instance, geodesic distances employed in the Isomap algorithm.

Experiments

The goal of the report is to compare the dimensionality reduction techniques, described in the previous section, on datasets that are as close as possible to 'real world' RNA sequencing (RNA-seq) data. In this section we describe the experiments that were performed, provide details about data simulation procedures and explain evaluation criteria that were used to compare different techniques.

Dataset Simulation

We decided to use simulated data in the experiments to be able to assess the variability in the output and performance of the considered techniques, because simulated datasets allow assessing methods many times. Since classical parametric simulation methods for RNA-seq data make assumptions that are not always satisfied in reality, we used SimSeq, a non-parametric approach, described in [8]. Herein, we briefly introduce this procedure and provide our implementation of it.

SimSeq is a data-driven method. It is motivated by overly optimistic results often reported when testing newly proposed statistical methods for the analysis of RNA-seq data. The authors of SimSeq argue that, frequently, methods are tested on the data which were generated with the model that is assumed [8]. Therefore, the obtained results do not generalise well to cases when the model does not hold. Moreover, an important advantage of SimSeq is that it retains a complex dependence structure between genes, resulting in more realistic simulated data.

The overall idea of SimSeq is to randomly simulate RNA-seq data from a given source dataset, preserving some of its statistical properties. Let Z denote the count matrix of the source dataset. Then Z_{git} corresponds to the count of gene $g \in \mathcal{G} = \{1, 2, ..., G\}$ in measurement unit $i \in \{1, 2, ..., N_t\}$ that has treatment $t \in \{1, 2\}$. We assume that all genes within every measurement unit can be normalised by a multiplicative unit-specific constant to match the expected count for the considered gene in the given treatment, i.e. $E[Z_{git}] = \mu_{gt} c_{it}$, where μ_{gt} is the expected count for gene g in treatment t and c_{it} is the unit-specific multiplicative effect. To estimate c_{it}, for example, trimmed mean of M values (TMM) method could be used. Counts Z_{git} and normalisation constants c_{it} are inputs to the SimSeq algorithm. We describe this algorithm further and demonstrate it applied to the RNA-Seq data obtained from lymphoblastoid cells of 69 unrelated individuals [9], available in `tweeDEseqCountData` package [10]:

```
library(tweeDEseqCountData)
data("pickrell1")
# Source count matrix
Z <- exprs(pickrell1.eset)
```

```
# Treatment variable
trts <- pickrell1.eset$gender
```

The algorithm consists of several steps [8]:

0. Compute normalisation constants c_{it}, using TMM.
1. Perform the two-sample Wilcoxon test for each gene and retain resulting p-values.
2. Compute local false discovery rate (FDR) with `fdrtool` package.
3. Compute sampling probabilities for each gene. Sampling probabilities are proportional to weights w given by subtracting local FDR from 1.
4. Randomly sample G_1 differentially expressed (DE) genes without replacement from $\mathcal{G}$ with probabilities computed in step 3. Let $\mathcal{G}_1$ denote the set of genes chosen to be DE.
5. Randomly sample G_0 genes to be equally expressed (EE) without replacement from $\mathcal{G} \backslash \mathcal{G}_1$ uniformly at random. We denote the set of sampled EE genes by $\mathcal{G}_0$. Moreover, let $\mathcal{G}^* = \mathcal{G}_0 \cup \mathcal{G}_1$.
6. Randomly sample column z without replacement from treatment group 1 from Z. Subset z down to genes in $\mathcal{G}^*$ and let it be new column x_1 of simulated count matrix X. Assign treatment 1 to column x_1.
7. Sample randomly without replacement columns z_1 and z_2 of Z from treatment groups 1 and 2, respectively.
8. Subset z_1 and z_2 down to genes in $\mathcal{G}^*$.
9. Let c_1 and c_2 be normalisation factors for columns z_1 and z_2. For each gene $g \in \mathcal{G}^*$, simulated column x_2 is formed as
$$x_{2g} = \begin{cases} z_{1g} & \text{if } g \in \mathcal{G}_0 \\ \lfloor z_{2g} \cdot c_1/c_2 + 0.5 \rfloor & \text{if } g \in \mathcal{G}_1 \end{cases}$$
Assign treatment 2 to column x_2.
10. Repeat steps 6-9 n times. Sample columns without replacement across *all* iterations.

As can be seen, G_0, G_1 and n are additional input arguments that need to be specified by the user. It is important to note, that this simulation allows producing datasets with only two treatment groups and with an unpaired design. These restrictions are made in the experiments of this report. The extensions to the algorithm are covered in [8]. Another noticeable limitation to the SimSeq approach is that the size of the generated dataset is limited by the sizes of treatment groups within the provided source data. Moreover, to achieve adequate variety between simulations, the size of the source needs to be sufficiently large.

We implemented SimSeq algorithm as `simSeqSim` function, which returns a simulated dataset based on the input provided by the user. Note, that we use `edgeR` package [11] for the TMM normalisation:

```
# SimSeq algorithm
simSeqSim <- function(Z, treatments, G0, G1, n, geneprobs = NULL) {
  # Z - source raw count matrix.
  # treatments - vector with treatment assignment for every sample.
  # G0 - number of equally expressed genes to be constructed.
  # G1 - number of differentially expressed genes to be constructed.
  # n - size of one treatment group in the simulated dataset.
  # geneprobs - pre-computed gene sampling probabilities. Input them
  #             to sample multiple datasets from one source faster.
  # Returns a list with simulated count matrix X; treatment assignments trt;
  # and gene sampling probabilities geneprobs.

  G <- 1 : nrow(Z)
```

```r
## 0.
normFactors <- edgeR::calcNormFactors(Z, method = "TMM")
## 1.
# Treatments 1 & 2
trt1 <- as.numeric(treatments) == 1
trt2 <- as.numeric(treatments) == 2
# Were pre-computed sampling probabilities provided?
if (is.null(geneprobs)) {
  # p-values for the Wilcoxon test.
  # Note, that we normalise and transform raw counts.
  pvals <- apply(X = Z, MARGIN = 1, FUN = function(x) {
    wilcox.test(log((x[trt1] + 1) / normFactors[trt1]),
                log((x[trt2] + 1) / normFactors[trt2]))$p.value})
  ## 2.
  locfdr <- fdrtool::fdrtool(x = pvals, statistic = "pvalue",
                             plot = FALSE, verbose = FALSE)$lfdr
  ## 3.
  w <- 1 - locfdr
  # Make sure that probabilities sum up to unity
  geneprobs <- w / sum(w)
}
## 4.
degenes <- sample(G, G1, replace = FALSE, prob = geneprobs)
## 5.
# Genes that were already sampled to be DE
sampled <- G %in% degenes
eegenes <- sample(G[!sampled], G0, replace = FALSE)
cols1 <- which(trt1)            # Columns with treatment 1
cols2 <- which(trt2)            # Columns with treatment 2
sampledCols <- c()              # Columns that have been sampled
simTreatment <- rep(0, 2 * n)   # Treatment assignments for simulated data
# Simulated dataset
X <- matrix(rep(0, (G0 + G1) * n * 2), ncol = 2 * n)
# Repeat n times
for (i in 1 : n) {
  ## 6.
  z <- sample(cols1[!(cols1 %in% sampledCols)], 1)
  sampledCols <- c(sampledCols, z)
  # Subset to sampled genes
  x1 <- Z[c(degenes, eegenes),z]
  simTreatment[2 * i - 1] <- 1
  ## 7.
  z1 <- sample(cols1[!(cols1 %in% sampledCols)], 1)
  z2 <- sample(cols2[!(cols2 %in% sampledCols)], 1)
  sampledCols <- c(sampledCols, z1, z2)
  ## 9.
  c1 <- normFactors[z1]
  c2 <- normFactors[z2]
  x2 <- floor(Z[degenes, z2] * c1 / c2 + 0.5)
  x2 <- c(x2, Z[eegenes, z1])
  # Assign treatment
  simTreatment[2 * i] <- 2
```

```
    # Store observations in the simulated count matrix
    X[, 2 * i - 1] <- x1
    X[, 2 * i] <- x2
  }

  return(list(X = X, trt = as.factor(simTreatment),
              geneprobs = unname(geneprobs), G0 = G0, G1 = G1))
}
```

To illustrate that the algorithm works as intended, we simulate a dataset from the Pickrell data. We choose 5000 genes to be equally expressed and 5000 genes to be expressed differentially; and we choose to have $n = 20$ observations in each treatment group:

```
set.seed(2812)
simout <- simSeqSim(Z, trts, 5000, 5000, 20)
str(simout)
```

```
## List of 5
##  $ X        : num [1:10000, 1:40] 2251 88 0 0 0 ...
##  $ trt      : Factor w/ 2 levels "1","2": 1 2 1 2 1 2 1 2 1 2 ...
##  $ geneprobs: num [1:38415] 1.69e-05 9.81e-06 7.23e-05 7.23e-05 1.53e-05 ...
##  $ G0       : num 5000
##  $ G1       : num 5000
```

We expect first 5000 genes to be differentially expressed. Figure 1 depicts unadjusted p-values for Wilcoxon tests performed for all genes:

```
normFactors <- edgeR::calcNormFactors(simout$X, method = "TMM")
trt1 <- as.numeric(simout$trt) == 1
trt2 <- as.numeric(simout$trt) == 2
pvals <- apply(X = simout$X, MARGIN = 1, FUN = function(x) {
  wilcox.test(log((x[trt1] + 1) / normFactors[trt1]),
         log((x[trt2] + 1) / normFactors[trt2]))$p.value})
plot(pvals, xlab = "Gene", ylab = "Wilcoxon p-value", pch = ".",
     cex.lab = 0.6, cex.axis = 0.6)
```

Clearly, the distribution of p-values for the first 5000 genes, which are truly differentially expressed, is right-skewed; whereas, for the equally expressed genes, the distribution is more uniform. This agrees with the input that we provided to the algorithm.

Evaluation and Comparison

To compare dimensionality reduction techniques, we apply them to simulated RNA-seq datasets obtained from the SimSeq algorithm described in the previous section. The source dataset we use in all simulations is from the experiment by Pickrell et al. [9] on lymphoblastoid cell lines from 69 unrelated Nigerian individuals. Two 'treatments' present in the data are male and female genders. We chose this dataset as a source, because it has quite large sample sizes within treatment groups (40 females and 29 males) and, thus, allows comparing techniques using larger numbers of observations.

Techniques are evaluated under four settings, which correspond to different percentages of differentially expressed genes:

1. Global shift, i.e. all genes are DE: $G_0 = 0$, $G_1 = 15000$.
2. Half of genes are DE: $G_0 = 7500$, $G_1 = 7500$.
3. Few genes are DE: $G_0 = 14000$, $G_1 = 1000$.
4. Global null, i.e. no differential expression: $G_0 = 15000$, $G_1 = 0$.

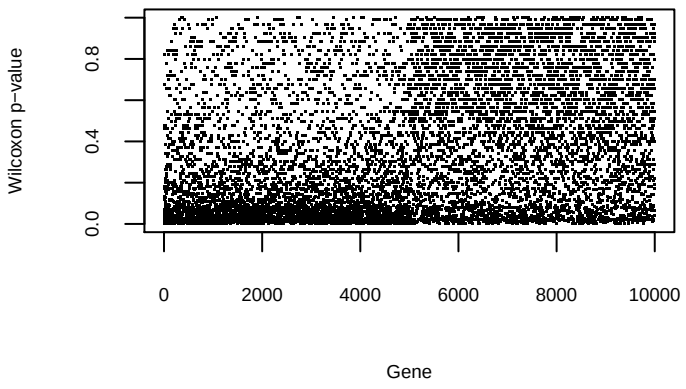

Figure 1: p-values of Wilcoxon tests for differential expression in simulated data.

In all cases, $\mathcal{G} = \{1, 2, ..., 15000\}$, and the size of every treatment group is $n = 15$. For each of these settings, $N = 100$ RNA-seq datasets are simulated from the Pickrell data. Every dimensionality reduction algorithm is run on N datasets, and *2-dimensional* outputs are evaluated using criteria that are introduced further. Algorithms that have tuning parameters are evaluated multiple times under different parameter values.

To facilitate comparison of dimensionality reduction techniques in terms of their ability to preserve biological signals present in data, we use quantitative evaluation criteria. The measures we employ are usually applied in assessment of clustering techniques. Intuitively, we want data points to cluster by their treatment in low-dimensional representation Y obtained from a dimensionality reduction algorithm that preserves and reflects treatment effects. We found the following criteria to be suitable.

Silhouette Width

Silhouette width is a measure which was proposed for assessing the quality of a clustering [12]. Herein, we explain how average silhouette width (ASW) can be computed to evaluate dimensionality reduction $Y = \begin{bmatrix} y_1 & y_2 & \cdots & y_n \end{bmatrix}$ of dataset X, given treatment assignments $t_1, t_2, ..., t_n \in \{1, 2\}$. Let $a(i)$ denote the average dissimilarity of point i in Y to all points within the same treatment group. Dissimilarity can be measured in different ways, we use the Euclidean distance to quantify it. Then, formally,

$$a(i) = \frac{1}{|\{j : t_j = t_i\}|} \sum_{j:t_j=t_i} ||y_i - y_j||.$$

Let $d(i, C)$ denote the average dissimilarity of data point i to all points in treatment group C, i.e.

$$d(i, C) = \frac{1}{|\{j : t_j = C\}|} \sum_{j:t_j=C} ||y_i - y_j||.$$

Based on $d(i, C)$, $b(i) = \min_{C \neq t_i} d(i, C)$ can be computed. Then the silhouette width of point i is defined as

$$s(i) = \frac{b(i) - a(i)}{\max\{a(i), b(i)\}}.$$

It holds that $-1 \leq s(i) \leq 1$. Finally, the ASW is just the arithmetic mean of $s(i)$ across all data points. An ASW close to 1 would suggest that points in low-dimensional representation Y cluster very well by treatment,

whereas a small or even negative ASW would mean that points do not cluster by treatment at all. To compute this quantity in R, we use `cluster` package [13]. For example, the ASW for the simulated dataset we generated with the SimSeq in the previous section can be computed in the following way:

```
library(cluster)
X <- simout$X
# Transform data
for (i in 1 : dim(X)[2]) {
  X[,i] <- log((X[,i] + 1) / normFactors[i])
}
# Compute pairwise Euclidean distances between points
D <- dist(t(X))
# Silhouette widths
s <- silhouette(x = as.numeric(simout$trt), dist = D)
# ASW
mean(s[,3])
```

```
## [1] 0.0208727
```

Davies-Bouldin Index

Davies-Bouldin (DB) index is a measure, originally proposed by Davies and Bouldin in 1979 [14], for assessing the appropriateness of a clustering. This index favours high inter-group and small intra-group dissimilarities. In the context considered herein Davies-Bouldin index is given by

$$DB = \frac{\sigma_1 + \sigma_2}{d(c_1, c_2)},$$

where $d(c_1, c_2)$ is the Euclidean distance between centroids of treatment groups 1 and 2; $c_1 = \frac{1}{N_1}\sum_{i:t_i=1} y_i$ and $c_2 = \frac{1}{N_2}\sum_{i:t_i=2} y_i$; σ_1 and σ_2 are average distances to centroids from points within treatment groups 1 and 2, respectively, i.e. $\sigma_1 = \frac{1}{N_1}\sum_{i:t_i=1} d(y_i, c_1)$ and $\sigma_2 = \frac{1}{N_2}\sum_{i:t_i=2} d(y_i, c_2)$. As opposed to the ASW, smaller values of the DB index correspond to better clustering of treatment groups in Y. To compute the DB index in R, we use the implementation available in package `clusterSim` [15]. Consider the following example of evaluating the DB index on the simulated data:

```
library(clusterSim)
db <- index.DB(t(X), as.numeric(simout$trt))
db$DB
```

```
## [1] 4.687478
```

Code

In this subsection we provide and explain the R code that was used for computations. All experiments were performed in R version 3.5.1 on the machine running 64-bit Windows with Intel Core i7-8750H CPU @2.20GHz 2.21GHz with 16.0 GB of RAM.

Firstly, we generate all $N = 100$ datasets that will be used to evaluate the considered techniques for the four different settings of gene expression. The datasets are simulated with the SimSeq algorithm from the Pickrell data under the inputs described before. In addition, simulated count matrices are log-transformed and normalised with the TMM method, genes with 0 variance are removed. This is a common pre-processing step for the dimensionality reduction algorithms that are applied further.

```
N <- 100                          # Number of datasets
G_0 <- c(0, 7500, 14000, 15000)   # Different settings for the number of EE genes
G_1 <- c(15000, 7500, 1000, 0)    # Different settings for the number of DE genes
n <- 15                           # Number of samples within each group
```

```r
set.seed(301201909)
# Lists to store simulation outputs under different settings
simouts15000 <- vector("list", N)
simouts7500 <- vector("list", N)
simouts1000 <- vector("list", N)
simouts0 <- vector("list", N)
# Pre-computed p-values
geneprobs <- NULL
for (k in 1 : 4) {      # For each setting...
  print(paste("Running setting #", k, sep = ""))
  # List to store simulation outputs
  simouts <- vector("list", N)
  for (i in 1 : N) {    # Simulate N times
    simouts[[i]] <- simSeqSim(Z, trts, G_0[k], G_1[k], n, geneprobs)
    geneprobs <- simouts[[i]]$geneprobs
    # Transform the count matrix
    X <- simouts[[i]]$X
    normFactors <- edgeR::calcNormFactors(X, method = "TMM")
    for (j in 1 : (2 * n)) {
      X[, j] <- log((X[, j] + 1) / normFactors[j])
    }
    vars <- apply(X, 1, var);
    X <- X[vars > 0,]
    simouts[[i]]$X <- X
  }

  # Asign outputs to the correct setting
  if (k == 1) {
    simouts15000 <- simouts
  } else if (k == 2) {
    simouts7500 <- simouts
  } else if (k == 3) {
    simouts1000 <- simouts
  } else {
    simouts0 <- simouts
  }
}
```

```
## [1] "Running setting #1"
## [1] "Running setting #2"
## [1] "Running setting #3"
## [1] "Running setting #4"
```

```r
# Remove the unused list
rm(simouts)
```

Once datasets are generated, we perform dimensionality reduction using various techniques, we reduce original datasets to two dimensions. Obtained low-dimensional representations are evaluated using the ASW and the DB index in-place. Additionally, the elapsed running time of algorithms is measured, to compare their scalability. The following utility functions are used for evaluating dimensionality reduction:

```r
evalDimRed <- function(Y, trt) {
  # This function performs the evaluation of a dimensionality reduction.
  # Y - low-dimensional representation.
  # trt - treatment labels.
```

```
  # Returns the average silhouette width asw; and the Davies-Bouldin index db.

  # Compute pairwise Euclidean distances between points
  D <- dist(Y)
  # Silhouette widths
  s <- cluster::silhouette(x = as.numeric(trt), dist = D)
  # ASW
  ASW <- mean(s[,3])
  # Davies-Bouldin index
  db <- clusterSim::index.DB(Y, as.numeric(trt))
  DB <- db$DB

  return(list(asw = ASW, db = DB))
}

evaluateNtimes <- function(simouts, funct, ...) {
  # This function performs the evaluation of dimensionality reduction technique
  # several times using given datasets.
  # simouts - list of simulated datasets.
  # funct - wrapper function that performs dimensionality reduction. It has to
  #         return the low-dimensional representation of the data.
  # Returns average silhouette widths; Davies-Bouldin indices; and running times.
  N <- length(simouts)
  asws <- numeric(N)
  dbs <- numeric(N)
  rtimes <- numeric(N)
  for (i in 1 : N) {
    rtime <- system.time(Y <- funct(simouts[[i]]$X, ...))
    evaluation <- evalDimRed(Y, simouts[[i]]$trt)
    asws[i] <- evaluation$asw
    dbs[i] <- evaluation$db
    rtimes[i] <- rtime[3]
  }

  return(list(asws = asws, dbs = dbs, rtimes = rtimes))
}
```

The first technique that we test is the principal component analysis. PCA is evaluated twice, because it is possible to run it on scaled data and on the original un-scaled points. To run PCA in R, we use function `prcomp` from `stats` package, which performs a singular value decomposition of the data matrix (rather than an eigendecomposition of the covariance matrix as in `princomp`).

```
# Wrapper function
pcfunct <- function(X, scale. = TRUE) {
  pc <- prcomp(t(X), scale. = scale.)
  return(pc$x[, 1 : 2])
}

# Scaled
evs_pc_sc_0 <- evaluateNtimes(simouts0, pcfunct, scale. = TRUE)
evs_pc_sc_1000 <- evaluateNtimes(simouts1000, pcfunct, scale. = TRUE)
evs_pc_sc_7500 <- evaluateNtimes(simouts7500, pcfunct, scale. = TRUE)
evs_pc_sc_15000 <- evaluateNtimes(simouts15000, pcfunct, scale. = TRUE)
# Unscaled
```

```
evs_pc_unsc_0 <- evaluateNtimes(simouts0, pcfunct, scale. = FALSE)
evs_pc_unsc_1000 <- evaluateNtimes(simouts1000, pcfunct, scale. = FALSE)
evs_pc_unsc_7500 <- evaluateNtimes(simouts7500, pcfunct, scale. = FALSE)
evs_pc_unsc_15000 <- evaluateNtimes(simouts15000, pcfunct, scale. = FALSE)
```

Another technique that we examine is the classical multidimensional scaling. We use correlation distance measure which we define as $d_{corr}(x_i, x_j) = 1 - Corr(x_i, x_j)$. To perform MDS, we apply `cmdscale` function from R package `stats`.

```
# Wrapper function
mdsfunct <- function(X) {
  # Correlation distance matrix
  d <- 1 - cor(X)
  mds <- cmdscale(d)
  return(mds)
}

evs_mds_0 <- evaluateNtimes(simouts0, mdsfunct)
evs_mds_1000 <- evaluateNtimes(simouts1000, mdsfunct)
evs_mds_7500 <- evaluateNtimes(simouts7500, mdsfunct)
evs_mds_15000 <- evaluateNtimes(simouts15000, mdsfunct)
```

Isomap embeddings of simulated datasets are computed. Bioconductor package `RDRToolbox` [16] contains an implementation of this algorithm. We run Isomap for different numbers of nearest neighbours $\mathtt{k} = 2, 8, 15, 21, 28$ used in the construction of the graph for computing geodesic distances.

```
# Wrapper function
isomapfunct <- function(X, k = 3) {
  res <- RDRToolbox::Isomap(t(X), k = k, verbose = FALSE)
  return(res$dim2)
}

# Different k
ks <- c(2, 8, 15, 21, 28)
evs_iso_0 <- vector("list", length(ks))
evs_iso_1000 <- vector("list", length(ks))
evs_iso_7500 <- vector("list", length(ks))
evs_iso_15000 <- vector("list", length(ks))
set.seed(1934)
# Run for every k
for (j in 1 : length(ks)) {
  evs_iso_0[[j]] <- evaluateNtimes(simouts0, isomapfunct, k = ks[j])
  evs_iso_1000[[j]] <- evaluateNtimes(simouts1000, isomapfunct, k = ks[j])
  evs_iso_7500[[j]] <- evaluateNtimes(simouts7500, isomapfunct, k = ks[j])
  evs_iso_15000[[j]] <- evaluateNtimes(simouts15000, isomapfunct, k = ks[j])
}
```

We apply t-SNE in the same setup. Since this algorithm has a tuning parameter, perplexity, t-SNE is run for a range of its values (the rest of hyperparameters are kept fixed at the default values). In particular, t-SNE is evaluated for the following perplexities: 1.0, 3.1, 5.3, 7.5, 9.6. We use the implementation from the `Rtsne` package in R [17].

```
# Wrapper function
tsnefunct <- function(X, perp = 9) {
  res <- Rtsne::Rtsne(t(X), perplexity = perp)
  return(res$Y)
```

```
}

# Perplexity values
perps <- c(1.0, 3.1, 5.3, 7.5, 9.6)
evs_tsne_0 <- vector("list", length(perps))
evs_tsne_1000 <- vector("list", length(perps))
evs_tsne_7500 <- vector("list", length(perps))
evs_tsne_15000 <- vector("list", length(perps))
set.seed(1418)
# Run for every perplexity value...
for (j in 1 : length(perps)) {
  evs_tsne_0[[j]] <- evaluateNtimes(simouts0, tsnefunct, perp = perps[j])
  evs_tsne_1000[[j]] <- evaluateNtimes(simouts1000, tsnefunct, perp = perps[j])
  evs_tsne_7500[[j]] <- evaluateNtimes(simouts7500, tsnefunct, perp = perps[j])
  evs_tsne_15000[[j]] <- evaluateNtimes(simouts15000, tsnefunct, perp = perps[j])
}
```

Last but not least, we use diffusion maps on the simulated data. The implementation in Bioconductor R package `destiny` [18] allows specifying the number of nearest neighbours `k` that are considered when constructing the graph for the Markov chain. We test the algorithm for $\texttt{k} = 1, 8, 15, 22, 29$.

```
# Wrapper function
dmfunct <- function(X, k = 3) {
  res <- destiny::DiffusionMap(t(X), k = k, n_eigs = 3)
  # Ignore the first eigenvalue
  return(cbind(res@eigenvalues[2] * res$DC2, res@eigenvalues[3] * res$DC3))
}

# Different k
ks <- c(1, 8, 15, 22, 29)
evs_dm_0 <- vector("list", length(ks))
evs_dm_1000 <- vector("list", length(ks))
evs_dm_7500 <- vector("list", length(ks))
evs_dm_15000 <- vector("list", length(ks))
set.seed(1514)
# Run for every k
for (j in 1 : length(ks)) {
  evs_dm_0[[j]] <- evaluateNtimes(simouts0, dmfunct, k = ks[j])
  evs_dm_1000[[j]] <- evaluateNtimes(simouts1000, dmfunct, k = ks[j])
  evs_dm_7500[[j]] <- evaluateNtimes(simouts7500, dmfunct, k = ks[j])
  evs_dm_15000[[j]] <- evaluateNtimes(simouts15000, dmfunct, k = ks[j])
}
```

Finally, we also evaluate ASW and DB index for the original datasets, i.e. without performing any dimensionality reduction. These evaluations serve as a baseline, to see if the compared techniques result in any improvement at all.

```
# Wrapper function
idfunct <- function(X) {
  # No dimensionality reduction
  return(t(X))
}

evs_id_0 <- evaluateNtimes(simouts0, idfunct)
evs_id_1000 <- evaluateNtimes(simouts1000, idfunct)
```

```
evs_id_7500 <- evaluateNtimes(simouts7500, idfunct)
evs_id_15000 <- evaluateNtimes(simouts15000, idfunct)
```

Results and Discussion

In this section we visualise calculated performance metrics, compare the techniques under various settings and discuss the results. All of the code used to execute necessary computations is provided in the previous section.

To begin with, it is important to compare the behaviour of the algorithms under the global null, i.e. when none of the genes are differentially expressed. Figure 2 contains boxplots of average silhouette widths for all tested techniques (under various parameter values) as well as the distribution of ASW for the original data.

```
# Set up colours for plotting
idcol <- "#FFFFFF"
colfunc_pc <- colorRampPalette(c("gray75", "gray48"))
pccol <- colfunc_pc(2)
mdscol <- "#FFD700"
colfunc_iso <- colorRampPalette(c("coral", "coral4"))
isocol <- colfunc_iso(5)
colfunc_tsne <- colorRampPalette(c("deepskyblue", "navy"))
tsnecol <- colfunc_tsne(5)
colfunc_dm <- colorRampPalette(c("chartreuse", "forestgreen"))
dmcol <- colfunc_dm(5)
allcols <- c(idcol, pccol, mdscol, isocol, tsnecol, dmcol)

# Boxplots
boxplot(evs_id_0$asws, evs_pc_unsc_0$asws, evs_pc_sc_0$asws, evs_mds_0$asws,
        evs_iso_0[[1]]$asws, evs_iso_0[[2]]$asws, evs_iso_0[[3]]$asws,
        evs_iso_0[[4]]$asws, evs_iso_0[[5]]$asws, evs_tsne_0[[1]]$asws,
        evs_tsne_0[[2]]$asws, evs_tsne_0[[3]]$asws, evs_tsne_0[[4]]$asws,
        evs_tsne_0[[5]]$asws, evs_dm_0[[1]]$asws, evs_dm_0[[2]]$asws,
        evs_dm_0[[3]]$asws, evs_dm_0[[4]]$asws, evs_dm_0[[5]]$asws,
        horizontal = TRUE, col = allcols, yaxt = "n", xlab = "ASW",
        cex.lab = 0.6, cex.axis = 0.6, outline = FALSE)
abline(v = 0, lty = 2, lwd = 2, col = "red")
legendlabs <- c("Original data", "PCA, unscaled", "scaled", "MDS", "Isomap, k=2",
                "k=8", "k=15", "k=21", "k=28", "t-SNE, perpl.=1.0", "perpl.=3.1",
                "perpl.=5.3", "perpl.=7.5", "perpl.=9.6", "Diffusion maps, k=1",
                "k=8", "k=15", "k=22", "k=29")
legend("bottomright", legend = legendlabs, fill = allcols, border = "black",
       cex = 0.6, bg = "white")
```

As can be seen, all ASW distributions concentrate around points close to 0. Overall, there appears to be little difference between distinct parameter values for the algorithms, e.g. for t-SNE or Isomap. Moreover, there is no considerable difference between algorithms either. It is interesting that for the most of techniques mean ASW is below 0. These results fully agree with our expectations. Generally, all the considered methods on average do not create spurious clustering by treatment. However, it is somewhat worrisome that distributions of ASW have high variances, and there exist some outliers with greater silhouette widths. This experiment demonstrates quite well that visualisation by dimensionality reduction should not be overinterpreted, since, even under the global null, we can obtain reduction with weak clustering by treatment (ASW of, approximately, 0.2). Results for the Davies-Bouldin index are very similar in this setting and, therefore, are omitted.

Another case that we consider is when only 1000 out of total 15000 genes are differentially expressed. Under

this setting, distributions of average silhouette widths are similar to the global null. Figure 3 shows boxplots of ASW values for all tested approaches.

```
boxplot(evs_id_1000$asws, evs_pc_unsc_1000$asws, evs_pc_sc_1000$asws,
        evs_mds_1000$asws, evs_iso_1000[[1]]$asws, evs_iso_1000[[2]]$asws,
        evs_iso_1000[[3]]$asws, evs_iso_1000[[4]]$asws, evs_iso_1000[[5]]$asws,
        evs_tsne_1000[[1]]$asws, evs_tsne_1000[[2]]$asws, evs_tsne_1000[[3]]$asws,
        evs_tsne_1000[[4]]$asws, evs_tsne_1000[[5]]$asws, evs_dm_1000[[1]]$asws,
        evs_dm_1000[[2]]$asws, evs_dm_1000[[3]]$asws, evs_dm_1000[[4]]$asws,
        evs_dm_1000[[5]]$asws, horizontal = TRUE, col = allcols, yaxt = "n",
        xlab = "ASW", cex.lab = 0.6, cex.axis = 0.6, outline = FALSE)
abline(v = 0, lty = 2, lwd = 2, col = "red")
legend("bottomright", legend = legendlabs, fill = allcols, border = "black",
       cex = 0.6, bg = "white")
```

All distributions have means close to 0. Overall, despite the fact, that simulated data contain differential expression, none of the algorithms manages to capture differences between treatment groups. This could be explained by the phenomenon of the curse of dimensionality [19], the techniques we apply might get overwhelmed with a large number of 'irrelevant' features. These results suggest a need for variable selection before performing dimensionality reduction, especially, when having datasets with only few informative dimensions. Similar conclusions can be made from the DB indices, which are plotted on log-scale in Figure 4 (recall that lesser values of DB correspond to better performance of clustering).

```
boxplot(evs_id_1000$dbs, evs_pc_unsc_1000$dbs, evs_pc_sc_1000$dbs,
        evs_mds_1000$dbs, evs_iso_1000[[1]]$dbs, evs_iso_1000[[2]]$dbs,
        evs_iso_1000[[3]]$dbs, evs_iso_1000[[4]]$dbs, evs_iso_1000[[5]]$dbs,
        evs_tsne_1000[[1]]$dbs, evs_tsne_1000[[2]]$dbs, evs_tsne_1000[[3]]$dbs,
        evs_tsne_1000[[4]]$dbs, evs_tsne_1000[[5]]$dbs, evs_dm_1000[[1]]$dbs,
        evs_dm_1000[[2]]$dbs, evs_dm_1000[[3]]$dbs, evs_dm_1000[[4]]$dbs,
        evs_dm_1000[[5]]$dbs, horizontal = TRUE, col = allcols, yaxt = "n",
        xlab = "DB", log = "x", cex.lab = 0.6, cex.axis = 0.6, outline = FALSE)
legend("bottomright", legend = legendlabs, fill = allcols, border = "black",
       cex = 0.6, bg = "white")
```

When applying dimensionality reduction to datasets with half of genes differentially expressed, we managed to obtain noticeably stronger clustering by treatment then in the raw data, despite that all techniques we use are unsupervised and, thus, do not consider treatment labels. Boxplots of ASW are depicted in Figure 5.

```
boxplot(evs_id_7500$asws, evs_pc_unsc_7500$asws, evs_pc_sc_7500$asws,
        evs_mds_7500$asws, evs_iso_7500[[1]]$asws, evs_iso_7500[[2]]$asws,
        evs_iso_7500[[3]]$asws, evs_iso_7500[[4]]$asws, evs_iso_7500[[5]]$asws,
        evs_tsne_7500[[1]]$asws, evs_tsne_7500[[2]]$asws, evs_tsne_7500[[3]]$asws,
        evs_tsne_7500[[4]]$asws, evs_tsne_7500[[5]]$asws, evs_dm_7500[[1]]$asws,
        evs_dm_7500[[2]]$asws, evs_dm_7500[[3]]$asws, evs_dm_7500[[4]]$asws,
        evs_dm_7500[[5]]$asws, horizontal = TRUE, col = allcols, yaxt = "n",
        xlab = "ASW", cex.lab = 0.6, cex.axis = 0.6, outline = FALSE)
abline(v = 0, lty = 2, lwd = 2, col = "red")
legend("bottomright", legend = legendlabs, fill = allcols, border = "black",
       cex = 0.6, bg = "white")
```

It is noteworthy, that, in this case, the ASW of obtained low-dimensional representations for Isomap and diffusion maps varies much more depending on the chosen number of neighbours, as opposed to the two previous settings discussed above (see Figures 3 and 2). On the other hand, the performance of t-SNE is quite stable across all tested perplexity values. Based on the ASW criterion, there is a visible improvement from using dimensionality reduction techniques; non-linear methods, Isomap, t-SNE and diffusion maps, being a bit more promising, on average, than the classical methods, MDS and PCA. For the Davies-Bouldin index,

we observe less gain from dimensionality reduction (see Figure 6). Nevertheless, the behaviour w.r.t. different parameter values is very similar.

```
boxplot(evs_id_7500$dbs, evs_pc_unsc_7500$dbs, evs_pc_sc_7500$dbs,
        evs_mds_7500$dbs, evs_iso_7500[[1]]$dbs, evs_iso_7500[[2]]$dbs,
        evs_iso_7500[[3]]$dbs, evs_iso_7500[[4]]$dbs, evs_iso_7500[[5]]$dbs,
        evs_tsne_7500[[1]]$dbs, evs_tsne_7500[[2]]$dbs, evs_tsne_7500[[3]]$dbs,
        evs_tsne_7500[[4]]$dbs, evs_tsne_7500[[5]]$dbs, evs_dm_7500[[1]]$dbs,
        evs_dm_7500[[2]]$dbs, evs_dm_7500[[3]]$dbs, evs_dm_7500[[4]]$dbs,
        evs_dm_7500[[5]]$dbs, horizontal = TRUE, col = allcols, yaxt = "n",
        xlab = "DB", log = "x", cex.lab = 0.6, cex.axis = 0.6, outline = FALSE)
legend("bottomright", legend = legendlabs, fill = allcols, border = "black",
       cex = 0.6, bg = "white")
```

Results for the experiment with all genes being differentially expressed (see Figures 7 and 8) are unexpected. It appears that under this setting the gain from dimensionality reduction is less than when only a half of genes are DE. Moreover, based on the DB index, t-SNE with perplexity 1.0 is even less informative than the raw data. Observe that, as opposed to the previous experiment, herein, t-SNE exhibits considerable amount of variability across different perplexities. Despite the fact that cases of complete differential expression are, probably, uncommon, it is worrisome that some techniques, under these conditions, can reflect even less information about the treatment than a raw dataset. A possible explanation could be that by reducing dimensionality we lose some differentiation that is present in *all* of genes. Nevertheless, unscaled PCA, MDS and Isomap still stably attain some gain compared to the high-dimensional space.

```
boxplot(evs_id_15000$asws, evs_pc_unsc_15000$asws, evs_pc_sc_15000$asws,
        evs_mds_15000$asws, evs_iso_15000[[1]]$asws, evs_iso_15000[[2]]$asws,
        evs_iso_15000[[3]]$asws, evs_iso_15000[[4]]$asws, evs_iso_15000[[5]]$asws,
        evs_tsne_15000[[1]]$asws, evs_tsne_15000[[2]]$asws, evs_tsne_15000[[3]]$asws,
        evs_tsne_15000[[4]]$asws, evs_tsne_15000[[5]]$asws, evs_dm_15000[[1]]$asws,
        evs_dm_15000[[2]]$asws, evs_dm_15000[[3]]$asws, evs_dm_15000[[4]]$asws,
        evs_dm_15000[[5]]$asws, horizontal = TRUE, col = allcols, yaxt = "n",
        xlab = "ASW", cex.lab = 0.6, cex.axis = 0.6, outline = FALSE)
abline(v = 0, lty = 2, lwd = 2, col = "red")
legend("bottomright", legend = legendlabs, fill = allcols, border = "black",
       cex = 0.6, bg = "white")
```

```
boxplot(evs_id_15000$dbs, evs_pc_unsc_15000$dbs, evs_pc_sc_15000$dbs,
        evs_mds_15000$dbs, evs_iso_15000[[1]]$dbs, evs_iso_15000[[2]]$dbs,
        evs_iso_15000[[3]]$dbs, evs_iso_15000[[4]]$dbs, evs_iso_15000[[5]]$dbs,
        evs_tsne_15000[[1]]$dbs, evs_tsne_15000[[2]]$dbs, evs_tsne_15000[[3]]$dbs,
        evs_tsne_15000[[4]]$dbs, evs_tsne_15000[[5]]$dbs, evs_dm_15000[[1]]$dbs,
        evs_dm_15000[[2]]$dbs, evs_dm_15000[[3]]$dbs, evs_dm_15000[[4]]$dbs,
        evs_dm_15000[[5]]$dbs, horizontal = TRUE, col = allcols, yaxt = "n",
        xlab = "DB", log = "x", cex.lab = 0.6, cex.axis = 0.6, outline = FALSE)
legend("bottomright", legend = legendlabs, fill = allcols, border = "black",
       cex = 0.6, bg = "white")
```

In summary, even though we observed some improvement from dimensionality reduction, it is, probably, not very practical to apply these techniques on unfiltered original variables, while expecting to obtain a strong treatment signal in the visualisation. It could be useful to pre-select variables in a supervised manner.

Another goal of this report is to consider scalability of the techniques that we compare. Figure 9 contains boxplots of elapsed running times for all algorithms. Note, that running times were aggregated across all four settings that we examined.

```
boxplot(c(evs_pc_unsc_0$rtimes, evs_pc_unsc_1000$rtimes, evs_pc_unsc_7500$rtimes,
          evs_pc_unsc_15000$rtimes), c(evs_pc_sc_0$rtimes, evs_pc_sc_1000$rtimes,
                                       evs_pc_sc_7500$rtimes, evs_pc_sc_15000$rtimes),
        c(evs_mds_0$rtimes, evs_mds_1000$rtimes, evs_mds_7500$rtimes,
          evs_mds_15000$rtimes), c(evs_iso_0[[1]]$rtimes, evs_iso_1000[[1]]$rtimes,
                                   evs_iso_7500[[1]]$rtimes, evs_iso_15000[[1]]$rtimes),
        c(evs_iso_0[[2]]$rtimes, evs_iso_1000[[2]]$rtimes, evs_iso_7500[[2]]$rtimes,
          evs_iso_15000[[2]]$rtimes), c(evs_iso_0[[3]]$rtimes, evs_iso_1000[[3]]$rtimes,
                                        evs_iso_7500[[3]]$rtimes, evs_iso_15000[[3]]$rtimes),
        c(evs_iso_0[[4]]$rtimes, evs_iso_1000[[4]]$rtimes, evs_iso_7500[[4]]$rtimes,
          evs_iso_15000[[4]]$rtimes), c(evs_iso_0[[5]]$rtimes, evs_iso_1000[[5]]$rtimes,
                                        evs_iso_7500[[5]]$rtimes, evs_iso_15000[[5]]$rtimes),
        c(evs_tsne_0[[1]]$rtimes, evs_tsne_1000[[1]]$rtimes, evs_tsne_7500[[1]]$rtimes,
          evs_tsne_15000[[1]]$rtimes), c(evs_tsne_0[[2]]$rtimes,
          evs_tsne_1000[[2]]$rtimes, evs_tsne_7500[[2]]$rtimes,
          evs_tsne_15000[[2]]$rtimes), c(evs_tsne_0[[3]]$rtimes,
          evs_tsne_1000[[3]]$rtimes, evs_tsne_7500[[3]]$rtimes,
          evs_tsne_15000[[3]]$rtimes), c(evs_tsne_0[[4]]$rtimes,
          evs_tsne_1000[[4]]$rtimes, evs_tsne_7500[[4]]$rtimes,
          evs_tsne_15000[[4]]$rtimes), c(evs_tsne_0[[5]]$rtimes,
          evs_tsne_1000[[5]]$rtimes, evs_tsne_7500[[5]]$rtimes,
          evs_tsne_15000[[5]]$rtimes), c(evs_dm_0[[1]]$rtimes, evs_dm_1000[[1]]$rtimes,
          evs_dm_7500[[1]]$rtimes, evs_dm_15000[[1]]$rtimes), c(evs_dm_0[[2]]$rtimes,
          evs_dm_1000[[2]]$rtimes, evs_dm_7500[[2]]$rtimes, evs_dm_15000[[2]]$rtimes),
        c(evs_dm_0[[3]]$rtimes, evs_dm_1000[[3]]$rtimes, evs_dm_7500[[3]]$rtimes,
          evs_dm_15000[[3]]$rtimes), c(evs_dm_0[[4]]$rtimes, evs_dm_1000[[4]]$rtimes,
          evs_dm_7500[[4]]$rtimes, evs_dm_15000[[4]]$rtimes), c(evs_dm_0[[5]]$rtimes,
          evs_dm_1000[[5]]$rtimes, evs_dm_7500[[5]]$rtimes, evs_dm_15000[[5]]$rtimes),
        horizontal = TRUE, col = allcols[2 : length(allcols)], yaxt = "n",
        xlab = "Elapsed Time", cex.lab = 0.6, cex.axis = 0.6, outline = FALSE)
legend("topleft", legend = legendlabs[2 : length(legendlabs)],
       fill = allcols[2 : length(allcols)], border = "black", cex = 0.6, bg = "white")
```

Based on these results, MDS and Isomap were the fastest. Observe that, as expected, PCA on unscaled data required less running time than PCA with scaling. The most time consuming algorithm is the t-distributed stochastic neighbour embedding. Overall, under rather small datasets that were simulated, the running time is, arguably, not a very important factor and is reasonable for all methods. However, with larger datasets, we might have to consider it more carefully.

Conclusions

In this report, we studied the behaviour of five data dimensionality reduction techniques in application to RNA-seq count matrices. In particular, the focus of the experiments is on the ability of and the gain from low-dimensional representation in reflecting treatment effects. We used the average silhouette width and the Davies-Bouldin index as comparison criteria. Techniques were studied under four settings with varying percentage of differentially expressed genes. Realistic RNA-seq datasets were simulated with the SimSeq, a non-parametric approach described in the literature.

The results show that dimensionality reduction can lead to better visualisations of treatment effects (in terms of clustering by treatment labels). However, the improvement was only observed when the number of differentially expressed genes was substantial (at least a half). Additionally, we demonstrated that this improvement might depend on algorithm parameter values, for Isomap, t-SNE and diffusion maps. Apart from this, the algorithms were compared based on their running times. We observed differences in scalability, which could be crucial in larger datasets. It would be interesting to extend this report by performing similar

investigations with *several* source datasets used in simulations, to see how well the observations we made generalise.

References

[1] W. Huber *et al.*, "Orchestrating high-throughput genomic analysis with Bioconductor," *Nat. Methods*, vol. 12, no. 2, pp. 115–121, 2015.

[2] L. Wang, Y. Wang, and Q. Chang, "Feature selection methods for big data bioinformatics: A survey from the search perspective," *Methods*, vol. 111, pp. 21–31, 2016.

[3] L. van der Maaten, E. O. Postma, and H. J. van den Herik, "Dimensionality reduction: A comparative review." 2008.

[4] F. Wang and J. Sun, "Survey on distance metric learning and dimensionality reduction in data mining," *Data Mining and Knowledge Discovery*, vol. 29, no. 2, pp. 534–564, 2015.

[5] C. O. S. Sorzano, J. Vargas, and A. P. Montano, "A survey of dimensionality reduction techniques." 2014.

[6] J. P. Cunningham and Z. Ghahramani, "Linear dimensionality reduction: Survey, insights, and generalizations," *Journal of Machine Learning Research*, vol. 16, pp. 2859–2900, 2015.

[7] L. van der Maaten and G. Hinton, "Visualizing data using t-SNE," *Journal of Machine Learning Research*, vol. 9, pp. 2579–2605, 2008.

[8] S. Benidt and D. Nettleton, "SimSeq: a nonparametric approach to simulation of RNA-sequence datasets," *Bioinformatics*, vol. 31, no. 13, pp. 2131–2140, 2015.

[9] J. K. Pickrell *et al.*, "Understanding mechanisms underlying human gene expression variation with RNA sequencing," *Nature*, vol. 464, no. 7289, pp. 768–772, 2010.

[10] J. R. Gonzalez and M. Esnaola, *TweeDEseqCountData: RNA-seq count data employed in the vignette of the tweeDEseq package. Available at http://www.creal.cat/jrgonzalez/software.htm*, 2018.

[11] M. D. Robinson, D. J. McCarthy, and G. K. Smyth, "EdgeR: A bioconductor package for differential expression analysis of digital gene expression data," *Bioinformatics*, vol. 26, no. 1, pp. 139–140, 2010.

[12] P. Rousseeuw, "Silhouettes: A graphical aid to the interpretation and validation of cluster analysis," *Journal of Computational and Applied Mathematics*, vol. 20, no. 1, pp. 53–65, 1987.

[13] M. Maechler, P. Rousseeuw, A. Struyf, M. Hubert, and K. Hornik, *Cluster: Cluster analysis basics and extensions.* 2018.

[14] D. L. Davies and D. W. Bouldin, "A cluster separation measure," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vols. PAMI-1, no. 2, pp. 224–227, 1979.

[15] M. Walesiak and A. Dudek, *ClusterSim: Searching for optimal clustering procedure for a data set.* 2018.

[16] C. Bartenhagen, *RDRToolbox: A package for nonlinear dimension reduction with isomap and lle.* 2018.

[17] J. H. Krijthe, *Rtsne: T-distributed stochastic neighbor embedding using barnes-hut implementation.* 2015.

[18] P. Angerer, L. Haghverdi, M. Büttner, F. Theis, C. Marr, and F. Büttner, "Destiny: Diffusion maps for large-scale single-cell data in r," *Bioinformatics*, vol. 32, no. 8, pp. 1241–1243, 2015.

[19] R. Clarke *et al.*, "The properties of high-dimensional data spaces: implications for exploring gene and protein expression data," *Nat. Rev. Cancer*, vol. 8, no. 1, pp. 37–49, 2008.

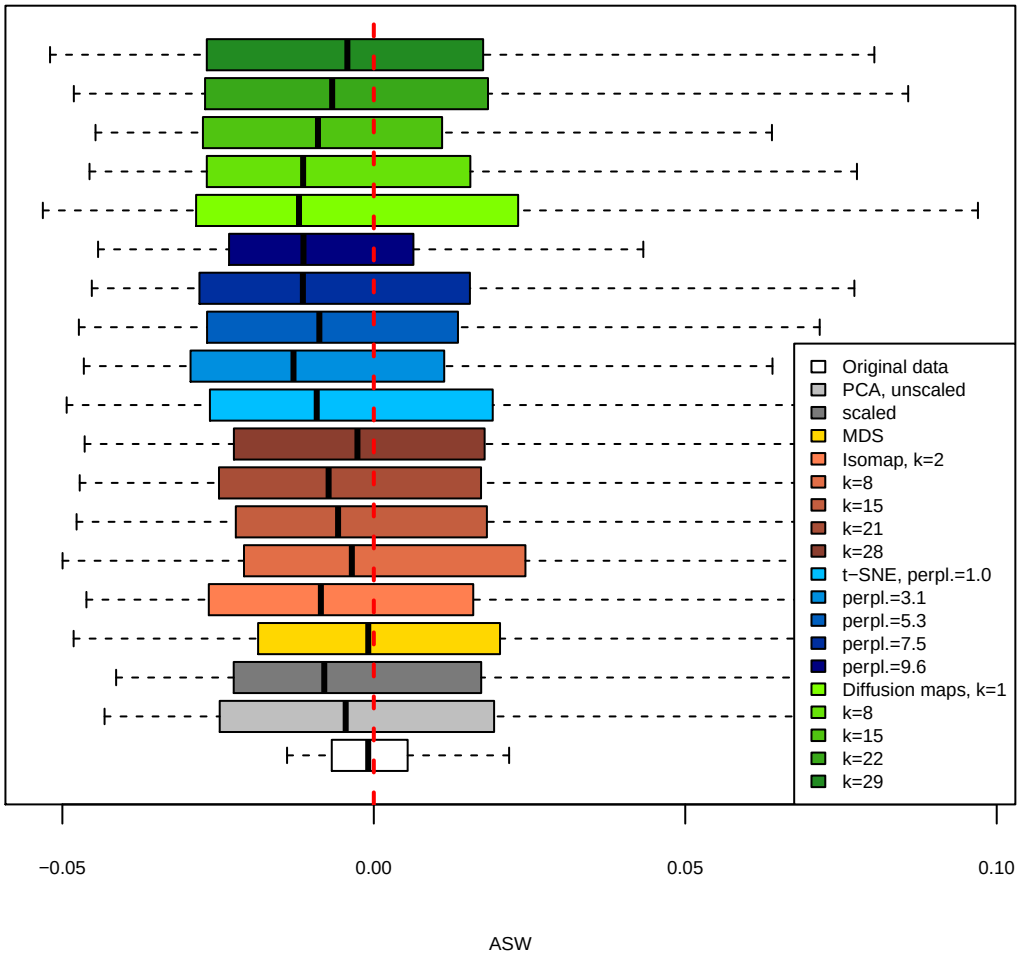

Figure 2: ASW values for dimensionality reduction techniques when applied to data with no differential expression.

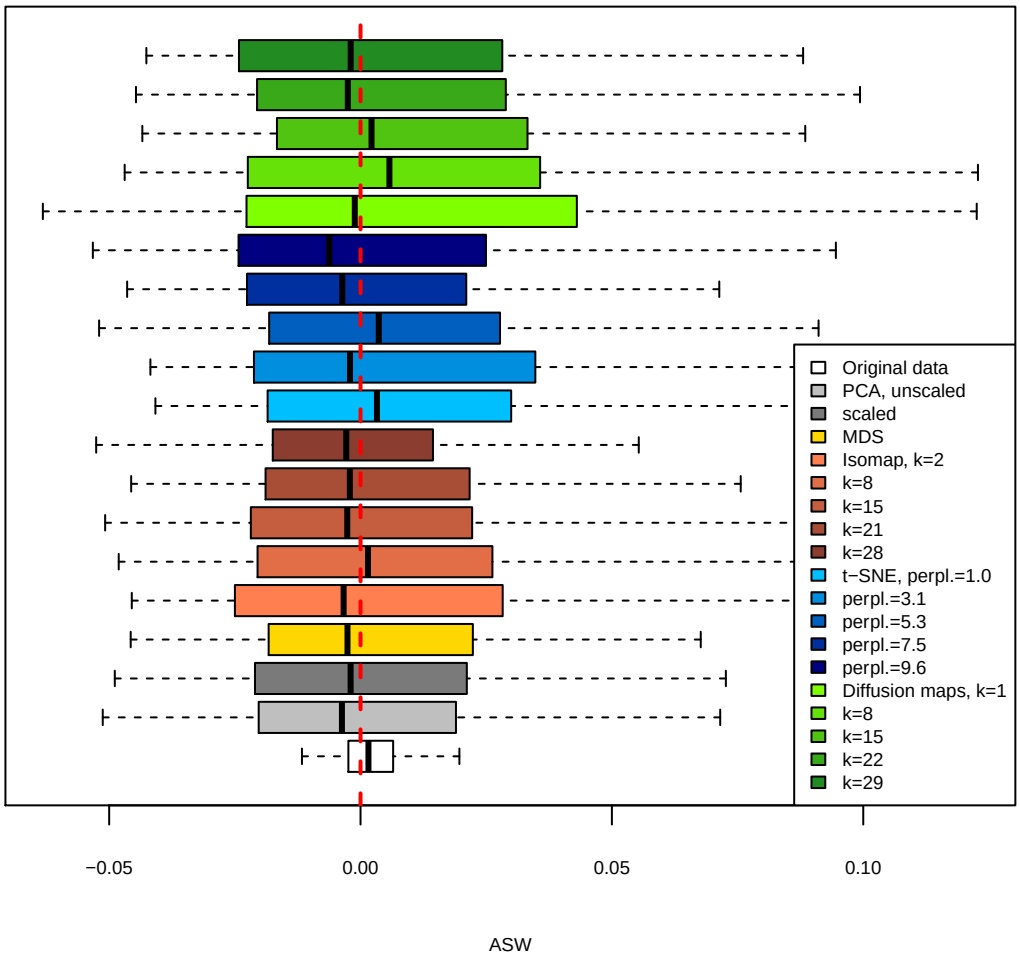

Figure 3: ASW values for dimensionality reduction techniques when applied to data with 1000 DE genes.

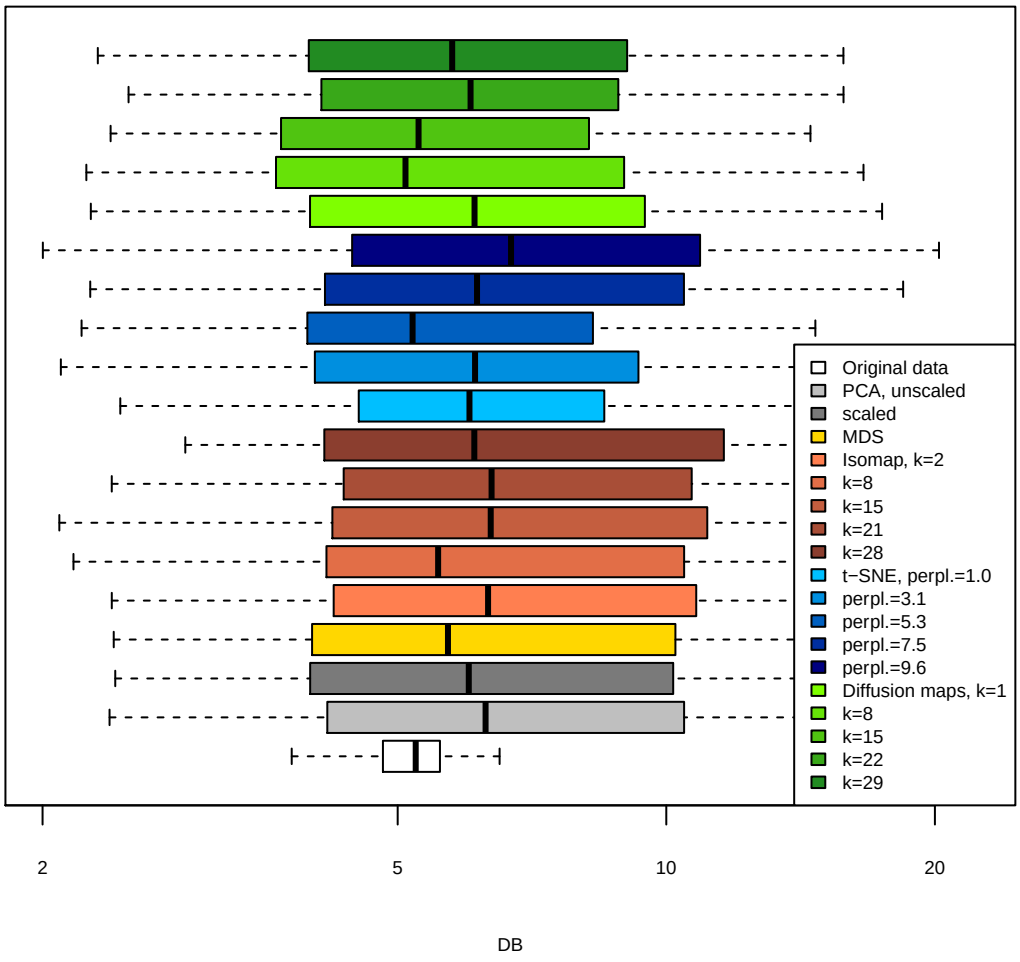

Figure 4: DB index values for dimensionality reduction techniques when applied to data with 1000 DE genes.

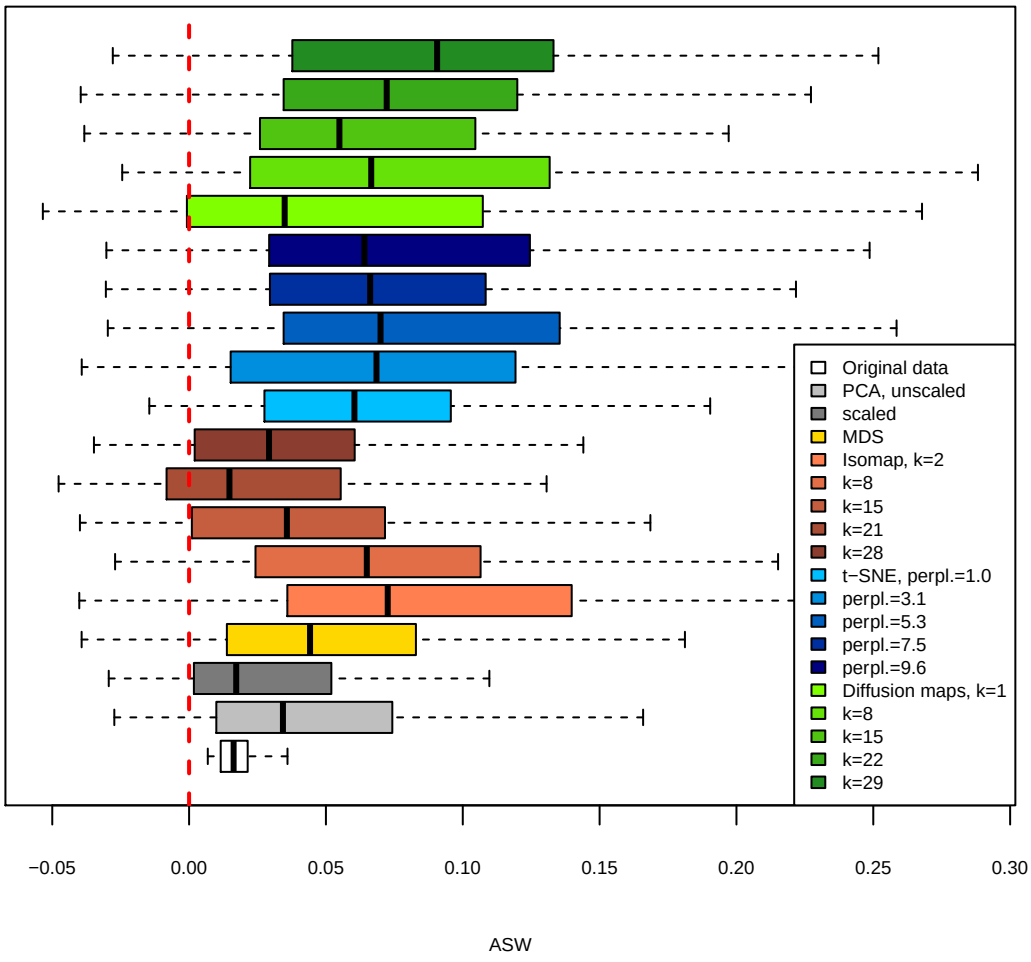

Figure 5: ASW values for dimensionality reduction techniques when applied to data with 7500 DE genes.

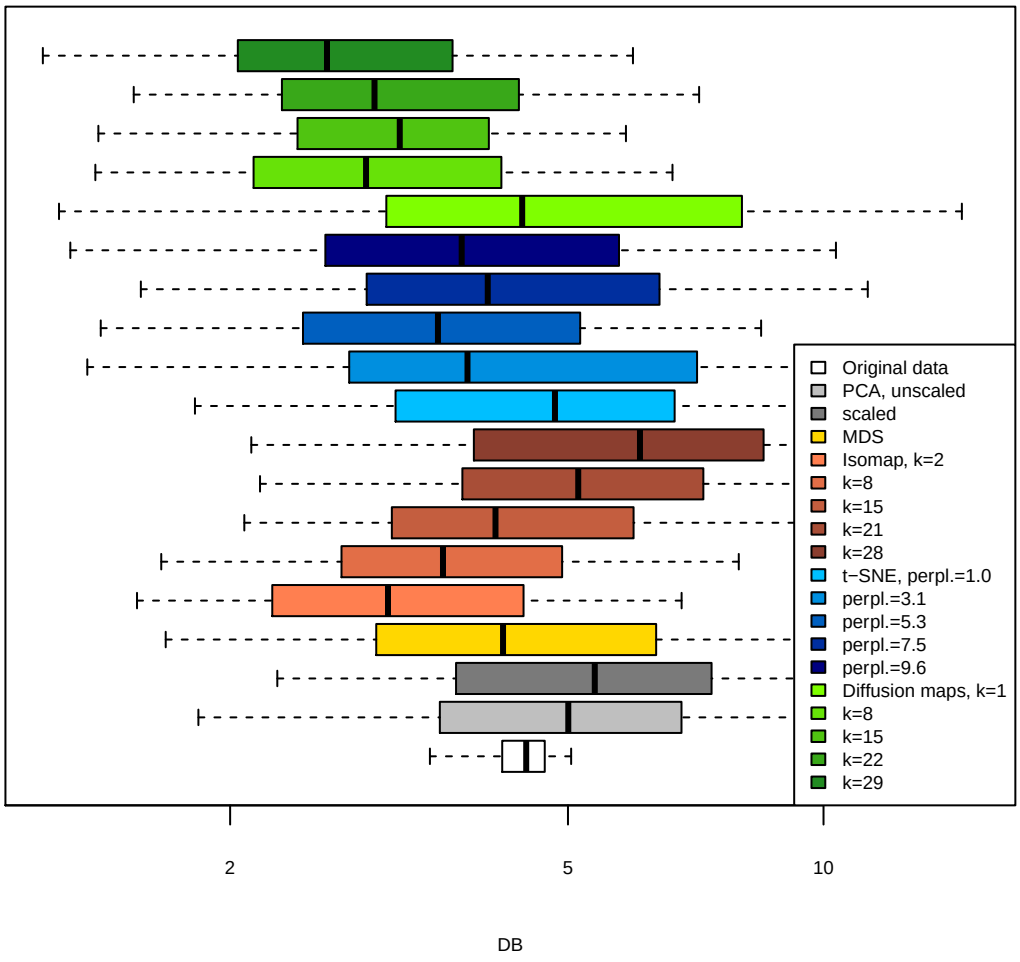

Figure 6: DB index values for dimensionality reduction techniques when applied to data with 7500 DE genes.

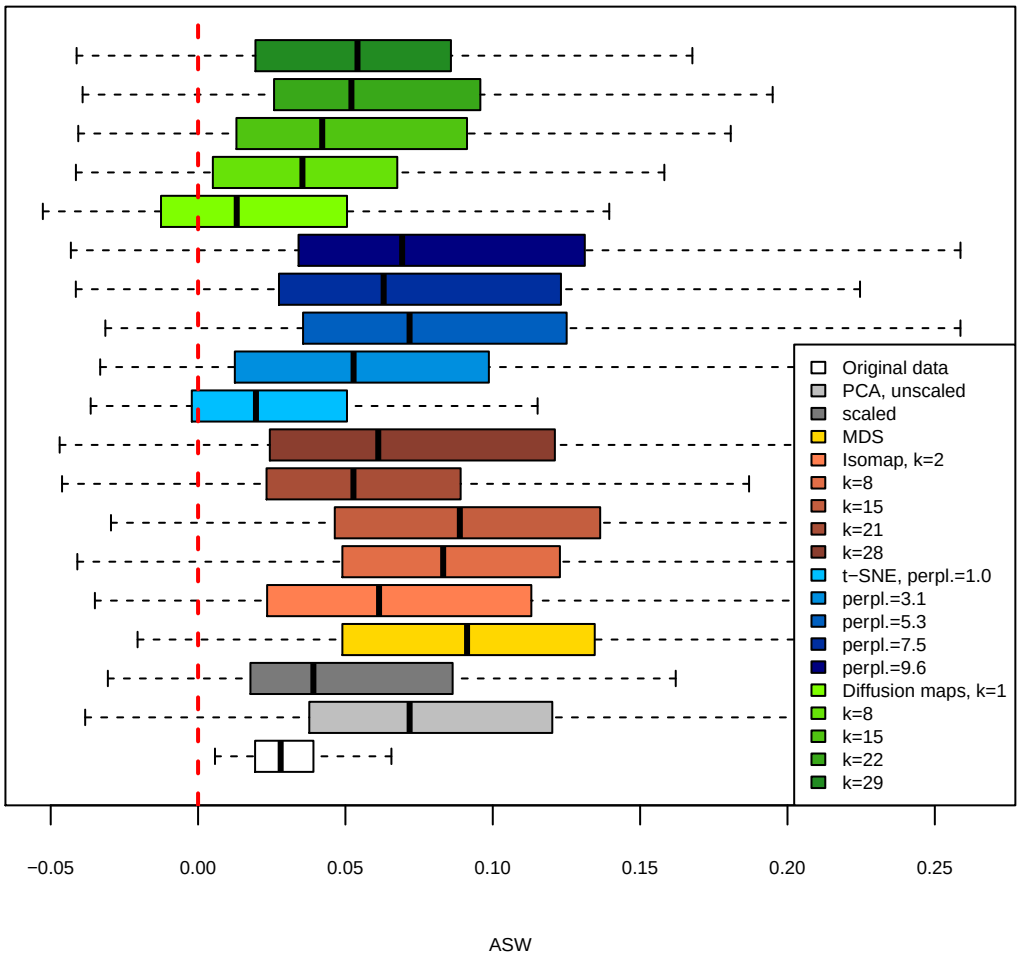

Figure 7: ASW values for dimensionality reduction techniques when applied to data with 15000 DE genes.

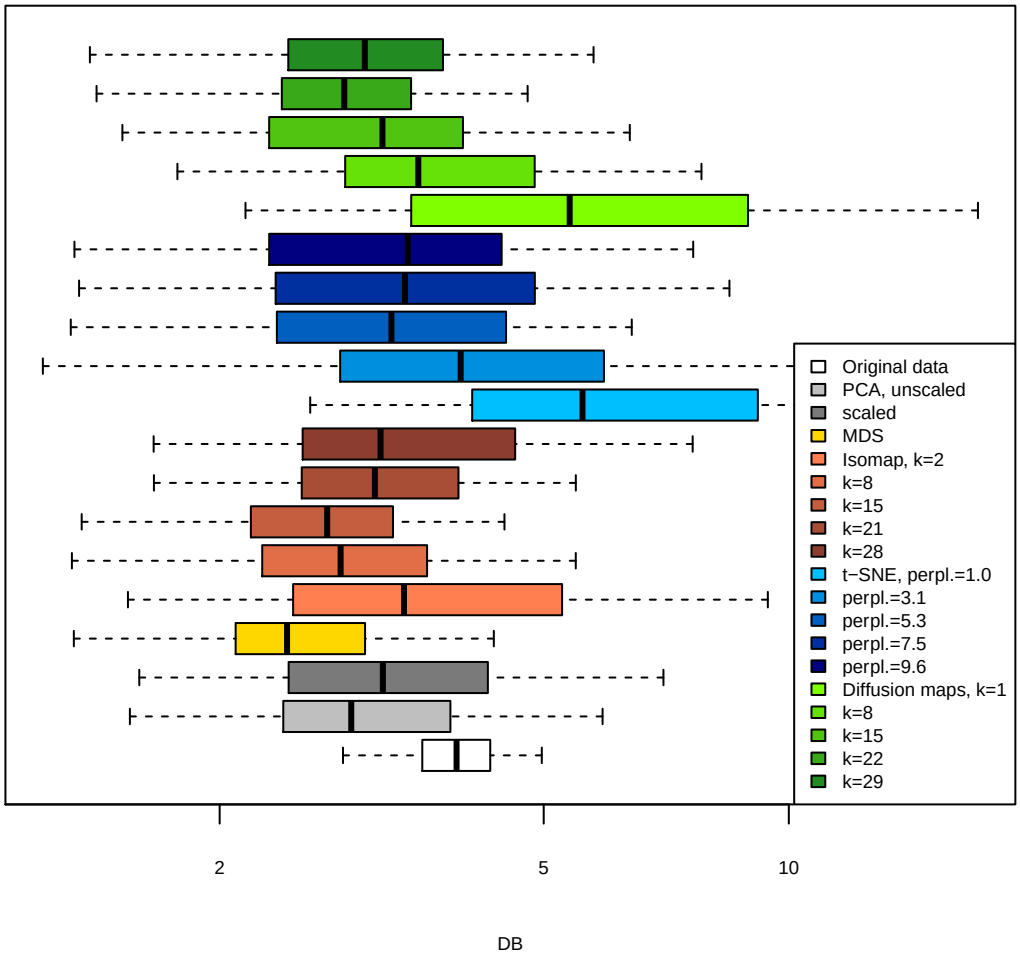

Figure 8: DB index values for dimensionality reduction techniques when applied to data with 15000 DE genes.

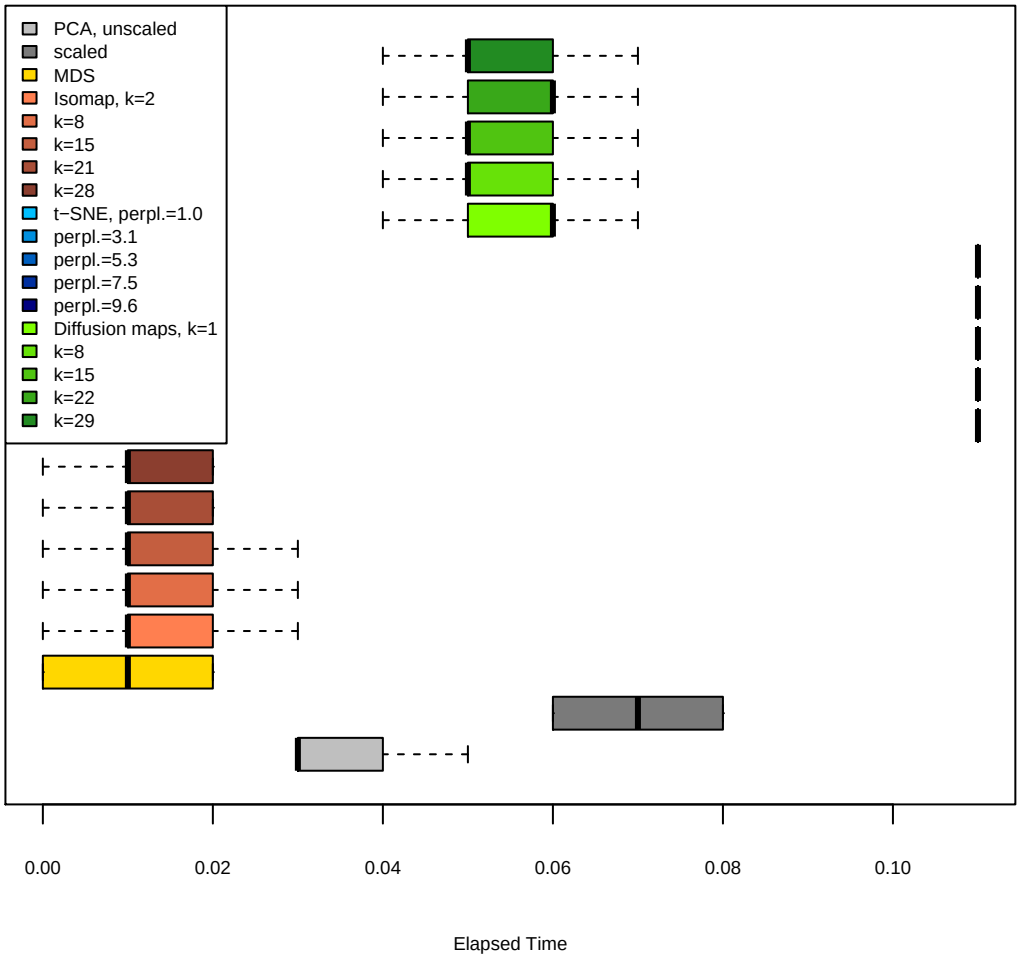

Figure 9: Elapsed running time of dimensionality reduction algorithms under different parameter values for one dataset with 30 observations and 15000 genes.