Dimensionality reduction

Martin Endler endlemar@fel.cvut.cz

Introduction

The goal of this tutorial is to get familiar with some basic methods for dimensionality reduction, complete your own implementation of the **Isomap algorithm** (in cooperation with *Multidimensional scaling*), experiment with its parameters and compare with other techniques of dimensionality reduction (**PCA**, **t-SNE**).

Background

The data you will be working with are vector representations of words in a latent (unknown) high-dimensional space. This representation of words, also known as word embedding, differs from standard bag-of-words (BoW, TFIDF, etc.) representations in that the meaning of the words is distributed across all the dimensions. Generally speaking, the word embedding algorithms seek to learn a mapping projecting the original BoW representation (simple word index into a given vocabulary) into a lower-dimensional (but still too high for our cause) continuous vector-space, based on their distributional properties observed in some raw text corpus. This distributional semantics approach to word representations is based on the basic idea that linguistic items with similar distributions typically have similar meanings, i.e. words that often appear in a similar context (words that surround them) tend to have similar (vector) representations.

Specifically, the data you are presented with are vector representations coming from the most popular algorithm for word embedding known as word2vec[1] by Tomas Mikolov (VUT-Brno alumni). *Word2vec* is a (shallow) neural model learning the projection of BoW word representations into a latent space by the means of gradient descend. Your task is to further reduce the dimensionality of the word representations to get a visual insight into what has been learned.

Data

You are given 300-dimensional word2vec vector embeddings in the file *data.csv* with corresponding word labels in *labels.txt* for each line. Each of these words comes from one of 10 selected classes of synonyms, which can be recognized (and depicted) w.r.t. labels denoted in the file *colors.csv*

Tasks

1. **Load the dataset of 165 words**, each represented as a 300-dimensional vector. Each word is assigned to one of 10 clusters.

```
# load the dataset of 165 words
mydata <- read.csv("data.csv", header = FALSE)
mylabels <- read.csv("labels.txt", header = FALSE)
mycolors <- read.csv("colors.csv", header = FALSE)
```

The data is in the matrix `mydata`, cluster assignment in `mycolors` and the actual words (useful for visualization) in `mylabels`. You can plot the data by using only the first 2 dimensions.

[1] Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg S Corrado, and Jeff Dean. Distributed representations of words and phrases and their compositionality. In *Advances in neural information processing systems*, pages 3111-3119, 2013.

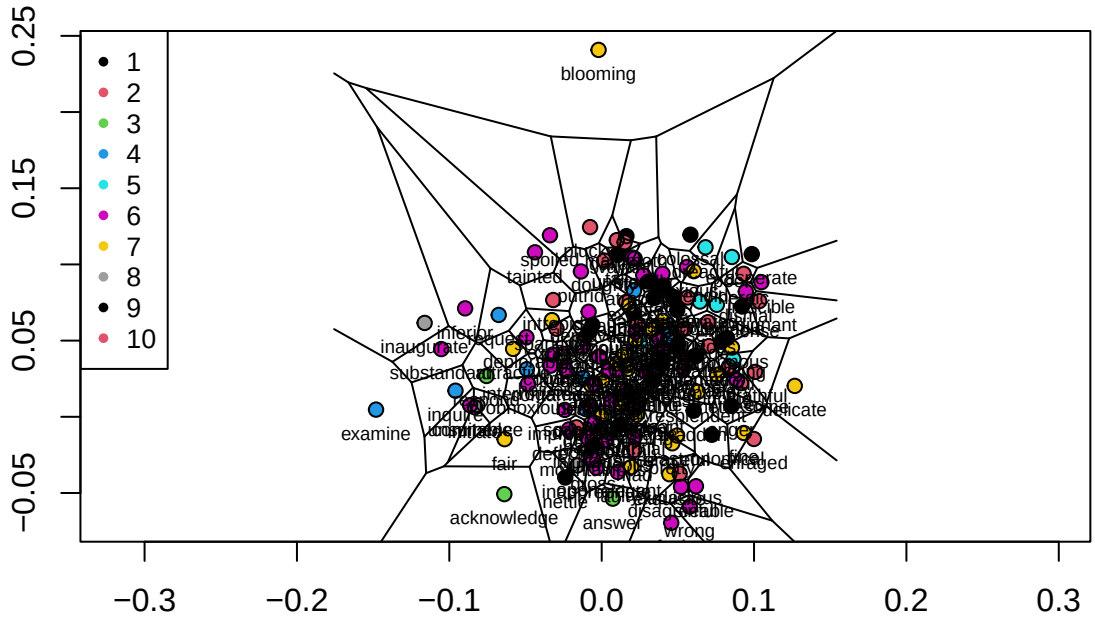
```
PlotPoints(mydata[, c(1, 2)], mylabels, mycolors)
```

```
## deldir 1.0-6      Nickname: "Mendacious Cosmonaut"
```

```
##
##       The syntax of deldir() has had an important change.
##       The arguments have been re-ordered (the first three
##       are now "x, y, z") and some arguments have been
##       eliminated.  The handling of the z ("tags")
##       argument has been improved.
##
##       The "dummy points" facility has been removed.
##       This facility was a historical artefact, was really
##       of no use to anyone, and had hung around much too
##       long.  Since there are no longer any "dummy points",
##       the structure of the value returned by deldir() has
##       changed slightly.  The arguments of plot.deldir()
##       have been adjusted accordingly; e.g. the character
##       string "wpoints" ("which points") has been
##       replaced by the logical scalar "showpoints".
##       The user should consult the help files.
```

```
## Warning in (function (x0, y0, x1 = x0, y1 = y0, col = par("fg"), lty =
## par("lty"), : "wpoints" is not a graphical parameter
```

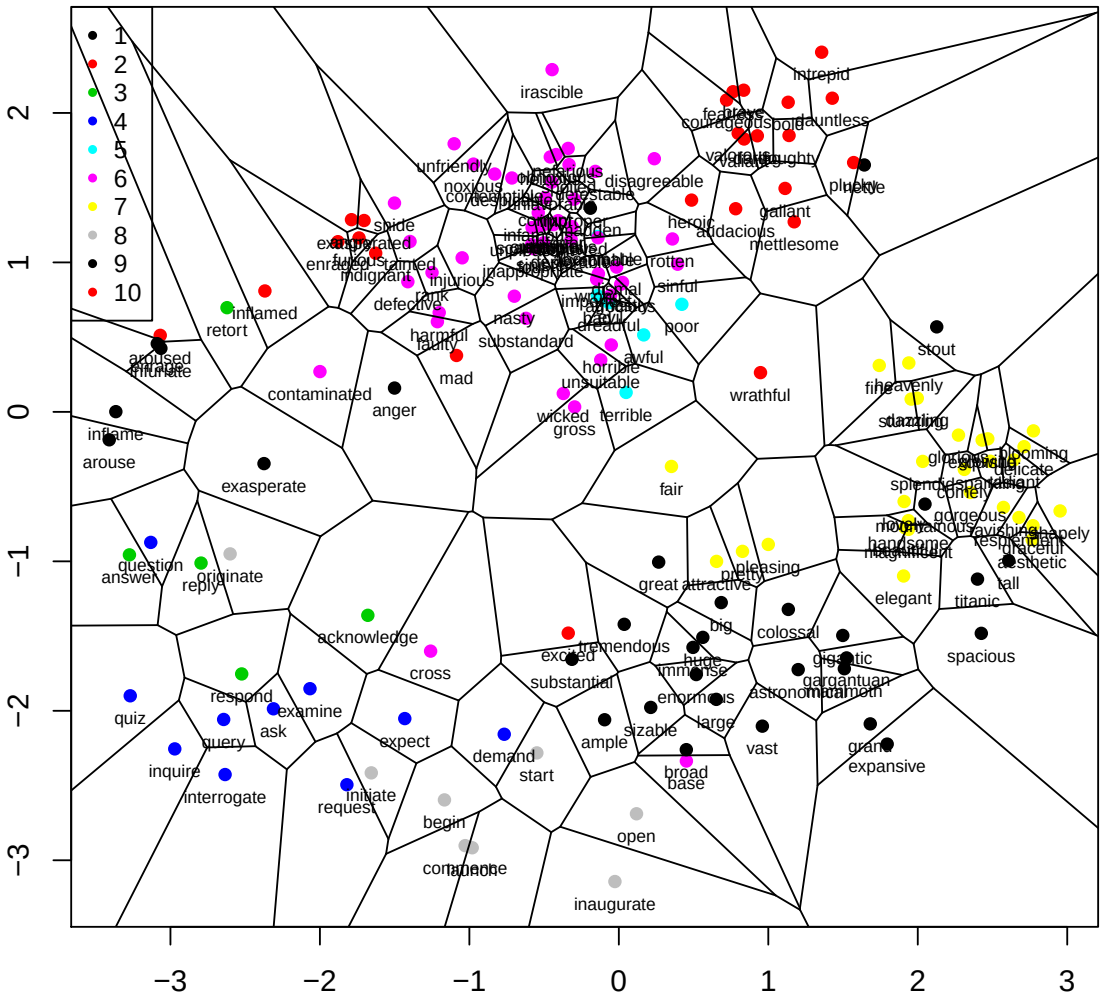
```
## Warning in plot.xy(xy.coords(x, y), type = type, ...): "wpoints" is not a
## graphical parameter
```



2. **Implement ISO-MAP dimensionality reduction procedure**.

- Use k-NN method to construct the neighborhood graph (sparse matrix).
  - For simplicity, you can use `get.knn` method available in `FNN` package.
- Compute shortest-paths (geodesic) matrix using your favourite algorithm.
  - Tip: Floyd-Warshall algorithm can be implemented easily here.
- Project the geodesic distance matrix into 2D space with (Classical) Multidimensional Scaling (`cmdscale` functions in R).
- Challenge: you may simply use PCA to do the same, but be careful to account for a proper normalization (centering) of the geodesic (kernel) matrix (see Kernel PCA for details).

An expected result (for $k = 5$) should look similar (not necessarily exactly the same) to following




```
MyIsomap2D <- function(data, knn.k) {

    k <- knn.k
```

```
    n <- nrow(data)

    # 1. Determine the neighbors of each point (k-NN)
    library(FNN)
    knn_output <- get.knn(mydata, k = k)

    # 2. Construct a neighborhood graph (i.e., adjacency matrix with edge lengths (Euclidean distances
    # NA when there is no edge (so that this matrix is compatiule with Rfast v2.0.6 Rfast::floyd
    # Each point is connected to other if it is a K nearest neighbor.
    # Edge length equal to Euclidean distance.
    neighborhood_graph <- matrix(NA, nrow = n, ncol = n)
    stopifnot(nrow(knn_output$nn.index) == n && nrow(knn_output$nn.dist) == n)
    stopifnot(ncol(knn_output$nn.index) == k && ncol(knn_output$nn.dist) == k)
    for (v1 in seq(n)) {
        for (i in seq(k)) {
            v2 <- knn_output$nn.index[v1, i]
            dist <- knn_output$nn.dist[v1, i]
            neighborhood_graph[v1, v2] <- dist
            neighborhood_graph[v2, v1] <- dist
        }
    }

    # 3. Compute shortest path between two nodes. (Floyd-Warshall algorithm)
    library(Rfast)
    D <- floyd(neighborhood_graph)

    # 4. (classical) multidimensional scaling
    fit <- cmdscale(D, eig = TRUE, k = 2)

    return(fit)

}
```

```
fitISOMAP <- MyIsomap2D(data = mydata, knn.k = 5)
```

```
## Loading required package: Rcpp
```

```
## Loading required package: RcppZiggurat
```

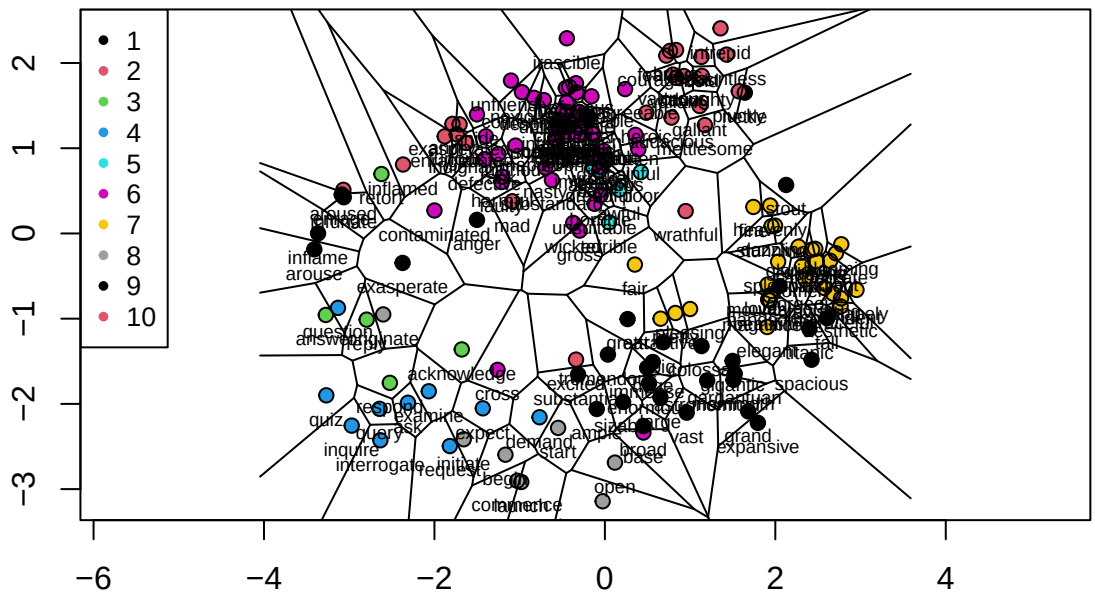
```
##
## Attaching package: 'Rfast'
```

```
## The following objects are masked from 'package:FNN':
##
##     knn, knn.cv
```

```
PlotPoints(fitISOMAP$points, mylabels, mycolors)
```

```
## Warning in (function (x0, y0, x1 = x0, y1 = y0, col = par("fg"), lty =
## par("lty"), : "wpoints" is not a graphical parameter
```

```
## Warning in plot.xy(xy.coords(x, y), type = type, ...): "wpoints" is not a
## graphical parameter
```

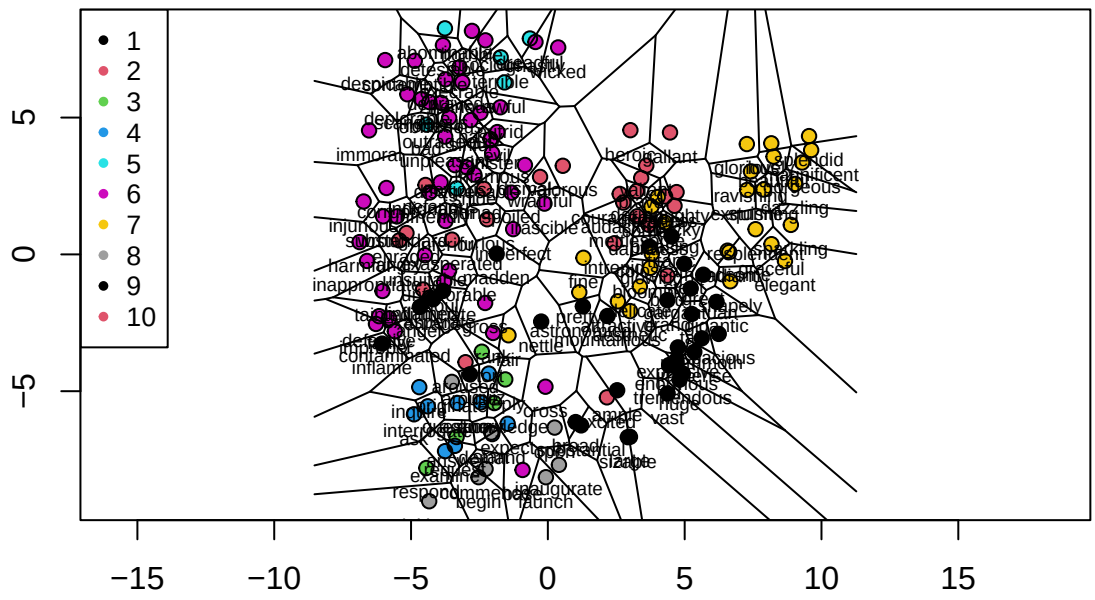


3. **Visually compare PCA, ISOMAP and t-SNE** by plotting the word2vec data, embedded into 2D using the `PlotPoints` function. Try finding the optimal k value for ISOMAP's nearest neighbour.

```
# Principal Component Analysis
fitPCA <- prcomp(mydata, center = TRUE, scale. = TRUE)
PlotPoints(fitPCA$x[, c(1, 2)], mylabels, mycolors)
```

```
## Warning in (function (x0, y0, x1 = x0, y1 = y0, col = par("fg"), lty =
## par("lty"), : "wpoints" is not a graphical parameter
```

```
## Warning in plot.xy(xy.coords(x, y), type = type, ...): "wpoints" is not a
## graphical parameter
```



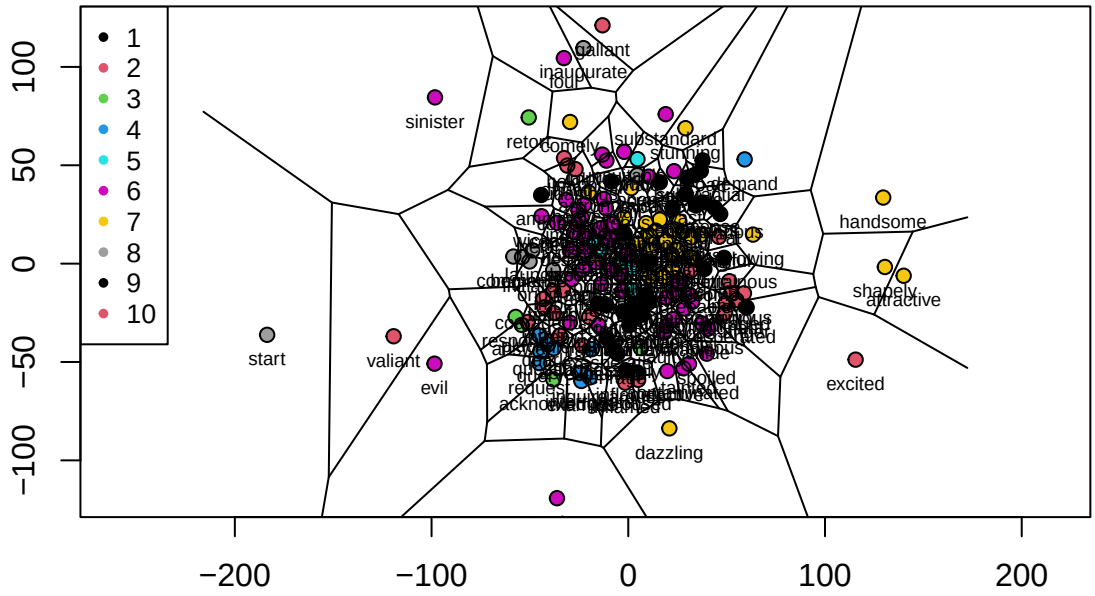

```
# t-SNE (T-Distributed Stochastic Neighbor Embedding)
# install.packages("tsne")
library(tsne)
fittsne <- tsne(mydata, k = 2)
```

```
## sigma summary: Min. : 0.524057890867544 |1st Qu. : 0.711912188370555 |Median : 0.813806386509011 |Mea
## Epoch: Iteration #100 error is: 15.327115523185
## Epoch: Iteration #200 error is: 0.790669467620458
## Epoch: Iteration #300 error is: 0.709845939337804
## Epoch: Iteration #400 error is: 0.642092256765428
## Epoch: Iteration #500 error is: 0.640737197056907
## Epoch: Iteration #600 error is: 0.622136977708454
## Epoch: Iteration #700 error is: 0.646337259118739
## Epoch: Iteration #800 error is: 0.692268564806337
## Epoch: Iteration #900 error is: 0.681115847943609
## Epoch: Iteration #1000 error is: 1.1920887396962
```

```
PlotPoints(fittsne, mylabels, mycolors)
```

```
## Warning in (function (x0, y0, x1 = x0, y1 = y0, col = par("fg"), lty =
## par("lty"), : "wpoints" is not a graphical parameter
```

```
## Warning in (function (x0, y0, x1 = x0, y1 = y0, col = par("fg"), lty =
## par("lty"), : "wpoints" is not a graphical parameter
```



All methods were able to *somehow* separate the given words and bring members of the similar synonym classes closer together.

One thing we can note is that some words are problematic for all the methods. These include especially ambivalent words from the *black* synonym class.

The results provided by Isomap seem to be visually the best. Synonym classes that have significantly different meanings and connotations are pushed far apart.

Second to the best are results provided by t-SNE. Some synonym classes are quite nicely separated (such as *blue* and *green* and *gray*).

Finally, PCA does not provide such good results. A lot of classes are mixed together. This will make the classification much more difficult.

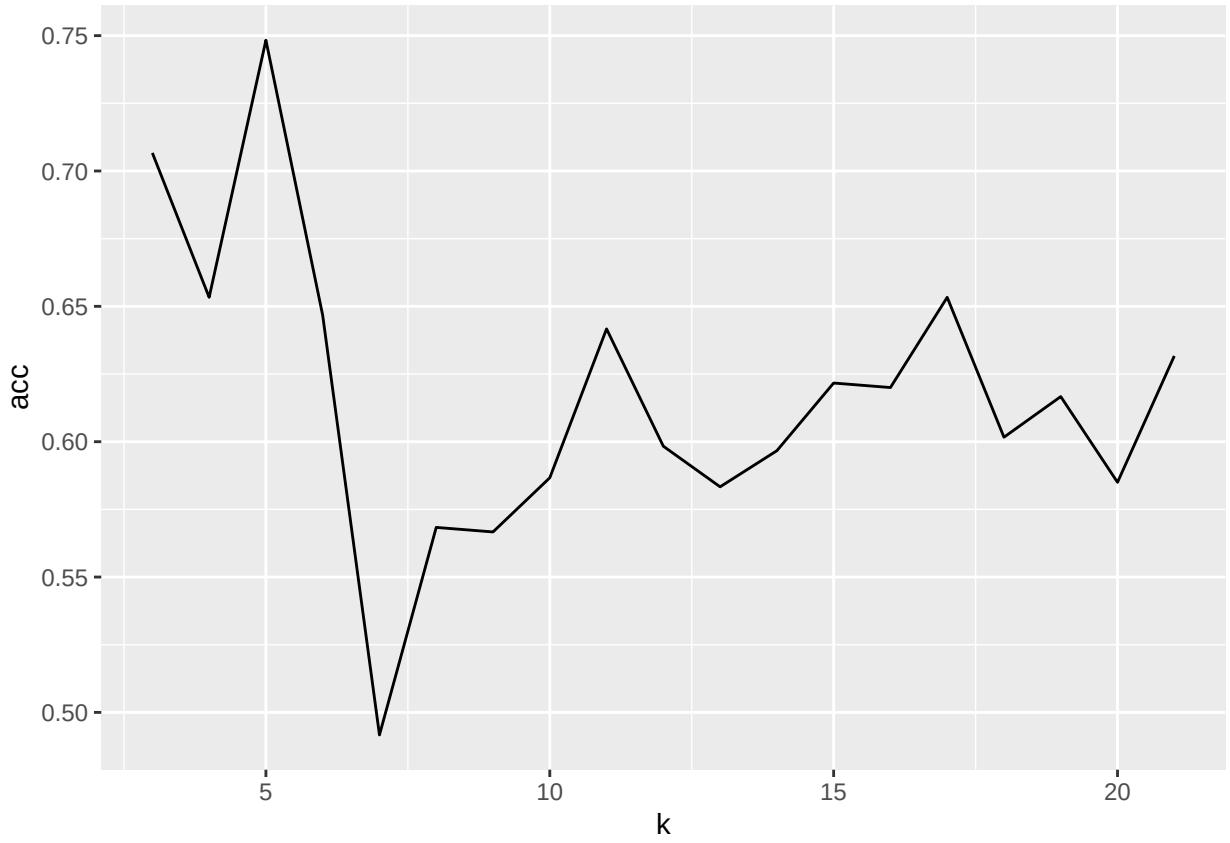
Next, we try finding the optimal k value for ISOMAP's nearest neighbor experimentally. We run our `MyIsomap2D` with different `knn.k` values ranging from 3 to 21. For each k, we run CV and calculate mean ACC. Finally, we plot the results.

```
k_data <- data.frame(
    k = 3:21, # try k from 3 to 21
    acc = 0 # initialize ACC to zero for all k
)

for (i in seq(dim(k_data)[1])) {
    k <- k_data$k[i]
```

```
    fit <- MyIsomap2D(data = mydata, knn.k = k)
    k_data$acc[i] <- mean(Classify(as.data.frame(fit$points), mycolors$V1, 50))
}

library(ggplot2)
ggplot(k_data, aes(x = k, y = acc)) +
    geom_line()
```



From the obtained results it seems that $\mathbf{k = 5}$ is the optimal value that produces the highest mean ACC.

Moreover, when we try the same process visually (observing the `PlotPoints` output for different $\mathbf{k}$), we can also see that $\mathbf{k = 5}$ yields the best result visually.

4. **Observe the effect of dimensionality reduction on a classification algorithm**. The supporting code in a function `Classify` performs training and testing of classification trees and gives the classification accuracy (percentage of correctly classified samples) as its result. Compare the accuracy of prediction on plain data, PCA, ISOMAP and t-SNE.

```
# classify plain data (using all original 300 dimensions)
accPlain <- Classify(mydata, mycolors$V1, 50)

# classify ISOMAP dim-reduced data (2 dimensions)
accISOMAP <- Classify(as.data.frame(fitISOMAP$points), mycolors$V1, 50)

# classify PCA dim-reduced data (2 dimensions)

# Note:
```

```
#   In the **original code (in the provided template)**, ALL dimensions of `fitPCA$x` were used as inpu
#   instead of just the 2 with the highest proportion of variance like it was done when comparing metho
#   in the previous step. We want to observe the effect of dimensionality reduction on a classification
#   when using different methods, so it only makes sense to consider the same number of dimensions.
#   Additionally, dimension of `fitPCA$x` is not 300 but actually just 165, because of how PCA works
#   (there cannot be more principal components than the number of samples).
accPCA <- Classify(as.data.frame(fitPCA$x[, c(1, 2)]), mycolors$V1, 50)
accPCA165 <- Classify(as.data.frame(fitPCA$x), mycolors$V1, 50)

# classify t-SNE dim-reduced data (2 dimensions)
accTSNE <- Classify(as.data.frame(fittsne), mycolors$V1, 50)

# print results
print(paste("plain 300D ACC:", mean(accPlain)))
```

```
## [1] "plain 300D ACC: 0.43"
```

```
print(paste("ISOMAP 2D ACC:", mean(accISOMAP)))
```

```
## [1] "ISOMAP 2D ACC: 0.748333333333333"
```

```
print(paste("PCA 2D ACC:", mean(accPCA)))
```

```
## [1] "PCA 2D ACC: 0.57"
```

```
print(paste("PCA 165D ACC:", mean(accPCA165)))
```

```
## [1] "PCA 165D ACC: 0.743333333333333"
```

```
print(paste("t-SNE 2D ACC:", mean(accTSNE)))
```

```
## [1] "t-SNE 2D ACC: 0.596666666666667"
```

From the mean accuracy values that we obtained trough cross-validation, we see that the classification model trained on the 2D data performed by far the best (mean accuracy of 74.8 %).

We can also clearly see the advantage of dimensionality reduction. The classification model trained on the data with all 300 dimensions (`plain 300D`) performed worse. This was expected as with high-dimensionality data we face the curse of dimensionality. We would need much more samples in order to be able to train a good classification model.

Next, the model trained on t-SNE 2D data has 63.3 % mean accuracy.

Lastly, model trained on PCA 2D data has 57.0 % mean accuracy.

Note: In the **original code (in the provided template)**, **ALL** dimensions of `fitPCA$x` were used as input for `Classify` instead of just the 2 with the highest proportion of variance like it was done when comparing methods visually in the previous step. We want to observe the effect of dimensionality reduction on a classification algorithm when using different methods, so it only makes sense to consider the same number of dimensions. Additionally, dimension of `fitPCA$x` is not 300 but actually just 165, because of how PCA works (there cannot be more principal components than the number of samples).