Dimension Reduction

D2K Course Staff

2/15/2019

Introduction

Often when we are working with big data sets, we may have significantly more information than we need. Whether that is redundant information or just uninformative data, it can be helpful for us to reduce the dimensions of the original data set. This can provide many advantages such as visualization, increased density of important information in your variables, and in some cases increased predictive ability when performing regression or classification tasks.

There are several methods that allow us to reduce the dimension of our original dataset. We will cover a handful of them in this module to understand what advantages and disadvantages different methods have for analyzing data.

```r
# Load libraries
library(dplyr)
#library(devtools)
#install_github("vqv/ggbiplot") # install from source if necessary
library(ggbiplot)
library(NMF)
library(Rtsne)
```

Principal Component Analysis (PCA)

PCA is one of the most widely used dimension reduction techniques for exploratory analysis, visualization, and pattern recognition. The objective of PCA is to find the low-dimensional representation of the data that captures the most variance in the data (i.e. the best low-dimensional representation of the data). In practice, this means determining the linear combinations of our features that preserve the most variance, or in other words, information gained. We often have this objective given a situation where our data has large p (recall our data is defined as a matrix $X_{n \times p}$ with n observations and p features) or when we wish to understand the underlying patterns in the data.

We calculate the *first* principal component as the linear combination of variables that maximizes the variance. Mathematically, we can describe is as the following:

$$\underset{v}{\text{maximize}} \ \text{Var}(Xv) \text{ subject to } ||v||_2 = 1$$

$$\underset{v}{\text{maximize}} \ v^T \text{Var}(X) v \text{ subject to } ||v||_2 = 1$$

$$\underset{v}{\text{maximize}} \ v^T \Sigma v \text{ subject to } ||v||_2 = 1$$

where $\Sigma = Cov(X)$.

We then calculate the next k principal components by finding the subsequent linear combinations that are orthogonal to the previous combinations. Mathematically, we describe it as:

$$\underset{v_k}{\text{maximize}} \ v_k^T \Sigma v_k \text{ subject to } ||v_k||_2 = 1 \ \& \ v_k^T v_j = 0 \ \forall j < k$$

We can arrive at the solution for the above optimization problem either through eigenvalue decomposition of the covariance matrix or singular value decomposition of the data matrix. We show the implementation of this solution in R below on the *mtcars* data set. Since PCA is used for continuous data, we will only select the numeric variables for our analyses.

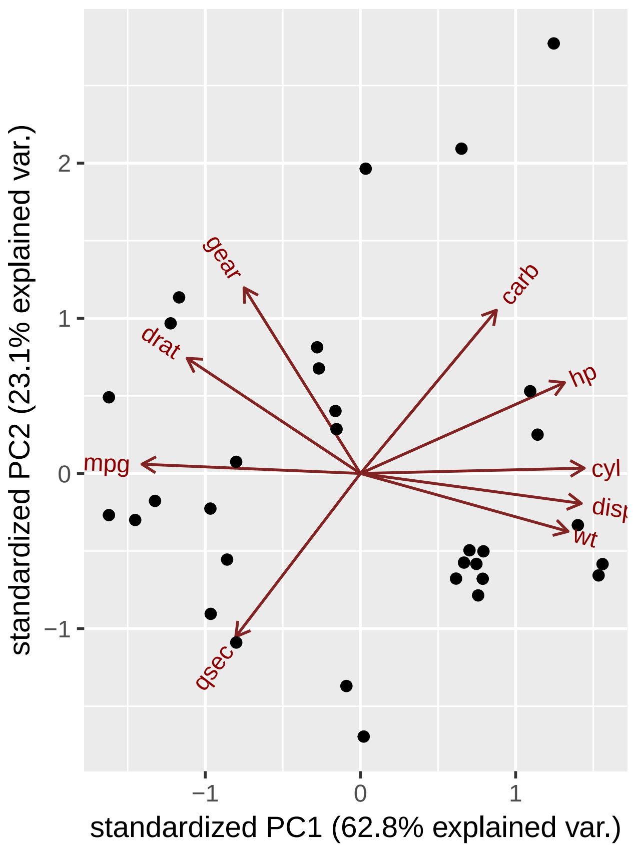
```
# Load data
data(mtcars)
head(mtcars)
```

```
##                    mpg cyl disp  hp drat    wt  qsec vs am gear carb
## Mazda RX4         21.0   6  160 110 3.90 2.620 16.46  0  1    4    4
## Mazda RX4 Wag     21.0   6  160 110 3.90 2.875 17.02  0  1    4    4
## Datsun 710        22.8   4  108  93 3.85 2.320 18.61  1  1    4    1
## Hornet 4 Drive    21.4   6  258 110 3.08 3.215 19.44  1  0    3    1
## Hornet Sportabout 18.7   8  360 175 3.15 3.440 17.02  0  0    3    2
## Valiant           18.1   6  225 105 2.76 3.460 20.22  1  0    3    1
```

```
# Select numeric variables
mtcars_x <- mtcars %>%
  select("mpg", "cyl", "disp", "hp", "drat", "wt", "qsec", "gear", "carb")
```

Note: one should always center features before performing PCA. It is also a good idea to scale features if they are measured differently. **Don't scale if features are measured in the same way!**

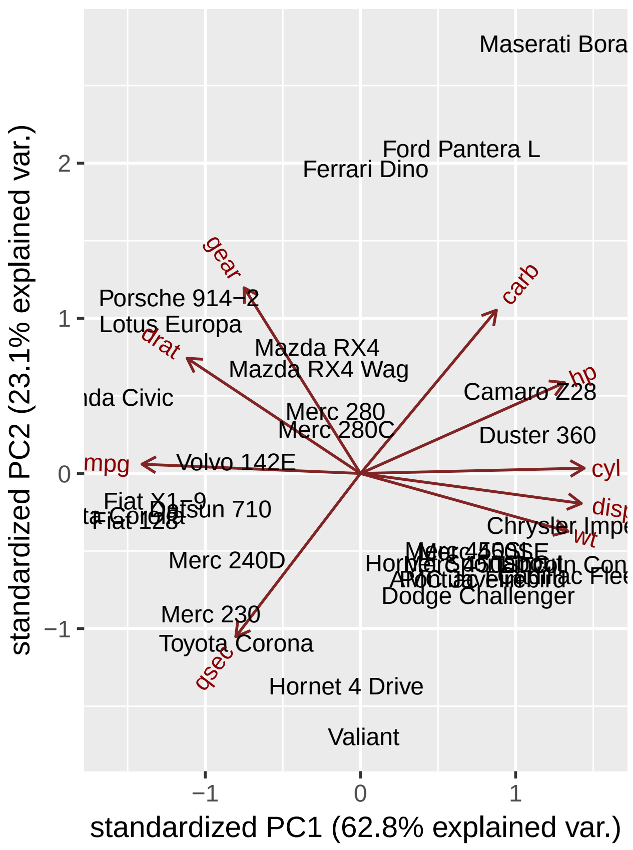
```
# Set seed for reproducibility
set.seed(123)

# Center and scale data
mtcars_x <- scale(mtcars_x, center = TRUE, scale = TRUE)

# Perform PCA with built-in function
mtcars_pca <- prcomp(mtcars_x)

# Visualize 1st and 2nd principal components
par(mfrow=c(1,1))
ggbiplot(mtcars_pca)
```



The above visualization shows how each variable contributes to the first and second principal components of the data. You can see that *cyl*, *disp*, and *wt* contribute significantly to the first principal component in the positive direction, and that *mpg* contributes significantly to the first principal component in the negative direction.

We can come to understand the underlying structure of the data more when we add labels to the data points. We replace each data point with the name of the car model and again visualize the first and second principal components.

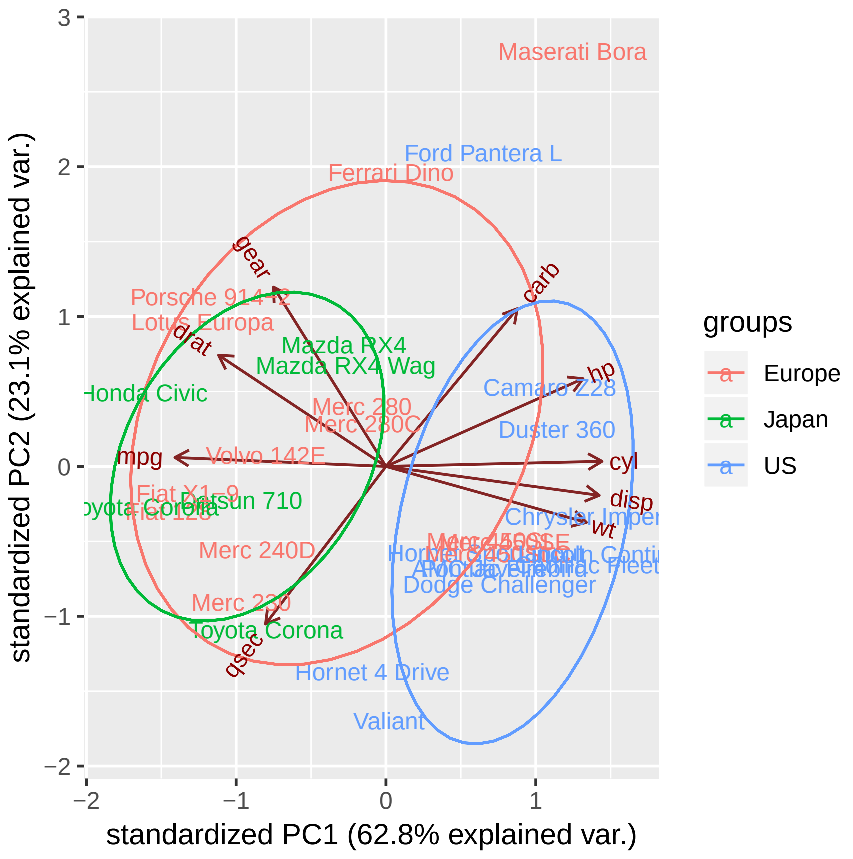
```
# Visualize 1st and 2nd principal components with labels
ggbiplot(mtcars_pca, labels = rownames(mtcars_x))
```



You can see patterns start to arise, like the Maserati Bora, Ford Pantera L, and Ferrari Dino all clustered together. This makes sense all of these models are sports cars that have high values of the features *carb* and *gear*.

Additionally, we can use group labels to understand underlying patterns in the data. We look to see if any clusters arise based on the place of manufacturing of the different model cars.

```
# Define place of manufacturing
mtcars_country <- c(rep("Japan", 3), rep("US",4), rep("Europe", 7),rep("US",3), "Europe", rep("Japan", 3

# Visualize 1st and 2nd principal components
ggbiplot(mtcars_pca, ellipse = TRUE, labels = rownames(mtcars_x), groups = mtcars_country)
```



Very interesting! We see some clear clusters arise based on the place of manufacturing of the model vehicle. This shows us that cars manufactured in the same places have similar characteristics.

As previously stated, PCA allows us to accomplish many different objectives that are common during the exploratory analysis phase of a data science project. It allows us visualize multivariate data sets in low-dimensional space easily, it allows us to generate new features that can increase our predictive ability, and it allows us to determine underlying patterns in our data. If you are working with numeric variables, PCA should be one of the first techniques you try in the exploratory phase.

Advantages

- Unique and global solution
- Ordered and orthognal components
- Best low-rank approximation of the data
- Yields best linear dimension reduction possible
- **Visualization!!!**

Disadvantages

- Computationally expensive for large square matrices
- Doesn't identify non-linear patterns
- Doesn't perform well in ultra high-dimensional spaces

Variance Explained

You may have noticed before on the plot above that next to each axis label is the "% explained variance". As explained earlier, PCA finds the linear combinations of the features that maximizes the variance. Thus, each principal component can be thought of as "explaining" a certain percentage of the overall variance in the data. We can make the following calculations:

Variance explained by the kth principal component:

$$d_k^2 = v_k^T X^T X v_k$$

Total variance of the data:

$$\Sigma_{k=1}^n d_k^2$$

Proportion of variance explained by kth principal component:

$$\frac{d_k^2}{\Sigma_{k=1}^n d_k^2}$$

Cumulative variance explained by first r principal components:

$$\frac{\Sigma_{k=1}^r d_k^2}{\Sigma_{k=1}^n d_k^2}$$

Non-Negative Matrix Factorization (NMF)

Just like PCA, Non-Negative Matrix Factorization is used to reduce the dimension of a dataset, visualize high-dimensional data, and find underlying patterns in the original data. It also uses linear dimension reduction (LDR) techniques in order to represent the data in a lower-dimension. In other words, we are trying to compute r basis elements such that the linear space spanned by the basis approximate the data points as closely as possible. All LDR techniques approximate the data matrix X with a low-rank matrix WH where $X \approx WH$, or in vector notation

$$x_j \approx \Sigma_{k=1}^r w_k h_j(k) \text{ for some weights } h_j \in \mathbb{R}^{\mathrm{r}}$$

where,

- the matrix $X \in \mathbb{R}^{\mathrm{p}\times\mathrm{n}}$ where each column is a data point, that is, $X(:,j) = x_j$ for $1 \leq j \leq n$
- the matrix $W \in \mathbb{R}^{\mathrm{p}\times\mathrm{r}}$ where each column is a basis element, that is, $W(:,k) = w_k$ for $1 \leq k \leq r$
- the matrix $H \in \mathbb{R}^{\mathrm{r}\times\mathrm{n}}$ where each column of H gives the coordinates of a data point $X(:,j)$ in the basis W, that is, $H(:,j) = hj$ for $1 \leq j \leq n$

Essentially, we are representing the original data that is in p-dimensions in an r-dimensional linear subspace. The subspace is spanned by the basis elements w_k's whose coordinates are given by the vectors h_j 's. This is what allows us to express the data in a *lower dimension.*

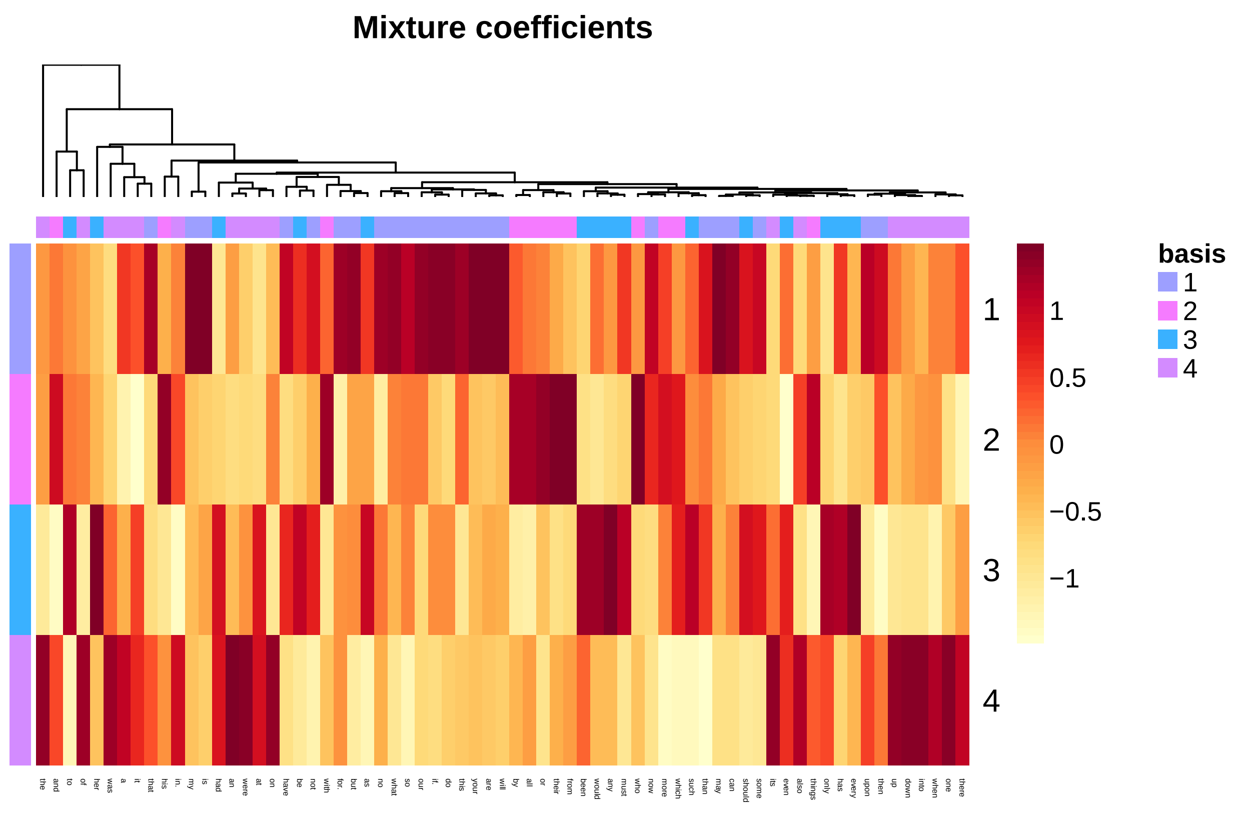
Unlike NMF, PCA makes no assumptions about W or H. However, NMF assumes that all of the basis elements w_k's and the weights h_j's are non-negative component-wise. Thus, it is a very useful technique if you are working with non-negative data like pixel values or text data (which we will explore in this example). It allows us to automatically extract sparse features that are easy to interpret. In order to compute the linear dimension reduction, we solve the following optimization problem:

$$\underset{W\in\mathbb{R}^{p\times r}, H\in\mathbb{R}^{r\times n}}{\text{minimize}} ||X - WH||_F^2 \text{ such that } W \geq 0 \text{ and } H \geq 0$$

Unfortunately, unlike PCA, we cannot solve the optimization problem with singular value decomposition due to the constraints. There are, however, several solutions that we will not cover in this module. Moreover, our solution depends on the rank k of matrix W that we choose. We must carefully consider what value we choose based on what our objective is since changing k can fundamentally change factors. We strongly suggest looking at additional literature on rank selection for NMF if this is a method you are trying to apply.

We now show how to implement NMF on author text data that we are providing for this module from the `authorship.csv` file. The dataset contains word counts for several books by a handful of authors. We can see if we can see any clear patterns arise when performing NMF on this data set. Furthermore, since we know there are four different authors in the data set we wish to analyze, we choose $k = 4$ when performing NMF. We can plot a heatmap of the consensus matrix derived earlier in order to see if any clusters of features arise and any patterns become apparent using hierarchical clustering.

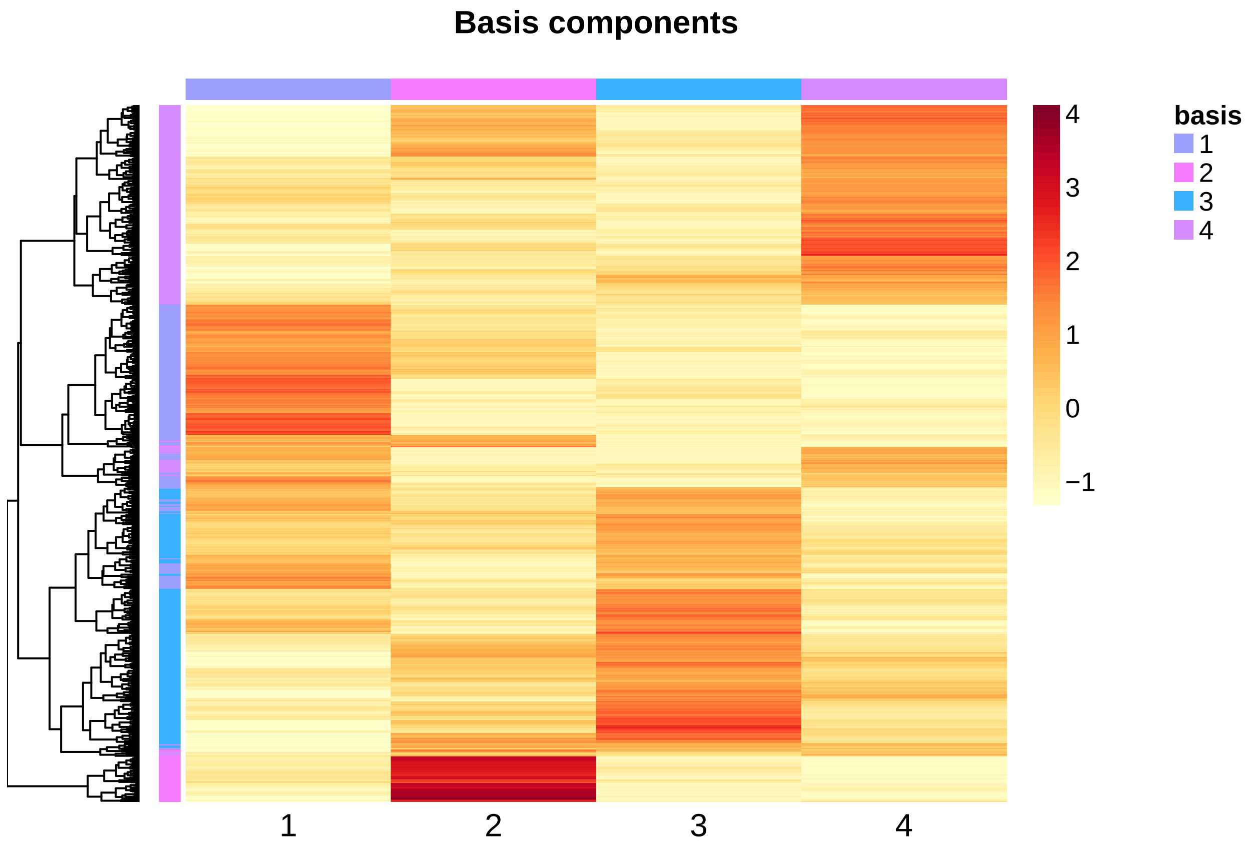
```r
# Load author text data
author_df <- read.csv("data/authorship.csv", header = TRUE, stringsAsFactors = FALSE)

# Separate data into features and labels
author_x <- as.matrix(author_df[,-c(70:71)])
author_labels <- as.factor(author_df[,71])

# Create matrix for analysis
X <- author_x
rownames(X) <- author_labels

# NMF
K <- 4
nmffit <- nmf(X,rank=K)
W <- basis(nmffit)
H <- coef(nmffit)

# Visualize coefficient matrix
coefmap(nmffit, scale="col", legend=TRUE)
```



We can see how each feature contributes to the low-dimensional representation of the data with this heatmap. Darker colored cells in the heat map correspond to greater values in coefficient matrix for that particular basis vector. This allows us to see which features should possibly be considered together in future analyses.

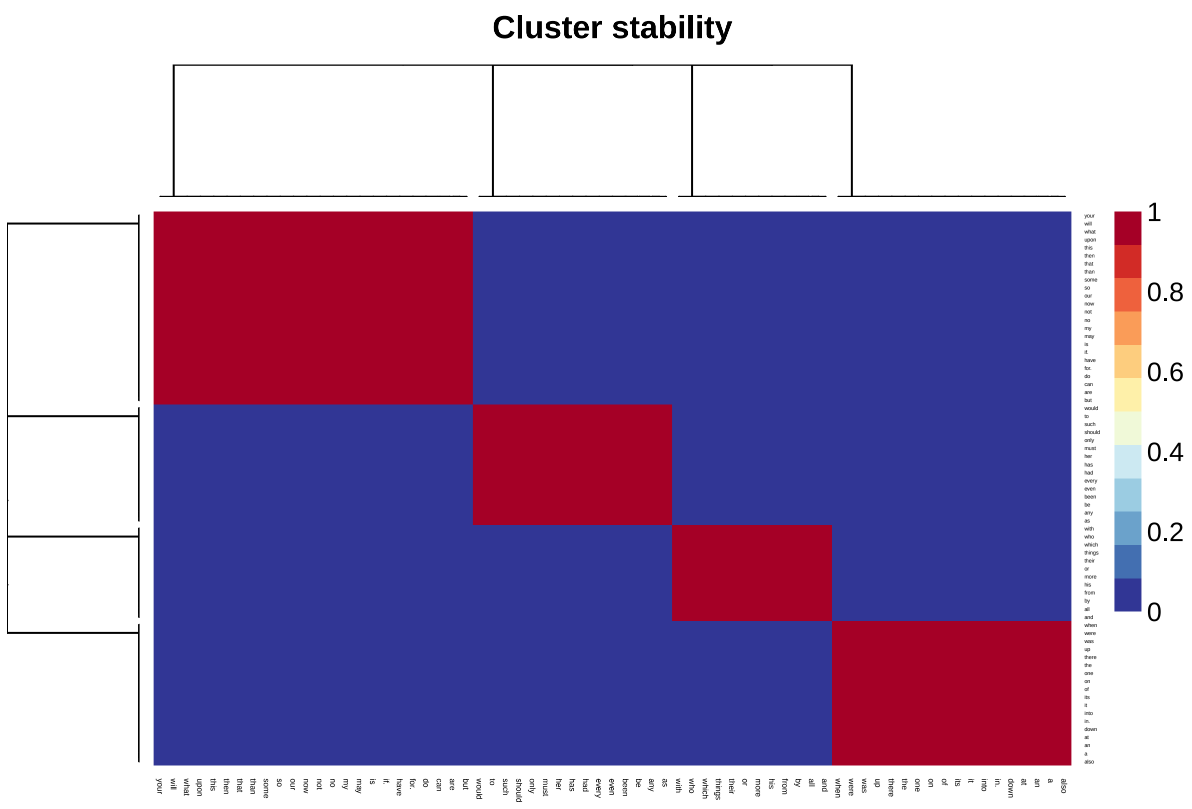
We can also visualize the basis matrix on a heatmap with hierarchical clustering as well.

```
# Visualize basis matrix
basismap(nmffit, scale="col", legend=TRUE)
```



And finally we can visualize the consensus matrix over multiple independent NMF runs with a heatmap to understand the stability of the clusters we previously observed.

```
consensusmap(nmffit, main='Cluster stability')
```



By doing this, we can see 4 distinct clusters of words from our original data set.

Multidimensional Scaling (MDS)

Unlike the first two techniques we have covered, MDS does not use the same *linear* dimension reduction framework that is used by PCA and NMF. Instead, MDS is *non-linear* and reduces the dimension of the data by representing it with the proximities between objects. In fact, you do not even need the original data if you have a matrix of similarities (or dissimilarities), $D_{n \times n}$. The objective of MDS is to find the projections (z_1,. . . ,z_K where $z \in \mathbb{R}^n$) that preserve the original distances in D in a lower dimensional space ($K << n$).

We perform MDS by solving an optimization problem where we seek to minimize a measure of distance called a *stress* or a *strain* function. There are multiple stress functions to choose from and they have different properties. We will cover a handful of commonly used ones here.

Kruskal-Shephard scaling: the most common stress function, also known as least squares scaling, is solved using the gradient descent algorithm. The idea is that it finds a low-dimensional mapping that preserves the pairwise distances as much as possible.

$$S_D(z_1, ..., z_K) = \Sigma_{i \neq i'} (d_{ii'} - ||z_i - z_{i'}||)^2$$

Sammon mapping: preserves smaller pairwise distances. Clearly, it is very similar to the Kruskal-Shephard scaling, but accounts for the size of the distance.

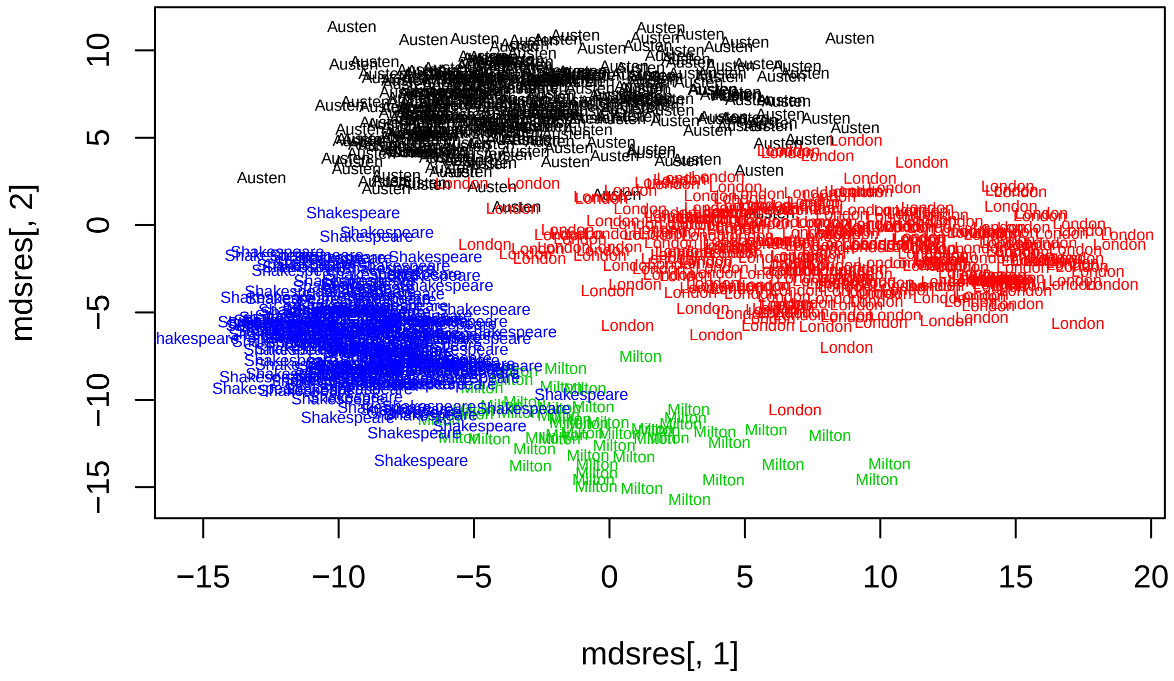
$$S_D(z_1, ..., z_K) = \frac{\Sigma_{i \neq i'} (d_{ii'} - ||z_i - z_{i'}||)^2}{d_{ii'}}$$

Shephard-Kruskal nonmetric scaling: effectively uses only ranks and not the approximated similarities. With θ, an arbitrary increasing function, me minimize over z_i by gradient descent. Then, with z_i fixed, we use monotonic regression (don't worry if you don't know what this is) to find the best monotonic approximation $\theta(d_{ii'})$ to $||z_i - z_{i'}||$. These two steps are iterated until we converge on a solution.

$$S_D(z_1, ..., z_K) = \frac{\Sigma_{i \neq i'}[||z_i - z_{i'}|| - \theta(d_{ii'})]^2}{\Sigma_{i \neq i'}||z_i - z_{i'}||^2}$$

One of the advantages of MDS is that it is extremely helpful for visualizing data in a lower dimension, especially when we are interested in identifying how similar certain groups are in our data. We will again use the authorship data to show how to perform MDS in R. Recall, we must first calculate the distance matrix from the original data, luckily this is quite easy in R. We elect to define our dissimilarities using the Canberra distance metric, however, there are many choices that may be worth exploring.

```r
# Compute distance matrix
Dmat <- dist(X, method="canberra")

# MDS
mdsres <- cmdscale(Dmat)

# Visualize proximities of our projections from MDS
par(mfrow=c(1,1))
plot(mdsres[,1],mdsres[,2],type="n")
text(mdsres[,1],mdsres[,2],rownames(X),col=as.numeric(author_labels),cex=.5)
```



We can see nice clear clusters arise from our projections using MDS. It is important to try different distance measures in order to see what allows you to identify the most distinct patterns.

t-Distributed Stochastic Neighbor Embedding (t-SNE)

A more modern method that has gained recent popularity as a dimension reduction technique is t-SNE. Compared to PCA, it often performs better in identifying clusters and is more helpful in exploratory analysis since it weighs data points that are closer together more heavily than linear methods. The key distinction between t-SNE and linear methods is that it takes the Euclidean distance in high-dimensional space, and then maps it to a conditional normal probability function to represent the distances, or similarities.

We can define the conditional probability $p_{j|i}$ as the similarity of data point x_j to x_i, or the probability that x_i would pick x_j as its neighbor. This probability is determined by a normal distribution which means that for nearby data points, the similarities are very high, but for highly separated data points this probability becomes extremely small. We can represent the conditional probability as,

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

where σ_i^2 is the variance of the normal distribution defined for x_i.

We can apply a similar logic to our lower-dimensional features to define a conditional probability metric for similarity between the lower-dimensional data points. However, we define the similarity metric using the Student's t-distribution with one degree of freedom as it allows us to model a moderate distance in the original data to a much larger distance in the lower-dimensional mapping. This allows us to eliminate the attractive forces between moderately dissimilar data points and thus achieve better separation for pattern recognition. We can represent this joint probability as,

$$q_{ij} = \frac{(1 + ||y_i - y_j||^2)^{-1}}{\Sigma_{k \neq i}(1 + ||y_i - y_k||^2)^{-1}}$$

where we there is only one degree of freedom.

We also define the cost function, or the objective equation we are seeking to minimize, is the sum of the Kullback-Liebler divergences. Mathematically, it can be represented as,

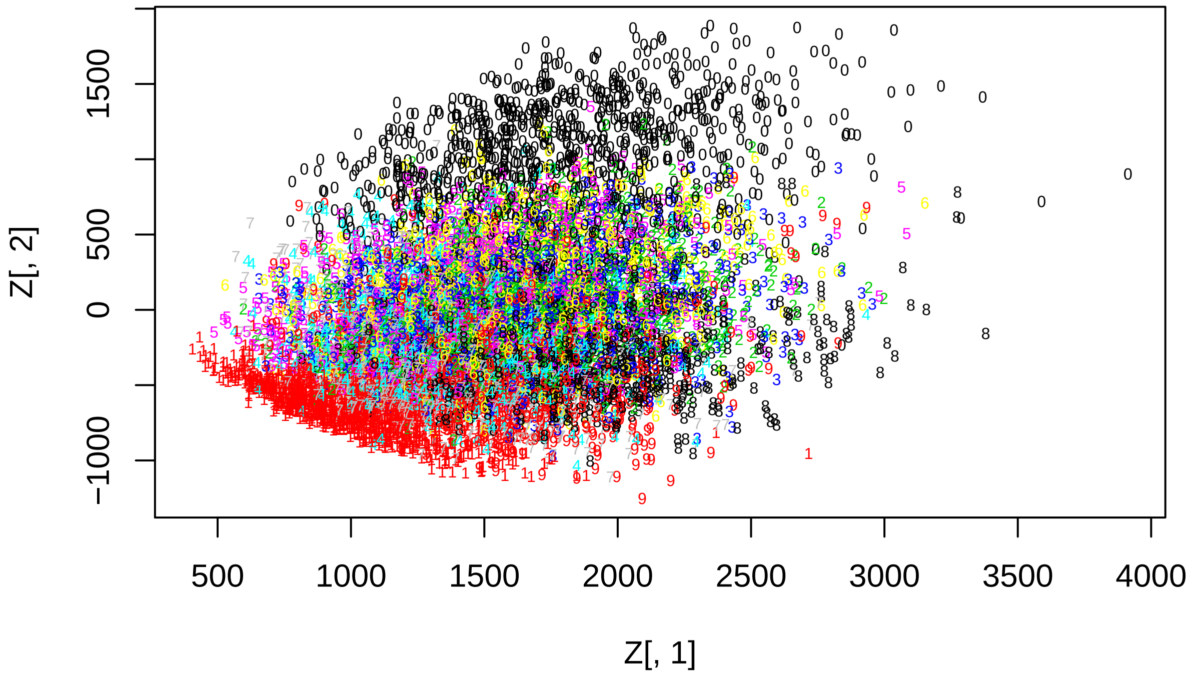
$$C = KL(P||Q) = \Sigma_i \Sigma_j p_{ij} log \frac{p_{ij}}{q_{ij}}$$

where we define the joint probability p_{ij} as the symmetrized conditional probabilities, $p_{ij} = \frac{p_{j|i} + p_{ij}}{2n}$. We solve this optimization problem using the gradient descent algorithm. Using normal distributions in the higher-dimensional space, and Student's t-distributions in the lower-dimensional space allows us to avoid the "crowding" problem that occurs in techniques like MDS.

We will now show an implementation of t-SNE in R. We use a dataset filled with pixel values to highlight how t-SNE performs better than techniques like PCA for visualization. First we show how PCA performs on the data set.

```r
# Load data
pixels <- read.csv("data/train.csv", header = TRUE, stringsAsFactors = FALSE)

# Create data matrix of numeric values
pixels_x <- as.matrix(pixels[,-1])

# Create vector of labels
pixel_labels <- as.factor(pixels[,1])

# Set seed for reproducibility
set.seed(123)

# PCA using SVD
X <- pixels_x
rownames(X) <- pixel_labels
sv <- svd(X);
```

```
V <- sv$v
Z <- X%*%V;

# Visualize PCA
par(mfrow=c(1,1))
plot(Z[,1],Z[,2],type="n")
text(Z[,1],Z[,2],rownames(X),col=as.numeric(pixel_labels),cex=.5)
```

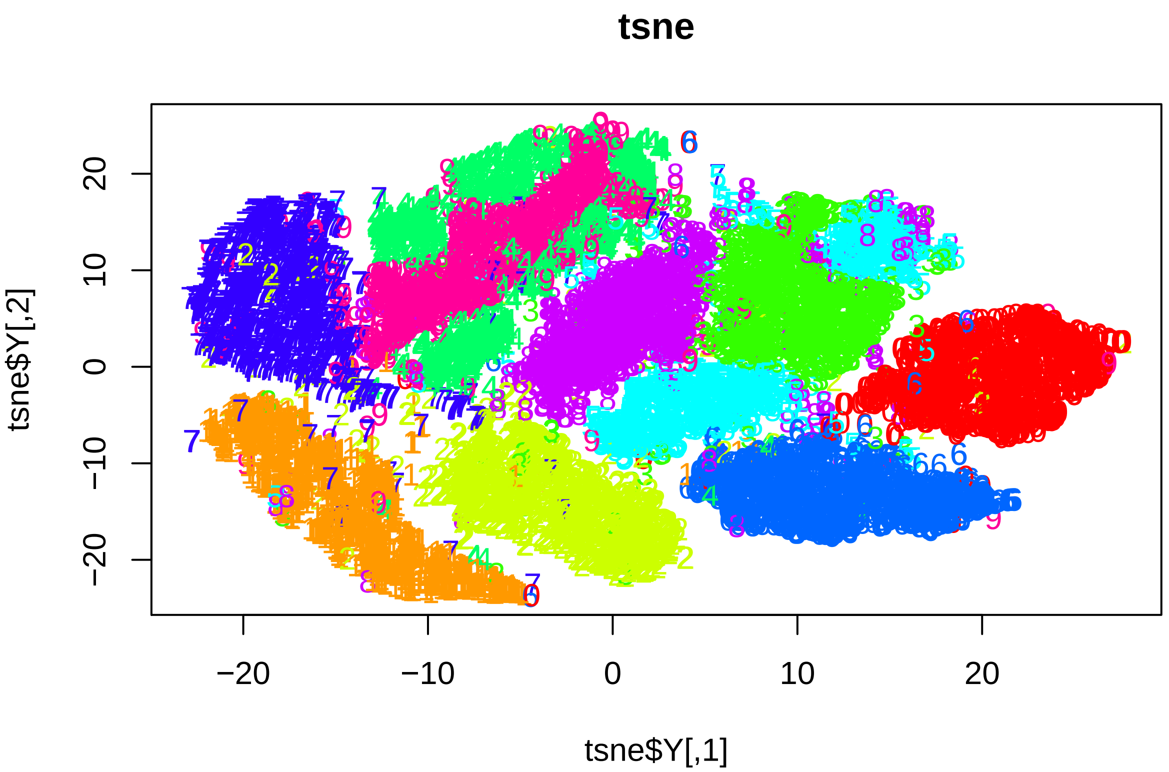


As you can see, some patterns start to arise. However, we clearly have a “crowding” since clusters of data are not clearly separated and there is much overlap between the clusters. We can see if t-SNE performs any better on the data and if it avoids the “crowding” problem.

```
# t-SNE
tsne <- Rtsne(pixels_x, dims = 2, perplexity=30, verbose=TRUE, max_iter = 500)
```

```
## Performing PCA
## Read the 10000 x 50 data matrix successfully!
## OpenMP is working. 1 threads.
## Using no_dims = 2, perplexity = 30.000000, and theta = 0.500000
## Computing input similarities...
## Building tree...
##  - point 10000 of 10000
## Done in 12.74 seconds (sparsity = 0.012259)!
## Learning embedding...
## Iteration 50: error is 97.574505 (50 iterations in 2.59 seconds)
## Iteration 100: error is 90.572766 (50 iterations in 4.24 seconds)
## Iteration 150: error is 86.514037 (50 iterations in 2.84 seconds)
## Iteration 200: error is 86.201899 (50 iterations in 3.00 seconds)
## Iteration 250: error is 86.160226 (50 iterations in 3.15 seconds)
## Iteration 300: error is 3.140585 (50 iterations in 2.44 seconds)
## Iteration 350: error is 2.744775 (50 iterations in 2.18 seconds)
## Iteration 400: error is 2.528705 (50 iterations in 2.43 seconds)
## Iteration 450: error is 2.386856 (50 iterations in 2.38 seconds)
## Iteration 500: error is 2.284919 (50 iterations in 2.83 seconds)
## Fitting performed in 28.09 seconds.
```

```
# Create color palette for visualization
colors <- rainbow(length(unique(pixel_labels)))
names(colors) <- unique(pixel_labels)

## Plot t-SNE
par(mfrow=c(1,1))
plot(tsne$Y, t='n', main="tsne")
text(tsne$Y, labels=pixel_labels, col=colors[pixel_labels])
```



Look how well t-SNE separates the data into distinct clusters compared to PCA! Generally, it is best practice to try both PCA and t-SNE, along with other methods, when trying to perform dimension reduction techniques for visualizaiton and exploration.

References

Gaujoux, Renaud. "Generating Heatmaps for Nonnegative Matrix Factorization." CRAN, 6 Feb. 2018, cran.r-project.org/web/packages/NMF/vignettes/heatmaps.pdf.

Gillis, Nicolas. "The Why and How of Nonnegative Matrix Factorization." ArXiv, Department of Mathematics and Operational Research Facult´e Polytechnique, Universit´e De Mons, 7 Mar. 2014, arxiv.org/pdf/1401.5226.pdf.

Hastie, Trevor, et al. The Elements of Statistical Learning Data Mining, Inference, and Prediction. Springer, 2009.

"PCA Analysis in R." DataCamp Community, www.datacamp.com/community/tutorials/pca-analysis-r.

Saurabh, and University of California. "Comprehensive Guide on t-SNE Algorithm with Implementation in R

& Python." Analytics Vidhya, 22 Mar. 2017, www.analyticsvidhya.com/blog/2017/01/t-sne-implementation-r-python/.

Van der Maaten, Laurens, and Geoffrey Hinton. "Visualizing Data Using t-SNE." Journal of Machine Learning Research, vol. 9, Nov. 2008, pp. 2579–2605., lvdmaaten.github.io/publications/papers/JMLR_2008.pdf.