Dimension Reduction Techniques

W. Evan Johnson, Ph.D.
Professor, Division of Infectious Disease
Director, Center for Data Science
Associate Director, Center for Biomedical Informatics and Health AI
Rutgers University – New Jersey Medical School

2025-11-06

Dimension reduction

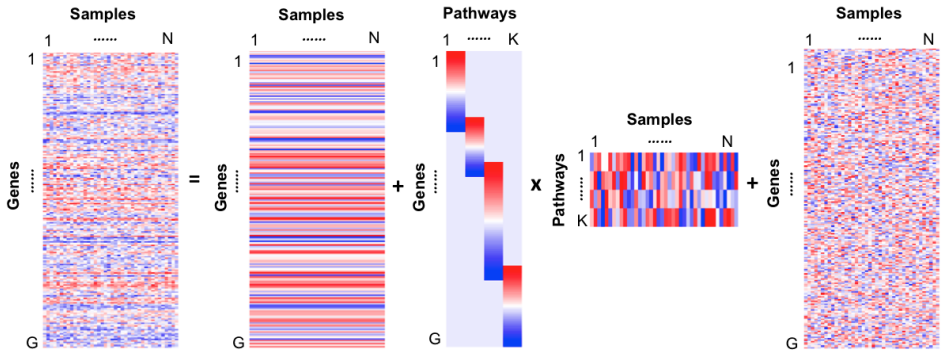
A typical machine learning challenge will include a large number of predictors, which makes visualization somewhat challenging. We have shown methods for visualizing univariate and paired data, but plots that reveal relationships between many variables are more complicated in higher dimensions.

Dimension reduction : motivation

Microbiome 16S data example: why we need dimension reduction

```
library(animalcules)
run_animalcules()
```

Dimension reduction : motivation



Dimension reduction

Here we describe powerful techniques useful for exploratory data analysis, among other things, generally referred to as **dimension reduction**.

The general idea is to reduce the dimension of the dataset while preserving important characteristics, such as the distance between features or observations.

The technique behind it all, the singular value decomposition, is also useful in other contexts. Principal component analysis (PCA) is the approach we will be showing first. Before applying PCA to high-dimensional datasets, we will motivate the ideas behind with a simple example.

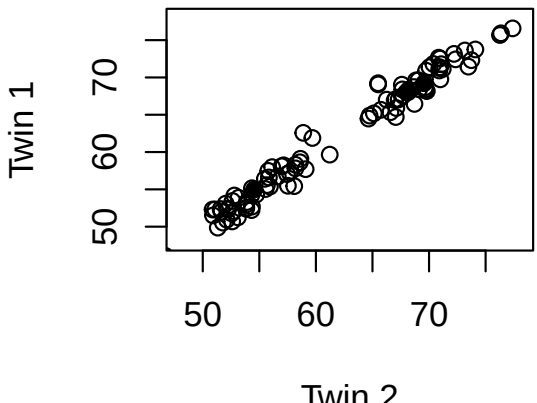
Dimension reduction: twin heights

We consider an= simulated example with twin heights (children and adults):

```
set.seed(1988)
library(MASS)
n <- 100
Sigma <- matrix(c(9, 9 * 0.9, 9 * 0.92, 9 * 1), 2, 2)
x <- rbind(mvrnorm(n / 2, c(69, 69), Sigma),
           mvrnorm(n / 2, c(55, 55), Sigma))
```

Dimension reduction: twin heights

A scatterplot reveals that the correlation is high and there are two groups of twins: adults and children:



Dimension reduction: twin heights

Our features are N two-dimensional points, the two heights, and, for illustrative purposes, we will act as if visualizing two dimensions is too challenging. We therefore want to reduce the dimensions from two to one, but still be able to understand important characteristics of the data, for example that the observations cluster into two groups: adults and children.

Dimension reduction: twin heights

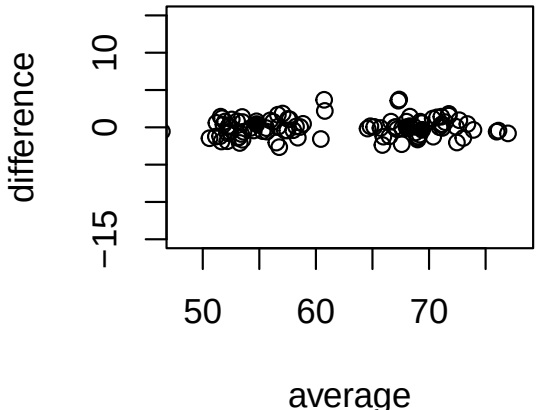
Now, can we pick a one-dimensional summary that makes this approximation even better?

If we look back at the previous scatterplot and visualize a line between any pair of points, the length of this line is the distance between the two points. These lines tend to go along the direction of the diagonal. Notice that if we instead plot the difference versus the average:

```
z  <- cbind(average=(x[,2] + x[,1])/2,
            difference=x[,2] - x[,1])
```

Dimension reduction: twin heights

We can see how the distance between points is mostly explained by the first dimension: the average.



Linear transformations

Note that each row of X was transformed using a linear transformation. For any row i, the first entry was:

$$Z_{i,1} = a_{1,1}X_{i,1} + a_{2,1}X_{i,2}$$

with $a_{1,1} = 0.5$ and $a_{2,1} = 0.5$.

The second entry was also a linear transformation:

$$Z_{i,2} = a_{1,2}X_{i,1} + a_{2,2}X_{i,2}$$

with $a_{1,2} = 1$ and $a_{2,2} = -1$.

Linear transformations

More formally, we can write the operation we just performed like this:

$$Z = XA \text{ with } A = \begin{pmatrix} 1/2 & 1 \\ 1/2 & -1 \end{pmatrix}.$$

```
A <- matrix(c(1/2,1/2,1,-1), nrow=2)
z <- x%*%A
head(z)
```

```
##           [,1]       [,2]
## [1,] 69.39688  0.3328979
## [2,] 68.01154 -0.0248272
## [3,] 65.96233  1.2495542
## [4,] 67.58309  2.2574218
```

Linear transformations

And that we can transform back by simply multiplying by A^{-1} as follows:

$$X = ZA^{-1} \text{ with } A^{-1} = \begin{pmatrix} 1 & 1 \\ 1/2 & -1/2 \end{pmatrix}.$$

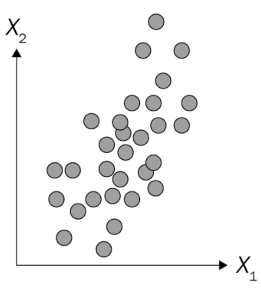
```
A <- matrix(c(1/2,1/2,1,-1), nrow=2)
x <- z%*%solve(A)
head(x)
```

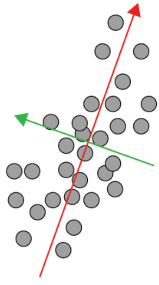
```
##           [,1]     [,2]
## [1,] 69.56333 69.23043
## [2,] 67.99913 68.02396
## [3,] 66.58711 65.33756
## [4,] 68.71180 66.45438
```

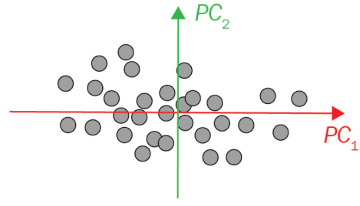
Linear transformations

Dimension reduction can often be described as applying a transformation A to a matrix X with many columns that *moves* or *rotates* the information contained in X to the first few columns of $Z = AX$, then keeping just these few informative columns, thus reducing the dimension of the vectors contained in the rows.

Principal component analysis







Data in feature space → Find principal components → Data in **p**rincipal **c**omponents space

Principal component analysis

The **first principal component (PC)** of a matrix X is the linear orthogonal transformation of X that maximizes the variability. The function `prcomp` provides this info:

```
pca <- prcomp(x)
pca
```

```
## Standard deviations (1, .., p=2):
## [1] 11.3574818  0.8811441
##
## Rotation (n x k) = (2 x 2):
##             PC1        PC2
## [1,] -0.7022354  0.7119448
## [2,] -0.7119448 -0.7022354
```

Principal component analysis

Note that the first PC is (almost) the same as the mean we used earlier and the second is (almost) the difference!

The function PCA returns both the rotation needed to transform X so that the variability of the columns is decreasing from most variable to least (accessed with `$rotation`) as well as the resulting new matrix (accessed with `$x`).

```
names(pca)
```

```
## [1] "sdev"     "rotation" "center"   "scale"    "x"
```

Principal component analysis

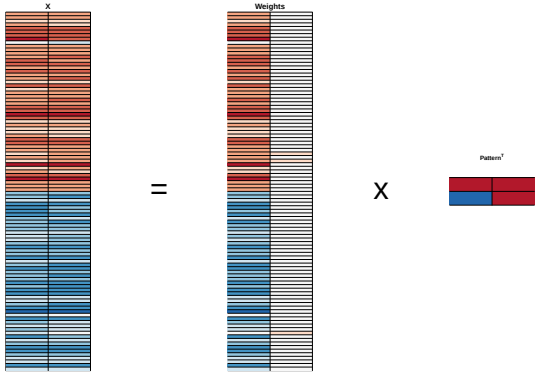
For example:

```
z  <- cbind(average=(x[,2] + x[,1])/2,
            difference=x[,2] - x[,1])
head(cbind(z, pca$x))
```

```
##        average difference        PC1         PC2
## [1,] 69.39688 -0.3328979 -10.520283  0.28543334
## [2,] 68.01154  0.0248272  -8.562911  0.01903868
## [3,] 65.96233 -1.2495542  -5.658769  0.90024455
## [4,] 67.58309 -2.2574218  -7.945915  1.62863430
## [5,] 72.67825  0.9519504 -15.166975 -0.59121000
## [6,] 71.74806  1.7963050 -13.855614 -1.19727644
```

Principal component analysis

We can visualize these to see how the components summarize the data:



Principal component analysis

It turns out that we can find this linear transformation not just for two dimensions but for matrices of any dimension p. Thus, for a matrix with X with p columns, we can find a transformation that creates Z for which the first column is the first principal component, the second column is the second principal component, and so on.

If after a certain number of columns, say k, the variances of the columns of Z_j, $j > k$ are very small, it means these dimensions have little to contribute to the distance and we can approximate distance between any two points with just k dimensions. If k is much smaller than p, then we can achieve a very efficient summary or **reduction** of our data.

Iris example

The iris data is a widely used example in data analysis courses. It includes four botanical measurements related to three flower species:

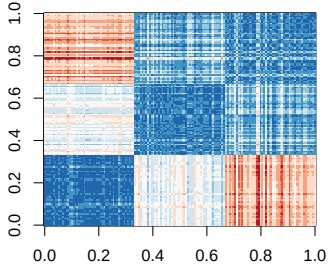
```
head(iris)
```

```
##   Sepal.Length Sepal.Width Petal.Length Petal.Width Species
## 1          5.1         3.5          1.4         0.2  setosa
## 2          4.9         3.0          1.4         0.2  setosa
## 3          4.7         3.2          1.3         0.2  setosa
## 4          4.6         3.1          1.5         0.2  setosa
## 5          5.0         3.6          1.4         0.2  setosa
## 6          5.4         3.9          1.7         0.4  setosa
```

Iris example

Let's compute the distance between each observation. You can clearly see the three species with one species very different from the other two:

```
x <- iris[,1:4] %>% as.matrix()
d <- dist(x)
image(as.matrix(d), col = rev(RColorBrewer::brewer.pal(9, "RdBu")))
```



Iris example

Our predictors here have four dimensions, but three are very correlated:

```
cor(x)
```

```
##              Sepal.Length Sepal.Width Petal.Length Petal.Width
## Sepal.Length    1.0000000  -0.1175698    0.8717538   0.8179411
## Sepal.Width    -0.1175698   1.0000000   -0.4284401  -0.3661259
## Petal.Length    0.8717538  -0.4284401    1.0000000   0.9628654
## Petal.Width     0.8179411  -0.3661259    0.9628654   1.0000000
```

If we apply PCA, we should be able to approximate this distance with just two dimensions, compressing the highly correlated dimensions.

Iris example

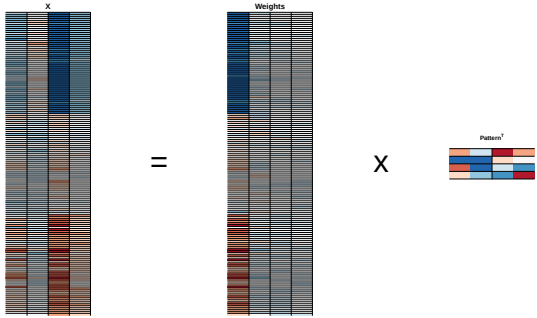
Using the `summary` function we can see the variability explained by each PC:

```
pca <- prcomp(x)
summary(pca)
```

```
## Importance of components:
##                           PC1     PC2    PC3     PC4
## Standard deviation     2.0563 0.49262 0.2797 0.15439
## Proportion of Variance 0.9246 0.05307 0.0171 0.00521
## Cumulative Proportion  0.9246 0.97769 0.9948 1.00000
```

Iris example

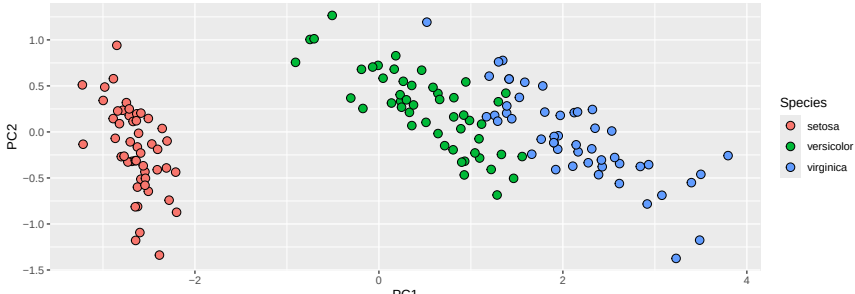
The first two dimensions account for 97% of the variability. Thus we should be able to approximate the distance very well with two dimensions. We can visualize the results of PCA:



Iris example

We plot the first two PCs with color representing the species:

```
data.frame(pca$x[,1:2], Species=iris$Species) %>%
  ggplot(aes(PC1,PC2, fill = Species))+
  geom_point(cex=3, pch=21) +
  coord_fixed(ratio = 1)
```



Non-linear transformations: UMAP

Check out the following links:

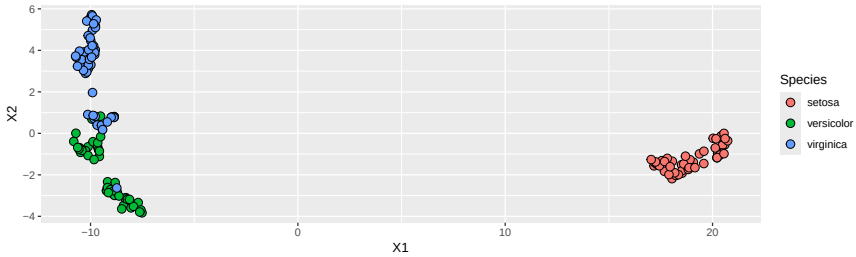
- https://pair-code.github.io/understanding-umap/
- https://pair-code.github.io/understanding-umap/supplement.html

Non-linear transformations: UMAP

The intuitions behind the core principles are actually quite simple: UMAP essentially constructs a weighted graph from the high dimensional data, with edge strength representing how “close” a given point is to another, then projects this graph down to a lower dimensionality. The advanced mathematics (topology) gives UMAP a solid footing with which to handle the challenges of doing this in high dimensions with real data.

Iris example

```
library(umap)
umap_iris <- umap(iris[,1:4])
data.frame(umap_iris$layout, Species=iris$Species) %>%
  ggplot(aes(X1,X2, fill = Species))+
  geom_point(cex=3, pch=21) +
  coord_fixed(ratio = 1)
```



Eigenvectors and Eigenvalues

An **eigenvalue** and **eigenvector** of a square matrix $\mathbf{A}$ are a scalar λ and a nonzero vector $\mathbf{x}$ so that

$$\mathbf{A}\mathbf{x} = \lambda\mathbf{x}.$$

Historical note: The prefix *eigen-* is adopted from the German word for "proper", "characteristic", "own". Eigenvalues and eigenvectors were originally used in physics to study principal axes of the rotational motion of rigid bodies, but have been found to be useful in a wide range of applications.

Eigenvectors and Eigenvalues

The eigenvalue-eigenvector equation for a square matrix can be written

$$(\mathbf{A} - \lambda\mathbf{I})\mathbf{x} = 0, \mathbf{x} \neq 0.$$

This implies that $\mathbf{A} - \lambda\mathbf{I}$ is singular and hence that

$$det(\mathbf{A} - \lambda\mathbf{I}) = 0.$$

This definition of an eigenvalue, which does not directly involve the corresponding eigenvector, is the **characteristic equation** or **characteristic polynomial** of $\mathbf{A}$.

Eigenvectors and Eigenvalues

Let $\lambda_1, \lambda_2, \ldots \lambda_n$ be the eigenvalues of a matrix $\mathbf{A}$ and denote Λ denote the n-by-n diagonal matrix with the λ_j on the diagonal. Also let $\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_n$ be a set of corresponding eigenvectors and and let $\mathbf{X}$ denote the n-by-n matrix whose jth column is $\mathbf{x}_j$.

Then note the following is true:

$$\mathbf{AX} = \mathbf{X}\Lambda.$$

Eigenvectors and Eigenvalues

Now make a key assumption that is not true for all matrices—assume that the eigenvectors are linearly independent. Then $\mathbf{X}^{-1}$ exists and

$$\mathbf{A} = \mathbf{X}\Lambda\mathbf{X}^{-1},$$

with nonsingular $\mathbf{X}$. This is known as the **eigenvalue decomposition** of the matrix $\mathbf{A}$. If it exists, it allows us to investigate the properties of $\mathbf{A}$ by analyzing the diagonal matrix Λ.

For example, matrix powers can be expressed in terms of powers of scalars:

$$\mathbf{A}^p = \mathbf{X}\Lambda^p\mathbf{X}^{-1}.$$

Eigenvectors and PCA

Principal Components Analysis (PCA) of an $N \times p$ matrix/dataset $\mathbf{Y}$ is determined by an eigenvalue decomposition on the **covariance matrix** of $\mathbf{Y}$:

$$Cov(\mathbf{Y}) = \mathbf{X}\Lambda\mathbf{X}^{-1},$$

where Λ is a diagonal matrix representing the ordered (decreasing) eigenvalues and $\mathbf{X}$ is a matrix of corresponding eigenvectors (same order as Λ). The eigenvectors in the columns of $\mathbf{X}$, provide the **rotation** or **loadings** that comprise the linear combinations of the original covariates. The eigenvalues represent the relative **variance contribution** of each of the linear combinations (eignevectors).

The rotation can then be applied to the original data to obtain the **principal components** or **PC**s, which are ordered by variance contribution (i.e., PC1 explains the most variation, etc.).

Singular value decomposition

A **Singular Value Decomposition (SVD)** is an extension of the ideas behind the eigenvalue decomposition and PCA, but factorizes **Y** directly. It is widely used in machine learning, both in practice and to understand the mathematical properties of some algorithms.

The SVD **decomposes** an $N \times p$ data matrix Y with $p < N$ as

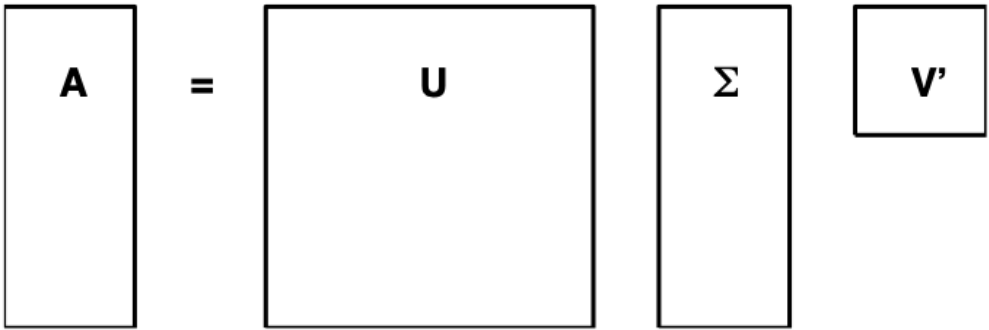
$$Y = UDV^\top$$

With U and V **orthogonal** of dimensions $N \times p$ and $p \times p$, respectively, and D a $p \times p$ **diagonal** matrix with the values of the diagonal decreasing:

$$d_{1,1} \geq d_{2,2} \geq \dots d_{p,p}.$$

Singular value decomposition

Visually, the SVD looks something like this:



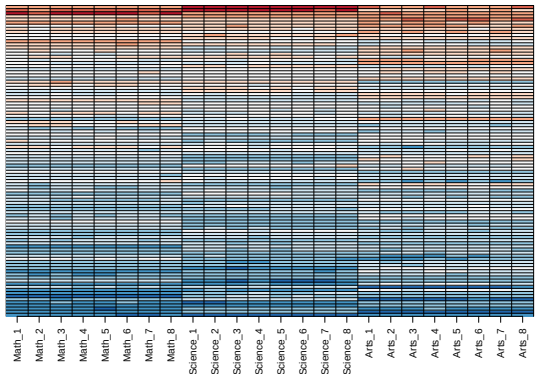
Singular value decomposition

We will construct a dataset of grade scores for 100 students in 24 different subjects. The overall average has been removed so this data represents the percentage point each student received above or below the average test score. So a 0 represents an average grade (C), a 25 is a high grade (A), and a -25 represents a low grade (F):

```
set.seed(1987)
n <- 100
k <- 8
Sigma <- 64  * matrix(c(1, .75, .5, .75, 1, .5, .5, .5, 1), 3, 3)
m <- MASS::mvrnorm(n, rep(0, 3), Sigma)
m <- m[order(rowMeans(m), decreasing = TRUE),]
y <- m %x% matrix(rep(1, k), nrow = 1) +
  matrix(rnorm(matrix(n * k * 3)), n, k * 3)
colnames(y) <- c(paste(rep("Math",k), 1:k, sep="_"),
  paste(rep("Science",k), 1:k, sep="_"), paste(rep("Arts",k), 1:k, sep="_"))
```

Singular value decomposition

We can visualize the 24 test scores for the 100 students below. Are all students just about as good? Does being good in one subject imply you will be good in another? How does the SVD help with all this?



Singular value decomposition

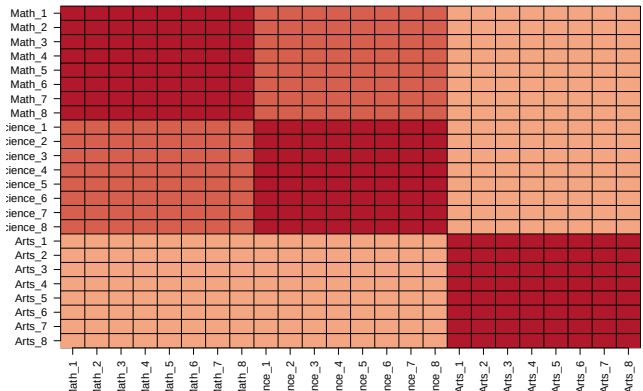
How would you describe the data based on this figure?

a. The test scores are all independent of each other.
b. The students that test well are at the top of the image and there seem to be three groupings by subject.
c. The students that are good at math are not good at science.
d. The students that are good at math are not good at humanities.

Singular value decomposition

We can examine the correlation between the test scores directly like this:

```
## [1] 0.4845344 1.0000000
```



Singular value decomposition

Which of the following best describes what you see?

a. The test scores are independent.
b. Math and science are highly correlated but the humanities are not.
c. There is high correlation between tests in the same subject but no correlation across subjects.
d. There is a correlation among all tests, but higher if the tests are in science and math and even higher within each subject.

Singular value decomposition

Use the function `svd` to compute the SVD of `y`. This function will return U, V and the diagonal entries of D.

```
s <- svd(y)
names(s)
```

```
## [1] "d" "u" "v"
```

Singular value decomposition

And we can look at the sizes of the outputs:

```
dim(s$d)
```

```
## NULL
```

```
dim(s$u)
```

```
## [1] 100  24
```

```
dim(s$v)
```

```
## [1] 24 24
```

Singular value decomposition

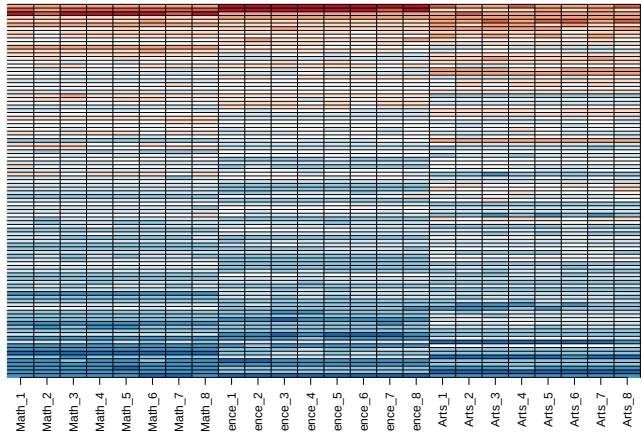
And we can check that the SVD works by typing:

```
y_svd <- s$u %*% diag(s$d) %*% t(s$v)
max(abs(y - y_svd))
```

```
## [1] 4.174439e-14
```

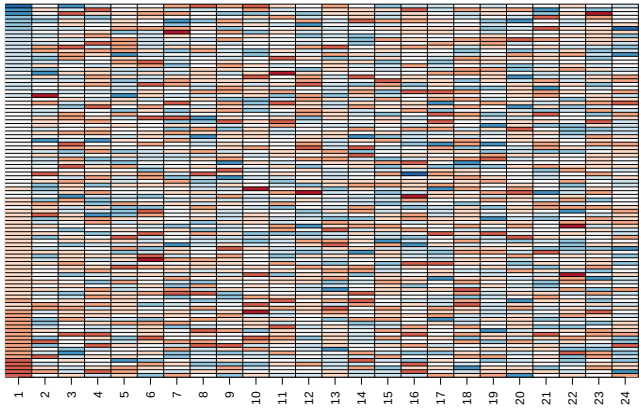
Singular value decomposition

Visualizing the SVD: Y



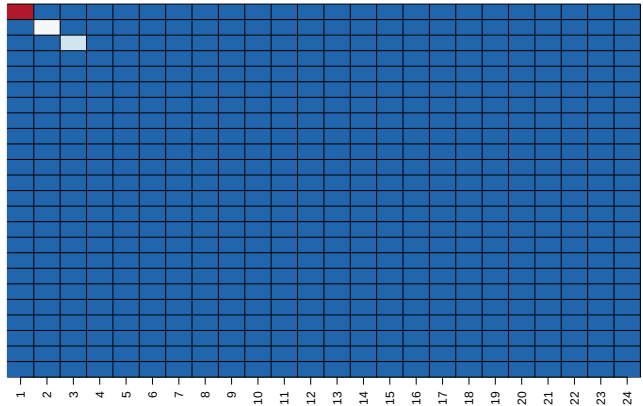
Singular value decomposition

Visualizing the SVD: U



Singular value decomposition

Visualizing the SVD: diag(D)



Singular value decomposition

Notice the following:

```
s$d
```

```
##  [1] 356.098658 173.727682 126.241777  14.942884  13.914703  12.772065
##  [7]  12.656324  12.046425  11.437718  10.904680  10.767620  10.480889
## [13]   9.800543   9.281666   9.128728   9.039113   8.446286   8.171682
## [19]   7.863741   7.481921   7.246042   6.898844   6.408354   6.001888
```

```
s$d^2/sum(s$d^2)
```

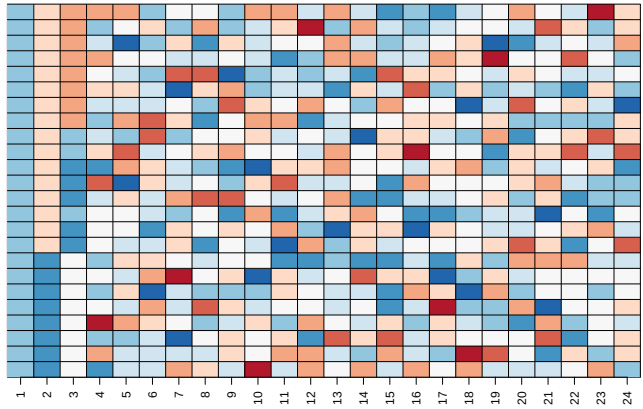
```
##  [1] 0.7243349532 0.1723998240 0.0910342816 0.0012754623 0.0011059783
##  [6] 0.0009317964 0.0009149850 0.0008289247 0.0007472700 0.0006792422
## [11] 0.0006622748 0.0006274729 0.0005486546 0.0004920969 0.0004760135
## [16] 0.0004667135 0.0004075025 0.0003814360 0.0003532296 0.0003197607
## [21] 0.0002999165 0.0002718638 0.0002345804 0.0002057664
```

```
cumsum(s$d^2)/sum(s$d^2)
```

```
##  [1] 0.7243350 0.8967348 0.9877691 0.9890445 0.9901505 0.9910823 0.9919973
##  [8] 0.9928262 0.9935735 0.9942527 0.9949150 0.9955425 0.9960911 0.9965832
## [15] 0.9970592 0.9975259 0.9979334 0.9983149 0.9986681 0.9989879 0.9992878
## [22] 0.9995597 0.9997942 1.0000000
```

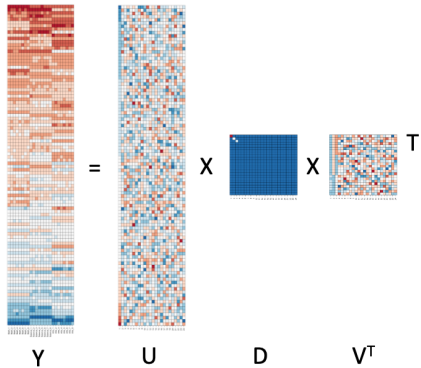
Singular value decomposition

Visualizing the SVD: V

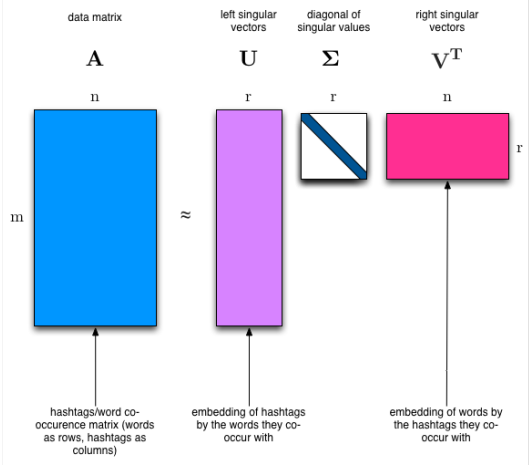


Singular value decomposition

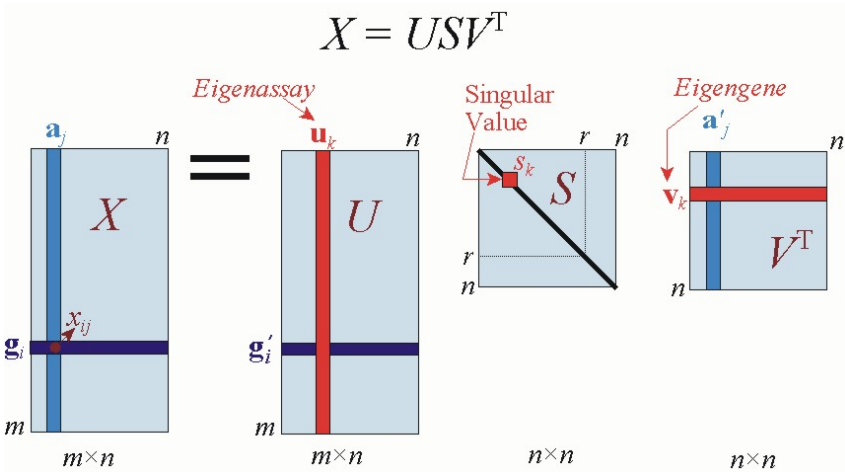
Putting them all together



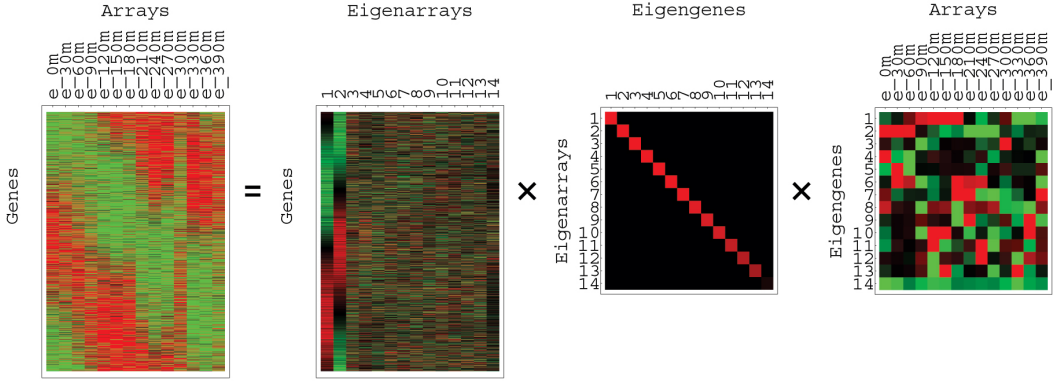
Singular value decomposition



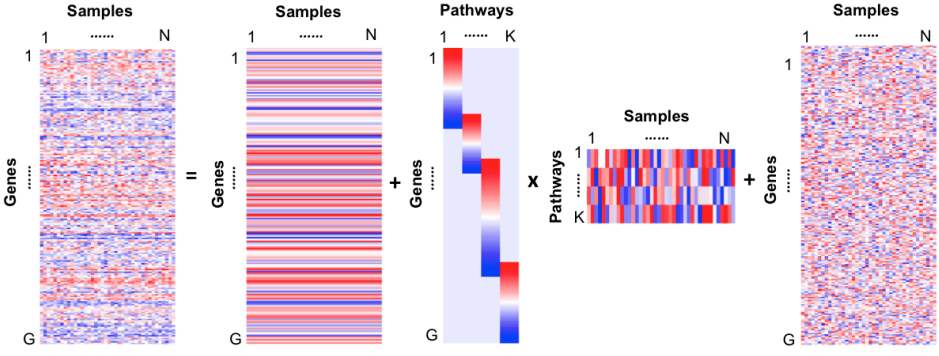
Singular value decomposition



Singular value decomposition



Factor analysis models



Non-Negative Matrix Factorization (NMF)

Similar to SVD, **Non-Negative Matrix Factorization (NMF)** is a linear dimensionality reduction technique.

- It decomposes a non-negative data matrix $\mathbf{V}$ into two lower-rank non-negative matrices, $\mathbf{W}$ and $\mathbf{H}$:

$$\mathbf{V} \approx \mathbf{WH}$$

  where $\mathbf{V}$ is $N \times P$, $\mathbf{W}$ is $N \times K$, and $\mathbf{H}$ is $K \times P$.
- The constraint that all elements in $\mathbf{W}$ (features/basis) and $\mathbf{H}$ (coefficients/weights) must be **non-negative** is key.

Non-Negative Matrix Factorization: Interpretation

The non-negativity constraint leads to a fundamentally different, often more **interpretable** factorization:

- **Additive, Parts-Based Representation:** Unlike PCA/SVD, NMF relies on a purely additive combination of components. It decomposes the data into meaningful **"parts"** that add up to form the original data.
- **Ideal for Non-Negative Data:** NMF is particularly well-suited for data where negative values are either impossible or meaningless, such as image pixel intensities, gene expression levels, or term frequency counts.

Non-Negative Matrix Factorization: Theory & Fitting

The core problem in NMF is finding two non-negative matrices, **W** and **H**, that best approximate the data **V**:

$$\mathbf{V} \approx \mathbf{WH}$$

This is a non-linear, non-convex **optimization problem** solved by minimizing a cost function (or objective function).

Non-Negative Matrix Factorization: Theory & Fitting

- **Common Cost Functions (Loss Functions):**
  1. **Frobenius Norm:** $\mathcal{D}_F(\mathbf{V}||\mathbf{WH}) = \frac{1}{2}\sum_{i,j}(V_{ij} - (WH)_{ij})^2$
  2. **Kullback-Leibler (KL) Divergence:** Often used for count data, as it is related to maximum likelihood estimation for the Poisson distribution.
- **Non-Convexity:** Since the problem is non-convex, the final solution can depend on the **initial random guess** for **W** and **H**, meaning different runs may converge to different local minima. Running the algorithm multiple times (as we will do with `nrun`=20) helps find a better solution.

NMF: Multiplicative Update Algorithm

The most popular method for minimizing the Frobenius Norm cost function is the **Multiplicative Update Rule**, developed by Lee and Seung (2001).

This method avoids standard gradient descent's subtraction step, ensuring that the non-negativity constraints for **W** and **H** are always met. It uses a sequence of iterative **multiplicative updates** until convergence.

NMF: Multiplicative Update Algorithm

The update rules for the $N \times K$ basis matrix $\mathbf{W}$ and the $K \times P$ coefficient matrix $\mathbf{H}$ are:

- **Update H:**

$$H_{a\mu} \leftarrow H_{a\mu} \frac{(\mathbf{W}^T \mathbf{V})_{a\mu}}{(\mathbf{W}^T \mathbf{W} \mathbf{H})_{a\mu}}$$

- **Update W:**

$$W_{i\alpha} \leftarrow W_{i\alpha} \frac{(\mathbf{V} \mathbf{H}^T)_{i\alpha}}{(\mathbf{W} \mathbf{H} \mathbf{H}^T)_{i\alpha}}$$

These updates are designed such that the cost function is non-increasing, guaranteeing convergence.

NMF vs. Other Techniques

Technique	Decomposition	Key Constraint	Best Suited For
PCA	$\mathbf{X} = \mathbf{T}\mathbf{P}^T + \mathbf{E}$	**Orthogonal** principal components.	General dimensionality reduction, variance maximization.
SVD	$\mathbf{Y} = \mathbf{U}\mathbf{D}\mathbf{V}^\top$	**U** and **V** are **orthogonal**.	Low-rank approximation, matrix structure analysis.
UMAP	Non-linear projection.	**Topological** preservation (preserving local/global data structure).	**Visualization** of complex, high-dimensional manifolds.
NMF	$\mathbf{V} \approx \mathbf{W}\mathbf{H}$	**V**, **W**, and **H** must be **non-negative**.	**Parts-based representation** in non-negative data.

When to Use NMF Over PCA, SVD, or UMAP?

- **Primary Condition:** Your data matrix **V** must be **non-negative** (e.g., counts, frequencies, magnitudes). If your data has negative values, you must shift it to be non-negative before applying NMF.
- **The Interpretability Advantage:** You should choose NMF when your goal is to achieve an **interpretable, additive, "parts-based" decomposition**.
  - **PCA/SVD** components are a mix of positive and negative values, making them "holistic" (like a linear combination of all features).
  - **NMF** components (**W**) are purely additive. They represent distinct, localized **parts** that combine to form the original observations.

When to Use NMF Over PCA, SVD, or UMAP?

- **Use Cases Where NMF Excels:**
  - **Topic Modeling:** Discovering how non-negative word frequencies combine into distinct, non-negative **topics**.
  - **Image Feature Extraction:** Decomposing images into a non-negative basis of facial **parts** (e.g., eyes, nose, mouth).
  - **Gene Expression:** Identifying non-negative **gene signatures** or pathways.

NMF: Iris Example

Since the **Iris dataset** features (sepal length/width, petal length/width) are all non-negative measurements, NMF is applicable.

We can apply NMF with a low rank $K = 2$ to visualize the data in two dimensions, similar to the PCA and UMAP examples.

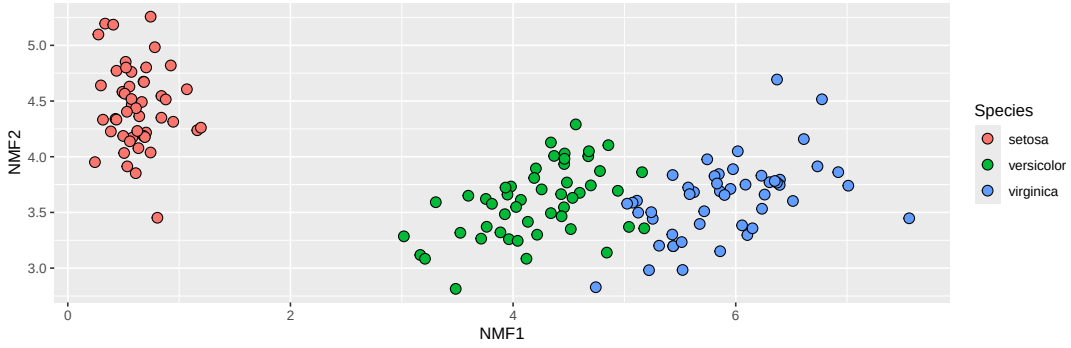
NMF: Iris Example

```r
library(NMF)
# The iris data is inherently non-negative
x_nmf <- iris[, 1:4] %>% as.matrix()
set.seed(42) # Set seed for reproducibility

# Perform NMF with rank k=2
# nrun=20 is recommended for better stability
nmf_res <- nmf(x_nmf, 2, nrun=20)
```

NMF: Iris Example

```r
# The transformed data is in the 'basis' matrix W
nmf_transformed <- basis(nmf_res)
colnames(nmf_transformed) <- c("NMF1", "NMF2")

# Visualization
data.frame(nmf_transformed, Species = iris$Species) %>%
  ggplot(aes(NMF1, NMF2, fill = Species)) +
  geom_point(cex = 3, pch = 21) +
  coord_fixed(ratio = 1)
```

NMF: Iris Example



NMF: Iris Example Interpretation

The resulting NMF plot:

- **Clustering:** The **Setosa** species (blue) is distinctly separate, largely driven by the first NMF component.
- **Non-Negative Axes:** The data is confined to the positive quadrant of the plot, which is characteristic of NMF and results in an additive decomposition.
- **Components (H):** The magnitude of the entries in the component matrix **H** (accessed via `coef(nmf_res)`) can reveal the contribution of each original feature to the separation.

Session Info

```
sessionInfo()
```

```
## R version 4.5.1 (2025-06-13)
## Platform: aarch64-apple-darwin20
## Running under: macOS Sequoia 15.6.1
##
## Matrix products: default
## BLAS:   /Library/Frameworks/R.framework/Versions/4.5-arm64/Resources/lib/libRblas.0.dylib
## LAPACK: /Library/Frameworks/R.framework/Versions/4.5-arm64/Resources/lib/libRlapack.dylib;  LAPACK version 3.12.1
##
## locale:
## [1] en_US.UTF-8/en_US.UTF-8/en_US.UTF-8/C/en_US.UTF-8/en_US.UTF-8
##
## time zone: America/Denver
## tzcode source: internal
##
## attached base packages:
## [1] parallel  stats     graphics  grDevices utils     datasets  methods
## [8] base
##
## other attached packages:
##  [1] doParallel_1.0.17   iterators_1.0.14    foreach_1.5.2
##  [4] NMF_0.28            Biobase_2.68.0      BiocGenerics_0.54.0
##  [7] generics_0.1.4      cluster_2.1.8.1     rngtools_1.5.2
## [10] registry_0.5-1      umap_0.2.10.0       MASS_7.3-65
## [13] lubridate_1.9.4     forcats_1.0.0       stringr_1.5.1
## [16] dplyr_1.1.4         purrr_1.1.0         readr_2.1.5
```