Singular value decomposition (SVD) and dimensionality reduction

Distributional Semantic Models

Stefan Evert[1] & Alessandro Lenci[2]

[1]University of Osnabrück, Germany
[2]University of Pisa, Italy

Remember PCA?

- Principal components analysis is based on an **eigenvalue decomposition** of the **covariance matrix** $\mathbf{C}$ into

$$\mathbf{C} = \mathbf{U} \cdot \mathbf{D} \cdot \mathbf{U}^T$$

where $\mathbf{U}$ is orthogonal and $\mathbf{D} = \mathrm{Diag}(\lambda_1, \ldots, \lambda_n)$.

Remember PCA?

- Principal components analysis is based on an **eigenvalue decomposition** of the **covariance matrix** $\mathbf{C}$ into

$$\mathbf{C} = \mathbf{U} \cdot \mathbf{D} \cdot \mathbf{U}^T$$

  where $\mathbf{U}$ is orthogonal and $\mathbf{D} = \mathrm{Diag}(\lambda_1, \dots, \lambda_n)$.

- The columns of $\mathbf{U}$ are **eigenvectors**

$$\mathbf{C}\mathbf{a}_i = \lambda_i \mathbf{a}_i$$

  for the ordered **eigenvalues** $\lambda_1 \geq \lambda_2 \geq \cdots \geq \lambda_n$

Remember PCA?

- Principal components analysis is based on an **eigenvalue decomposition** of the **covariance matrix** $\mathbf{C}$ into

$$\mathbf{C} = \mathbf{U} \cdot \mathbf{D} \cdot \mathbf{U}^T$$

  where $\mathbf{U}$ is orthogonal and $\mathbf{D} = \text{Diag}(\lambda_1, \dots, \lambda_n)$.

- The columns of $\mathbf{U}$ are **eigenvectors**

$$\mathbf{C}\mathbf{a}_i = \lambda_i \mathbf{a}_i$$

  for the ordered **eigenvalues** $\lambda_1 \geq \lambda_2 \geq \cdots \geq \lambda_n$

- Interesting link: $\mathbf{u}^T\mathbf{C}\mathbf{v}$ describes a general **inner product**
  - $\sigma_{\mathbf{v}}$ is the norm of $\mathbf{v}$ with respect to this general inner product
  - the eigenvalue decomposition corresponds to a transformation into Cartesian coordinates where $\mathbf{C}$ has diagonal form
  - eigenvalues λ_i are the "squashing factors" of the unit circle

Singular value decomposition (SVD)

- The idea of eigenvalue decomposition can be generalised to an arbitrary (non-symmetric, non-square) matrix $\mathbf{A}$
  - ☞ need not have any eigenvalues

Singular value decomposition (SVD)

- The idea of eigenvalue decomposition can be generalised to an arbitrary (non-symmetric, non-square) matrix $\mathbf{A}$
  - ☞ need not have any eigenvalues
- **Singular value decomposition** (**SVD**) factorises $\mathbf{A}$ into

$$\mathbf{A} = \mathbf{U} \cdot \mathbf{\Sigma} \cdot \mathbf{V}^T$$

where $\mathbf{U}$ and $\mathbf{V}$ are orthogonal coordinate transformations and $\mathbf{\Sigma}$ is a rectangular-diagonal matrix of **singular values** (with customary ordering $\sigma_1 \geq \sigma_2 \geq \cdots \geq \sigma_n \geq 0$)

Singular value decomposition (SVD)

- The idea of eigenvalue decomposition can be generalised to an arbitrary (non-symmetric, non-square) matrix $\mathbf{A}$
  - ☞ need not have any eigenvalues
- **Singular value decomposition** (**SVD**) factorises $\mathbf{A}$ into

$$\mathbf{A} = \mathbf{U} \cdot \mathbf{\Sigma} \cdot \mathbf{V}^T$$

  where $\mathbf{U}$ and $\mathbf{V}$ are orthogonal coordinate transformations and $\mathbf{\Sigma}$ is a rectangular-diagonal matrix of **singular values** (with customary ordering $\sigma_1 \geq \sigma_2 \geq \cdots \geq \sigma_n \geq 0$)
- SVD is an important tool in linear algebra and statistics
  - ☞ in particular, PCA can be computed from SVD decomposition

SVD illustration

$$\begin{array}{cc} & n \\ k & \begin{bmatrix} \\ \mathbf{A} \\ \\ \end{bmatrix} \end{array} = \begin{array}{cc} & k \\ k & \begin{bmatrix} \\ \mathbf{U} \\ \\ \end{bmatrix} \end{array} \cdot \begin{array}{cc} & n \\ k & \begin{bmatrix} \sigma_1 & & \\ & \ddots & \\ & \mathbf{\Sigma} & \sigma_n \\ & & \end{bmatrix} \end{array} \cdot \begin{array}{cc} & n \\ n & \begin{bmatrix} \\ \mathbf{V}^T \\ \\ \end{bmatrix} \end{array}$$

PCA and the DSM matrix

- Take a closer look at the covariance matrix

$$\mathbf{C} = \frac{1}{k-1} \sum_{i=1}^{k} \mathbf{x}_i \mathbf{x}_i^T$$

PCA and the DSM matrix

- Take a closer look at the covariance matrix

$$\mathbf{C} = \frac{1}{k-1} \sum_{i=1}^{k} \mathbf{x}_i \mathbf{x}_i^T$$

- With $\mathbf{x}_i^T = \left[x_{i1}, \ldots, x_{in} \right]$ we find that

$$\mathbf{x}_i \mathbf{x}_i^T = \begin{bmatrix} x_{i1} \\ \vdots \\ x_{in} \end{bmatrix} \cdot \begin{bmatrix} x_{i1} & \cdots & x_{in} \end{bmatrix} = \begin{bmatrix} (x_{i1})^2 & x_{i1}x_{i2} & \cdots & x_{i1}x_{in} \\ x_{i2}x_{i1} & (x_{i2})^2 & \cdots & x_{i2}x_{in} \\ \vdots & \vdots & \ddots & \vdots \\ x_{in}x_{i1} & x_{in}x_{i2} & \cdots & (x_{in})^2 \end{bmatrix}$$

PCA and the DSM matrix

$$\sum_{i=1}^{k} \mathbf{x}_i \mathbf{x}_i^T = \begin{bmatrix} \sum_i (x_{i1})^2 & \sum_i x_{i1}x_{i2} & \cdots & \sum_i x_{i1}x_{in} \\ \sum_i x_{i2}x_{i1} & \sum_i (x_{i2})^2 & \cdots & \sum_i x_{i2}x_{in} \\ \vdots & \vdots & \ddots & \vdots \\ \sum_i x_{in}x_{i1} & \sum_i x_{in}x_{i2} & \cdots & \sum_i (x_{in})^2 \end{bmatrix}$$

PCA and the DSM matrix

$$\sum_{i=1}^{k} \mathbf{x}_i \mathbf{x}_i^T = \begin{bmatrix} \sum_i (x_{i1})^2 & \sum_i x_{i1}x_{i2} & \cdots & \sum_i x_{i1}x_{in} \\ \sum_i x_{i2}x_{i1} & \sum_i (x_{i2})^2 & \cdots & \sum_i x_{i2}x_{in} \\ \vdots & \vdots & \ddots & \vdots \\ \sum_i x_{in}x_{i1} & \sum_i x_{in}x_{i2} & \cdots & \sum_i (x_{in})^2 \end{bmatrix}$$

- If the $\mathbf{x}_i$ are the row vectors of a DSM matrix $\mathbf{M}$, then the sums above are inner products between its column vectors

PCA and the DSM matrix

$$\sum_{i=1}^{k} \mathbf{x}_i \mathbf{x}_i^T = \begin{bmatrix} \sum_i (x_{i1})^2 & \sum_i x_{i1}x_{i2} & \cdots & \sum_i x_{i1}x_{in} \\ \sum_i x_{i2}x_{i1} & \sum_i (x_{i2})^2 & \cdots & \sum_i x_{i2}x_{in} \\ \vdots & \vdots & \ddots & \vdots \\ \sum_i x_{in}x_{i1} & \sum_i x_{in}x_{i2} & \cdots & \sum_i (x_{in})^2 \end{bmatrix}$$

- If the $\mathbf{x}_i$ are the row vectors of a DSM matrix $\mathbf{M}$, then the sums above are inner products between its column vectors

➥ $\mathbf{C}$ can efficiently be computed by matrix multiplication (similar to cosine similarities, but for column vectors)

$$\mathbf{C} = \tfrac{1}{k-1} \sum_{i=1}^{k} \mathbf{x}_i \mathbf{x}_i^T = \tfrac{1}{k-1} \mathbf{M}^T \mathbf{M}$$

PCA by singular value decomposition

- Up to an irrelevant scaling factor $\frac{1}{k-1}$, we are thus looking for an eigenvalue decomposition of $\mathbf{M}^T\mathbf{M}$ (which is symmetric!)
- Like every matrix, $\mathbf{M}$ has a singular value decomposition

$$\mathbf{M} = \mathbf{U}\mathbf{\Sigma}\mathbf{V}^T$$

- By inserting the SVD, we obtain

$$\begin{aligned} \mathbf{M}^T\mathbf{M} &= \left(\mathbf{U}\mathbf{\Sigma}\mathbf{V}^T\right)^T\mathbf{U}\mathbf{\Sigma}\mathbf{V}^T \\ &= (\mathbf{V}^T)^T\mathbf{\Sigma}^T\underbrace{\mathbf{U}^T\mathbf{U}}_{\mathbf{I}}\mathbf{\Sigma}\mathbf{V}^T \\ &= \mathbf{V}(\underbrace{\mathbf{\Sigma}^T\mathbf{\Sigma}}_{\mathbf{\Sigma}^2})\mathbf{V}^T \end{aligned}$$

PCA by singular value decomposition

- We have found the eigenvalue decomposition

$$\mathbf{M}^T\mathbf{M} = \mathbf{V}\mathbf{\Sigma}^2\mathbf{V}^T$$

with

$$\mathbf{\Sigma}^2 = \mathbf{\Sigma}^T\mathbf{\Sigma} = \begin{bmatrix} (\sigma_1)^2 & n & \\ n & \ddots & \\ & & (\sigma_n)^2 \end{bmatrix}$$

PCA by singular value decomposition

- We have found the eigenvalue decomposition

$$\mathbf{M}^T\mathbf{M} = \mathbf{V}\mathbf{\Sigma}^2\mathbf{V}^T$$

with

$$\mathbf{\Sigma}^2 = \mathbf{\Sigma}^T\mathbf{\Sigma} = \begin{bmatrix} (\sigma_1)^2 & n & \\ n & \ddots & \\ & & (\sigma_n)^2 \end{bmatrix}$$

- The column vectors of **V** are **latent dimensions**

PCA by singular value decomposition

- We have found the eigenvalue decomposition

$$\mathbf{M}^T\mathbf{M} = \mathbf{V}\mathbf{\Sigma}^2\mathbf{V}^T$$

with

$$\mathbf{\Sigma}^2 = \mathbf{\Sigma}^T\mathbf{\Sigma} = \begin{bmatrix} (\sigma_1)^2 & n & \\ n & \ddots & \\ & & (\sigma_n)^2 \end{bmatrix}$$

- The column vectors of **V** are **latent dimensions**
- The corresponding squared **singular values** partition variance: $(\sigma_1)^2 / \sum_i (\sigma_i)^2$ = proportion along first latent dimension
  - ☞ intuitively, singular value shows importance of latent dimension

PCA by singular value decomposition

- We have found the eigenvalue decomposition

$$\mathbf{M}^T\mathbf{M} = \mathbf{V}\boldsymbol{\Sigma}^2\mathbf{V}^T$$

with

$$\boldsymbol{\Sigma}^2 = \boldsymbol{\Sigma}^T\boldsymbol{\Sigma} = \begin{bmatrix} (\sigma_1)^2 & n & \\ n & \ddots & \\ & & (\sigma_n)^2 \end{bmatrix}$$

- The column vectors of **V** are **latent dimensions**
- The corresponding squared **singular values** partition variance: $(\sigma_1)^2 / \sum_i (\sigma_i)^2$ = proportion along first latent dimension
  - ☞ intuitively, singular value shows importance of latent dimension
- Interpretation of **U** is less intuitive (**latent families** of words?)

Transforming the DSM matrix

- We can directly transform the columns of the DSM matrix $\mathbf{M}$:

$$\mathbf{MV} = \mathbf{U\Sigma}(\mathbf{V}^T\mathbf{V}) = \mathbf{U\Sigma}$$

Transforming the DSM matrix

- We can directly transform the columns of the DSM matrix $\mathbf{M}$:

$$\mathbf{MV} = \mathbf{U\Sigma}(\mathbf{V}^T\mathbf{V}) = \mathbf{U\Sigma}$$

- For "noise reduction", project into m-dimensional subspace by dropping all but the first $m \ll n$ columns of $\mathbf{U\Sigma}$

➥ Sufficient to calculate the first m **singular values** $\sigma_1, \dots, \sigma_m$ and **left singular vectors** $\mathbf{a}_1, \dots, \mathbf{a}_m$ (columns of $\mathbf{U}$)

Transforming the DSM matrix

- We can directly transform the columns of the DSM matrix $\mathbf{M}$:

$$\mathbf{MV} = \mathbf{U\Sigma}(\mathbf{V}^T\mathbf{V}) = \mathbf{U\Sigma}$$

- For "noise reduction", project into m-dimensional subspace by dropping all but the first $m \ll n$ columns of $\mathbf{U\Sigma}$

➥ Sufficient to calculate the first m **singular values** $\sigma_1, \dots, \sigma_m$ and **left singular vectors** $\mathbf{a}_1, \dots, \mathbf{a}_m$ (columns of $\mathbf{U}$)

- What is the difference between SVD and PCA?

Transforming the DSM matrix

- We can directly transform the columns of the DSM matrix $\mathbf{M}$:

$$\mathbf{MV} = \mathbf{U\Sigma}(\mathbf{V}^T\mathbf{V}) = \mathbf{U\Sigma}$$

- For "noise reduction", project into m-dimensional subspace by dropping all but the first $m \ll n$ columns of $\mathbf{U\Sigma}$

➥ Sufficient to calculate the first m **singular values** $\sigma_1, \ldots, \sigma_m$ and **left singular vectors** $\mathbf{a}_1, \ldots, \mathbf{a}_m$ (columns of $\mathbf{U}$)

- What is the difference between SVD and PCA?
  - ☞ we forgot to center and rescale the data!
  - ☞ most DSM matrices contain only non-negative values
  - ☞ first latent dimension points towards "positive" sector, and was often found to be "uninteresting" in early SVD studies

SVD with R

```
# As an example, we will use the unscaled matrix M again
> M1 <- M[c(1, 2, 4, 6), ]
> M1
      eat get hear kill see use
 boat   0  59    4    0  39  23
 cat    6  52    4   26  58   4
 dog   33 115   42   17  83  10
 pig    9  12    2   27  17   3

# svd() function returns data structure with decomposition
> SVD <- svd(M1)

> SVD$d  # singular values
[1] 186.57942  34.92487  28.18571  12.03908
```

SVD with R

```
# Extract matrices U, Σ and V
> Sigma <- diag(SVD$d)   # reduced to square matrix
> U <- SVD$u   # coordinate transformations U and V
> V <- SVD$v   # recall that V contains the latent dimensions

# Now reconstruct M from decomposition
> round(U %*% Sigma %*% t(V), 2)
     [,1] [,2] [,3] [,4] [,5] [,6]
[1,]    0   59    4    0   39   23
[2,]    6   52    4   26   58    4
[3,]   33  115   42   17   83   10
[4,]    9   12    2   27   17    3
```

SVD with R

```
# Coordinates of target nouns in latent DSM space
> U %*% Sigma

> M1 %*% V   # this version preserves row names
           [,1]       [,2]       [,3]       [,4]
boat  -69.97214 -12.570114  21.760062  4.4036025
cat   -78.87562  21.092424   9.865719 -6.9580067
dog  -151.85390  -9.004136 -14.673158  0.1279540
pig   -25.19541  23.146798  -2.880942  8.7816522
```

SVD with R

```
# Compute rank-m approximations of the original matrix M
> svd.approx <- function (m) {
+   U[,1:m, drop=FALSE] %*% Sigma[1:m,1:m, drop=FALSE] %*%
+   t(V)[1:m,, drop=FALSE]
+ }
> round(svd.approx(1), 1)
     [,1]  [,2] [,3] [,4] [,5] [,6]
[1,] 11.5  52.3 14.1 10.7 40.9  7.1
[2,] 12.9  58.9 15.9 12.0 46.1  8.0
[3,] 24.9 113.4 30.6 23.2 88.7 15.4
[4,]  4.1  18.8  5.1  3.8 14.7  2.5
> round(svd.approx(2), 1)
     [,1]  [,2] [,3] [,4] [,5] [,6]
[1,] 11.1  56.4 17.2  0.2 37.0  9.4
[2,] 13.6  51.9 10.8 29.7 52.6  4.1
[3,] 24.6 116.4 32.8 15.6 85.9 17.0
[4,]  4.8  11.2 -0.6 23.2 21.9 -1.7
```

Scaling up to the real world

- So far, we have worked on small **toy models**
  - DSM matrix restricted to 2,000 – 5,000 rows and columns
  - small corpora (or dependency sets) can be processed within **R**

Scaling up to the real world

- So far, we have worked on small **toy models**
  - DSM matrix restricted to 2,000 – 5,000 rows and columns
  - small corpora (or dependency sets) can be processed within **R**
- Now we need to scale up to **real world** data sets
  - for most statistical models, more data are better data!
  - cf. success of Google-based NLP techniques (even if simplistic)

Scaling up to the real world

- So far, we have worked on small **toy models**
  - DSM matrix restricted to 2,000 – 5,000 rows and columns
  - small corpora (or dependency sets) can be processed within **R**
- Now we need to scale up to **real world** data sets
  - for most statistical models, more data are better data!
  - cf. success of Google-based NLP techniques (even if simplistic)
- Example 1: window-based DSM on BNC content words
  - 83,926 lemma types with $f \geq 10$
  - term-term matrix with $83{,}926 \cdot 83{,}926 = 7$ billion entries
  - standard representation requires 56 GB of RAM (8-byte floats)

Scaling up to the real world

- So far, we have worked on small **toy models**
  - DSM matrix restricted to 2,000 – 5,000 rows and columns
  - small corpora (or dependency sets) can be processed within **R**
- Now we need to scale up to **real world** data sets
  - for most statistical models, more data are better data!
  - cf. success of Google-based NLP techniques (even if simplistic)
- Example 1: window-based DSM on BNC content words
  - 83,926 lemma types with $f \geq 10$
  - term-term matrix with $83{,}926 \cdot 83{,}926 = 7$ billion entries
  - standard representation requires 56 GB of RAM (8-byte floats)
  - only 22.1 million non-zero entries ($= 0.32\%$)

Scaling up to the real world

- So far, we have worked on small **toy models**
  - DSM matrix restricted to 2,000 – 5,000 rows and columns
  - small corpora (or dependency sets) can be processed within **R**
- Now we need to scale up to **real world** data sets
  - for most statistical models, more data are better data!
  - cf. success of Google-based NLP techniques (even if simplistic)
- Example 1: window-based DSM on BNC content words
  - 83,926 lemma types with $f \geq 10$
  - term-term matrix with $83{,}926 \cdot 83{,}926 = 7$ billion entries
  - standard representation requires 56 GB of RAM (8-byte floats)
  - only 22.1 million non-zero entries ($= 0.32\%$)
- Example 2: Google Web 1T 5-grams (1 trillion words)
  - more than 1 million word types with $f \geq 2500$
  - term-term matrix with 1 trillion entries requires 8 TB RAM
  - only 400 million non-zero entries ($= 0.04\%$)

Handling large data sets: three approaches

1. Sparse matrix representation
   - full DSM matrix does not fit into memory
   - but much smaller number of non-zero entries can be handled

Handling large data sets: three approaches

1. Sparse matrix representation
   - full DSM matrix does not fit into memory
   - but much smaller number of non-zero entries can be handled

2. Feature selection
   - reduce DSM matrix to subset of columns (usu. 2,000 – 10,000)
   - select most frequent, salient, discriminative, ... features

Handling large data sets: three approaches

1. Sparse matrix representation
   - full DSM matrix does not fit into memory
   - but much smaller number of non-zero entries can be handled
2. Feature selection
   - reduce DSM matrix to subset of columns (usu. 2,000 – 10,000)
   - select most frequent, salient, discriminative, … features
3. Dimensionality reduction
   - also reduces number of columns, but maps vectors to subspace
   - singular value decomposition (usu. ca. 300 dimensions)
   - random indexing (2,000 or more dimensions)
   - performed with external tools ➜ **R** can handle reduced matrix

Sparse matrix representation

- Invented example of a **sparsely populated** DSM matrix

	eat	get	hear	kill	see	use
boat	·	59	·	·	39	23
cat	·	·	·	26	58	·
cup	·	98	·	·	·	·
dog	33	·	42	·	83	·
knife	·	·	·	·	·	84
pig	9	·	·	27	·	·

Sparse matrix representation

- Invented example of a **sparsely populated** DSM matrix

	eat	get	hear	kill	see	use
boat	·	59	·	·	39	23
cat	·	·	·	26	58	·
cup	·	98	·	·	·	·
dog	33	·	42	·	83	·
knife	·	·	·	·	·	84
pig	9	·	·	27	·	·

- Store only non-zero entries in compact **sparse matrix format**

row	col	value
1	2	59
1	5	39
1	6	23
2	4	26
2	5	58
3	2	98

row	col	value
4	1	33
4	3	42
4	5	83
5	6	84
6	1	9
6	4	27

Working with sparse matrices

- Compressed format: each row index (or column index) stored only once, followed by non-zero entries in this row (or column)
  - convention: **column-major** matrix (data stored by columns)

Working with sparse matrices

- Compressed format: each row index (or column index) stored only once, followed by non-zero entries in this row (or column)
  - convention: **column-major** matrix (data stored by columns)

- Specialised algorithms for sparse matrix algebra
  - especially matrix multiplication, solving linear systems, etc.
  - take care to avoid operations that create a dense matrix!

Working with sparse matrices

- Compressed format: each row index (or column index) stored only once, followed by non-zero entries in this row (or column)
  - convention: **column-major** matrix (data stored by columns)
- Specialised algorithms for sparse matrix algebra
  - especially matrix multiplication, solving linear systems, etc.
  - take care to avoid operations that create a dense matrix!
- **R** implementation: `Matrix` package (from CRAN)
  - can build sparse matrix from (row, column, value) table
  - unfortunately, no implementation of sparse SVD so far

Working with sparse matrices

- Compressed format: each row index (or column index) stored only once, followed by non-zero entries in this row (or column)
  - convention: **column-major** matrix (data stored by columns)
- Specialised algorithms for sparse matrix algebra
  - especially matrix multiplication, solving linear systems, etc.
  - take care to avoid operations that create a dense matrix!
- **R** implementation: `Matrix` package (from CRAN)
  - can build sparse matrix from (row, column, value) table
  - unfortunately, no implementation of sparse SVD so far
- Other software packages: Matlab, Octave (recent versions)

Feature selection

- Many published models use **feature selection** to reduce the size of a term-term DSM matrix

Feature selection

- Many published models use **feature selection** to reduce the size of a term-term DSM matrix
- Selection criteria:
  - most frequent context terms
  - most informative contxt terms (tf.idf)
  - most discriminative context terms (variance, entropy)
  - term restricted by part of speech (e.g. only verbs)

Feature selection

- Many published models use **feature selection** to reduce the size of a term-term DSM matrix
- Selection criteria:
  - most frequent context terms
  - most informative contxt terms (tf.idf)
  - most discriminative context terms (variance, entropy)
  - term restricted by part of speech (e.g. only verbs)
- Features often selected *before* co-occurrence counts
  - only a moderately-sized DSM matrix has to be built
  - allows simple in-memory algorithm for co-occurrence counts

Feature selection

- Many published models use **feature selection** to reduce the size of a term-term DSM matrix
- Selection criteria:
  - most frequent context terms
  - most informative contxt terms (tf.idf)
  - most discriminative context terms (variance, entropy)
  - term restricted by part of speech (e.g. only verbs)
- Features often selected *before* co-occurrence counts
  - only a moderately-sized DSM matrix has to be built
  - allows simple in-memory algorithm for co-occurrence counts
- Alternative: build DSM matrix only for relevant target terms
  - i.e. reduce the number of rows instead of number of columns

Feature selection

- Many published models use **feature selection** to reduce the size of a term-term DSM matrix
- Selection criteria:
  - most frequent context terms
  - most informative contxt terms (tf.idf)
  - most discriminative context terms (variance, entropy)
  - term restricted by part of speech (e.g. only verbs)
- Features often selected *before* co-occurrence counts
  - only a moderately-sized DSM matrix has to be built
  - allows simple in-memory algorithm for co-occurrence counts
- Alternative: build DSM matrix only for relevant target terms
  - i.e. reduce the number of rows instead of number of columns
- **Disadvantage**: useful information may be discarded
  - aggressive feature selection is common in the DSM literature

Dimensionality reduction: SVD

- Feature selection is a simple form of **dimensionality reduction** for managing high-dimensional spaces
  - information from discarded features is completely lost

Dimensionality reduction: SVD

- Feature selection is a simple form of **dimensionality reduction** for managing high-dimensional spaces
  - information from discarded features is completely lost
- Better strategy: only discard irrelevant information by orthogonal projection into subspace of latent dimensions
  - subspace of first *m* principal components or singular vectors
  - recall that this subspace preserves original distances as well as possible ➜ minimal amount of information discarded

Dimensionality reduction: SVD

- Feature selection is a simple form of **dimensionality reduction** for managing high-dimensional spaces
  - information from discarded features is completely lost
- Better strategy: only discard irrelevant information by orthogonal projection into subspace of latent dimensions
  - subspace of first m principal components or singular vectors
  - recall that this subspace preserves original distances as well as possible ➜ minimal amount of information discarded
- Key ingredient: implementation of **sparse-matrix SVD**
  - SVDPACK with various algorithms developed by Michael Berry
  - most convenient implementation: SVDLIBC `http://tedlab.mit.edu/~dr/svdlibc/`
  - standard input format: compressed column-major sparse matrix
  - only calculates first m singular values and vectors

Dimensionality reduction: SVD

- Feature selection is a simple form of **dimensionality reduction** for managing high-dimensional spaces
  - information from discarded features is completely lost
- Better strategy: only discard irrelevant information by orthogonal projection into subspace of latent dimensions
  - subspace of first *m* principal components or singular vectors
  - recall that this subspace preserves original distances as well as possible ➜ minimal amount of information discarded
- Key ingredient: implementation of **sparse-matrix SVD**
  - SVDPACK with various algorithms developed by Michael Berry
  - most convenient implementation: SVDLIBC `http://tedlab.mit.edu/~dr/svdlibc/`
  - standard input format: compressed column-major sparse matrix
  - only calculates first *m* singular values and vectors
- SVD components $\mathbf{U}$, $\mathbf{\Sigma}$ and $\mathbf{V}$ are stored in separate files

Dimensionality reduction: Random Indexing

- SVD is computationally expensive for large DSM matrix
  - even if the matrix is sparsely populated

Dimensionality reduction: Random Indexing

- SVD is computationally expensive for large DSM matrix
  - even if the matrix is sparsely populated
- Cheap method: orthogonal projection into **random** subspace
  - it can be shown that this preserves original distances with high probability (though not as well as SVD)
  - intuition: if dimensionality m of subspace is large enough, some vector should be close to $\mathbf{a}_1$, another close to $\mathbf{a}_2$, etc.

➥ **random indexing** (**RI**)

Dimensionality reduction: Random Indexing

- SVD is computationally expensive for large DSM matrix
  - even if the matrix is sparsely populated
- Cheap method: orthogonal projection into **random** subspace
  - it can be shown that this preserves original distances with high probability (though not as well as SVD)
  - intuition: if dimensionality m of subspace is large enough, some vector should be close to $\mathbf{a}_1$, another close to $\mathbf{a}_2$, etc.

➥ **random indexing** (**RI**)

- Further simplication: use **random basis** vectors for subspace
  - saves additional cost of constructing an orthonormal basis
  - if dimensionality n of original DSM space is large enough, two random vectors are likely to be almost orthogonal
  - intuition: inner product between random vectors = covariance of two independent samples of random numbers (should be 0)

Dimensionality reduction: Random Indexing

- SVD is computationally expensive for large DSM matrix
  - even if the matrix is sparsely populated
- Cheap method: orthogonal projection into **random** subspace
  - it can be shown that this preserves original distances with high probability (though not as well as SVD)
  - intuition: if dimensionality m of subspace is large enough, some vector should be close to $\mathbf{a}_1$, another close to $\mathbf{a}_2$, etc.

➥ **random indexing** (**RI**)

- Further simplication: use **random basis** vectors for subspace
  - saves additional cost of constructing an orthonormal basis
  - if dimensionality n of original DSM space is large enough, two random vectors are likely to be almost orthogonal
  - intuition: inner product between random vectors = covariance of two independent samples of random numbers (should be 0)
- SVD identifies latent dimensions ("noise reduction"), but RI only preserves distances ➜ requires higher dimensionality m