Computational Intelligence Lab FS2011 [1]

Stefan Heule

August 22, 2011

Contents

1 Introduction

The core problem we consider is *matrix factorization* (MF) of a given data matrix into two or three unknown matrices. Many important data analysis techniques can be written as MF problems by specifying:

- **Type of data:** e.g. boolean ($x_{d,n} \in \{0,1\}$) or non-negative real.
- **Approximation:** e.g. exact $\boldsymbol{X} = \boldsymbol{A} \cdot \boldsymbol{B}$ or minimal Frobenius norm $\min_{\boldsymbol{A},\boldsymbol{B}} \|\boldsymbol{X} - \boldsymbol{A} \cdot \boldsymbol{B}\|_F^2$.
- **Constraints on unknown matrices:** e.g. $\boldsymbol{A}$ has to be a basis or $\boldsymbol{B}$ must be sparse.

Often, MF is computationally NP-hard, therefore we need *efficient* and *low-error* approximation algorithms. We study the following techniques:

1. Dimension reduction
2. Single-assignment clustering
3. Multi-assignment clustering
4. Sparse coding

Definition 1.1 (Orthogonality)**.** *Two vectors in an inner product space V are* orthogonal *if their inner product is zero.*

An orthogonal *matrix is a square matrix with real entries whose column (or rows) are orthogonal unit vectors.*

Theorem 1.1. *The inverse of an orthogonal matrix is its transpose.*

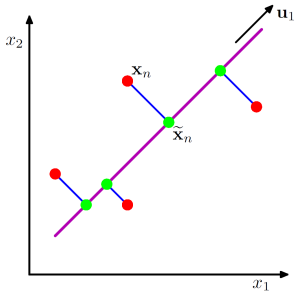

Figure 1: Projection of data points.

2 Dimension Reduction

Given N vectors in D dimensions, we want to find the K most important axes and project them. We reduce the dimensions because:

- Some features may be irrelevant
- We want to visualize high dimensional data, e.g., in 2 or 3 dimensions.
- The "intrinsic" dimensionality may be smaller than the number of features recorded in the data.

We want the *most interesting* dimensions:

- data compression
- feature selection
- complexity reduction

2.1 Principle Component Analysis (PCA)

Orthogonal linear projection of high dimensional data onto low dimensional subspace, as shown in Figure 1.

We have the following objectives when performing a PCA:

1. Minimize error $\|\boldsymbol{x}_n - \tilde{\boldsymbol{x}}_n\|_2$ of point $\boldsymbol{x}_n$ and its approximation $\tilde{\boldsymbol{x}}_n$.
2. Reveal "interesting" information: maximize *variance*.

It is possible to show that both objectives are formally equivalent.

2.1.1 Computing PCA

To perform PCA on a data matrix $\boldsymbol{X}$ of size $D \times N$:

1. Compute the zero-centered data $\overline{\boldsymbol{X}}$ by subtracting the mean of the sample, $\overline{\boldsymbol{X}} = \boldsymbol{X} - \boldsymbol{M}$.
2. Calculate the covariance matrix $\boldsymbol{\Sigma} = \frac{1}{N}\overline{\boldsymbol{X}}\overline{\boldsymbol{X}}^T$.

3. Compute the eigenvectors and eigenvalues of the covariance matrix.

4. Compute the projection of $\overline{\boldsymbol{X}}$ on the K largest principal components $\boldsymbol{U}_K = [\boldsymbol{u}_1, \ldots, \boldsymbol{u}_K]$. By $\overline{\boldsymbol{Z}}_K = \boldsymbol{U}_K^T \overline{\boldsymbol{X}}$.

2.1.2 Successive Variance Maximization

Consider a set of observations $\{\boldsymbol{x}_n\}$, $n = 1, \ldots, N$ and $\boldsymbol{x}_n \in \mathbb{R}^D$. We want to project data onto a $K < D$ dimensional space while maximizing the variance of the projected data.

For $K = 1$ we define the direction of the projection as $\boldsymbol{u}_1$ and set $\|\boldsymbol{u}_1\|_2 = 1$; only the direction of the projection is important.

For the *original data* with sample mean $\overline{\boldsymbol{x}}$, we have the following covariance:

$$\boldsymbol{\Sigma} = \frac{1}{N} \sum_{n=1}^{N} (\boldsymbol{x}_n - \overline{\boldsymbol{x}})(\boldsymbol{x}_n - \overline{\boldsymbol{x}})^T$$

For the *projected data* with new mean $u_1^T \overline{\boldsymbol{x}}$, the variance is given by

$$\begin{aligned} \frac{1}{N} \sum_{n=1}^{N} \left\{ \boldsymbol{u}_1^T \boldsymbol{x}_n - \boldsymbol{u}_1^T \overline{\boldsymbol{x}} \right\}^2 &= \frac{1}{N} \sum_{n=1}^{N} \left\{ \boldsymbol{u}_1^T (\boldsymbol{x}_n - \overline{\boldsymbol{x}}) \right\}^2 \\ &= \frac{1}{N} \sum_{n=1}^{N} \boldsymbol{u}_1^T (\boldsymbol{x}_n - \overline{\boldsymbol{x}})(\boldsymbol{x}_n - \overline{\boldsymbol{x}})^T \boldsymbol{u}_1 \\ &= \boldsymbol{u}_1^T \boldsymbol{\Sigma} \boldsymbol{u}_1 \end{aligned}$$

We can formulate this as a maximization problem, where we maximize the variance of the projected data. We seek

$$\max_{\boldsymbol{u}_1} \boldsymbol{u}_1^T \boldsymbol{\Sigma} \boldsymbol{u}_1$$

such that $\|\boldsymbol{u}_1\|_2 = 1$. We use the Lagrangian of this problem

$$\mathcal{L} := \boldsymbol{u}_1^T \boldsymbol{\Sigma} \boldsymbol{u}_1 + \lambda_1 (1 - \boldsymbol{u}_1^T \boldsymbol{u}_1)$$

and set its partial derivative to zero:

$$\frac{\partial}{\partial \boldsymbol{u}_1} \mathcal{L} \stackrel{!}{=} 0 \qquad \Longrightarrow \qquad \boldsymbol{\Sigma} \boldsymbol{u}_1 = \lambda_1 \boldsymbol{u}_1$$

Therefore, $\boldsymbol{u}_1$ is an *eigenvector* of $\boldsymbol{\Sigma}$ and λ_1 is its associated *eigenvalue*. Furthermore, λ_1 is also the variance of the projected data, since $\lambda_1 = \boldsymbol{u}_1^T \boldsymbol{\Sigma} \boldsymbol{u}_1$. This is called the *principle direction*.

With a similar approach we will find that the second principle direction is the second largest eigenvector.

The *eigenvalue decomposition* of the covariance matrix

$$\boldsymbol{\Sigma} = \boldsymbol{U} \boldsymbol{\Lambda} \boldsymbol{U}^T$$

contains all relevant information.

For a $K \leq D$ dimensional projection space we can choose the K eigenvectors with the largest associated eigenvalues.

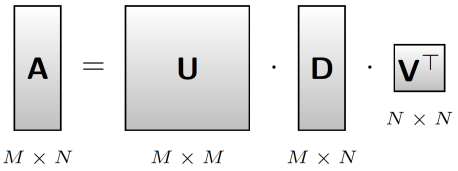

Figure 2: Singular Value Decomposition

2.1.3 Minimum Error Formulation

We define an *orthonormal* basis $\{\boldsymbol{u}_d\}$, $d = 1, \ldots, D$ of $\mathbb{R}^D$. The scalar projection of $\boldsymbol{x}_n$ onto $\boldsymbol{u}_d$ (magnitude) is given by

$$z_{n,d} = \boldsymbol{x}_n^T \boldsymbol{u}_d$$

The associated projection onto $\boldsymbol{u}_d$ amounts to $z_{n,d}\boldsymbol{u}_d$, and thus we can represent every data point by

$$\boldsymbol{x}_n = \sum_{d=1}^{D} z_{n,d}\boldsymbol{u}_d = \sum_{d=1}^{D} (\boldsymbol{x}_n^T \boldsymbol{u}_d)\boldsymbol{u}_d$$

2.1.4 Matrix Viewpoint

We can also consider PCA from a matrix factorization viewpoint by representing the data as a matrix

$$\boldsymbol{X} = [\boldsymbol{x}_1, \ldots, \boldsymbol{x}_N]$$

The corresponding zero-centered data is

$$\overline{\boldsymbol{X}} = \boldsymbol{X} - \boldsymbol{M} \qquad \text{where} \quad \boldsymbol{M} = [\overline{\boldsymbol{x}}, \ldots, \overline{\boldsymbol{x}}]$$

We factor $\overline{\boldsymbol{X}}$ by computing the projection of $\overline{\boldsymbol{X}}$ on $\boldsymbol{U}_K = [\boldsymbol{u}_1, \ldots, \boldsymbol{u}_K]$ (eigenvectors with K highest eigenvalues).

$$\overline{\boldsymbol{Z}}_K = \boldsymbol{U}_K^T \overline{\boldsymbol{X}}$$

To approximate $\overline{\boldsymbol{X}}$ we return to the original basis:

$$\tilde{\overline{\boldsymbol{X}}} = \boldsymbol{U}_K \cdot \overline{\boldsymbol{Z}}_K$$

Of course, if we use $K = D$, we obtain a perfect reconstruction.

2.2 Singular Value Decomposition

The *singular value decomposition* (SVD) is a widely used technique to decompose a matrix into several component matrices, exposing many of the useful and interesting properties of the original matrix like rank, nullspace, orthogonal basis of column and row space, etc. Applications which deploy SVD include computing pseudoinverses, least square fitting of data, matrix approximation, and determining the rank, range and null space of a matrix.

Theorem 2.1 (SVD Theorem)**.** *Let $\boldsymbol{A}$ be any real M by N matrix, $\boldsymbol{A} \in \mathbb{R}^{M \times N}$, then $\boldsymbol{A}$ can be decomposed as $\boldsymbol{A} = \boldsymbol{U}\boldsymbol{D}\boldsymbol{V}^T$.*

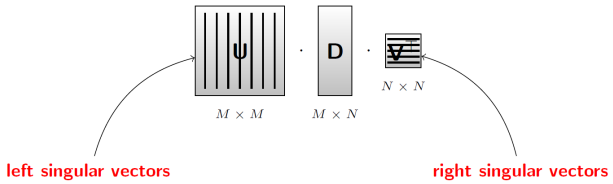

Figure 3: Left and right singular vectors

- $\boldsymbol{U}$ *is an* $M \times M$ *orthogonal matrix such that* $\boldsymbol{U}^T\boldsymbol{U} = I_{(M)}$.
- $\boldsymbol{D}$ *is an* $M \times N$ *diagonal matrix.*
- $\boldsymbol{V}^T$ *is an* $N \times N$ *orthogonal matrix such that* $\boldsymbol{V}^T\boldsymbol{V} = I_{(N)}$.

2.2.1 SVD Component Matrices

The elements of $\boldsymbol{D}$ are only non-zero on the diagonal, and are called the *singular values*. An important relation to the rank r of a matrix is

$$\begin{cases} d_{i,i} > 0 & 1 \leq i \leq r \\ d_{i,i} = 0 & r+1 \leq i \leq n \\ d_{i,j} = 0 & i \neq j \end{cases}$$

By convention, the ordering of the singular vectors is determined by high-to-low sorting of the singular values, with the highest singular value in the upper left corner of $\boldsymbol{D}$.

$$d_1 \geq d_2 \geq \cdots \geq d_r > d_{r+1} = \cdots = d_n = 0$$

The first r columns of $\boldsymbol{U}$ are called the *left singular vectors*, they form an orthogonal basis for the space spanned by the columns of the original matrix $\boldsymbol{A}$. Similarly the first r rows of $\boldsymbol{V}^T$ are the right singular vectors, they form an orthonormal basis for the row space of $\boldsymbol{A}$. This is illustrated in Figure 3.

We can also interpret the columns of $\boldsymbol{U}$ as the eigenvectors of $\boldsymbol{A}\boldsymbol{A}^T$. This can be verified by the SVD:

$$\boldsymbol{A}\boldsymbol{A}^T = \boldsymbol{U}\boldsymbol{D}\boldsymbol{V}^T\boldsymbol{V}\boldsymbol{D}\boldsymbol{U}^T = \boldsymbol{U}\boldsymbol{D}^2\boldsymbol{U}^T$$

Similarly for the rows of $\boldsymbol{V}^T$ (or the columns of $\boldsymbol{V}$), which are the eigenvectors of $\boldsymbol{A}^T\boldsymbol{A}$, since

$$\boldsymbol{A}^T\boldsymbol{A} = \boldsymbol{V}\boldsymbol{D}\boldsymbol{U}^T\boldsymbol{U}\boldsymbol{D}\boldsymbol{V}^T = \boldsymbol{V}\boldsymbol{D}^2\boldsymbol{V}^T$$

2.2.2 Singular Values

The singular values of a matrix $\boldsymbol{A}$ represent the *lengths* of the semi-axes of the hyper-ellipsoid E defined as $E = \{\boldsymbol{A}\boldsymbol{x} \mid \|\boldsymbol{x}\|_2 = 1\}$. Hence, they are non-negative.

2.2.3 SVD Interpretation

Suppose we are given a matrix $\boldsymbol{A}$ that records the ratings of movies by users (rows correspond to users, columns to movies). Then,

- $\boldsymbol{U}$: user-to-concepts affinity matrix.
- $\boldsymbol{V}$: movie-to-concepts affinity matrix.
- $\boldsymbol{D}$: the diagonal elements of $\boldsymbol{D}$ represent the "expressiveness" of each concept in the data.

2.2.4 Range, Nullspace, Rank

SVD provides an explicit representation of the range and nullspace of a matrix.

- The right singular vectors corresponding to vanishing singular values of $\boldsymbol{A}$ span the nullspace of $\boldsymbol{A}$.
- The left singular vectors correspond to the non-zero singular values of $\boldsymbol{A}$ and span the range of $\boldsymbol{A}$.

2.2.5 SVD to Reveal Structure

SVD can be viewed from three mutually compatible points of view:

1. SVD is a method for transforming correlated variables into a set of uncorrelated ones that better expose the various relationships among the original data items.
2. SVD is a method for identifying and ordering the dimensions along which data points exhibit the most variation.
3. SVD as data reduction method, once we have identified the directions of most variation we can obtain the best approximation of the original data points using fewer dimensions.

2.2.6 Closest Rank-k Matrix

We can use this decomposition to find the *k-closest approximation*. Let $\boldsymbol{A}$ be a matrix of rank R. If we wish to approximate $\boldsymbol{A}$ using a matrix of lower rank K, then

$$\boldsymbol{A}_k = \sum_{k=1}^{K} d_k \boldsymbol{u}_k \boldsymbol{v}_k^T$$

is the closest matrix in the L_2 norm. More formally, we have

$$\min_{\boldsymbol{B}, \operatorname{rank}(\boldsymbol{B})=k} \|\boldsymbol{A} - \boldsymbol{B}\|_2 = \|\boldsymbol{A} - \boldsymbol{A}_k\|_2$$

Moreover, the magnitudes of the nonzero singular values provide a measure of how close $\boldsymbol{A}$ is to a matrix of lower rank in the Euclidean norm:

$$\|\boldsymbol{A} - \boldsymbol{A}_k\|_2 = d_{k+1}$$

The *spectral norm* of a matrix $\boldsymbol{A}$ is the largest singular value of $\boldsymbol{A}$:

$$\|\boldsymbol{A}\|_2 = \sigma_{\max}(\boldsymbol{A})$$

2.2.7 Computing the SVD

Given a matrix $\boldsymbol{A}$

1. Compute its transpose $\boldsymbol{A}^T$ and then $\boldsymbol{A}^T\boldsymbol{A}$.
2. Determine the eigenvalues of $\boldsymbol{A}^T\boldsymbol{A}$ and sort them in descending order. Square the roots to obtain the singular values of $\boldsymbol{A}$.
3. Construct the diagonal matrix $\boldsymbol{D}$ by placing the singular values in descending order along its diagonal.
4. Use the eigenvalues from the second step to compute the eigenvectors of $\boldsymbol{A}^T\boldsymbol{A}$. Place them (in the right order) along the columns of $\boldsymbol{V}$ and compute $\boldsymbol{V}^T$.
5. Compute the inverse of the diagonal matrix, $\boldsymbol{D}^{-1}$, using $(\boldsymbol{D}^{-1})_{ii} = 1/D_{ii}^{-1}$. Compute the matrix $\boldsymbol{U} = \boldsymbol{A}\boldsymbol{V}\boldsymbol{D}^{-1}$.

2.2.8 SVD of Symmetric Matrices

Theorem 2.2. *If $\boldsymbol{S}$ is a real and symmetric matrix (i.e., $\boldsymbol{S} = \boldsymbol{S}^T$), then $\boldsymbol{S} = \boldsymbol{U}\boldsymbol{D}\boldsymbol{U}^T$ where the columns of $\boldsymbol{U}$ are the eigenvectors of $\boldsymbol{S}$ and $\boldsymbol{D}$ is a diagonal matrix with the eigenvalues of $\boldsymbol{S}$.*

2.2.9 Relationship to PCA

Given any $m \times n$ matrix $\boldsymbol{A}$,

- The matrix $\boldsymbol{A}\boldsymbol{A}^T$ is a symmetric matrix describing the similarity between the rows of the matrix $\boldsymbol{A}$. The eigenvectors $[\boldsymbol{u}_1, \ldots, \boldsymbol{u}_n]$ of $\boldsymbol{A}\boldsymbol{A}^T$ are the principle components in a PCA of $\boldsymbol{A}$ (equal up to a constant, assuming $\boldsymbol{A}$ is centered) and are the left singular vectors in the SVD decomposition of $\boldsymbol{A}$.
- The squared singular values of $\boldsymbol{A}$ are the eigenvalues of $\boldsymbol{A}\boldsymbol{A}^T$ and $\boldsymbol{A}^T\boldsymbol{A}$.

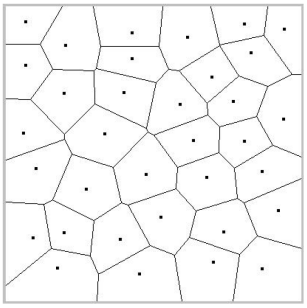

Figure 4: Quantization of the space $\mathbb{R}^D$

3 Clustering

Given a set of data points in a d-dimensional Euclidean space, we want to find a "meaningful" partition of the data. The partition should group together similar data points (put them in one group/cluster), while the different clusters should be as dissimilar as possible. Partitioning the data in this way can uncover similarities between data points and give rise to data compression schemes.

We can quantize the space $\mathbb{R}^D$ (as shown in Figure 4). A data point $\boldsymbol{x} \in \mathbb{R}^D$ is represented by one of the K *centroids* $\boldsymbol{u}_k \in \mathbb{R}^D$ for $k = 1, \ldots, K$. We assign $\boldsymbol{x}$ to the centroid $\boldsymbol{u}_{k^*}$ that has minimal distance.

Definition 3.1 (The Clustering Problem)**.** *Consider N data points in a D-dimensional space. Each data vector is denoted by $\boldsymbol{x}_n$, $n = 1, \ldots, N$. The goal is to partition the data into K clusters. More formally, we seek vectors $\boldsymbol{u}_1, \ldots, \boldsymbol{u}_K$ that represent the centroid of each cluster. A data point $\boldsymbol{x}_n$ belongs to the cluster k if the Euclidean distance between $\boldsymbol{x}_n$ and $\boldsymbol{u}_k$ is smaller than the distance to any other centroid.*

We want to minimize the following cost function

$$J(\boldsymbol{U}, \boldsymbol{Z}) = \|\boldsymbol{X} - \boldsymbol{U}\boldsymbol{Z}\|_2^2 = \sum_{n=1}^{N} \sum_{k=1}^{K} z_{k,n} \|\boldsymbol{x}_n - \boldsymbol{u}_k\|_2^2$$

The vectors $\boldsymbol{z}_n$ are the assignments of data points to clusters, and we can have various constraints on $\boldsymbol{Z}$. For now, we consider *hard assignments*, i.e., $\boldsymbol{Z} \in \{0, 1\}^{K \times N}$ with $\sum_k z_{k,n} = 1$ for all n. That is, one entry per column is set to 1.

We have two problems:

- Determine the set of cluster centroids $\boldsymbol{u}_k$.
- Compute the assignment matrix $\boldsymbol{Z}$.

Finding a solution to one of them (given the other) is analytically solvable.

We can write assignments (for hard assignments) down in two alternative notations. Either, we use a vector $\hat{\boldsymbol{z}} \in \{1, \ldots, K\}^N$ where every entry indicates (for each data point) to which cluster

index it is assigned to, e.g.,

$$\hat{\boldsymbol{z}} = \begin{pmatrix} 2 \\ 3 \\ 4 \\ 2 \\ 2 \\ 1 \end{pmatrix}$$

Alternatively, we can use a matrix $\boldsymbol{Z} \in \{0,1\}^{K \times N}$, e.g.,

$$\boldsymbol{Z} = \begin{pmatrix} 0 & 0 & 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 1 & 1 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \end{pmatrix}$$

3.1 The K-means Algorithm

We *alternate* between two steps:

- Assigning data points to clusters. Let $k^*(\boldsymbol{x}_n)$ denote the cluster index with the minimal distance between a cluster centroid and the data point:

$$k^*(\boldsymbol{x}_n) = \arg\min_k \left\{ \|\boldsymbol{x}_n - \boldsymbol{u}_1\|_2^2, \ldots, \|\boldsymbol{x}_n - \boldsymbol{u}_K\|_2^2 \right\}$$

- updating the cluster centroids based on the data points assigned to it. We compute the mean/centroid of a cluster as

$$\boldsymbol{u}_k = \frac{\sum_{n=1}^N z_{k,n} \boldsymbol{x}_n}{\sum_{n=1}^N z_{k,n}} \qquad \text{for all } k$$

Then, the K-means algorithm can be described as

1. Initiate with random choice for $\boldsymbol{u}_1^{(0)}, \ldots, \boldsymbol{u}_K^{(0)}$ (or use data points, or some other initialization) and set $t = 1$.
2. **Cluster assignment.** Solve for all n

$$k^*(\boldsymbol{x}_n) = \arg\min_k \left\{ \left\|\boldsymbol{x}_n - \boldsymbol{u}_1^{(t-1)}\right\|_2^2, \ldots, \left\|\boldsymbol{x}_n - \boldsymbol{u}_K^{(t-1)}\right\|_2^2 \right\}$$

3. **Centroid update.** The centroids are given by

$$\boldsymbol{u}_k^{(t)} = \frac{\sum_{n=1}^N z_{k,n}^{(t)} \boldsymbol{x}_n}{\sum_{n=1}^N z_{k,n}^{(t)}} \qquad \text{for all } k$$

4. Increment t and repeat from step 2 until $\left\|\boldsymbol{u}_k^{(t)} - \boldsymbol{u}_k^{(t-1)}\right\|_2^2 < \varepsilon$ for all k or $t = t_{\text{finish}}$.

The computational cost of each iteration is $\mathcal{O}(KN)$ and convergence is guaranteed. K-means optimizes a non-convex objective and hence we can only guarantee to find a local minimum. The algorithms can be used to compress data (lossy, if $K < N$ by only storing the centroids and the assignments of data points to clusters.

Problems of the approach include the following

- The non-convex objective guarantees only to find local minima, and the algorithm is sensitive to initializations. To some extent, this can be overcome by restarts.
- The optimal number of clusters K is unknown. This problem is referred to as *model order selection* or model validation.

3.2 Model Order Selection

We have a trade-off between two conflicting goals:

- **Data fit.** We want to predict the data well, e.g., for clustering

$$J(\boldsymbol{U}, \boldsymbol{Z}) = \|\boldsymbol{X} - \boldsymbol{U}\boldsymbol{Z}\|_2^2 = \sum_{n=1}^{N} \sum_{k=1}^{K} z_{k,n} \|\boldsymbol{x}_n - \boldsymbol{u}_k\|_2^2$$

  should ideally be zero.

- **Complexity.** We want a model that is not very complex, which is often measured by the number of free parameters.

We can use the *negative log-likelihood* of data for K-means:

$$-\ell(\boldsymbol{U}, \boldsymbol{Z} \mid \boldsymbol{X}) = J(\boldsymbol{U}, \boldsymbol{Z}) = \|\boldsymbol{X} - \boldsymbol{U}\boldsymbol{Z}\|_2^2 = \sum_{n=1}^{N} \sum_{k=1}^{K} z_{k,n} \|\boldsymbol{x}_n - \boldsymbol{u}_k\|_2^2$$

We can use (at least) two heuristics for choosing the parameter K. We achieve a balance between data fit (measured by the cost function $-\ell(\boldsymbol{U}, \boldsymbol{Z} \mid \boldsymbol{X})$) and complexity, which we measure by the number of free parameters $\kappa(\boldsymbol{U}, \boldsymbol{Z}) = K \cdot D$.

- **Akaike Information Criterion** (AIC)

$$\mathrm{AIC}(\boldsymbol{U}, \boldsymbol{Z} \mid \boldsymbol{X}) = -\ell(\boldsymbol{U}, \boldsymbol{Z} \mid \boldsymbol{X}) + \kappa(\boldsymbol{U}, \boldsymbol{X})$$

- **Bayesian Information Criterion** (BIC)

$$\mathrm{BIC}(\boldsymbol{U}, \boldsymbol{Z} \mid \boldsymbol{X}) = -\ell(\boldsymbol{U}, \boldsymbol{Z} \mid \boldsymbol{X}) + \tfrac{1}{2}\kappa(\boldsymbol{U}, \boldsymbol{X}) \cdot \ln N$$

We can then repeat the analysis for different values of K and choose the smallest AIC/BIC value.

3.3 Clustering Stability

The idea behind the notion of *stability* is the following concept. If we repeatedly sample data points from the same underlying model or of the same data generating process and apply a

clustering algorithm then a "good" algorithm should return clusterings that do not vary much from one sample to another. Using the K-means algorithm, this selection strategy reduces to choosing the optimal number of clusters for which stability is given.

We can use the following high-level stability test for a given set of data points and a given number of clusters:

1. Generate perturbed versions of the set (e.g., by adding noise or drawing sub-samples).
2. Apply the clustering algorithm on all versions.
3. Compute the pair-wise distances between all clusterings.
4. Compute the *instability* as the means distance between all clusterings.

This procedure can be repeated for different numbers of clusters and then one chooses the number that minimizes the instability.

To measure the distance between to clusterings that are defined on the same data points, we compute the distance between clusterings as follows:

1. Compute the distance between two by counting points on which the two clusterings disagree.
2. Repeat over all permutations of the cluster labels (as the same cluster might have different labels)
3. Choose the permutation with minimal distance.

In other words

$$d = \min_{\pi} \left\| \boldsymbol{Z} - \pi(\boldsymbol{Z}') \right\|_0$$

4 Soft Clustering

The term soft clustering is ambiguous, as it can refer to the algorithm or the model:

1. Algorithmic: soft K-means. In the K-means algorithms, a data point is assigned to one and only one cluster. A better estimation of the means would be to consider assigning to each cluster a probability that the data point belongs to it. This is known as the Gaussian mixture model (GMM). Note that the model still assumes that a data point belongs to only one cluster.
2. Model relaxation: One can also relax the "hard" constraint given by

   $$z_{k,n} \in \{0, 1\} \quad \text{and} \quad \sum_{k=1}^{K} z_{k,n} = 1$$

   and replace it with the following "soft" constraint:

   $$z_{k,n} \in [0, 1] \quad \text{and} \quad \sum_{k=1}^{K} z_{k,n} = 1$$

   This is known as *semi-NMF*.

Two modifications of our objective are possible:

1. **Gaussian mixture models.** Different clustering objective which arises from a generative probabilistic model. GMMs can not be written in matrix factorization form. The "softness" refers to the probability of hypotheses and, therefore, its nature is algorithmic.
2. **Semi non-negative matrix factorization.** Relax the constraints of the K-means objective to $z_{k,n} \in [0, \infty)$ and drop the constraint on the sum $\sum_{k=1}^{K} z_{k,n}$.

The motivation is to have probabilistic clustering model. We have a generative model:

- Goal: Explain the observed data $\{\boldsymbol{x}_n\}_{n=1}^{N}$ by a probabilistic model $p(\boldsymbol{x})$.
- We assume the parametric form of the model to be chosen a-priori and look at the GMM.
- The model has *parameters* that need to be learned in order to explain the observed data well. For instance, the K-means centroids $\boldsymbol{u}_k$ are an example of such parameters.

4.1 Probability Basics

4.1.1 Categorical Distribution

A categorical distribution is a discrete distribution over K events that assigns the probability π_k to the kth event. There are two codings for the random variable; the vector or index.

- Index coding. The random variable takes the index of the event that occurred.
- Vector coding. The random variable is a vector of size K and if event k occurs then the kth element is set to one, all other to zero.

We will use the vector coding, where the sample space is the set of 1-of-K encoded random vectors $\boldsymbol{z}$ of dimension K with

$$\sum_{k=1}^{K} z_k = 1 \quad \text{and} \quad z_k \in \{0, 1\}$$

The probability mass function is defined as

$$p(\boldsymbol{z} \mid \boldsymbol{\pi}) = \prod_{k=1}^{K} \pi_k^{z_k}$$

where π_k represents the probability of seeing element k. We have

$$\pi_k \geq 0 \quad \text{and} \quad \sum_{k=1}^{K} \pi_k = 1$$

4.1.2 Gaussian Distribution

The probability density function of the Gaussian distribution is given by

$$p(x \mid \mu, \sigma) = \mathcal{N}(x \mid \mu, \sigma) = \frac{1}{\sigma\sqrt{2\pi}} \exp\left(-\frac{(x-\mu)^2}{2\sigma^2}\right)$$

The variable μ stands for the mean of the distribution and is at the same place as the mode and the median. σ^2 is the variance. The probability for $X \in [a, b]$ is given as

$$P(a < X < b) = \int_a^n p(x)\, dx$$

4.1.3 Multivariate Gaussian Distribution

The multivariate Gaussian distribution is a generalization to higher dimension with sample space $\boldsymbol{\mathcal{X}} \subseteq \mathbb{R}^D$. A random variable $\boldsymbol{X} = (X_1, \ldots, X_D)^T$ has a multivariate normal distribution if every linear combination of its components (i.e., $Y = a_1 X_1 + \ldots + a_D X_D$ has a univariate normal distribution. Its definition is

$$p(\boldsymbol{x} \mid \boldsymbol{\mu}, \boldsymbol{\Sigma}) = \mathcal{N}(\boldsymbol{x} \mid \boldsymbol{\mu}, \boldsymbol{\Sigma}) = \frac{1}{(\sqrt{2\pi})^D |\boldsymbol{\Sigma}|^{1/2}} \exp\left(-\tfrac{1}{2}(\boldsymbol{x} - \boldsymbol{\mu})^T \boldsymbol{\Sigma}^{-1} (\boldsymbol{x} - \boldsymbol{\mu})\right)$$

4.1.4 Multidimensional Moment Statistics

The expectation is the vector of component expectations

$$\mathbb{E}[\boldsymbol{X}] = (\mathbb{E}[X_1], \ldots, \mathbb{E}[X_D])^T$$

The variance can then be generalized to the *covariance*:

$$\mathrm{Cov}(X, Y) = \int_{\mathcal{X}} \int_{\mathcal{Y}} p(x, y)(x - \mu_X)(y - \mu_Y)\, dxdy \qquad = \mathbb{E}_{X,Y}[(x - \mu_X)(y - \mu_Y)]$$

For random variables $X_1, \ldots, X_D$ we record the covariances in the covariance matrix $\boldsymbol{\Sigma}$ as $\boldsymbol{\Sigma}_{ij} = \mathrm{Cov}(X_i, X_j)$. This generalizes the notation of variance to sets of random variables for multiple dimensions.

4.1.5 Conditional Probability

Bayes' rule. The conditional probability of A given B (posterior) is given by

$$p(A \mid B) = \frac{p(B \mid A)p(A)}{p(B)}$$

We call $p(A)$ prior, $p(B \mid A)$ likelihood and $p(B)$ evidence.

4.2 Introduction to Mixture Models

Mixture of K probability densities is defined as

$$p(\boldsymbol{x}) = \sum_{k=1}^{K} \pi_k p(\boldsymbol{x} \mid \boldsymbol{\theta}_k)$$

Each probability distribution $p(\boldsymbol{x} \mid \boldsymbol{\theta}_k)$ is a *component* of the mixture and has its own parameter $\boldsymbol{\theta}_k$. Almost any continuous density can be approximated using a sufficient number of component

distributions. For a Gaussian component distribution the parameters $\boldsymbol{\theta}_k$ are given by the mean $\boldsymbol{\mu}_k$ and the covariance $\boldsymbol{\Sigma}_k$.

Mixture models are constructed from (1) the component distributions of the form $p(\boldsymbol{x} \mid \boldsymbol{\theta}_k)$ and (2) the mixing coefficient π_k that give the probability of each component. In order for $p(\boldsymbol{x})$ to be a proper distribution we have to ensure that

$$\pi_k \geq 0 \quad \text{and} \quad \sum_{k=1}^{K} \pi_k = 1$$

Thus, the parameters π_k define a categorical distribution which represents the probability of each component distribution.

4.3 Gaussian Mixture Models

The Gaussian mixture model (GMM) uses Gaussians as the component distributions; thus

$$p(\boldsymbol{x}) = \sum_{k=1}^{K} \pi_k \mathcal{N}(\boldsymbol{x} \mid \boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k)$$

Given data points $\{\boldsymbol{x}_1, \ldots, \boldsymbol{x}_N\}$ we then aim at learning the parameters $\boldsymbol{\mu}_k$ and $\boldsymbol{\Sigma}_k$, such that we approximate the data as closely as possible. This is equivalent to finding the parameters that maximize the likelihood of the data.

Generative viewpoint. Given the model parameters $\boldsymbol{\Sigma}$, $\boldsymbol{\mu}$, and $\boldsymbol{\pi}$, sample the data $\boldsymbol{x}_n$ as follows:

1. Sample a cluster index k according to the probabilities π_k.
2. Sample a data point $\boldsymbol{x}_n$ from the distribution $p(\boldsymbol{x}_n \mid \boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k)$.

Parameter estimation is then to revert this process and go from the data to the parameters.

To estimate the parameters, we assume that the data points $\boldsymbol{x}$ are independent and identically distributed (i.i.d.). The probability or likelihood of the observed data $\boldsymbol{X}$, given the parameters can thus be written as

$$p(\boldsymbol{X} \mid \boldsymbol{\pi}, \boldsymbol{\mu}, \boldsymbol{\Sigma}) = \prod_{n=1}^{N} p(\boldsymbol{x}_n) = \prod_{n=1}^{N} \sum_{k=1}^{N} \pi_k \mathcal{N}(\boldsymbol{x} \mid \boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k)$$

We aim at finding at finding the parameters that maximize the likelihood of the data:

$$(\hat{\boldsymbol{\pi}}, \hat{\boldsymbol{\mu}}, \hat{\boldsymbol{\Sigma}}) \in \arg\max_{\boldsymbol{\pi}, \boldsymbol{\mu}, \boldsymbol{\Sigma}} p(\boldsymbol{X} \mid \boldsymbol{\pi}, \boldsymbol{\mu}, \boldsymbol{\Sigma})$$

To simplify the expression, we take the logarithm, such that the product becomes a sum:

$$(\hat{\boldsymbol{\pi}}, \hat{\boldsymbol{\mu}}, \hat{\boldsymbol{\Sigma}}) \in \arg\max_{\boldsymbol{\pi}, \boldsymbol{\mu}, \boldsymbol{\Sigma}} \sum_{n=1}^{N} \ln \left[\sum_{k=1}^{N} \pi_k \mathcal{N}(\boldsymbol{x} \mid \boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k) \right]$$

This equation does not have a closed-form analytic solution, and here we present a solution based on *expectation maximization*. The intuition is as follows: if we know to which clusters the data points are assigned, then computing the maximum likelihood is easy. First, we need to introduce a latent (or hidden) variable for the assignments of data points to clusters:

- We define a K-dimensional binary random variable $\boldsymbol{z}$ having a 1-of-K representation with

$$\sum_{k=1}^{K} z_k = 1 \quad \text{and} \quad z_k \in \{0, 1\}$$

- The marginal distribution over $\boldsymbol{z}$ is specified in terms of the mixing coefficients π_k, i.e.,

$$p(z_k = 1) = \pi_k$$

Because $\boldsymbol{z}$ uses a 1-of-K representation, we can write this distribution in the form

$$p(\boldsymbol{z}) = \prod_{k=1}^{K} \pi_k^{z_k}$$

Similarly, the conditional distribution of $\boldsymbol{x}$ given a particular value for $\boldsymbol{z}$ is a Gaussian

$$p(\boldsymbol{x} \mid z_k = 1) = \mathcal{N}(\boldsymbol{x} \mid \boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k)^{z_k}$$

The conditional probability can be written as

$$p(\boldsymbol{x} \mid \boldsymbol{z}) = \prod_{k=1}^{K} \mathcal{N}(\boldsymbol{x} \mid \boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k)^{z_k}$$

We can use Bayes' rule to get the probability of assigning a data point to a cluster:

$$\begin{aligned} \gamma(z_k) := p(z_k = 1 \mid \boldsymbol{x}) &= \frac{p(z_k = 1)p(x \mid z_k = 1)}{\sum_{j=1}^{K} p(z_j = 1)p(x \mid z_j = 1)} \\ &= \frac{\pi_k \mathcal{N}(\boldsymbol{x} \mid \boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k)}{\sum_{j=1}^{K} \pi_j \mathcal{N}(\boldsymbol{x} \mid \boldsymbol{\mu}_j, \boldsymbol{\Sigma}_j)} \end{aligned}$$

4.4 Expectation Maximization for GMM

Expectation maximization is a powerful method for finding maximum likelihood solutions for models with latent variables. Setting the derivatives of $\ln p(\boldsymbol{X} \mid \boldsymbol{\pi}, \boldsymbol{\mu}, \boldsymbol{\Sigma})$ with respect to the means $\boldsymbol{\mu}_k$ to zero and multiplying with $\boldsymbol{\Sigma}_k$ we obtain:

$$\boldsymbol{\mu}_k = \tfrac{1}{N_k} \sum_{n=1}^{N} \gamma(z_{k,n}) \boldsymbol{x}_n \qquad N_k = \sum_{n=1}^{N} \gamma(z_{k,n})$$

The mean $\boldsymbol{\mu}_k$ is obtained by taking a weighted mean of all the points in the data set. And further

$$\pi_k = \frac{N_k}{N}$$

Informally, we first choose some initial values for the means and mixing coefficients. Then, we alternate between two updates called the E and M step:

1. In the expectation step, the current values for the parameters are used to evaluate the posterior probability $\gamma(z_{k,n})$.

2. In the maximization step, these probabilities are used to re-estimate the means and mixing coefficients.

More formally, given a GMM, the goal is to maximize the likelihood function with respect to the parameters.

1. Initialize the means $\boldsymbol{\mu}_k$, and the mixing coefficients π_k. Set the $\boldsymbol{\Sigma}_k$ to the given covariances.
2. **E-step.** Evaluate the responsibilities using the current parameter values:

$$\gamma(z_{k,n}) = \frac{\pi_k \mathcal{N}(\boldsymbol{x}_n \mid \boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k)}{\sum_{j=1}^{K} \pi_j \mathcal{N}(\boldsymbol{x}_n \mid \boldsymbol{\mu}_j, \boldsymbol{\Sigma}_j)}$$

3. **M-step.** Re-estimate the parameters using the current responsibilities.

$$\begin{aligned} \boldsymbol{\mu}_k^{\text{new}} &= \frac{1}{N_k} \sum_{n=1}^{N} \gamma(z_{k,n}) \boldsymbol{x}_n \\ \pi_k^{\text{new}} &= \frac{N_k}{N} \qquad N_k = \sum_{n=1}^{N} \gamma(z_{k,n}) \end{aligned}$$

4. Evaluate the log likelihood and check for convergence of either the parameters or the log likelihood.

5 Sampling

The basic idea is to *randomly* draw configurations from the probability distribution. Probable configurations appear frequently. This method is convenient, as

- solutions are in the hypothesis space,
- solutions obey to the probability distribution they are sampled from, and
- it works with difficult distributions (e.g., when normalization is infeasible).

5.1 Monte Carlo Methods

The first problem is to generate samples. That is, for a given probability distribution $p(\boldsymbol{X})$, generate samples $\{\boldsymbol{x}^{(r)} \mid 1 \leq r \leq R\}$. Instead of performing an integration we can evaluate a sum:

$$\frac{1}{R} \sum_{r=1}^{R} f(\boldsymbol{x}^{(r)}) \overset{R \to \infty}{\longrightarrow} \int_{\Omega} f(\boldsymbol{x}) \, d^D \boldsymbol{x}$$

The second problem is expectation. Estimate the expectation of a given function $f(\boldsymbol{x})$ under this distribution:

$$\mathbb{E}\left[f(\boldsymbol{x})\right] = \int p(\boldsymbol{x}) f(\boldsymbol{x}) \, d^D \boldsymbol{x}$$

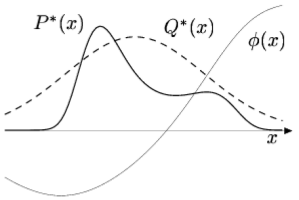

Figure 5: Importance Sampling.

5.1.1 Expectation from Samples

The empirical distribution is given as

$$\hat{p}(\boldsymbol{x}) = \frac{1}{R}\sum_{r=1}^{R}\delta(\boldsymbol{x} - \boldsymbol{x}^{(r)})$$

If we can solve the sampling problem, then we can solve the expectation problem by using the empirical mean

$$\hat{\mathbb{E}}[f(\boldsymbol{x})] = \int \hat{p}(\boldsymbol{x}) f(\boldsymbol{x})\, d^D\boldsymbol{x} = \frac{1}{R}\sum_{r=1}^{R} f(\boldsymbol{x}^{(r)})$$

If $\{\boldsymbol{x}^{(r)} \mid 1 \leq r \leq R\}$ are generated from $p(\boldsymbol{x})$, then the expectation of $\hat{\mathbb{E}}[f(\boldsymbol{x})]$ is $\mathbb{E}[f(\boldsymbol{x})]$ by the law of large numbers.

5.2 Importance Sampling

Importance sampling is a method for estimating the expectation of the function $f(\boldsymbol{x})$. Assume we can evaluate $p^*(\boldsymbol{x})$ at any given point $\boldsymbol{x}$:

$$p(x) = \frac{p^*(\boldsymbol{x})}{Z_p}$$

where Z_p is unknown. Assume that we cannot sample from $p(\boldsymbol{x})$ but we have a simpler density $q(\boldsymbol{x})$ from which we can sample.

$$q(x) = \frac{q^*(\boldsymbol{x})}{Z_q}$$

is called the sampler density.

Some samples are under represented, other are over represented, so we correct by weights

$$w_r = \frac{p^*(\boldsymbol{x}^{(r)})}{q^*(\boldsymbol{x}^{(r)})}$$

which adjust the *importance* of each point. The expectation is thus

$$\hat{\mathbb{E}}[f(\boldsymbol{x})] = \frac{\sum_r w_r f(\boldsymbol{x}^{(r)})}{\sum_r w_r}$$

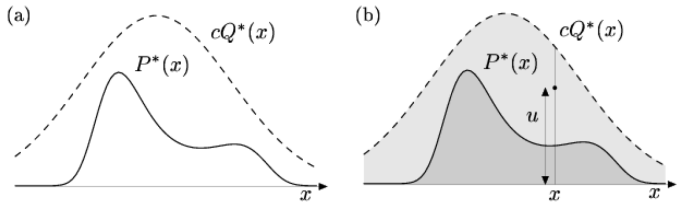

Figure 6: Rejection Sampling.

5.3 Rejection Sampling

Rejection sampling allows us to sample from relatively complex distributions. Assume that we can evaluate $p^*(\boldsymbol{x})$ at any point $\boldsymbol{x}$,

$$p(\boldsymbol{x}) = \frac{p^*(\boldsymbol{x})}{Z_p}$$

where Z_p is unknown. Assume we know a constant c such that

$$c \cdot q^*(x) > p^*(x) \qquad \text{for all } x$$

The algorithm then is as follows:

1. Generate x from the proposal density $q(x)$.
2. Generate a uniformly distributed random variable u from the interval $[0, c \cdot q^*(x)]$.
3. If $u > p^*(x)$ then x is rejected, otherwise it is added to our samples $\{x^{(r)}\}$.

Rejection sampling works best if q is a good approximation of p. If q is very different from p, then c may be very large (especially in heigh dimensions). Estimating the value of c may be hard, since we may not know where the modes of p^* are, or how high they may be.

6 Role Mining and Multi-Assignment Clustering

A simple way to manage access control is called *discretionary access control*. Assume we have N users and D permissions, then we use a binary access control matrix $\boldsymbol{X} \in \{0,1\}^{D \times N}$ where x_{di} indicates whether user i has access to resource d.

This simple idea has the problem of being tedious and expensive to maintain, and irregularities (i.e., potential errors) are very difficult to detect. Such irregularities might be special permissions that are given to someone on special-duty (need to be revoked later) or are simply mistakes.

Missing permissions are of course not a problem, but if a user has unnecessary permissions, someone might cause damage accidentally or on purpose. Thus, the *principle of least privilege* demands that users have only the privileges needed to fulfil their job. Detecting non-regular permissions and assessing their risk increases the security on IT systems.

6.1 Role-Based Access Control

Role-based access control (RBAC) simplifies access control and is, today, the method of choice. The basic entities of RBAC are:

- USERS, ROLES, OBJ, OPS, the set of users, roles, objects, and operations.
- PRMS $\subseteq \{(obj, op) \mid obj \in \text{OBJ} \wedge op \in \text{OPS}\}$, the set of permissions.
- UA $\subseteq$ USERS $\times$ ROLES, a user-role assignment relation.
- PA $\subseteq$ PRMS $\times$ ROLES, a permission-role assignment relation.
- UPA $\subseteq$ USERS $\times$ PRMS, a user-permission assignment relation.
- assigned_users$(R) = \{u \in \text{USERS} \mid (u, R) \in \text{UA}\}$
- assigned_permissions$(R) = \{p \in \text{PRMS} \mid (p, R) \in \text{PA}\}$

Each role defines a set of permissions. Users are assigned to a set of roles and get all permissions of these roles.

We define the boolean matrix product, denoted by $\otimes$ as

$$\boldsymbol{X} = \boldsymbol{U} \otimes \boldsymbol{Z} \qquad \Longleftrightarrow \qquad x_{di} = \bigvee_k (u_{dk} \wedge z_{ki})$$

6.2 Problem Definition

We are interested in finding good roles and role assignment based on a given user-permission matrix $\boldsymbol{X}$. That is, given $\boldsymbol{X}$, find $\boldsymbol{U}$ and $\boldsymbol{Z}$ such that

$$\boldsymbol{X} \approx \boldsymbol{U} \otimes \boldsymbol{Z}$$

with $\boldsymbol{X} \in \{0,1\}^{D \times N}$, $\boldsymbol{U} \in \{0,1\}^{D \times K}$, and $\boldsymbol{Z} \in \{0,1\}^{K \times N}$. Note that $\boldsymbol{X}$ is typically noisy, i.e., it contains permission assignments without an underlying role.

There are three different definitions of the role-mining problem:

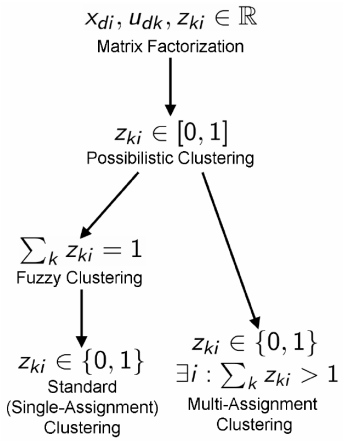

Figure 7: The matrix factorization landscape. Role mining is a multi-assignment clustering problem with $x_{di}, u_{dk} \in \{0,1\}$.

1. **Min-noise approximation.** Given K, find matrices $\boldsymbol{U} \in \{0,1\}^{D \times K}$ and $\boldsymbol{Z} \in \{0,1\}^{K \times N}$ as
$$(\hat{\boldsymbol{U}}, \hat{\boldsymbol{Z}}) = \arg\min_{\boldsymbol{U},\boldsymbol{Z}} \|\boldsymbol{X} - \boldsymbol{U} \otimes \boldsymbol{Z}\|_1$$

2. δ**-approximation.** Given $\delta \geq 0$, find the minimal number of roles K and matrices $\boldsymbol{U}$ and $\boldsymbol{Z}$ such that
$$\|\boldsymbol{X} - \boldsymbol{U} \otimes \boldsymbol{Z}\|_1 < \delta$$

3. **Inference.** Assume that the given matrix $\boldsymbol{X}$ has an underlying structure $\boldsymbol{X}^S = \boldsymbol{U} \otimes \boldsymbol{Z}$ which was altered by a noise process $\boldsymbol{X}^S \sim P(\boldsymbol{X}^S; \Theta^S)$, $\boldsymbol{X} \sim P(X; \boldsymbol{X}^S; \Theta^N)$. Find the structure encoding in $\boldsymbol{U}$ and $\boldsymbol{Z}$.

6.3 Evaluation Criteria for Boolean Matrix Decomposition

For each role mining problem definition, there is a set of evaluation criteria:

1. **Matrix reconstruction.** How accurately does the solution reconstruct the input matrix $\boldsymbol{X}$?

2. **Number of sources.** How many sources K do you need to obtain a given accuracy?

3. **Inference quality.** How accurately can you infer the underlying structure?

4. **Generalization.** How precisely can you roles $\boldsymbol{U}$ reconstruct a *new* matrix $\boldsymbol{X}'$ (with the same distribution as $\boldsymbol{X}$)?

5. **Stability.** How consistent are the solutions of $\boldsymbol{X}'$ and $\boldsymbol{X}$.

6.3.1 Evaluating a Matrix Reconstruction

To measure the discrepancy between $\boldsymbol{X}$ and its inferred solution, we have several possibilities:

- **Deviation.** Compute (used to define the min-noise approximation):

$$\frac{1}{N \cdot D} \|\boldsymbol{X} - \boldsymbol{U} \otimes \boldsymbol{Z}\|_1$$

- **Coverage.** The ratio of true 1's which are reconstructed as 1's:

$$\text{Cov} := \frac{|\{(i,j) \mid \hat{x}_{ij} = x_{ij} = 1\}|}{|\{(i,j) \mid x_{ij} = 1\}|}$$

- **Deviating ones (zeros).** The number of deviating 1's (0's) is the number of matrix elements which are wrongly reconstructed divided by the total number of 1's (0's) in $\boldsymbol{X}$.

$$d_\xi := \frac{|\{(i,j) \mid \hat{x}_{ij} = 1 - \xi \wedge x_{ij} = \xi\}|}{|\{(i,j) \mid x_{ij} = \xi\}|} \qquad \xi \in \{0,1\}$$

Note that these measures have trivial optimal solutions, and if $\boldsymbol{X}$ is noisy, one might not be interested in reconstructing every single bit.

6.3.2 Evaluation Inference Quality

Three more adequate (but more involved) measures are:

- **Parameter accuracy.** If you know the true parameters Θ describing the roles, compare with your estimators $\hat{\Theta}$.

  Compute the Hamming distance d_H between the estimated roles $\hat{\boldsymbol{U}}$ and the true roles $\boldsymbol{U}$:

$$d_H(\boldsymbol{U}, \hat{\boldsymbol{U}}) := \min_{\pi \in P_K} \frac{1}{K \cdot D} \sum_{k=1}^{K} \sum_{d=1}^{D} \left| \hat{u}_{d,k} - u_{d,\pi(k)} \right|$$

  where P_K is the set of all permutations over K elements. Note that the true source must be given, which is usually not the case.

- **Stability** of the clustering solution. Cluster two matrices $\boldsymbol{X}, \boldsymbol{X}' \sim P(\boldsymbol{X}; \Theta)$. Do you get similar clusterings?

  In unsupervised learning, there is no fixed labelling of the clusters. Assume we have 2 clustering solutions $\mathcal{K}_1$ and $\mathcal{K}_2$ with clusters numbered $1, \ldots, K$ and they are defined by the elements they contain. Also, we know the costs $c(k_1, k_2)$ for matching cluster k_1 from $\mathcal{K}_1$ to k_2 from $\mathcal{K}_2$. To measure the match between $\mathcal{K}_1$ and $\mathcal{K}_2$ we need to find the optimal mapping π^* between the clusters, or more formally

$$\pi^* = \arg\min_{\pi} \left\{ \sum_k c(k, \pi(k)) \right\}$$

  The permutation π^* can be efficiently found using the *Hungarian* method.

- **Generalization ability.** How well can you explain new entries with your estimated roles?

  To measure how well the inferred set of roles generalizes to a new data matrix $\boldsymbol{X}'$, we can select a random subset $\mathcal{D}^* \subset \{1, \ldots, D\}$ of the permissions (usually approx. half of the permissions). For every column i of $\boldsymbol{X}'$ (i.e., for every new user)

  1. Compute $\hat{\boldsymbol{z}}_{\cdot,i}$ the optimal role combination of user i:

  $$\hat{\boldsymbol{z}}_{\cdot,i} := \arg\min_{\boldsymbol{z}_i} \left\{ \sum_{d \in \mathcal{D}^*} \left| x'_{d,i} - \boldsymbol{u}_{d,\cdot} \otimes \boldsymbol{z}_{\cdot,i} \right| \right\}$$

  2. Compute the Hamming distance between the predicted and the actual permissions of user i

  $$g_i := \frac{1}{D - |\mathcal{D}^*|} \sum_{d \notin \mathcal{D}^*} \left| x'_{d,i} - \boldsymbol{u}_{d,\cdot} \otimes \hat{\boldsymbol{z}}_{\cdot,i} \right|$$

  The average over all values is the generalization ability G:

  $$G := \tfrac{1}{N} \sum_i g_i$$

6.4 SVD for Role Mining

Naively taking the SVD of $\boldsymbol{X}$ and then rounding the left and right singular values to boolean values is a bad idea and does not work well. However, it can be used for *denoising*:

1. Compute the singular value decomposition:

$$\boldsymbol{X} = \boldsymbol{U} \cdot \boldsymbol{S} \cdot \boldsymbol{V}^T$$

2. Discard columns $K+1, \ldots, D$ of $\boldsymbol{U}$ to get $\boldsymbol{U}_{(K)}$, discard rows and columns $K+1, \ldots, N$ of $\boldsymbol{S}$ to get $\boldsymbol{S}_{(K)}$ and discard rows $K+1, \ldots, N$ of $\boldsymbol{V}$ to get $\boldsymbol{V}_{(K)}$.

3. Round the reconstruction

$$\boldsymbol{X}_{(K)} := \boldsymbol{U}_{(K)} \cdot \boldsymbol{S}_{(K)} \cdot \boldsymbol{V}_{(K)}^T$$

   to get

$$\hat{\boldsymbol{X}}_{(K)} = (\boldsymbol{X}_{(K)} > t_{\boldsymbol{X}})$$

   where $t_{\boldsymbol{X}}$ is a threshold.

For this to work, we actually need three values:

- a threshold $t_{\boldsymbol{X}}$ to be chosen depending on the data
- a *hyper-parameter* K, the number of dimensions
- an estimate for the reconstruction error

6.5 Cross-Validation

For choosing the right parameters, we need three disjoint data sets. If there is only one data set available, then one needs to split it into three parts:

- The *training data* is used to estimate the best value of the parameters θ. For the application above, $\theta = t_{\boldsymbol{X}}$, given a fixed K.
- The *validation data* is used to determine the best value of the hyper-parameters θ^H (for the best value of θ). In our setting $\theta^H = K$.
- The *test data* is then used to estimate the reconstruction error.

On a high level, the algorithm for choosing a hyper-parameter works as follows:

1. Get three disjoint data (sub)sets.
2. For all possible values of the hyper-parameter θ^H:
   i) Fix θ^H, and
   ii) *train* you method on the training data to get θ^*, and
   iii) *validate* your method on the validation data.
3. Set
$$\theta^{H*} = \begin{pmatrix}\text{Value of } \theta^H \text{ that maximized the performance on the} \\ \text{validation data (with respective best value for } \theta)\end{pmatrix}$$

Note that the test data is *not* used in the complete process.
Once the optimal hyper-parameter θ^{H*} has been found, the following procedure can be used to evaluate the model:

1. Fix θ^{H*}.
2. Train your method to get the optimal value θ^* on the training data and the validation data.
3. Test your method with θ^{H*} and θ^* on the *test data*. This is the final result of your method.

It is (implicitly) assume that the data in the three parts has the *same distribution*. The division into three parts is random and reduces the number of training data. To avoid too much dependency on this choice, one often uses several splits. This is called *cross-validation*.
Cross-validation allows one to estimate how well a method will perform on a new, unseen data set. This procedure avoids overfitting, i.e., a too detailed adaptation to the training data which renders the method very specific to the training data. Therefore, the test data is only considered at the very end.

6.6 Simple Approach Using a Variant of K-means

We can use the idea of K-means and adapt the algorithm to boolean data:

- Adapt distance to *Hamming distance* (0-norm):

$$R(\boldsymbol{U}, \boldsymbol{Z}) = \|\boldsymbol{X} - \boldsymbol{U}\boldsymbol{Z}\|_0 = \sum_{i=1}^{N} \sum_{k=1}^{K} z_{ki} \|\boldsymbol{x}_i - \boldsymbol{u}_k\|_0$$

- Restrict centroids $\boldsymbol{u}_k$ to boolean values and adapt the centroid update step:

$$u_{kd} = \text{median}(\{x_{dn} \mid z_{kn} = 1\}) \qquad \text{for all } k \text{ and } d$$

However, this approach has several problems:

- The parameter K has to be given (use cross-validation)
- K-means yields *disjoint* clustering, but in RBAC a user can have more than one role.

6.7 The RoleMiner Algorithm

RoleMiner is one of the first role mining approaches (heuristic) and comes in two variants; CompleteMiner and FastMiner.

The algorithms uses the observation that users are described by sets of permissions, and that roles are defined as sets of permissions. The key idea is that a user's permissions are generated by a combination of roles, and thus all possible intersections of permission sets of all users should give the underlying components (the *atomic* roles, so to say). For instance, for users (represented as permission sets) $\{1, 2, 5\}$ and $\{2, 3, 5\}$ we would have the following candidate roles: $\{1, 2, 5\}$, $\{2, 3, 5\}$ and $\{2, 5\}$.

Algorithm 1 The CompleteMiner algorithm

Input: x

1: Extract all unique permission sets (columns) $\boldsymbol{x}'$ from the input.
2: Compute all unique set intersections between the members of $\boldsymbol{x}'$ and add them to the set of k candidate roles $\boldsymbol{u}$.
3: **for** each candidate in $\boldsymbol{u}$ **do**
4: Count how many users have the permission set of the candidate as a subset.
5: **end for**
6: **return** The prioritized set of k roles $\boldsymbol{u}$, where the counts $c = (c_1, \ldots, c_k)$ give the priority.

A variant of the CompleteMiner is FastMiner, where only the set of *pairwise* intersections is computed. If t is the number of initial roles (i.e., the number of users with unique permission sets), then CompleteMiner has to compute $2^t - t - 1$ set intersections, and FastMiner only $\frac{t(t-1)}{2}$.

Clearly, there are various problems with the RoleMiner algorithm:

- The algorithm is very sensitive to noise:
  - If only a few individual bits are noisy, the number of candidate roles gets much larger.
  - Slightly different noise leads to a completely different solution.
- No assignments of users to roles, only candidate for the columns of $\boldsymbol{U}$ are given ($\boldsymbol{Z}$ is neglected).

6.8 DBPsolver

DBPsolver approximately solves the *discrete basis problem.*

Definition 6.1. *For a given boolean matrix* $\boldsymbol{X} \in \{0,1\}^{D \times N}$ *and a number* K *of basis vectors, find a boolean matrix* $\boldsymbol{U} \in \{0,1\}^{D \times K}$ *and a boolean matrix* $\boldsymbol{Z} \in \{0,1\}^{K \times N}$ *minimizing*

$$\|\boldsymbol{X} - \boldsymbol{U} \otimes \boldsymbol{Z}\|_1$$

where $\|\cdot\|_1$ *is the* L_1 *norm:* $\|M\|_1 = \sum_{i=1}^{m} \sum_{n=1}^{n} |M_{ij}|$

Note that the discrete basis problem and the min-noise role mining problem are identical.

The idea of DBPsolver is the following. To cover most of the 1's in $\boldsymbol{X}$ with as few basis vectors as possible, candidates should have 1's at dimensions (e.g., permissions) that are frequently 1 simultaneously.

DBPsolver computes a $D \times D$ association matrix $\boldsymbol{A}$ where $a_{dd'}$ is the *pairwise association* between dimensions (e.g., permissions) d and d':

$$a_{dd'} := a(d \Rightarrow d') = \frac{\langle \boldsymbol{x}_{d,\cdot}, \boldsymbol{x}_{d',\cdot} \rangle}{\langle \boldsymbol{x}_{d,\cdot}, \boldsymbol{x}_{d,\cdot} \rangle}$$

We have $a_{dd'} = \hat{p}\{d' = 1 \mid d = 1\}$. In role mining $a_{dd'}$ is the empirical probability of having permission d' conditioned on having d.

The algorithm now works greedily as follows:

1. Compute the association matrix $\boldsymbol{A}$.
2. Compute $\boldsymbol{A}' = (\boldsymbol{A} > \tau)$ by thresholding $\boldsymbol{A}$ with a predefined τ. Each column in $\boldsymbol{A}'$ is a candidate basis vector. In role mining, it represents a candidate role.
3. Initialize $\boldsymbol{U} = \{0\}^{D \times K}$ and $\boldsymbol{Z} = \{0\}^{K \times N}$.
4. For $k = 1, \ldots, K$: substitute column k in $\boldsymbol{U}$ with column t from A' and modify $\boldsymbol{z}_k$ such that

$$\begin{aligned} R(\boldsymbol{X}, \boldsymbol{Z}, \boldsymbol{U}, w^+, w^-) = & w^+ \cdot \left| \left\{ (i,d) \mid x_{di} = 1 \wedge \left(\boldsymbol{U}_{(t)} \otimes \boldsymbol{Z}_{(t)} \right)_{di} = 1 \right\} \right| - \\ & w^- \cdot \left| \left\{ (i,d) \mid x_{di} = 0 \wedge \left(\boldsymbol{U}_{(t)} \otimes \boldsymbol{Z}_{(t)} \right)_{di} = 1 \right\} \right| \end{aligned}$$

   is maximized.

w^+ and w^- are predefined parameters that reward and penalize covering 0's and 1's with a 1.

DBPsolver has the following problems:

- The parameters τ, w^+, w^- and K must be chosen (cross-validation).
- Greedy optimization does not find the *best* combination of basis vectors, but only picks a single best candidate at every step and ignores possible combinations with other vectors.
- Finding the best matrix approximation differs from discovering structure and noise! The best fit to $\boldsymbol{X}$ might include noise ("overfitting"). This cannot be avoided and is intrinsic to the problem being solved (i.e., we asked the wrong question).

7 Sparse Coding

A signal and its representation are not the same thing, and there is an infinite number of possible representations.

Given a signal f and an orthonormal basis $\{u_1, \ldots, u_L\}$, we can represent the signal by coefficients

$$z_l = \langle f, u_l \rangle$$

The signal f can then be reconstructed as

$$f = \sum_{l=1}^{L} z_l u_l = \sum_{l=1}^{L} \langle f, u_l \rangle u_l$$

In sparse coding, we hope that we get only few non-zero coefficients. We can compress by dropping some of the coefficients and only using certain basis functions for a subset σ:

$$\hat{f} = \sum_{k \in \sigma} z_k u_k$$

In this case, the reconstruction error is defined as

$$\left\| f - \hat{f} \right\|^2 = \sum_{k \notin \sigma} |\langle f, u_k \rangle|^2$$

since $\langle u_k, u_k \rangle = 0$ for $k \neq l$.

Of course, the choice of basis is critical, and there is no choice of transform that is better than all others, as it depends on the type of the signal. For instance, the Fourier basis has global support and is good for "sine like" signals, but performs poor for localized signals. On the other hand, a Wavelet basis has local support but is poor for non-vanishing signals.

7.1 Comparison to Principle Component Analysis

We can compress a set of vectors $\boldsymbol{X} = [\boldsymbol{x}_1, \ldots, \boldsymbol{x}_n]$ by performing PCA. That is, we compute the (centred) covariance matrix

$$\boldsymbol{\Sigma} = \tfrac{1}{n} (\boldsymbol{X} - \boldsymbol{M})(\boldsymbol{X} - \boldsymbol{M})^T$$

and compute its eigenvector decomposition, giving us $\boldsymbol{U}$ and $\boldsymbol{\Lambda}$. We then choose the k eigenvectors corresponding to the largest eigenvalues as $\boldsymbol{U}_k$, and compress a given signal $\boldsymbol{x}$ as

$$\hat{\boldsymbol{z}} = \boldsymbol{U}_k \boldsymbol{x}$$

PCA basis: Since $\boldsymbol{U}_k$ is dependent on the data, we need to transmit both the basis $\boldsymbol{U}$ and the elements of $\hat{\boldsymbol{z}}$. However, the basis is optimal for the given covariance.

On the other hand, using a *fixed basis*, both parties (compressor and decompressor) can agree upon a particular basis and then transmit only the non-zero elements of the transformed signal $\hat{\boldsymbol{z}}$.

7.2 Compressive Sensing

When we gather information (e.g., in photography, MRI technology or other applications), we do not want to collect huge amounts of data if we compress most of it away right afterwards. Instead, compressing the data while gathering it is called *compressive sensing* and helps to decrease acquisition time, required battery and storage.

Given an original signal $\boldsymbol{x} \in \mathbb{R}^D$ which is assume to be sparse in some orthonormal basis $\boldsymbol{U}$ with K large coefficients in its sparse representation $\boldsymbol{z}$:

$$\boldsymbol{x} = \boldsymbol{U}\boldsymbol{z} \qquad \text{such that } |I_{\boldsymbol{z}} \gg 0| = K$$

The main idea is to acquire the set of M linear combinations $y_k = \langle w_k, \boldsymbol{x} \rangle$ of the initial signal instead of the signal itself, where $M \ll D$ and $\boldsymbol{W} = \{w_1, \ldots, w_M\}$ is a matrix with linear combinations coefficients

$$y_k = \langle w_k, \boldsymbol{x} \rangle \qquad k \in M$$
$$\boldsymbol{y} = \boldsymbol{W}\boldsymbol{x} = \boldsymbol{W}\boldsymbol{U}\boldsymbol{z} \overset{\Theta=\boldsymbol{W}\boldsymbol{U}}{=} \Theta\boldsymbol{z}$$

Let all matrix elements $w_{i,j}$ be independent and i.i.d. random variables from a Gaussian probability density function with mean zero and variance $\frac{1}{D}$. Surprisingly, regardless of the choice of orthonormal basis $\boldsymbol{U}$ then M

$$M \leq cK \log(\frac{d}{K})$$

where c is some constant, is enough to obtain a stable solution for any K-sparse and compressible signal.

Reconstruction the initial signal of length D having M measurements means inverting the following system:

$$\boldsymbol{y} = \boldsymbol{W}\boldsymbol{x} = \boldsymbol{W}\boldsymbol{U}\boldsymbol{z} = \Theta\boldsymbol{z}$$

This problems seems to be *ill-posed*, as $M \ll D$ we have many more unknowns than equations. We can look at the sparsest solution such that

$$\boldsymbol{z}^* = \arg\min_{\boldsymbol{z}} \|\boldsymbol{z}\|_0 \qquad \text{such that } \Theta\boldsymbol{z} = \boldsymbol{y}$$

This can be solved using the *matching pursuit* algorithm

7.3 Summary of Sparse Coding

Sparse coding works by coding via orthogonal transforms. That is, given an original signal $\boldsymbol{x}$ and an orthogonal matrix $\boldsymbol{A}$, we compute the linear transformation (change of basis) $\boldsymbol{z} = \boldsymbol{A}\boldsymbol{x}$ and truncate "small" values, giving us $\hat{\boldsymbol{z}}$. The inverse is then computed as $\hat{\boldsymbol{x}} = \boldsymbol{A}^T\hat{\boldsymbol{z}}$ (since $\boldsymbol{A}^{-1} = \boldsymbol{A}^T$).

To measure the performance, we can measure the *reconstruction error* $\|\boldsymbol{x} - \hat{\boldsymbol{x}}\|$ or measure the *sparsity* of the coding vector $\boldsymbol{z}$, i.e., $\text{nnz}(\boldsymbol{z})$.

7.4 Overcomplete Dictionaries

No single basis (e.g., Fourier or Wavelets) is optimally sparse for *all* signal classes. We could use more atoms (dictionary elements) than dimensions, i.e., make use of *overcompleteness* ($L > D$).

We have two possibilities:

1. We use a union of orthogonal bases $[\boldsymbol{U}_1, \dots, \boldsymbol{U}_B] \in \mathbb{R}^{D \times D \cdot D}$:

$$\boldsymbol{x} = [\boldsymbol{U}_1, \dots, \boldsymbol{U}_B] \cdot \boldsymbol{z}$$

   The coding vector $\boldsymbol{z}$ is now also larger, but we hope that it is very sparsely populated.

2. We can also use a general overcomplete dictionary with some number $L > D$ of atoms.

When we increase the overcompleteness factor $\frac{L}{D}$ we increase (potentially) the sparsity of the coding, but also increase the linear dependency between atoms. We measure this linear dependency using the notion of *coherence*:

$$m(\boldsymbol{U}) = \max_{i \neq j} \left| \boldsymbol{u}_i^T \boldsymbol{u}_j \right|$$

We have $m(\boldsymbol{B}) = 0$ for an orthogonal basis $\boldsymbol{B}$, and $m([\boldsymbol{B}, \boldsymbol{u}]) \geq \frac{1}{\sqrt{D}}$ if an atom $\boldsymbol{u}$ is added to $\boldsymbol{B}$.

7.4.1 Signal Coding

Given an overcomplete dictionary $\boldsymbol{U} \in \mathbb{R}^{D \times L}$, solving the equation

$$\boldsymbol{x} = \boldsymbol{U} \cdot \boldsymbol{z}$$

is an ill-posed problem, as we have more unknowns than equations. We can add the constraint that we want to find the sparsest $\boldsymbol{z}$. We solve

$$\boldsymbol{z}^* = \arg\min_{\boldsymbol{z}} \|\boldsymbol{z}\|_0 \qquad \text{such that} \quad \boldsymbol{x} = \boldsymbol{U}\boldsymbol{z}$$

This is the same formulation as for compressive sensing.

7.4.2 Noisy Observations

Additive noise. Assume each dimension is independently corrupted by zero-mean Gaussian noise with variance σ^2:

$$\boldsymbol{x} = \boldsymbol{U}\boldsymbol{z} + \boldsymbol{n} \qquad \text{with} \quad n_d \sim \mathcal{N}(0, \sigma^2)$$

Then, we want to find

$$\boldsymbol{z}^* = \arg\min_{\boldsymbol{z}} \|\boldsymbol{z}\|_0 \qquad \text{such that} \quad \|\boldsymbol{x} - \boldsymbol{U}\boldsymbol{z}\|_2 < \sigma$$

That is, we maximize the sparsity of $\boldsymbol{z}$ while the residual (approximation error) remains below σ.

Alternatively, we can also solve

$$\boldsymbol{z}^* = \arg\min_{\boldsymbol{z}} \|\boldsymbol{x} - \boldsymbol{U}\boldsymbol{z}\|_2 \qquad \text{such that} \quad \|\boldsymbol{z}\|_0 \leq K$$

That is, we minimize the residual while selecting less than K atoms from the dictionary.

Approximate sparse coding. We can also code the noise in terms of $\boldsymbol{U}$:

$$\boldsymbol{x} = \boldsymbol{U}\boldsymbol{z} + \boldsymbol{n} = \boldsymbol{U}\boldsymbol{z} + \boldsymbol{U}\boldsymbol{y} = \boldsymbol{U}(\boldsymbol{z} + \boldsymbol{y})$$

where $\boldsymbol{y}$ is the coding of the noise $\boldsymbol{n}$ in $\boldsymbol{U}$. The noise cannot be sparsely coded in any dictionary, therefore $\boldsymbol{y}$ has many small coefficients. However, the few large coefficients are still due to $\boldsymbol{z}$.

Geometry of sparse coding solution $\boldsymbol{z}^*$. As shown in Figure 8, the orthogonal projection of $\boldsymbol{x}$ onto the subspace spanned by selected atoms $\{\boldsymbol{u}_d \mid z_d^* \neq 0\}$ minimizes $\|\boldsymbol{x} - \boldsymbol{U}\boldsymbol{z}\|_2$.

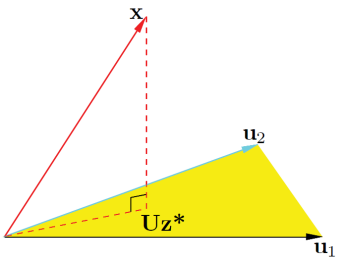

Figure 8: Geometry of sparse coding solution $\boldsymbol{z}^*$.

7.4.3 Matching Pursuit (MP)

We consider a greedy algorithm to approximate the NP hard problem iteratively, each time taking the action which is short-term optimal. Applied to sparse coding, we do the following:

1. Start with the zero vector $\boldsymbol{z} = \boldsymbol{0}$ and residual $\boldsymbol{r}^0 = \boldsymbol{x}$.
2. At each iteration t, take a step in the direction of the atom $\boldsymbol{u}_{d^*(t)}$ that maximally reduces the residual $\|\boldsymbol{x} - \boldsymbol{U}\boldsymbol{z}\|_2$.

The atom selection in iteration t is done as follows:

$$d^*(t) = \arg\max_d \left|\langle \boldsymbol{r}^t, \boldsymbol{u}_d \rangle\right|$$

The objective of the algorithm is (as seen before):

$$\boldsymbol{z}^* = \arg\min_{\boldsymbol{z}} \|\boldsymbol{x} - \boldsymbol{U}\boldsymbol{z}\|_2 \qquad \text{such that} \quad \|\boldsymbol{z}\|_0 \leq K$$

Algorithm 2 The matching pursuit algorithm

Objective: $\boldsymbol{z}^* = \arg\min_{\boldsymbol{z}} \|\boldsymbol{x} - \boldsymbol{U}\boldsymbol{z}\|_2$, such that $\|\boldsymbol{z}\|_0 \leq K$

1: $\boldsymbol{z} \leftarrow \boldsymbol{0}; \boldsymbol{r} \leftarrow \boldsymbol{x}$

2: **while** $\|\boldsymbol{z}\|_0 < K$ **do**

3: Select the atom with the maximum absolute correlation to the residual:

$$d^* = \arg\max_d |\langle \boldsymbol{r}, \boldsymbol{u}_d \rangle|$$

4: Update the coefficient vector and the residual:

$$z_{d^*} \leftarrow z_{d^*} + \boldsymbol{u}_{d^*}^T \boldsymbol{r}$$
$$\boldsymbol{r} \leftarrow \boldsymbol{r} - \left(\boldsymbol{u}_{d^*}^T \boldsymbol{r}\right) \boldsymbol{u}_{d^*}$$

5: **end while**

7.4.4 Support Recovery of MP

Assume $\boldsymbol{x} = \boldsymbol{U}\boldsymbol{z}$ has a K sparse coding

$$\boldsymbol{x} = \sum_{d \in \Delta} z_d \boldsymbol{u}_d$$

Under which condition is MP sucessful in recovering the true *support*, i.e., $\Delta_{\text{MP}} = \Delta$, where

$$\Delta_{\text{MP}} = \{d \mid z_d^* \neq 0\}$$

is the set of coding atoms for the MP solution $\boldsymbol{z}^*$. The *exact recovery condition* is

$$K < \frac{1}{2}\left(1 + \frac{1}{m(\boldsymbol{U})}\right)$$

where $m(\boldsymbol{U})$ is the coherence. The intuition is that if the coherence is small, then explaining a generating atom with other atoms is not sparse. Therefore, sparse coding recovers the support. We have a trade-off for increasing L; this leads to sparser coding (smaller possible K) but increases the coherence $m(\boldsymbol{U})$.

7.4.5 Sparse Coding for Inpainting

To use sparse coding for image inpainting, we can sparse code the known parts of the images, and then predict the missing parts by reconstructing from the sparse code. An algorithm by Elad et al. (2005) works as follows:

1. Define the diagonal masking matrix $\boldsymbol{M}$ where $m_{d,d} = 1$ if the pixel d is known and $m_{d,d} = 0$ if it is missing.
2. Sparse coding of the known parts in an overcomplete dictionary $\boldsymbol{U}$:

$$\boldsymbol{z}^* = \arg\min_{\boldsymbol{z}} \|\boldsymbol{z}\|_0 \qquad \text{such that} \quad \|\boldsymbol{M}(\boldsymbol{x} - \boldsymbol{U}\boldsymbol{z})\|_2 < \sigma \tag{1}$$

3. Image reconstruction using the mask:

$$\hat{\boldsymbol{x}} = \boldsymbol{M}\boldsymbol{x} + (I - \boldsymbol{M})\boldsymbol{U}\boldsymbol{z}^*$$

The question that remains is how we can modify MP to solve Equation 1. We can take an EM algorithm, as follows:

1. Assume initial sparse coding $\boldsymbol{z}^0$ of the image.
2. Reconstruct the image using the mask

$$\hat{\boldsymbol{x}}^1 = \boldsymbol{M}\boldsymbol{x} + (I - \boldsymbol{M})\boldsymbol{U}\boldsymbol{z}^0$$

3. Sparse coding of reconstructed image $\boldsymbol{x}^1$:

$$\boldsymbol{z}^1 = \arg\min_{\boldsymbol{z}} \|\boldsymbol{z}\|_0 \qquad \text{such that} \quad \left\|\boldsymbol{x}^1 - \boldsymbol{U}\boldsymbol{z}\right\|_2 < \sigma$$

4. Iterate steps two and three t times until convergence.
5. Image reconstruction using the mask:

$$\hat{\boldsymbol{x}} = \boldsymbol{M}\boldsymbol{x} + (I - \boldsymbol{M})\boldsymbol{U}\boldsymbol{z}^t$$

7.5 Dictionary Learning

So far, we have seen two variants:

- Fixed orthonormal basis. The advantage is that coding is efficient and can be done by matrix multiplication $\boldsymbol{z} = \boldsymbol{U}^T\boldsymbol{x}$. However, the result is only sparse for a single class of signals.
- Fixed overcomplete basis. Now, sparse coding is possible for many signal classes, but finding the sparsest code requires an approximation algorithm and is problematic if the dictionary size L and the coherence $m(\boldsymbol{U})$ are large.

Instead, we now consider *learning the dictionary*. This allows us to adapt the dictionary to our signal's characteristics. However, we have to solve a matrix factorization problem

$$\boldsymbol{X}_{D\times N} \approx \boldsymbol{U}_{D\times L} \cdot \boldsymbol{Z}_{L\times N}$$

subject to the sparsity constraint on $\boldsymbol{Z}$ and atom norm constraint for $\boldsymbol{U}$.

We have the following objective

$$\arg\min_{\boldsymbol{U},\boldsymbol{Z}} \|\boldsymbol{X} - \boldsymbol{U}\cdot\boldsymbol{Z}\|_F^2$$

where we use the Frobenius norm $\|\boldsymbol{R}\|_F^2 = \sum_{i,j} r_{i,j}^2$ (sum-square error). This objective is not convex in both $\boldsymbol{U}$ and $\boldsymbol{Z}$ (local minima), but convex in either $\boldsymbol{U}$ or $\boldsymbol{Z}$ (unique minimum). We can use an iterative greedy minimization:

1. Coding step: $\boldsymbol{Z}^{(t+1)} = \arg\min_{\boldsymbol{Z}} \left\|\boldsymbol{X} - \boldsymbol{U}^{(t)}\cdot\boldsymbol{Z}\right\|_F^2$, subject to $\boldsymbol{Z}$ being sparse.
2. Dictionary update step: $\boldsymbol{U}^{(t+1)} = \arg\min_{\boldsymbol{U}} \left\|\boldsymbol{X} - \boldsymbol{U}\cdot\boldsymbol{Z}^{(t+1)}\right\|_F^2$ subject to $\|\boldsymbol{u}_d\|_2 = 1$ for all $d = 1,\dots,D$.

7.5.1 Coding Step

The residual is column separable, i.e.,

$$\|\boldsymbol{R}\|_F^2 = \sum_{i,j} r_{i,j}^2 = \sum_j \|\boldsymbol{r}_j\|_2^2$$

Thus, we can perform N *independent* sparse coding steps. For all $n = 1,\dots,N$:

$$\boldsymbol{z}_n^{(t+1)} = \arg\min_{\boldsymbol{z}} \|\boldsymbol{z}\|_0 \qquad \text{such that} \quad \left\|\boldsymbol{x}_n - \boldsymbol{U}^{(t)}\boldsymbol{z}\right\|_2 \le \sigma\cdot\|\boldsymbol{x}_n\|_2$$

7.5.2 Dictionary Update

Unfortunately, the residual for the dictionary update step, i.e.,

$$\boldsymbol{U}^{(t+1)} = \arg\min_{\boldsymbol{U}} \left\|\boldsymbol{X} - \boldsymbol{U}\cdot\boldsymbol{Z}^{(t+1)}\right\|_F^2$$

is not separable in atoms (columns of $\boldsymbol{U}$). Therefore, we approximate by updating one atom at a time. For all $d = 1,\dots,D$

1. Fix all atoms except $\boldsymbol{u}_d$:
$$\boldsymbol{U} = \left[\boldsymbol{u}_1^{(t)}, \ldots, \boldsymbol{u}_d, \ldots, \boldsymbol{u}_D^{(t)}\right]$$

2. Isolate $\boldsymbol{R}_d^{(t)}$, the residual that is due to atom $\boldsymbol{u}_d$:
$$\begin{aligned}
&\left\|\boldsymbol{X} - \left[\boldsymbol{u}_1^{(t)}, \ldots, \boldsymbol{u}_d, \ldots, \boldsymbol{u}_D^{(t)}\right] \cdot \boldsymbol{Z}^{(t+1)}\right\|_F^2 \\
&= \left\|\boldsymbol{X} - \left\{\sum_{e \neq d} \boldsymbol{u}_e^{(t)} \left(\boldsymbol{z}_e^{(t+1)}\right)^T + \boldsymbol{u}_d^{(t)} \left(\boldsymbol{z}_d\right\} \cdot \boldsymbol{Z}^{(t+1)}\right\}\right\|_F^2 \\
&= \left\|\boldsymbol{R}_d^{(t)} - \boldsymbol{u}_d \left(\boldsymbol{z}_d^{(t+1)}\right)^T\right\|_F^2
\end{aligned}$$

3. Find $\boldsymbol{u}_d^*$ that minimizes $\boldsymbol{R}_d^{(t)}$, subject to $\left\|\boldsymbol{u}_d^*\right\|_2 = 1$:
   - $\boldsymbol{u}_d \left(\boldsymbol{z}_d^{(t+1)}\right)^T$ is an outer product, i.e., a matrix.
   - Minimizing the residual
   $$\left\|\boldsymbol{R}_d^{(t)} - \boldsymbol{u}_d \left(\boldsymbol{z}_d^{(t+1)}\right)^T\right\|_F^2$$
   by approximating $\boldsymbol{R}_d^{(t)}$ with rank-1 matrix $\boldsymbol{u}_d \left(\boldsymbol{z}_d^{(t+1)}\right)^T$.
   - Achieved by the SVD of $\boldsymbol{R}_d^{(t)}$:
   $$\boldsymbol{R}_d^{(t)} = \tilde{\boldsymbol{U}} \boldsymbol{S} \tilde{\boldsymbol{V}}^T = \sum_i s_i \tilde{\boldsymbol{u}}_i \tilde{\boldsymbol{v}}_i^T$$
   - $\boldsymbol{u}_d^* = \tilde{\boldsymbol{u}}_1$ is the first left-singular vector.
   - $\left\|\boldsymbol{u}_d^*\right\|_2 = 1$ is naturally satisfied.

7.5.3 Initialization

The approach is sensitive to the choice of $\boldsymbol{U}^{(0)}$; the initial candidate solution is optimized locally and greedily until no progress is possible any more. Possible choices include:

1. **Random atoms.** Sampling $\left\{\boldsymbol{u}_d^{(0)}\right\}$ on the unit sphere.
   i) Sample D dimensional standard normal distribution: $\boldsymbol{u}_d^{(0)} \sim \mathcal{N}(\boldsymbol{0}, \boldsymbol{I}_D)$.
   ii) Scale to unit length:
   $$\boldsymbol{u}_d^{(0)} \leftarrow \frac{\boldsymbol{u}_d^{(0)}}{\left\|\boldsymbol{u}_d^{(0)}\right\|_2}$$

2. **Samples from $\boldsymbol{X}$.**
   i) $\boldsymbol{u}_d^{(0)} \leftarrow \boldsymbol{x}_n$ where $n \sim \mathcal{U}(1, N)$ is sampled uniformly.
   ii) Scale to unit length.

3. Use **fixed overcomplete dictionary**, e.g., overcomplete DCT.