Unsupervised learning

Christophe Troalen

July 2024

Disclaimer: These notes were originally created for personal reference during my studies at Imperial College London and Télécom SudParis. As such, they may not be exhaustive or fully accurate, serving mainly as quick notes rather than comprehensive explanations.

1 Curse of Dimensionality

In many real-world scenarios, datasets feature hundreds or even thousands of dimensions, presenting significant challenges for machine learning algorithms. This phenomenon, known as the *curse of dimensionality*, particularly affects distance-based algorithms like k-nearest neighbors (k-NN).

1.1 Challenges in High-Dimensional Spaces

The k-NN algorithm operates on the premise that similar data points generally share similar labels. However, in high-dimensional spaces, the distance between points tends to increase, and points become almost equidistant from each other. This effect dilutes the meaning of "nearest neighbors" since all points appear to be roughly the same distance from each other.

1.2 Illustrative Example

Consider a unit cube $[0,1]^d$ where each dimension d represents a feature of the dataset. Assume that the training data consists of points sampled uniformly within this cube. For instance, when considering the $k = 10$ nearest neighbors of a test point, we analyze the edge length l of the smallest hyper-cube that contains all these neighbors. Mathematically, l is approximated by:

$$l \approx \left(\frac{k}{n}\right)^{\frac{1}{d}}$$

If $d = 100$, $l = 0.955$; if $d = 1,000$, $l = 0.995$. With such high values of d, the nearest neighbors are not significantly closer—and thus not more similar—than any other points in the training set. This undermines the fundamental assumption of the k-NN that nearest neighbors share similar labels.

1.3 Implications

Increasing the number of training samples n might seem like a solution to ensure that the nearest neighbors are truly close to the test point. However, this would require n to grow exponentially, which is impractical:

$$n = \frac{k}{l^d}$$

This example highlights the curse of dimensionality: as the number of dimensions increases, maintaining the effectiveness of algorithms like k-NN becomes exponentially harder.

2 Dimensionality Reduction

Now that we have observed the challenges posed by high-dimensional data, it may be beneficial to seek a low-dimensional representation that retains the essential statistical properties of the high-dimensional data.

The main applications are among the following:

- Compress and visualize data
- Preprocess data before supervised learning to improve model performance and reduce overfitting
- Simplify the description of massive datasets by removing uninformative dimensions
- Reduce storage and computational costs

Some dimensionality reduction methods include *Principal Components Analysis* (PCA), *Kernel PCA*, *Nonnegative Matrix Factorisation* (NMF), *Linear Discriminant Analysis* (LDA), *Generalised Discriminant Analysis* (GDA), *Nonlinear independent component analysis* (ICA), *Uniform manifold approximation* (UMA), *Locally Linear Embedding* (LLE), and also *Random Forests.*

3 Principal Component Analysis (PCA)

Principal component analysis is a multivariate technique which allows us to analyze the statistical structure of high dimensional dependent observations by representing data using orthogonal variables called principal components.

There are several equivalent ways of deriving the principal components mathematically. The simplest one is by finding the projections which maximize the variance. The first principal component is the direction in space along which projections have the largest variance. The second principal component is the direction which maximizes variance among all directions orthogonal to the first. The kth component is the variance-maximizing direction orthogonal to the previous $k-1$ components. There are p principal components in all. Rather than maximizing variance, it might sound more plausible to look for the projection with the smallest average (mean-squared) distance between the original vectors and their projections onto the principal components; this turns out to be equivalent to maximizing the variance.

Let $(X_i)_{1\leq i\leq n}$ be iid random variables in $\mathbb{R}^d$ and consider the matrix $X \in \mathbb{R}^{n\times d}$ such that the i-th row of X is the observation X_i^T.

We assume that data are preprocessed so that the columns of X are centered. Let Σ_n be the empirical covariance matrix:

$$\Sigma_n = \frac{1}{n}\sum_{i=1}^{n} X_i X_i^T$$

We can reduce the dimensionality of the observations (X_i) using a compression matrix $W \in \mathbb{R}^{p\times d}$ with $p \leq d$ so that for each $1 \leq i \leq n, WX_i$ is a low-dimensional representation of X_i.

The original observation may then be partially recovered using another matrix $U \in \mathbb{R}^{d\times p}$.

PCA computes U and W using the least squares approach:

$$(U_\star, W_\star) \quad \underset{(U,W)\in\mathbb{R}^{d\times p}\times\mathbb{R}^{p\times d}}{\operatorname{argmin}} \sum_{i=1}^{n} \|X_i - UWX_i\|^2$$

Let $(U_*, W_*) \in \mathbb{R}^{d\times p} \times \mathbb{R}^{p\times d}$ be a solution. Then, the columns of U_* are orthonormal and $W_* = U_*^T$.

For all $U \in \mathbb{R}^{d\times p}$ such that $U^TU = I_p$, we have:

$$\sum_{i=1}^{n} \|X_i - UU^TX_i\|^2 = \sum_{i=1}^{n} \|X_i\|^2 - \operatorname{trace}(U^TXX^TU)$$

Therefore, solving the PCA problem boils down to computing

$$U_\star \in \underset{U\in\mathbb{R}^{d\times p}, U^TU=I_p}{\operatorname{argmax}} \operatorname{trace}\left(U^T\Sigma_n U\right)$$

Let $v_1, \ldots, v_d$ be orthonormal eigenvectors associated with the eigenvalues $\lambda_1 \geq \ldots \geq \lambda_d$ of Σ_n. Then a solution to the PCA problem is given by the matrix U_* with columns $\{v_1, \ldots, v_p\}$.

4 PCA Algorithm

The following is the pseudo-code for the Principal Component Analysis (PCA) algorithm:

Algorithm 1 PCA Algorithm

```
1. **Center** the data matrix $X \in \mathbb{R}^{n \times d}$ by subtracting the mean of each column.
2. **Compute** the covariance matrix $\Sigma_n$ and obtain its eigenvectors $v_i \in \mathbb{R}^d$ sorted by eigenvalues in decreasing order.
3. **Construct** the matrix $U_* = (v_1, \ldots, v_p) \in \mathbb{R}^{d \times p}$ by stacking the top $p$ eigenvectors $v_i$.
4. **Project** the data to the reduced dimensionality, giving the compressed representation $Z = XU_* \in \mathbb{R}^{n \times p}$.
```

PCA only allows dimensionality reduction based on principal components which are linear combinations of the variables. When the data has more complex structures which cannot be well represented in a linear subspace, standard PCA fails. Kernel PCA allows us to generalize standard PCA to nonlinear dimensionality reduction.

5 Cluster Analysis

- The *K-means algorithm* classifies n points into k clusters in a vector space, based on their distance to each other. It starts by randomly choosing the representatives of each cluster—sometimes referred to as *centroids*—and then iteratively assigns each of the n points to its closest centroid to obtain k clusters. The centroid of the latter will be updated as the barycenter of the cluster's points.
- *Hierarchical clustering* aims to create a hierarchy of clusters by either merging small clusters into larger ones (agglomerative clustering) or by breaking larger clusters down (divisive clustering).
- The *DBSCAN algorithm* forms clusters based on the density of points in space. Unlike K-means, it can find arbitrarily shaped clusters and can filter out noise points that are not part of any cluster.

Cluster analysis can be applied in a variety of fields, from marketing (for customer segmentation) to biology (for grouping similar species) and beyond.

6 Expectation-Maximization (EM) Algorithm

The expectation-maximization (EM) algorithm is a statistical technique for finding maximum likelihood estimates of parameters in probabilistic models, particularly when the model depends on unobserved latent variables.

The EM algorithm iteratively optimizes a likelihood function, typically in two steps:

1. **Expectation Step (E-Step)**: Calculate the expected value of the log-likelihood function, given the current parameter estimates and the observed data.
2. **Maximization Step (M-Step)**: Maximize the expected log-likelihood found in the E-Step to obtain new parameter estimates.

The procedure repeats until convergence, providing a method to handle incomplete data or latent variables.

6.1 Application to Gaussian Mixture Models

In the context of Gaussian Mixture Models (GMMs), the EM algorithm is particularly useful. A GMM assumes that the data is generated from a mixture of several Gaussian distributions with unknown parameters.

- Let $X = \{x_1, x_2, \ldots, x_n\}$ be the observed data.
- Assume X is generated from K Gaussian components with parameters $\theta = \{\mu_k, \sigma_k^2, \pi_k\}_{k=1}^K$ where μ_k is the mean, σ_k^2 is the variance, and π_k is the mixing coefficient.

In the E-Step, compute the responsibilities r_{nk}, which represent the probability that data point x_n belongs to component k:

$$r_{nk} = \frac{\pi_k \mathcal{N}(x_n|\mu_k, \sigma_k^2)}{\sum_{j=1}^K \pi_j \mathcal{N}(x_n|\mu_j, \sigma_j^2)}$$

In the M-Step, update the parameters:

- Mixing coefficients:

$$\pi_k = \frac{1}{N} \sum_{n=1}^{N} r_{nk}$$

- Means:

$$\mu_k = \frac{\sum_{n=1}^N r_{nk} x_n}{\sum_{n=1}^N r_{nk}}$$

- Variances:

$$\sigma_k^2 = \frac{\sum_{n=1}^N r_{nk}(x_n - \mu_k)^2}{\sum_{n=1}^N r_{nk}}$$

Repeat the E-Step and M-Step until convergence to find the parameters that best describe the observed data.

7 Detailed Derivations for M-Step in Gaussian Mixture Models

The M-Step involves finding the maximum likelihood estimates of the parameters given the responsibilities calculated in the E-Step.

To derive the M-Step equations, we need to maximize the expected log-likelihood:

$$Q(\theta|\theta^{(t)}) = \mathbb{E}[\log p(X|\theta)|X, \theta^{(t)}]$$

where $\theta^{(t)}$ are the current estimates of the parameters. The M-step estimates are obtained by maximizing Q.

7.1 Deriving the Updating Equations

1. **For the Mixing Coefficients**: The mixing coefficients must satisfy the constraint $\sum_{k=1}^{K} \pi_k = 1$. To maximize Q:

$$Q(\theta|\theta^{(t)}) = \sum_{n=1}^{N} \sum_{k=1}^{K} r_{nk} \log(\pi_k) + \text{constant}$$

The update for π_k can be derived as:

$$\pi_k^{(t+1)} = \frac{1}{N} \sum_{n=1}^{N} r_{nk}$$

2. **For the Means**: The expected contribution of data point x_n to the Gaussian with mean μ_k is:

$$Q(\theta|\theta^{(t)}) = \sum_{n=1}^{N} \sum_{k=1}^{K} r_{nk} \log(\mathcal{N}(x_n|\mu_k, \sigma_k^2))$$

Taking the derivative with respect to μ_k and setting it to zero yields:

$$\mu_k^{(t+1)} = \frac{\sum_{n=1}^{N} r_{nk} x_n}{\sum_{n=1}^{N} r_{nk}}$$

3. **For the Variances**: The update for variances follows a similar reasoning by considering the expected distance of points from the mean:

$$\sigma_k^{2(t+1)} = \frac{\sum_{n=1}^{N} r_{nk} (x_n - \mu_k^{(t+1)})^2}{\sum_{n=1}^{N} r_{nk}}$$

The complete set of parameter updates is performed until the convergence of the likelihood function or the parameters.