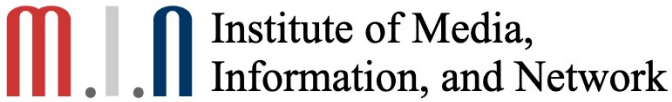


交叉创新课程

Unsupervised Learning

Wenrui Dai
戴文睿
http://min.sjtu.edu.cn

计算机科学工程系
上海交通大学

2021 年 04 月 19日

Outline

- Cluster Analysis
- Principal Component Analysis
- Non-negative Matrix Factorization
- The Google PageRank Algorithm
- Contrastive Learning

上海交通大學
SHANGHAI JIAO TONG UNIVERSITY

Introduction

Introduction

Supervised learning or *"learning with a teacher"*

- The predictions are based on the training sample $(x_1, y_1), \ldots, (x_N, y_N)$ of previously solved cases, where the joint values of all of the variables are known.
- The supervisor or "teacher" provides either the correct answer and/or an error associated with the student's answer.

Unsupervised learning or *"learning without a teacher"*

- In this case one has a set of N observations $(x_1, x_2, \ldots, x_N)$ of a random p-vector X having joint density $\Pr(X)$.
- The goal is to directly infer the properties of this probability density without the help of a supervisor or teacher providing correct answers or degree-of-error for each observation.

上海交通大學
SHANGHAI JIAO TONG UNIVERSITY

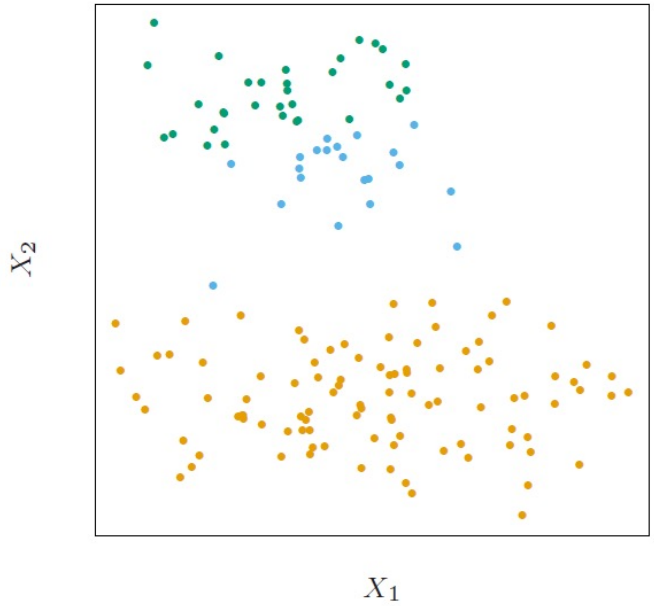
Cluster Analysis

Cluster Analysis

- Cluster analysis, also called data segmentation, has a variety of goals.

- All relate to grouping or segmenting a collection of objects into subsets or "clusters," such that those within each cluster are more closely related to each other than objects assigned to different clusters.

- Central to all of the goals of cluster analysis is the notion of the degree of similarity (or dissimilarity) between the individual objects being clustered.



Simulated data in the plane, clustered into three classes (represented by orange, blue and green) by the K-means clustering algorithm

Proximity Matrices

- Sometimes the data are represented directly in terms of the proximity (alikeness or affinity) between pairs of objects. These can be either similarities or dissimilarities (difference or lack of affinity).

- The data can be represented by an $N \times N$ matrix $\boldsymbol{D}$,
  - N is the number of objects,
  - Each element $d_{ii'}$ records the proximity between the i-th and i'-th objects.

- Most algorithms presume a matrix of dissimilarities with nonnegative entries and zero diagonal elements: $d_{ii} = 0, i = 1,2, \ldots, N$.

Dissimilarities Based on Attributes

- Given measurements on variables x_{ij}, for $i = 1,2,\dots,N$, on ***variables*** (***attributes***) $j = 1,2,\dots,p$, dissimilarity between two objects is

$$D(x_i, x_{i'}) = \sum_{j=1}^{p} d(x_{ij}, x_{i'j})$$

$$d(x_{ij}, x_{i'j}) = (x_{ij} - x_{i'j})^2$$ ← Most common distance

Quantitative Variables $d(x_{ij}, x_{i'j}) = l(|x_i - x_{i'}|)$

Treated as quantitative variables on this scale.

Ordinal Variables Replacing their M original values with $\frac{i-\frac{1}{2}}{M}, i = 1,\dots,M$

Categorical Variables If the variable assumes M distinct values, these can be arranged in a symmetric M×M matrix with elements

$$L_{rr'} = L_{r'r}, L_{rr} = 0, L_{rr'} \geq 0$$

The most common choice $L_{rr'} = 1$, for all $r \neq r'$

Object Dissimilarity

- Dissimilarity $D(x_i, x_{i'})$ between two objects or observations

$$D(x_i, x_{i'}) = \sum_{j=1}^{p} w_j d(x_{ij}, x_{i'j}); \sum_{j=1}^{p} w_j = 1$$

- The average object dissimilarity measure over all pairs of observations in the data set

$$\bar{D} = \frac{1}{N^2} \sum_{i=1}^{N} \sum_{i'=1}^{N} D(x_i, x_{i'}) = \sum_{j=1}^{p} w_j \bar{d}_j$$

$$\bar{d}_j = \frac{1}{N^2} \sum_{i=1}^{N} \sum_{i'=1}^{N} d_j(x_{ij}, x_{i'j})$$

Clustering Algorithms

Combinatorial Algorithms

- Work directly on the observed data with no direct reference to an underlying probability model.

Mixture Modeling

- Supposes that the data is an *i.i.d* sample from some population described by a probability density function. This density function is characterized by a parameterized model taken to be ***a mixture of component density functions***; each component density describes one of the clusters. This model is then fit to the data by maximum likelihood or corresponding Bayesian approaches.

Mode Seekers

- Take a nonparametric perspective, attempting to directly estimate distinct modes of the probability density function. Observations "closest" to each respective mode then define the individual clusters.

Clustering Algorithms

Combinatorial Algorithms

- Assign each observation to a group or cluster without regard to a probability model describing the data.

$$\{x_i\}_{i=1}^{N} \quad x_i \Rightarrow C(i) = k, k = 1, \dots, K$$

- The Loss function

$$W(C) = \frac{1}{2}\sum_{k=1}^{K}\sum_{C(i)=k}\sum_{C(i')=k} d(x_i, x_{i'})$$

This criterion characterizes the extent to which observations assigned to the same cluster tend to be close to one another.

Clustering Algorithms

Combinatorial Algorithms

- It is sometimes referred to as the "***within cluster***" ***point scatter*** since

$$T = \frac{1}{2}\sum_{i=1}^{N}\sum_{i'=1}^{N} d_{ii'} = \frac{1}{2}\sum_{k=1}^{K}\sum_{C(i)=k}\left(\sum_{C(i')=k} d_{ii'} + \sum_{C(i')\neq k} d_{ii'}\right)$$

$$\downarrow$$

$$T = W(C) + B(C)$$

Here T is the total point scatter, which is a constant given the data, independent of cluster assignment.

- Between-Cluster Scatter:

$$B(C) = \frac{1}{2}\sum_{k=1}^{K}\sum_{C(i)=k}\sum_{C(i')\neq k} d_{ii'}$$

Minimizing W(C) is equivalent to maximizing B(C).

K-Means Clustering

- The within-point scatter can be written as

$$W(C) = \frac{1}{2}\sum_{k=1}^{K}\sum_{C(i)=k}\sum_{C(i')=k} d(x_i, x_{i'}) = \sum_{k=1}^{K} N_k \sum_{C(i)=k} \|x_i - \bar{x}_k\|^2$$

where $\bar{x}_k = (\bar{x}_{1k}, \ldots, \bar{x}_{pk})$ is the mean vector associated with the kth cluster, and N_k is the sample number of kth class.

- An iterative descent algorithm for solving $C^* = \sum_{k=1}^{K} N_k \sum_{C(i)=k} \|x_i - \bar{x}_k\|^2$ can be obtained by noting that for any set of observations S

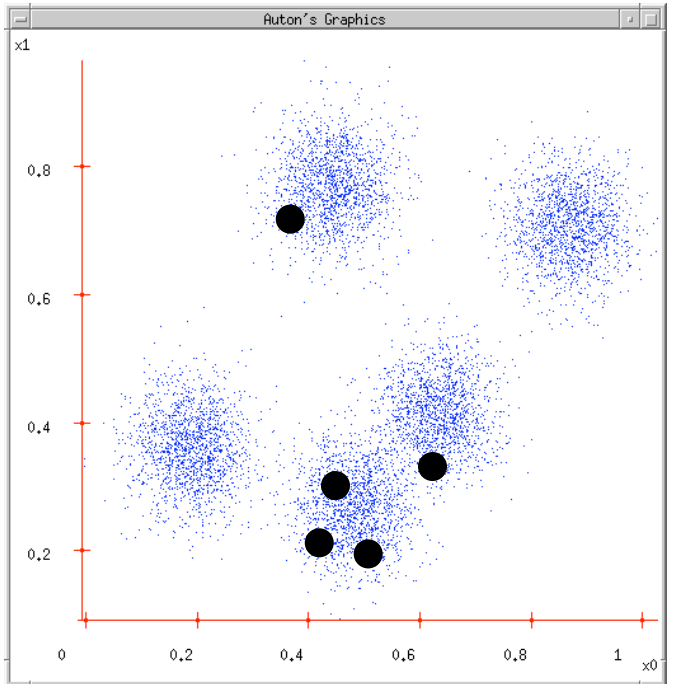
$$\bar{x}_S = \arg\min_m \sum_{i \in S} \|x_i - m\|^2$$

K-Means Clustering

Step 0: Initialization

Input: Data $\{x_i\}$, number of clusters K

- Randomly select K examples as the centroids of clusters
- Loop until convergence
  - Assign each example to the closest centroid (cluster mean) by minimizing the cluster variance
  - Yield the means of currently assigned clusters



[http://www.autonlab.org/tutorials/]

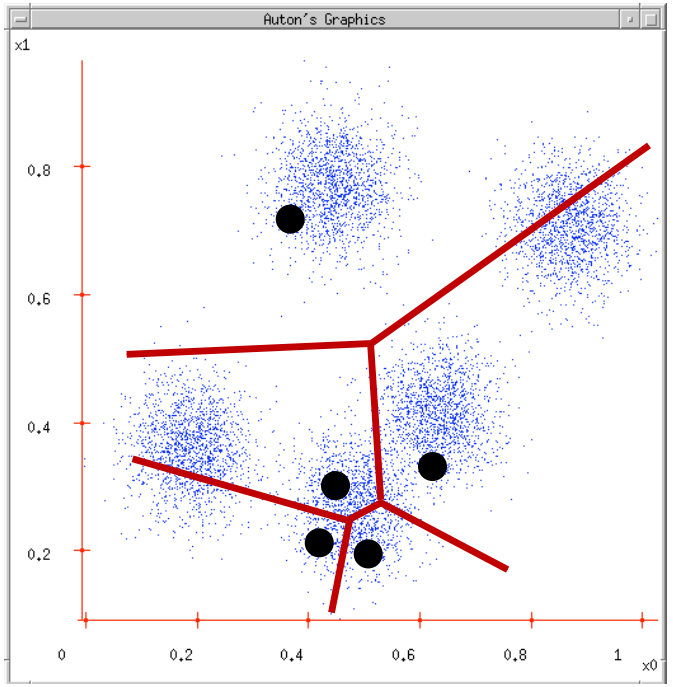
K-Means Clustering

Step 1: Segmentation

Input: Data $\{x_i\}$, number of clusters K

- Randomly select K examples as the centroids of clusters
- Loop until convergence
  - Assign each example to the closest centroid (cluster mean) by minimizing the cluster variance
  - Yield the means of currently assigned clusters

$$C(i) = k \; if \; k = \arg\min_k \|x_i - m_k\|^2$$



[http://www.autonlab.org/tutorials/]

K-Means Clustering

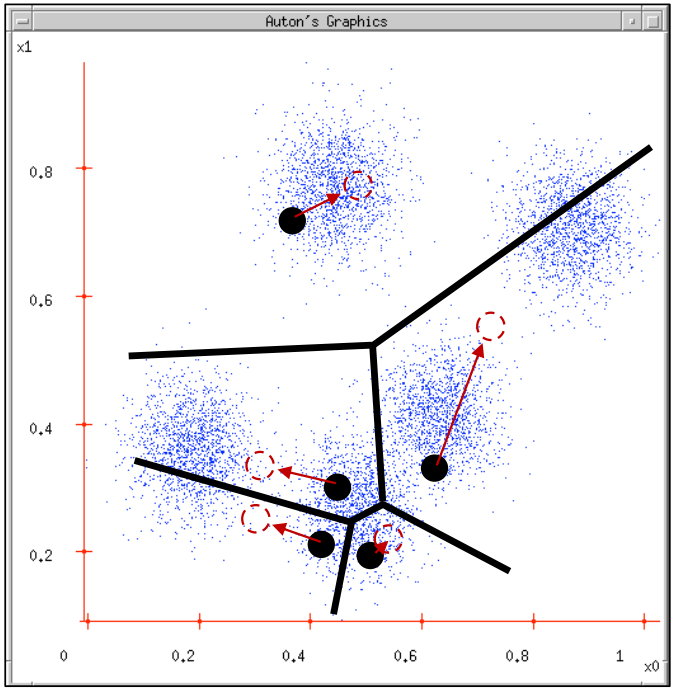
Step 2: Segmentation

Input: Data $\{x_i\}$, number of clusters K

- Randomly select K examples as the centroids of clusters
- Loop until convergence
  - Assign each example to the closest centroid (cluster mean) by minimizing the cluster variance
  - Yield the means of currently assigned clusters

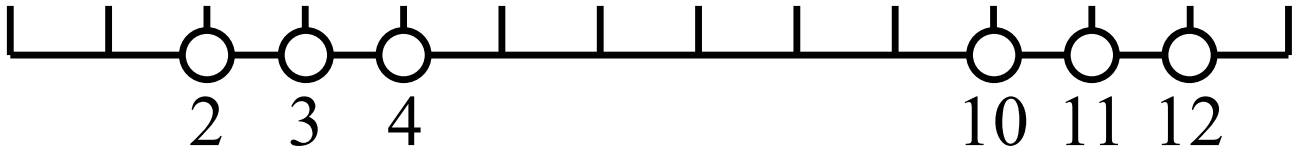
$$m_k = \arg\min_{m_k} \sum_{C(i)=k} \|x_i - m_k\|^2$$

$$m_k = \sum_{C(i)=k} x_i / N_k$$



[http://www.autonlab.org/tutorials/]

Example



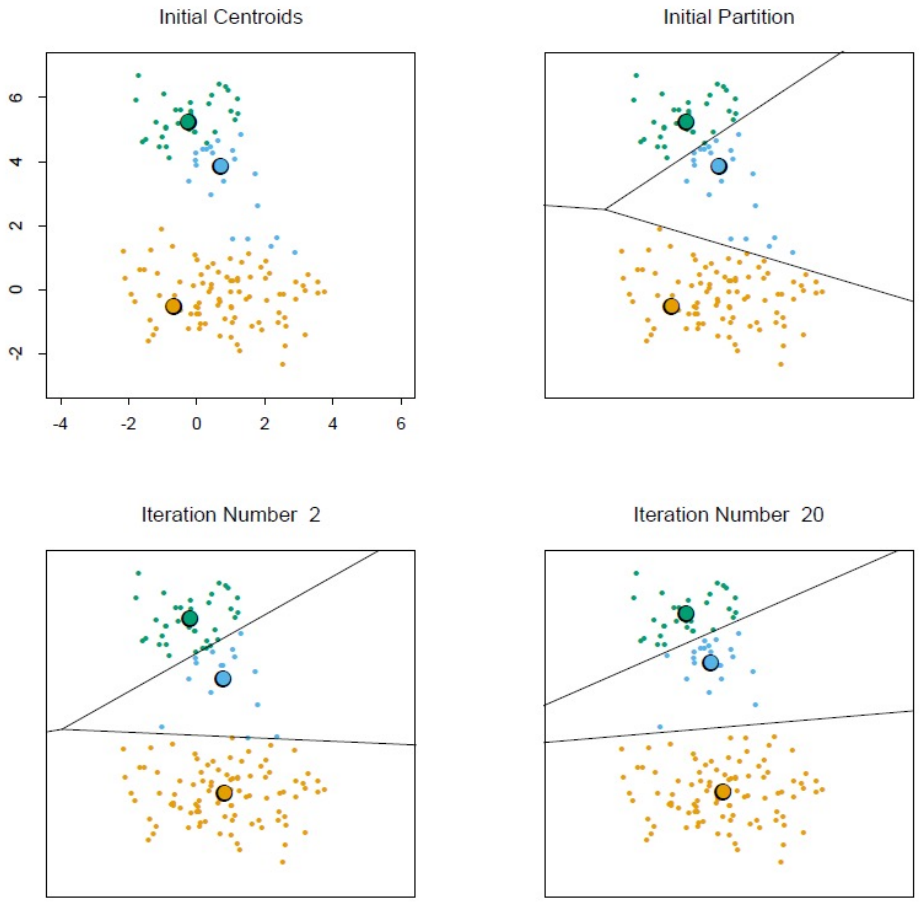
K-means clustering on one-dimensional data

- Segment the six data points into K=2 clusters
- Initial centroids: $m1$=2, $m2$=4

$$C\left(\begin{bmatrix}1\\2\\3\\4\\5\\6\end{bmatrix}\right)=\begin{bmatrix}1\\1\\2\\2\\2\\2\end{bmatrix}\rightarrow \begin{matrix}m_1=1.5\\m_2=9.25\end{matrix}\rightarrow C\left(\begin{bmatrix}1\\2\\3\\4\\5\\6\end{bmatrix}\right)=\begin{bmatrix}1\\1\\1\\2\\2\\2\end{bmatrix}\rightarrow \begin{matrix}m_1=3\\m_2=11\end{matrix}\rightarrow C\left(\begin{bmatrix}1\\2\\3\\4\\5\\6\end{bmatrix}\right)=\begin{bmatrix}1\\1\\1\\2\\2\\2\end{bmatrix}$$

K-Means Clustering

Some of the K-means iterations for the simulated data. The centroids are depicted by "O"s. The straight lines show the partitioning of points, each sector being the set of points closest to each centroid. This partitioning is called the Voronoi tessellation. After 20 iterations the procedure has converged.



Gaussian Mixtures as Soft K-means Clustering

- The K-means clustering procedure is closely related to the EM algorithm for estimating a certain Gaussian mixture model.

- Suppose we specify K mixture components, each with a Gaussian density having scalar covariance matrix $\sigma^2 I$. Then the relative density under each mixture component is a monotone function of the Euclidean distance between the data point and the mixture center.

- Hence in this setup EM is a "soft" version of K-means clustering, making probabilistic (rather than deterministic) assignments of points to cluster centers. As the variance $\sigma^2 \to 0$, these probabilities become 0 and 1, and the two methods coincide.

上海交通大學
SHANGHAI JIAO TONG UNIVERSITY

Principal Component Analysis

Principal Component Analysis (PCA)

- PCA is a technique that is widely used for applications such as dimensionality reduction, lossy data compression, feature extraction, and data visualization. It is also known as the *Karhunen-Loeve (KL)* transform.
- Two commonly used definitions of PCA:
  - PCA can be defined as the orthogonal projection of the data onto a lower dimensional linear space, known as the *principal subspace*, such that the variance of the projected data is maximized
  - Equivalently, it can be defined as the linear projection that minimizes the average projection cost, defined as the mean squared distance between the data points and their projections

Principal Component Analysis (PCA)

- Maximum variance formulation
  - Consider a dataset of observations $\{x_n\}_{n=1,\cdots,N}$ and x_n is a Euclidean variable with dimensionality D.
  - Our goal is to project the data onto a space having the dimensionality $M < D$ while maximizing the variance of the projected data.
  - The mean of the projected data is $\mathrm{u}_1^{\mathrm{T}}\bar{x}$ where $\bar{x}$ is the mean of observations and the variance of the projected data are given by:

$$\overline{\mathbf{x}} = \frac{1}{N}\sum_{n=1}^{N}\mathbf{x}_n \qquad \frac{1}{N}\sum_{n=1}^{N}\left\{\mathbf{u}_1^{\mathrm{T}}\mathbf{x}_n - \mathbf{u}_1^{\mathrm{T}}\overline{\mathbf{x}}\right\}^2 = \mathbf{u}_1^{\mathrm{T}}\mathbf{S}\mathbf{u}_1$$

where S is the data covariance matrix defined by: $\mathbf{S} = \frac{1}{N}\sum_{n=1}^{N}(\mathbf{x}_n - \overline{\mathbf{x}})(\mathbf{x}_n - \overline{\mathbf{x}})^{\mathrm{T}}$

Principal Component Analysis (PCA)

- Maximum variance formulation
  - We now maximize the projected variance $u_1^T S u_1$ with respect to u_1. We introduce a Lagrange multiplier λ_1 and then make an unconstrained maximization of:

$$\mathbf{u}_1^{\mathrm{T}}\mathbf{S}\mathbf{u}_1 + \lambda_1\left(1 - \mathbf{u}_1^{\mathrm{T}}\mathbf{u}_1\right)$$

  - By setting the derivative with respect to u_1 to zero, we have a stationary point when

$$\mathbf{S}\mathbf{u}_1 = \lambda_1\mathbf{u}_1$$

u_1 must be an eigenvector of S. If we left-multiply u_1^T and make use of $u_1^T u_1 = 1$, the variance is given by

$$\mathbf{u}_1^{\mathrm{T}}\mathbf{S}\mathbf{u}_1 = \lambda_1$$

Principal Component Analysis (PCA)

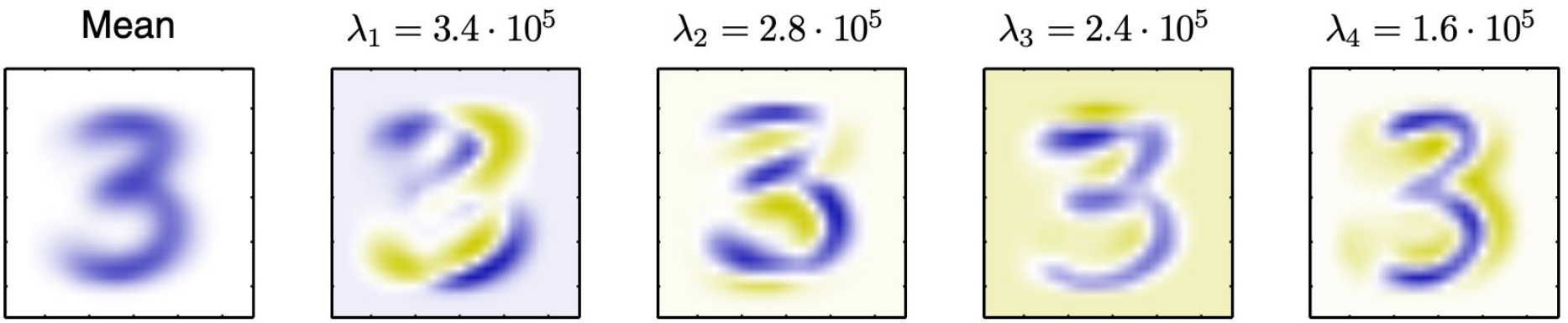
- Maximum variance formulation
  - The variance will be a maximum when we set u_1 equal to the eigenvector having the largest eigenvalue λ_1. This eigenvector is known as the first principal component. We can define additional principal components in an incremental fashion by choosing each new direction that maximizes the projected variance. Then the PCA approximation to a data vector x_n can be written in the form

$$\tilde{x}_n = \sum_{i=1}^{M} (x_n^T \mathrm{u}_i)\mathrm{u}_i + \sum_{i=M+1}^{D} (\bar{x}^T \mathrm{u}_i)\mathrm{u}_i = \bar{x} + \sum_{i=1}^{M} (x_n^T \mathrm{u}_i - \bar{x}^T \mathrm{u}_i)\mathrm{u}_i$$

  - To summarize, PCA evaluates the mean $\bar{x}$ and the covariance matrix S of the data set and then find the M eigenvectors of S corresponding to the M largest eigenvalues. Algorithms for finding eigenvectors and eigenvalues, as well as additional theorems related to eigenvector decomposition.

Applications of PCA

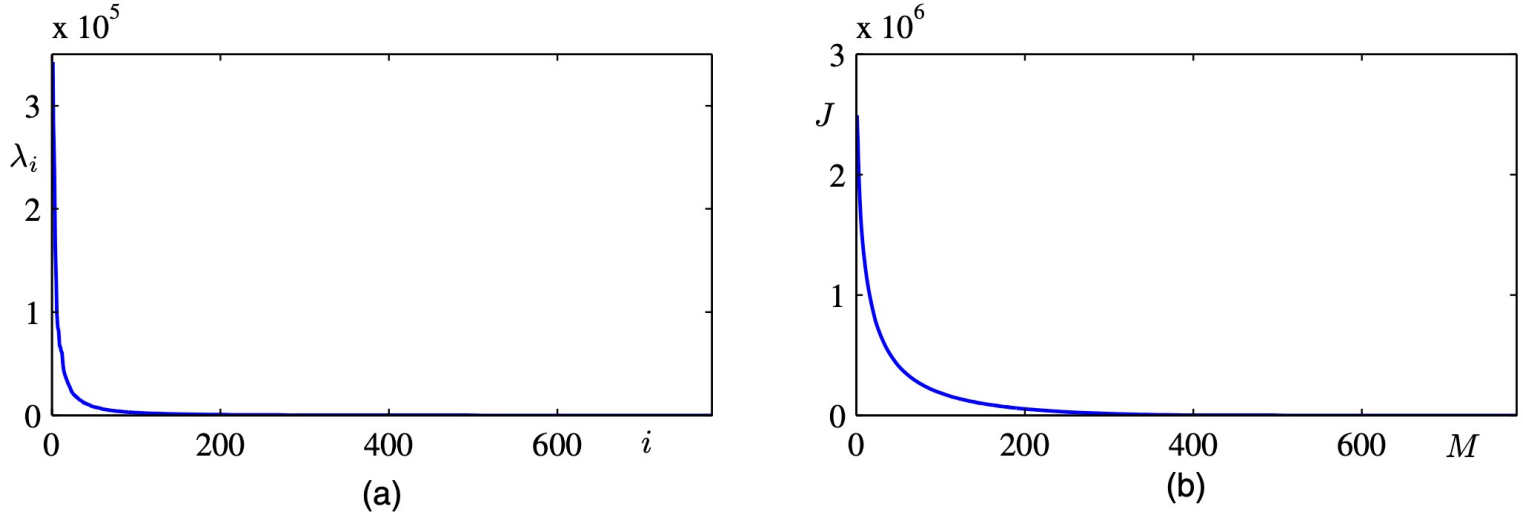
- Compression



The mean vector x along with the first four PCA eigenvectors $u_1, \cdots, u_4$ for the off-line digits data set, together with the corresponding eigenvalues.

Applications of PCA

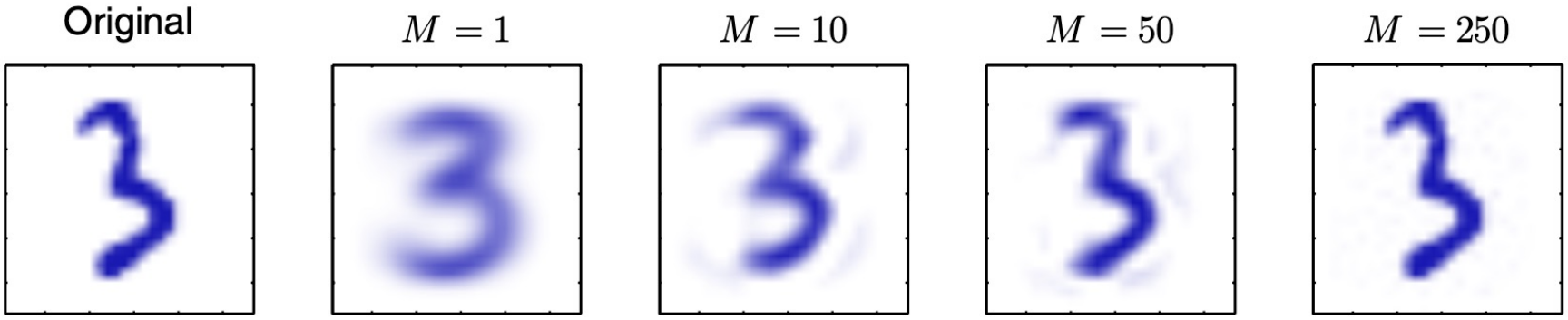
- Compression



(a): Plot of the eigenvalue spectrum for the off-line digits data set.
(b): Plot of the sum of the discarded eigenvalues, which represents the sum-of-squares distortion J introduced by projecting the data onto a principal component subspace of dimensionality M.

Applications of PCA

- Compression



An original example from the off-line digits data set together with its PCA reconstructions obtained by retaining M principal components. As M increases, the reconstruction becomes more accurate and would become perfect when $M = D = 28 \times 28 = 784$.

Applications of PCA

- Data pre-processing
  - The goal is not dimensionality reduction but the transformation of a data set in order to standardize certain of its properties.
  - The *correlation* matrix of the original data

$$\rho_{ij} = \frac{1}{N} \sum_{n=1}^{N} \frac{(x_{ni} - \overline{x}_i)}{\sigma_i} \frac{(x_{nj} - \overline{x}_j)}{\sigma_j}$$

  - If two components x_i and x_j of the data are perfectly correlated, then $\rho_{ij} = 1$, and if they are uncorrelated, then $\rho_{ij} = 0$.

Applications of PCA

- Data pre-processing
  - Using PCA we can make a more substantial normalization of the data to give it zero mean and unit covariance, so that different variables become decorrelated.
  - To do this, we first write the eigenvector equation in the form:

$$\mathbf{SU} = \mathbf{UL}$$

where L is a $D \times D$ diagonal matrix with elements λ_i, and U is a $D \times D$ orthogonal matrix with columns given by u_i.

  - For each data point x_n, a transformed value given by

$$\mathbf{y}_n = \mathbf{L}^{-1/2}\mathbf{U}^{\mathrm{T}}(\mathbf{x}_n - \overline{\mathbf{x}})$$

where $\bar{x}$ is the sample mean.

Applications of PCA

- Data pre-processing
  - Clearly, the set $\{y_n\}$ has zero mean, and its covariance is given by the identity matrix because

$$\begin{aligned}\frac{1}{N}\sum_{n=1}^{N}\mathbf{y}_n\mathbf{y}_n^{\mathrm{T}} &= \frac{1}{N}\sum_{n=1}^{N}\mathbf{L}^{-1/2}\mathbf{U}^{\mathrm{T}}(\mathbf{x}_n-\overline{\mathbf{x}})(\mathbf{x}_n-\overline{\mathbf{x}})^{\mathrm{T}}\mathbf{U}\mathbf{L}^{-1/2}\\ &= \mathbf{L}^{-1/2}\mathbf{U}^{\mathrm{T}}\mathbf{S}\mathbf{U}\mathbf{L}^{-1/2} = \mathbf{L}^{-1/2}\mathbf{L}\mathbf{L}^{-1/2} = \mathbf{I}.\end{aligned}$$

  - This operation is known as *whitening* or *sphering* the data

Applications of PCA

- Data pre-processing

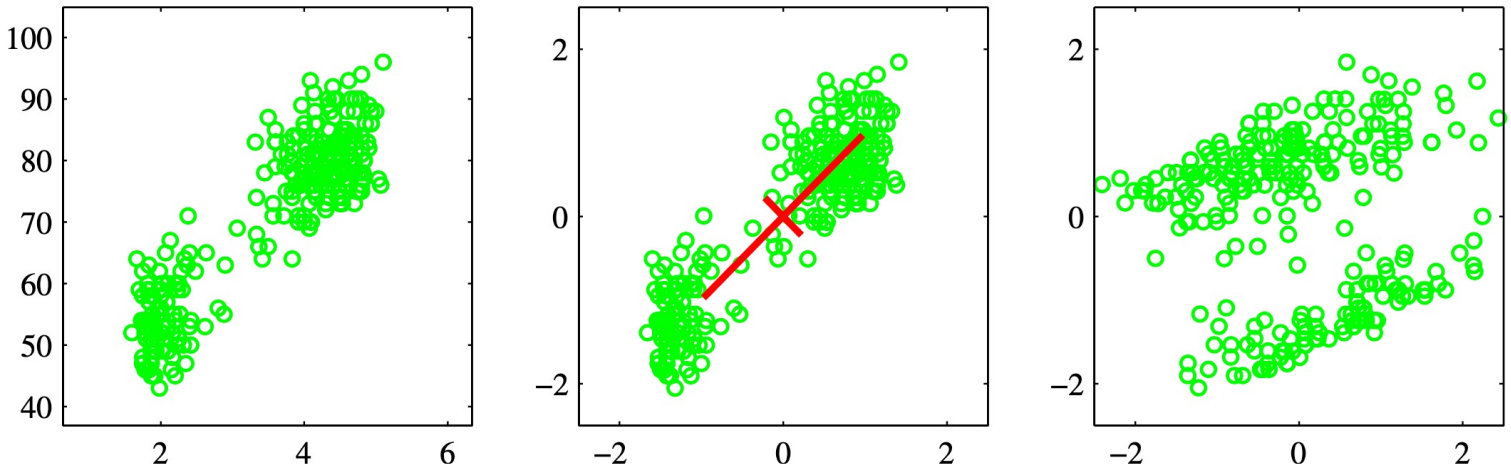


Illustration of the effects of linear pre-processing applied to the Old Faithful data set. The plot on the left shows the original data. The centre plot shows the result of standardizing the individual variables to zero mean and unit variance. Also shown are the principal axes of this normalized data set, plotted over the range $\pm\lambda_i^{1/2}$. The plot on the right shows the result of whitening of the data to give it zero mean and unit covariance.

Applications of PCA

- PCA for high-dimensional data
  - In some applications of PCA, the number of data points is smaller than the dimensionality of the data space. We can resolve this problem as follows.
  - First, let us define X as the $(N \times D)$-dimensional centered data matrix, whose n-th row is given by $(x_n - \bar{x})^T$. The covariance matrix is $S = X^T X / N$, and the corresponding eigenvector equation becomes

$$\frac{1}{N}\mathbf{X}^{\mathrm{T}}\mathbf{X}\mathbf{u}_i = \lambda_i \mathbf{u}_i$$

  - Pre-multiply both sides by X to give $\frac{1}{N} XX^T(X\mathrm{u}_i) = \lambda_i(X\mathrm{u}_i)$. Let $v_i = Xu_i$, we obtain

$$\frac{1}{N}\mathbf{X}\mathbf{X}^{\mathrm{T}}\mathbf{v}_i = \lambda_i \mathbf{v}_i$$

  which is an eigenvector equation for the matrix XX^T/N

Applications of PCA

- PCA for high-dimensional data
  - In order to determine the eigenvectors, we multiply both sides of previous equation by X^T to give

$$\left(\frac{1}{N}\mathbf{X}^{\mathrm{T}}\mathbf{X}\right)(\mathbf{X}^{\mathrm{T}}\mathbf{v}_i) = \lambda_i(\mathbf{X}^{\mathrm{T}}\mathbf{v}_i)$$

  - $(X^T v_i)$ is an eigenvector of S with eigenvalue λ_i. Note that these eigenvectors are not normalized. To determine the appropriate normalization, we re-scale $u_i \propto X^T v_i$ by a constant such that $||u_i|| = 1$, which, assuming v_i has been normalized to unit length, gives

$$\mathbf{u}_i = \frac{1}{(N\lambda_i)^{1/2}}\mathbf{X}^{\mathrm{T}}\mathbf{v}_i$$

上海交通大學
SHANGHAI JIAO TONG UNIVERSITY

Non-negative Matrix Factorization

Non-negative Matrix Factorization

- Non-negative matrix factorization (Lee and Seung, 1999) is an **alternative approach to PCA**, in which the data and components are assumed to be non-negative. It is useful for modeling non-negative data such as images.

- The $N \times p$ data matrix $\boldsymbol{X}$ is approximated by $\boldsymbol{X} \approx \boldsymbol{WH}$, where $\boldsymbol{W}$ is $N \times r$ and $\boldsymbol{H}$ is $r \times p$, $r \leq \max(N, p)$. We assume that $x_{ij}, w_{ik}, h_{kj} \geq 0$.

- The matrices $\boldsymbol{W}$ and $\boldsymbol{H}$ are found by maximizing

$$L(\boldsymbol{W}, \boldsymbol{H}) = \sum_{i=1}^{N} \sum_{j=1}^{p} \left[x_{ij} \log(\boldsymbol{WH})_{ij} - (\boldsymbol{WH})_{ij} \right]$$

This is the log-likelihood from a model in which x_{ij} has a Poisson distribution with mean $(\boldsymbol{WH})_{ij}$—quite reasonable for positive data.

Non-negative Matrix Factorization

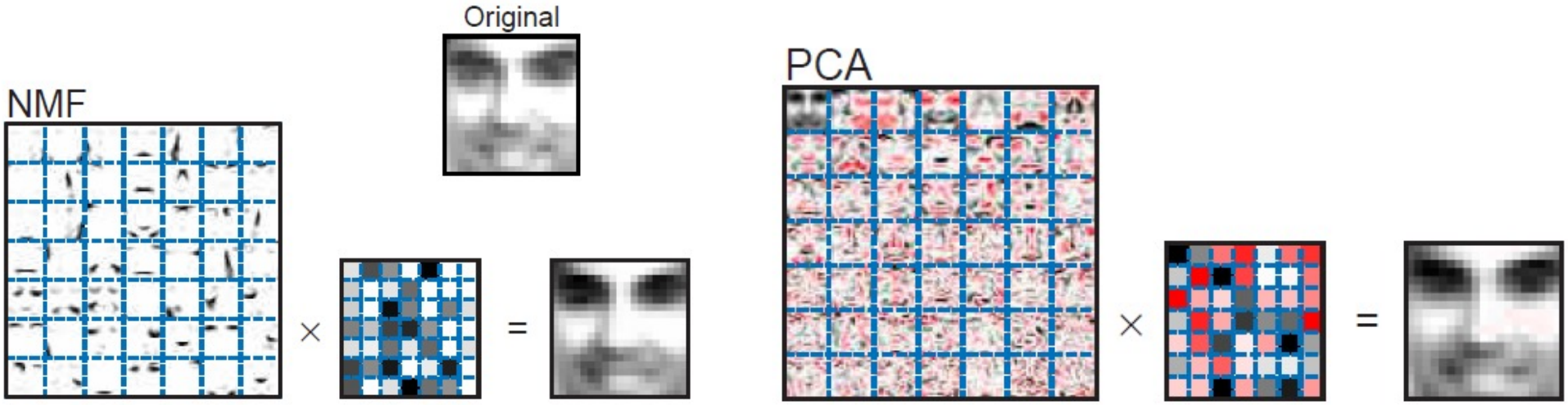
- The following alternating algorithm (Lee and Seung, 2001) converges to a local maximum of $L(\boldsymbol{W}, \boldsymbol{H})$:

$$w_{ik} \leftarrow w_{ik} \frac{\sum_{j=1}^{p} h_{kj} x_{ij} / (\mathbf{WH})_{ij}}{\sum_{j=1}^{p} h_{kj}}$$

$$h_{kj} \leftarrow h_{kj} \frac{\sum_{i=1}^{N} w_{ik} x_{ij} / (\mathbf{WH})_{ij}}{\sum_{i=1}^{N} w_{ik}}$$

This algorithm can be derived as a minorization procedure for maximizing $L(\boldsymbol{W}, \boldsymbol{H})$ and is also related to the iterative-proportional-scaling algorithm for log-linear models.

Non-negative Matrix Factorization



The above figure shows an example taken from Lee and Seung (1999), comparing non-negative matrix factorization (NMF) and principal components analysis (PCA). Unlike PCA, NMF learns to represent faces with a set of basis images resembling parts of faces.

上海交通大學
SHANGHAI JIAO TONG UNIVERSITY

The Google PageRank Algorithm

The Google PageRank Algorithm

- The PageRank algorithm considers a webpage to be important if many other webpages point to it.

- The linking webpages that point to a given page are not treated equally: the algorithm also takes into account both the importance (PageRank) of the linking pages and the number of outgoing links that they have.

- Linking pages with higher PageRank are given more weight, while pages with more outgoing links are given less weight.

The Google PageRank Algorithm

- Let $L_{ij} = 1$ if page j points to page i, and zero otherwise. Let $c_j = \sum_{i=1}^{N} L_{ij}$ denote the number of pages pointed to by page j (number of out-links). Then the *Google PageRanks* p_i are defined by the recursive relationship

$$p_i = (1-d) + d\sum_{j=1}^{N}\left(\frac{L_{ij}}{c_j}\right)p_j$$

where d is a positive constant that ensures that each page gets a *PageRank* of at least $1-d$. In matrix notation,

$$\boldsymbol{p} = (1-d)\boldsymbol{e} + d \cdot \boldsymbol{L}\boldsymbol{D}_c^{-1}\boldsymbol{p}$$

where $\boldsymbol{e}$ is a vector of N ones and $\boldsymbol{D}_c = \mathrm{diag}(\boldsymbol{c})$ is a diagonal matrix with diagonal elements c_j.

The Google PageRank Algorithm

$$\boldsymbol{p} = (1-d)\boldsymbol{e} + d \cdot \boldsymbol{L}\boldsymbol{D}_c^{-1}\boldsymbol{p}$$

- Introducing the normalization $\boldsymbol{e}^T\boldsymbol{p} = N$ (i.e., the average PageRank is 1), rewrite the above equation

$$\boldsymbol{p} = \left[\frac{(1-d)\boldsymbol{e}\boldsymbol{e}^T}{\boldsymbol{N}} + d\boldsymbol{L}\boldsymbol{D}_c^{-1}\right]\boldsymbol{p} = \boldsymbol{A}\boldsymbol{p}$$

- Exploiting a connection with Markov chains (see below), the matrix **A** has a real eigenvalue equal to one, and one is its largest eigenvalue. This means that we can find $\hat{\boldsymbol{p}}$ by the power method: starting with some $\boldsymbol{p} = \boldsymbol{p}_0$ we iterate

$$\boldsymbol{p}_k \leftarrow \boldsymbol{A}\boldsymbol{p}_{k-1};\ \boldsymbol{p}_k \leftarrow \frac{N\boldsymbol{p}_k}{\boldsymbol{e}^T\boldsymbol{p}_k}.$$

The fixed points $\hat{p}$ are the desired *PageRanks*.

The Google PageRank Algorithm $\boldsymbol{p} = (1-d)\boldsymbol{e} + d \cdot \boldsymbol{L}\boldsymbol{D}_c^{-1}\boldsymbol{p}$

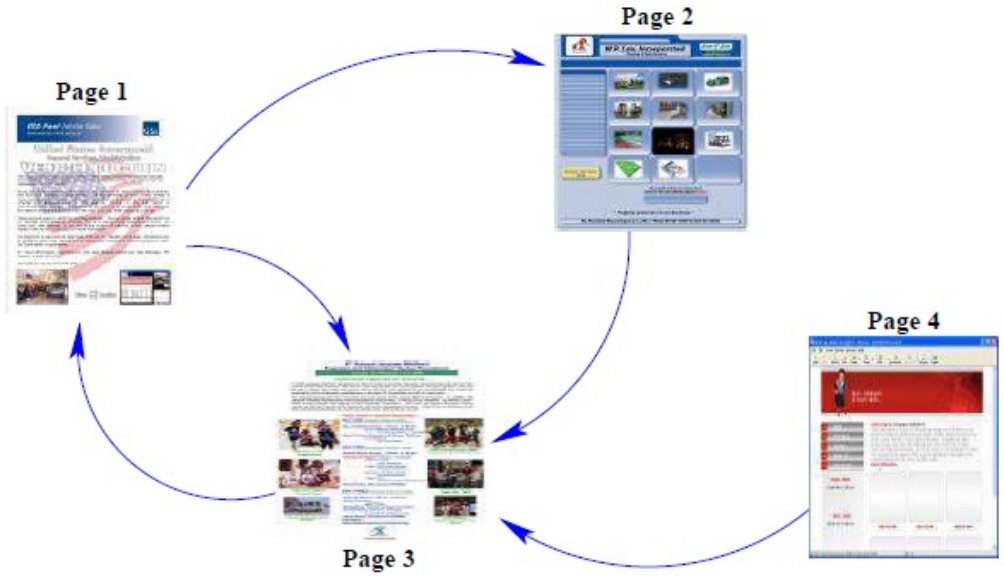
- The factor $1 - d$ is the probability that he does not click on a link, but jumps instead to a random webpage.

- Some descriptions of *PageRank* have $(\mathbf{1} - \boldsymbol{d})/\boldsymbol{N}$ as the first term in the above equation, which would better coincide with the random surfer interpretation. Then the page rank solution (divided by N) is the stationary distribution of an irreducible, aperiodic Markov chain over the N webpages.

- Viewing PageRank as a Markov chain makes clear why the matrix **A** has a maximal real eigenvalue of 1. Since $\boldsymbol{A}$ has positive entries with each column summing to one, Markov chain theory tells us that it has a unique eigenvector with eigenvalue one, corresponding to the stationary distribution of the chain.

The Google PageRank Algorithm

Simple Example



PageRank algorithm: example of a small network

$$L = \begin{pmatrix} 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 \\ 1 & 1 & 0 & 1 \\ 0 & 0 & 0 & 0 \end{pmatrix}$$

The number of outlinks $c = (2,1,1,1)$.

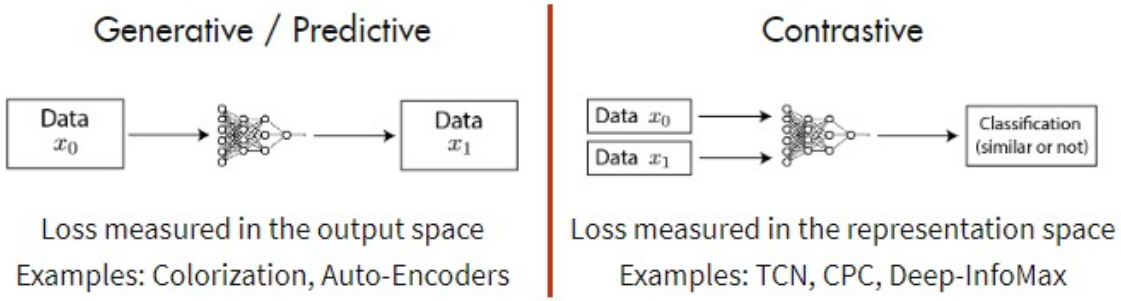
$p = (1.49,0.78,1.58,0.15)$.

上海交通大學
SHANGHAI JIAO TONG UNIVERSITY

Contrastive Learning

Contrastive Learning

- Contemporary self-supervised learning methods can roughly be broken down into two classes of methods:



- Contrastive methods, as the name implies, ***learn representations by contrasting positive and negative examples***. They differ from the more traditional generative methods to learn representations, which focus on reconstruction error in the pixel space to learn representations.

Contrastive Learning

- Contrastive methods trained on unlabelled ImageNet data and evaluated with a linear classifier now surpass the accuracy of supervised AlexNet. They also exhibit significant data efficiency when learning from labelled data compared to purely supervised learning (Data-Efficient CPC, Hénaff et al., 2019).
- Contrastive pre-training on ImageNet successfully transfers to other downstream tasks and outperforms the supervised pre-training counterparts (MoCo, He et al., 2019).
- Using pixel-level losses can lead to such methods being overly focused on pixel-based details, rather than more abstract latent factors.
- Pixel-based objectives often assume independence between each pixel, thereby reducing their ability to model correlations or complex structure.

Contrastive Learning

How do contrastive methods work?

For any data point x, contrastive methods aim to learn an encoder *f* such that:

$$\text{score}(f(x), f(x^+)) >> \text{score}(f(x), f(x^-))$$

- x^+ is data point similar or congruent to x, referred to as a positive sample.
- x^- is a data point dissimilar to xx, referred to as a negative sample.
- The **score** function is a metric that measures the similarity between two features.

x is referred to as an "anchor" data point. To optimize for this property, we can construct a softmax classifier that classifies positive and negative samples correctly. It encourages the score function to assign large values to positive examples and small values to negative examples:

$$\mathcal{L}_N = -\mathbb{E}_X \left[\log \frac{\exp\left(f(x)^T f\left(x^+\right)\right)}{\exp\left(f(x)^T f\left(x^+\right)\right) + \sum_{j=1}^{N-1} \exp\left(f(x)^T f\left(x_j\right)\right)} \right]$$

Contrastive Learning

How do contrastive methods work?

- The denominator terms consist of one positive, and $N-1$ negative samples. Here, we use the dot product as the score function:

$$\text{score}(f(x), f(x^+)) = f(x)^T f(x^+)$$

This is the familiar cross-entropy loss for an N-way softmax classifier, and commonly called the **InfoNCE loss** in the contrastive learning literature.

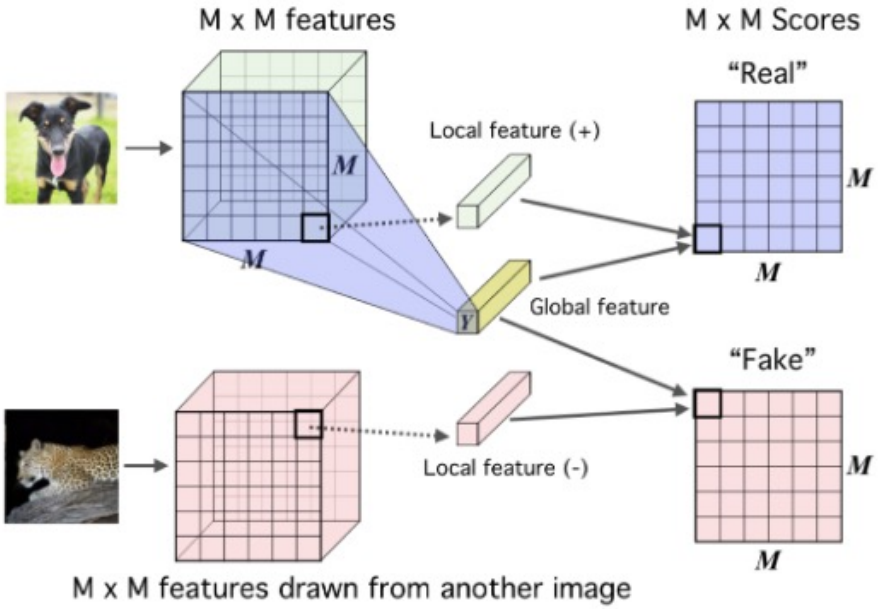
Interpretation: The InfoNCE objective is also connected to mutual information. Specifically, minimizing the InfoNCE loss maximizes a lower bound on the mutual information between $f(X)$ and $f(X^+)$.

Institute of Media, Information, and Network

生命科学技术学院
School of Life Sciences and Biotechnology

Contrastive Learning

Different contrastive methods



The contrastive task in Deep InfoMax. Image source: Hjelm et al., 2018

Deep InfoMax (DIM, Hjelm et al., 2018) learns representations of images by leveraging the local structure present in an image. The contrastive task behind DIM is to classify whether a pair of global features and local features are from the same image or not. Here, global features are the final output of a convolutional encoder (a flat vector, Y) and local features are the output of an intermediate layer in the encoder (an M x M feature map). Each local feature map has a limited receptive field. So, intuitively this means to do well at the contrastive task the global feature vector must capture information from all the different local regions.

Contrastive Learning

Different contrastive methods

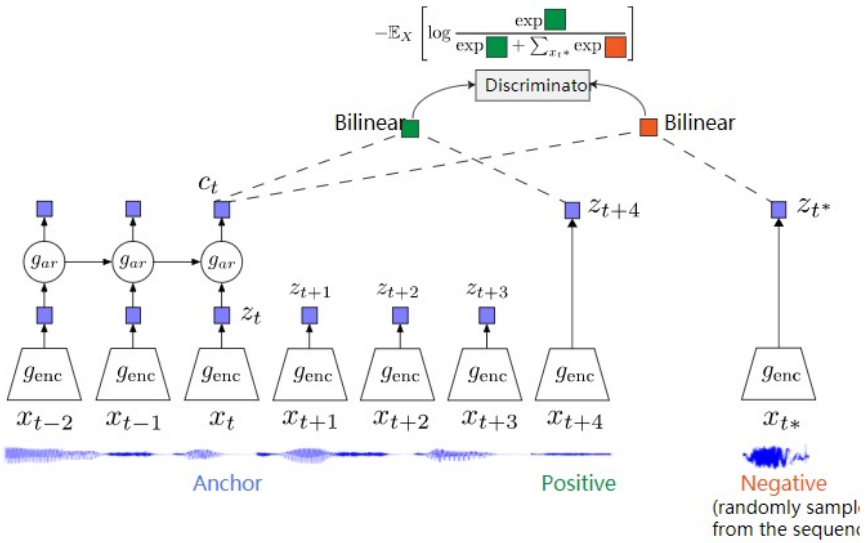


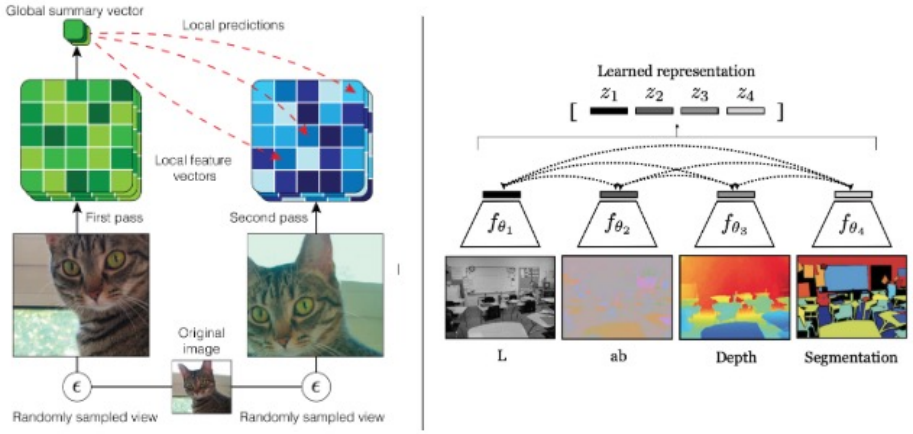
Illustration of the contrastive task in CPC wih an audio input . Image adapted from van den Oord et al., 2018

- Contrastive Predictive Coding (CPC, van den Oord et al., 2018) is a contrastive method that can be applied to any form of data that can be expressed in an ordered sequence: text, speech, video, even images (an image can be seen as a sequence of pixels or patches).

- CPC learns representations by encoding information that's shared across data points multiple time steps apart, discarding local information. These features are often called "slow features": features that don't change too quickly across time. Examples include identity of a speaker in an audio signal, an activity carried out in a video, an object in an image etc.

Contrastive Learning

Different contrastive methods



Left: AMDIM learns representations that are invariant across data augmentations such as random-crop. Right: CMC learns representations that are invariant across different views (channels) of an image. Image source: Bachman et al., 2019 and Tian et al., 2019

- Contrastive Learning provides an easy way to impose invariances in the representation space. Suppose we want a representation to be invariant to a transformation T (for example crops, grayscaling), we can simply construct a contrastive objective where given an anchor data point x, (a) T(x) is a positive sample. (b) T(x′) where x′ is a random image or data point is a negative sample.

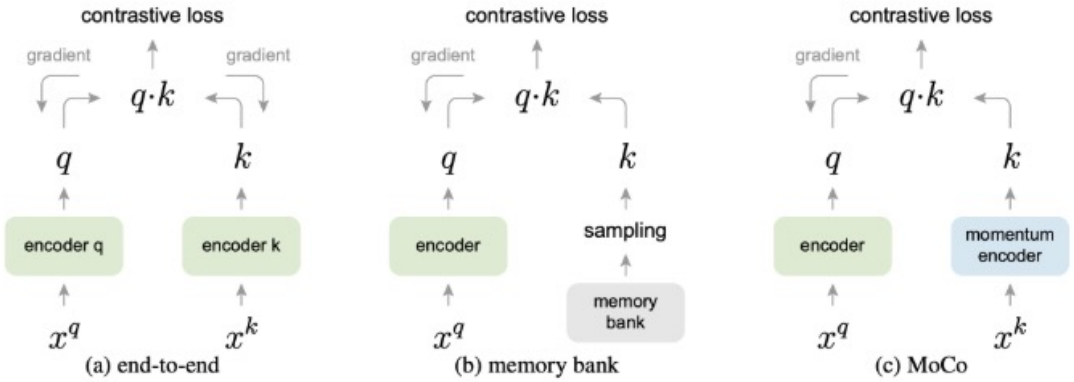
- Augmented Multiscale DIM (AMDIM, Bachman et al., 2019) uses standard data augmentation techniques as the set of transformations a representation should be invariant to.
- Contrastive MultiView Coding (CMC, Tian et al., 2019) uses different views of the same image (depth, luminance, luminance, chrominance, surface normal, and semantic labels) as the set of transformations the representation should be invariant to.

Institute of Media, Information, and Network

生命科学技术学院
School of Life Sciences and Biotechnology

Contrastive Learning

Different contrastive methods



Strategies of using negative samples in contrastive methods. Here, xqxq *are positive examples, and* xkxk *negative examples. The gradient does not flow back through the momentum encoder in MoCo. Image source: He et al., 2019*

- Contrastive methods tend to work better with more number of negative examples, since presumably larger number of negative examples may cover the underlying distribution more effectively and thus give a better training signal. In the usual formulation of contrastive learning, the gradients flow back through the encoders of both the positive and negative samples. This means that the number of negative samples is restricted to the size of the mini-batch.

- Momentum Contrast (MoCo, He et al., 2019) gets around this effectively by maintaining a large queue of negative samples, and not using backpropagation to update the negative encoder. Instead it periodically updates the negative encoder using a momentum update:

$$\theta_k \leftarrow m\theta_k + (1-m)\theta_q$$

θ_k: the weights of the encoder for negative examples

θ_q: the weights of the encoder for positive examples

Institute of Media, Information, and Network

生命科学技术学院
School of Life Sciences and Biotechnology

Q & A

Many Thanks