Data Analysis

Linear Algebra Refresher

Thm. **0.0.1 Rank Theorem:**

- Let $A \in \mathbb{R}^{m\times n}, B \in \mathbb{R}^{n\times k}$, with $\text{rank}(B) = n$. Then $\text{rank}(AB) = \text{rank}(A)$.
- Let $A \in \mathbb{R}^{m\times n}, C \in \mathbb{R}^{\ell\times m}$, with $\text{rank}(C) = n$. Then $\text{rank}(CA) = \text{rank}(A)$.

1 Week 1

1.1 Principal Component Analysis (PCA)

1.1.1 Statistical POV of PCA

We treat $\boldsymbol{x} \in \mathbb{R}^D$ as a random vector, aka a multivariate random variable. We assume features are scaled, i.e. $E[x_i] = 0$ and $\text{Var}[x_i] = 1$. Generally, $E[x_i x_j] \neq 0$, therefore the features are correlated. We wish to find a representation $\boldsymbol{y} \in \mathbb{R}^d$, $d << D$, given as a linear combination of $\boldsymbol{x}$, i.e. $\boldsymbol{y} = A\boldsymbol{x}, A \in \mathbb{R}^{d\times D}$, such that the features are uncorrelated, i.e.

$$\text{Cov}[y_i, y_j] = E[y_i y_j] - E[y_i]E[y_j] = 0$$

and since $E[y_i] = E[y_j] = 0$, this means we look for $E[y_i y_j] = 0$. In addition, we look for the most "meaningful" uncorrelated directions - those directions along which $\boldsymbol{x}$ varies the most. How do we capture this? Recall that with the random matrix:

$$P_x = \boldsymbol{x}\boldsymbol{x}^T$$

and $\text{Cov}[\boldsymbol{x}] = E[\boldsymbol{x}\boldsymbol{x}^T]$ we can measure the variance of the projection of $\boldsymbol{x}$ along a vector $\boldsymbol{u}$ by:

$$\text{Var}[\boldsymbol{u}^T\boldsymbol{x}] = \boldsymbol{u}^T E[\boldsymbol{x}\boldsymbol{x}^T]\boldsymbol{u} = E[\boldsymbol{u}^T\boldsymbol{x}\boldsymbol{x}^T\boldsymbol{u}] = E[(\boldsymbol{u}^T\boldsymbol{x})^2]$$

This gives the objective function:

$$\boldsymbol{u}_i^\star = \arg\max_{\boldsymbol{u}_i} E[(\boldsymbol{u}_i^T\boldsymbol{x})^2]$$
$$\text{s.t. } \boldsymbol{u}_i^T\boldsymbol{u}_j = \delta_{ij} \text{ for } j \leq i$$

where the condition means that we require the directions to be uncorrelated, since

$$E[(y_2)(y_1)] = E[(\boldsymbol{u}_1^T\boldsymbol{x})(\boldsymbol{u}_2^T\boldsymbol{x})] = E(\boldsymbol{u}_1^T\boldsymbol{x}\boldsymbol{x}^T\boldsymbol{u}_2)] = \boldsymbol{u}_1 P_x \boldsymbol{u}_2 = \lambda_1 \boldsymbol{u}_1^T\boldsymbol{u}_2$$

so $E[(y_2)(y_1)] = 0$ iff $\boldsymbol{u}_1 \perp \boldsymbol{u}_2$.

Since $\text{Var}[\boldsymbol{u}^T\boldsymbol{x}] = \boldsymbol{u}^T P_x \boldsymbol{u}$, we have from Courant-Fischer that the largest eigenvectors of P_x are those that maximize it.

1.1.2 Sample POV of PCA

Assume we have n samples of dimension d, each sample denoted by $\boldsymbol{x}^{(i)} \in \mathbb{R}^d$. Denote $X \in \mathbb{R}^{n\times d}$, where each row contains a sample. Assume the samples are centered. PCA seeks to find the directions (principal components) that maximize the variance of the projected data. Let S denote the sample covariance matrix:

$$S \overset{\text{def}}{=} \frac{1}{n-1}X^T X = \frac{1}{n-1}\sum_{i=1}^n \boldsymbol{x}^{(i)}(\boldsymbol{x}^{(i)})^T = \frac{1}{n-1}\sum_{i=1}^n \boldsymbol{x}^{(i)} \otimes \boldsymbol{x}^{(i)}$$

Then $S \in \mathbb{R}^{d\times d}$.

Objective Function PCA solves the following optimization problem:

$$\max_{\boldsymbol{w}} \boldsymbol{w}^T S \boldsymbol{w} \quad \text{subject to } \|\boldsymbol{w}\| = 1$$

The principal components are the eigenvectors $\boldsymbol{w}$ corresponding to the largest eigenvalues of S. Since S is PSD, it has real non-negative eigenvalues, and we write:

$$S = V\Lambda V^T$$

with $V \in \mathbb{R}^{d\times d}$, and its columns, the eigenvectors of S are called the **principal directions**.

Limitations: PCA can only capture linear relations when the covariance matrix is computed in the original feature space. Hence for data with nonlinear structure, PCA may not reduce dimensionality effectively. The solution is to use Feature Mapping.

Remark (Summary): Let $X \in \mathbb{R}^{n\times d}$ be a matrix of n samples with d features each. Assume the data is centered. We perform PCA by computing $S = \frac{1}{n-1}X^TX = V\Lambda V^T$. We call the columns of V the **principal directions**. We can then embed each sample of X into a lower dimension k by projecting it into the first k principal directions, getting k **principal scores**. We can then reconstruct the sample by using the k principal scores to take a weighted sum of the k principal directions, producing a reconstruction. Note - we don't use the eigenvalues in Λ in the embedding/reconstruction process. They only provide measurement of the variance of the i-th principal direction.

Remark (Transpose Trick): Let $X \in \mathbb{R}^{m\times n}$, with $m >> n$, with m features. To find principal directions, the eigenvalues of XX^T, we need to compute XX^T, an $m \times m$ matrix, which is computationally expensive. Instead, we can use the following relation: Let v be an eigenvector XX^T, i.e.

$$\begin{aligned} XX^Tv &= \lambda v \\ \Rightarrow X^TX(X^Tv) &= \lambda X^Tv \\ \Rightarrow X^TXw &= \lambda w \end{aligned}$$

I.e. $w := X^Tv$ is an eigenvector of X^TX with eigenvalue λ. Additionally, we have that:

$$\begin{aligned} Xw &= XX^Tv = \lambda v \\ \Rightarrow v &= \lambda^{-1}Xw \end{aligned}$$

Therefore, we can compute X^TX eigenpairs (w, λ) and get XX^T eigenpairs (v, λ) by $v = \lambda^{-1}Xw$.

1.2 Feature Mapping

Feature Space By mapping our data to higher feature space $\mathcal{F}$, we can separate non-linear data, as in Fig. 1. Denote by

$$\phi : \mathbb{R}^d \to \mathcal{F} \qquad \boldsymbol{x}^{(i)} \to \phi(\boldsymbol{x}^{(i)})$$

recall that $\boldsymbol{x}^{(i)} \in \mathbb{R}^d$.

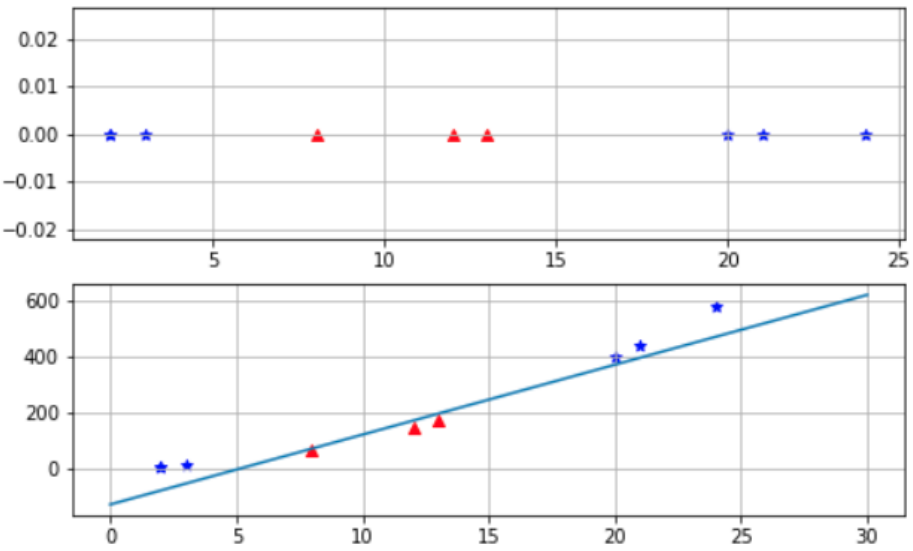


Figure 1: Non-linear data. Here, the function is $\phi : \mathbb{R} \to \mathbb{R}^2$, with $\phi(x) = (x, x^2)$.

In this new representation, we **can** linearly separate the samples. Note - we do need to recenter the projected samples. We will write $\phi(\boldsymbol{x}^{(i)})$ again assuming it is centered. Another assumption is that $\left\{\phi(\boldsymbol{x}^{(i)})\right\}_{i=1}^n$ are linearly independent.

Now we can use PCA, only we do so on the new covariance matrix:

$$S_\phi \overset{\text{def}}{=} \frac{1}{n}\sum_{i=1}^{n}\phi(\boldsymbol{x}^{(i)})\phi(\boldsymbol{x}^{(i)})^T$$

Continuing similarly to PCA, we wish to find the (largest) eigenvectors of S_ϕ. If we denote $\mathcal{F}$ dimension by m, then $S_\phi \in \mathbb{R}^{m\times m}$. This suggest a new problem - if $m >> n$, then computations become expensive. Luckily, we notice that:

Thm. **1.2.1 The kernel theorem:**

Computing the (nonzero) eigenvectors of S_ϕ requires only knowing the inner products

$$k(\boldsymbol{x}^{(i)}, \boldsymbol{x}^{(j)}) := \left\langle \phi(\boldsymbol{x}^{(i)}) \, , \, \phi(\boldsymbol{x}^{(j)}) \right\rangle$$

Hence, if computing $k(\boldsymbol{x}^{(i)}, \boldsymbol{x}^{(j)})$ can be done easily, then we reduced the complexity of the problem

Proof. We will first prove a lemma.

Lemma 1.2.2: Range of S_ϕ: Recall that:

$$S_\phi = \frac{1}{n}\sum_{i=1}^{n} \phi(\boldsymbol{x}^{(i)})\phi(\boldsymbol{x}^{(i)})^T$$

Let $\boldsymbol{w} \in \mathcal{F}$ be any vector. Then:

$$S_\phi \boldsymbol{w} = \frac{1}{n}\sum_{i=1}^{n} \phi(\boldsymbol{x}^{(i)})\phi(\boldsymbol{x}^{(i)})^T \boldsymbol{w} = \frac{1}{n}\sum_{i=1}^{n} \phi(\boldsymbol{x}^{(i)}) \left\langle \phi(\boldsymbol{x}^{(i)}) \, , \, \boldsymbol{w} \right\rangle$$

Therefore, $S_\phi \boldsymbol{w}$ is a linear combination of $\phi(\boldsymbol{x}^{(i)})$, with the coefficients being $\langle \phi(\boldsymbol{x}^{(i)}, \boldsymbol{w}\rangle$. Therefore, we can express $S_\phi \boldsymbol{w}$ as a linear combination of $\left\{\phi(\boldsymbol{x}^{(i)})\right\}_{i=1}^{n}$. From the assumption that $\left\{\phi(\boldsymbol{x}^{(i)})\right\}_{i=1}^{n}$ are linearly independent, this representation is unique.

End of Lemma

Continuing - denote by $\boldsymbol{v}$ a (yet unknown) eigenvector of S_ϕ, i.e.:

$$S_\phi \boldsymbol{v} = \lambda \boldsymbol{v} \tag{1}$$

Using the lemma, we can write:

$$\boldsymbol{v} = \sum_{j=1}^{n} \alpha_j \phi(\boldsymbol{x}^{(j)}) \tag{2}$$

Where $\alpha_j \in \mathbb{R}$ are the coefficients we wish to determine, and we write $\boldsymbol{\alpha} = [\alpha_1, \ldots, \alpha_n]^T$. Plugging this into Eq. (1), we get:

$$\begin{aligned} S_\phi \boldsymbol{v} &= \frac{1}{n}\Big(\sum_{i=1}^{n} \phi(\boldsymbol{x}^{(i)})\phi(\boldsymbol{x}^{(i)})^T\Big)\boldsymbol{v} \\ &= \frac{1}{n}\Big(\sum_{i=1}^{n} \phi(\boldsymbol{x}^{(i)})\phi(\boldsymbol{x}^{(i)})^T\Big)\sum_{j=1}^{n} \alpha_j \phi(\boldsymbol{x}^{(j)}) \\ &= \frac{1}{n}\sum_{i=1}^{n} \phi(\boldsymbol{x}^{(i)}) \sum_{j=1}^{n} \alpha_j \langle \phi(\boldsymbol{x}^{(i)}) \, , \, \phi(\boldsymbol{x}^{(j)})\rangle \end{aligned}$$

Define a new matrix $K \in \mathbb{R}^{n\times n}$ by:

$$K_{ij} = \langle \phi(\boldsymbol{x}^{(i)}), \phi(\boldsymbol{x}^{(j)})\rangle$$

Then we get that:

$$S_\phi \boldsymbol{v} = \frac{1}{n}\sum_{i=1}^{n} \phi(\boldsymbol{x}^{(i)}) \sum_{j=1}^{n} \alpha_j K_{ij}$$

Define a new variable:

$$\beta_i \equiv \sum_{j=1}^{n} \alpha_j K_{ij}$$

Then:

$$S_\phi \boldsymbol{v} = \frac{1}{n}\sum_{i=1}^{n} \phi(\boldsymbol{x}^{(i)})\beta_i$$

Developing the RHS of Eq. (1), we have:

$$\lambda \boldsymbol{v} = \lambda \sum_{i=1}^{n} \alpha_i \phi(\boldsymbol{x}^{(i)})$$

Rewriting Eq. (1), we get:

$$\frac{1}{n}\sum_{i=1}^{n} \phi(\boldsymbol{x}^{(i)})\beta_i = \lambda \sum_{i=1}^{n} \alpha_i \phi(\boldsymbol{x}^{(i)})$$

From the linear independence of $\left\{\phi(\boldsymbol{x}^{(i)})\right\}_{i=1}^{n}$, we know the coefficient for each $\phi(\boldsymbol{x}^{(i)})$ must be equal, hence we get:

$$\begin{aligned} &\frac{1}{n}\beta_i = \lambda\alpha_i \\ \Rightarrow &\sum_{j=1}^{n} K_{ij}\alpha_j = n\lambda\alpha_i \\ \Rightarrow &\left(K\boldsymbol{\alpha}\right)_i = n\lambda\alpha_i \\ \Rightarrow &K\boldsymbol{\alpha} = n\lambda\boldsymbol{\alpha} \end{aligned}$$

Therefore, the original problem of finding eigenvectors for the $m \times m$ matrix S_ϕ is actually equivalent to finding the eigenvectors of the $n \times n$ matrix K. Since $m >> n$, the second task is computationally easier. Using k, we compute matrix K, find its eigenvalues $\boldsymbol{\alpha}_1, \ldots \boldsymbol{\alpha}_m$ with eigenvalues $\lambda_1, \ldots, \lambda_m$, and proceed.

Note that K is defined by the inner product $K_{ij} = \langle \phi(\boldsymbol{x}^{(i)}), \phi(\boldsymbol{x}^{(j)}) \rangle$. Hence, in situations where such a computation is easy, i.e. in situations where we have a **kernel function** $k(\boldsymbol{x}^{(i)}, \boldsymbol{x}^{(j)})$ such that:

$$k(\boldsymbol{x}^{(i)}, \boldsymbol{x}^{(j)}) = \langle \phi(\boldsymbol{x}^{(i)}), \phi(\boldsymbol{x}^{(j)}) \rangle$$

We reduced the complexity of the problem. □

1.3 Kernel PCA

We finish by performing kernel PCA, projecting the lifted samples $\phi(\boldsymbol{x}^{(i)})$ onto the m-th largest eigenvector of the (centered) kernel matrix K, using **only the kernel function**:

$$k(\boldsymbol{x}^{(i)}, \boldsymbol{x}^{(j)}) = \langle \phi(\boldsymbol{x}^{(i)}, \phi(\boldsymbol{x}^{(j)}) \rangle$$

Let $\boldsymbol{\alpha}_m = [(\boldsymbol{\alpha}_m)_1, \ldots, (\boldsymbol{\alpha}_m)_n]$ denote the m-th largest eigenvector of the centered kernel matrix K, and $\boldsymbol{v}_m$ the m-th largest eigenvector of S_ϕ. Remember that $\boldsymbol{v}_m = \sum_{j=1}^{n} \phi(\boldsymbol{x}^{(j)})\,(\boldsymbol{\alpha}_m)_j$. Then we have that:

$$\begin{aligned} \langle\, \phi(\boldsymbol{x}^{(i)})\,,\, \boldsymbol{v}_m \,\rangle &= \langle\, \phi(\boldsymbol{x}^{(i)})\,,\, \sum_{j=1}^{n} \phi(\boldsymbol{x}^{(j)})(\boldsymbol{\alpha}_m)_j \,\rangle \\ &= \sum_{j=1}^{n} (\boldsymbol{\alpha}_m)_j \,\langle\, \phi(\boldsymbol{x}^{(i)})\,,\, \phi(\boldsymbol{x}^{(j)})\,\rangle \\ &= \sum_{j=1}^{n} (\boldsymbol{\alpha}_m)_j \, k(i,j) \end{aligned}$$

And therefore, we can project the samples, and hence perform kernel PCA, using **only** the kernel function $k(i,j)$.

@ Example: Consider a dataset consisting of points lying on a circle on $\mathbb{R}^2$:

$$\boldsymbol{x}^{(i)} = \begin{bmatrix} \cos(\theta_i) \\ \sin(\theta_i) \end{bmatrix}, \quad \theta_i = \frac{2\pi i}{n}, \quad i = 1, \ldots, n$$

clearly this is a 1 dimensional data, however it currently in $\mathbb{R}^2$. Since this data is has a non-linear structure, classic PCA would not help in reducing its dimension. We use the RBF function:

$$k(\boldsymbol{x}^{(i)}, \boldsymbol{x}^{(j)}) = e^{-\frac{\|\boldsymbol{x}^{(i)} - \boldsymbol{x}^{(j)}\|}{2\sigma^2}}$$

For simulation, The un-centered kernel matrix K is computed for $n = 100$ and $\sigma = 0.5$.

The kernel matrix is centered as:

$$K_{\text{centered}} = K - \frac{1}{n}\mathbf{1}K - K\frac{1}{n}\mathbf{1} + \frac{1}{n}\mathbf{1}K\frac{1}{n}\mathbf{1},$$

where $\mathbf{1}$ is the all-ones matrix.

The eigenvalues and eigenvectors of K_{centered} are computed. The eigenvalues in descending order are:

$$\lambda_1 = 17.8751, \quad \lambda_2 = 17.8751, \quad \lambda_3 = 11.7627, \ldots$$

The projection onto the first principal component is given by:

$$\boldsymbol{z} = K_{\text{centered}}\boldsymbol{v}_1,$$

where $\boldsymbol{v}_1$ is the eigenvector corresponding to λ_1.

Below is a plot of the projection:

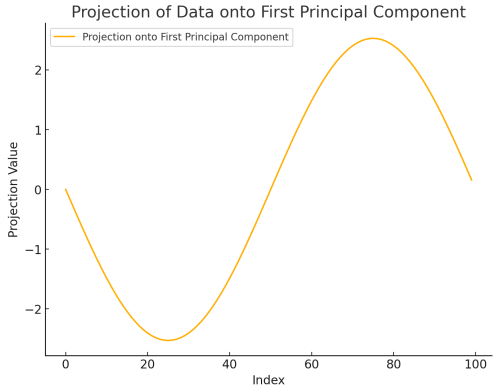


Figure 2: Projecting the points on the circle using the largest eigenvector of the centered kernel matrix. Clearly we are projecting $\boldsymbol{x}^{(i)}$, associated to θ_i, to a slightly scaled version of $-\sin(\theta_i)$

Remark ((Classic) PCA is linear): (Classic) PCA can only find linear relations in the data. Assume $\boldsymbol{x}^{(i)}$ is a sample, and $W \in \mathbb{R}^{k\times d}$ is the projection matrix containing the k largest eigenvectors of the sample covariance matrix $S = \frac{1}{n}X^TX$. Let $\boldsymbol{y}^{(i)}$ denote the projection of $\boldsymbol{x}^{(i)}$, i.e.:

$$\boldsymbol{y}^{(i)} = W\boldsymbol{x}^{(i)}$$

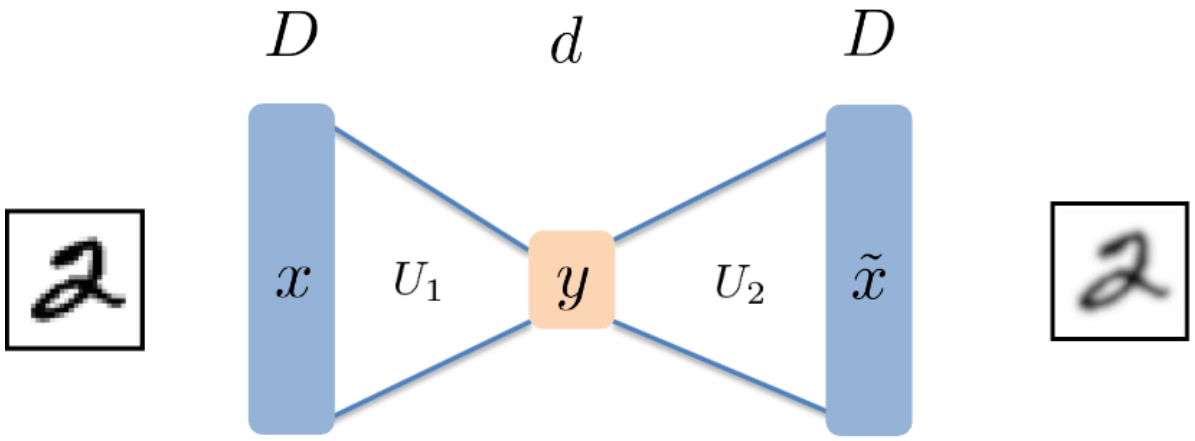
Notice we are only applying a linear transformation on $\boldsymbol{x}^{(i)}$, hence PCA is linear.

Kernel PCA is nonlinear.

2 Week 2

2.1 Encoder Decoder

Let $x \in \mathbb{R}^D$ be some sample. A common task is to do dimensionality reduction, to some representation $y \in \mathbb{R}^d$, and then do reconstruction to $\widetilde{x} \in \mathbb{R}^D$. This is an encoder-decoder framework.



- Setup: Let $X \in \mathbb{R}^{D\times n}$, where D is number of features and n is number of samples. Assume the data is centered, there are no redundant features, and $n \geq D$, i.e. $\text{rank}(X) = D$. Denote by $Y \in \mathbb{R}^{d\times n}$ an encoding of X, and denote by $\widetilde{X} \in \mathbb{R}^{D\times n}$ a reconstruction of X using Y.
- Objective: We wish, using linear operators, to find $\widetilde{X}$ s.t. :

  $$\min_{\widetilde{X}} \|X - \widetilde{X}\|_F^2$$

  Since both the encoding and decoding is done via linear operators, we can write:

  $$\min_{\widetilde{X}} \|X - \widetilde{X}\|_F^2 \text{ s.t. } \widetilde{X} = U_2 Y = U_2 U_1 X \quad U_1 \in \mathbb{R}^{d\times D}, U_2 \in \mathbb{R}^{D\times d}$$

- Solution: PCA. Let $\widehat{P} = \frac{1}{n} XX^T$ be the covariance matrix, and let U be the first d largest eigenvectors of $\widehat{P}$. Then $U_1 = U^T, U_2 = U$.
- Geometric PCA is not-unique: Notice that for every orthogonal matrix $R \in \mathbb{R}^{d\times d}$, we have that $U_1 = (UR)^T, U_2 = (UR)$ is also a minimizer. Denote $\widetilde{U} = UR$. For example, if our data really only has entries in the first two entries, then we can encode it to $X - Y$ plane. However every rotation following this projection would also serve, provided the decoder also includes the inverse rotation.

  I.e. the encoding Y isn't unique. However, statistical PCA is unique - how can this be? The answer is that in statistical PCA, we add a requirement that there is no correlation between the features (i.e. they are orthogonal), i.e. every row (feature) of Y is uncorrelated with every other row. If $Y = U^T X \in \mathbb{R}^{d\times n}$, then we have that $E[YY^T] = \Lambda = \text{diag}(\lambda_1, \ldots, \lambda_d)$, i.e. the encodings are not correlated. However, for $\widetilde{Y} = \widetilde{U}^T X = (UR)^T X$, we have that:

  $$E[\widetilde{Y}\widetilde{Y}^T] = R^T \Lambda R$$

  i.e. there can be non-zero off-diagonal elements, and hence the features can be correlated, unlike (statistical) PCA.

2.2 Relating Statistical and Geometrical PCA

2.3 SVD, PCA

Thm.**2.3.1 SVD:** Let $X \in \mathbb{R}^{m\times n}$, with m samples and n features per sample, $q = \min\{m, n\}$ and $r = \text{rank}(M)$. SVD decomposes X to

$$X = U\Sigma V^T$$

with $U \in \mathbb{R}^{m\times m}, \Sigma \in \mathbb{R}^{m\times n}, V \in \mathbb{R}^{n\times n}$. Σ is a rectangular diagonal matrix, with non-increasing entries.

Thm. **2.3.2 PCA and SVD:** Recall that in PCA (assuming X is centered), we compute:

$$\widehat{P} = \frac{1}{n-1} X^T X = \frac{1}{n-1} V\Sigma^T U^T U\Sigma V^T = V\frac{\Sigma^2}{n-1} V^T$$

I.e. the right singular vectors in V (from the SVD) are the principal directions (eigenvectors) from PCA, and the PCA eigenvalues λ_i are related to σ_i via $\lambda_i = \frac{\sigma_i^2}{n-1}$.

Since the columns of V are the principal directions, we can project onto them to get principal scores, i.e. an optimal lower dimension embedding. The rank k embedding $Y \in \mathbb{R}^{m\times k}$ is given by:

$$Y = XV_{n\times k} = U\Sigma V^T V_{n\times k} = U\Sigma_{m\times k} = U_{m\times k}\Sigma_{k\times k}$$

where:

$$\Sigma_{k\times k} = \text{diag}(\sigma_1, \ldots, \sigma_k) = \begin{bmatrix} \sigma_1 & 0 & \ldots & 0 \\ 0 & \sigma_2 & 0 & 0 \\ 0 & \ddots & \ddots & 0 \\ 0 & \cdots & \cdots & \sigma_k \end{bmatrix}$$

Note that $Y = U_{m\times k}\Sigma_{k\times k}$ are the principal scores. Therefore we can produce a reconstruction $\widetilde{X} \in \mathbb{R}^{m\times n}$ by using the principal scores to take a weighted sum of the principal directions:

$$\widetilde{X} = Y(V^T)_{k\times n} = (U_{m\times k}\Sigma_{k\times k})(V^T)_{k\times n}$$

This can be written as:

$$\widetilde{X} = Y(V^T)_{k\times n} = U_{m\times k}\Sigma_{k\times k}(V^T)_{k\times n} = \tag{1}$$

$$= U_{m\times k}\begin{bmatrix} - & \sigma_1 v_1^T & - \\ - & \vdots & - \\ - & \sigma_k v_k^T & - \end{bmatrix} = \sum_{i=1}^{k} \sigma_i u_i v_i^T \tag{2}$$

Alternatively, we can think of the reconstruction $\widetilde{X}$ of rank k as taking the first k largest singular values, and zeroing the rest. Let $\Sigma^{(k)} \in \mathbb{R}^{m\times n}$ be defined by:

$$\Sigma^{(k)} = \begin{cases} \sigma_i, & i \leq k \\ 0, & i \geq k \end{cases}$$

Then:

$$\widetilde{X} = U\Sigma^{(k)}V^T$$

This follows from the RHS of Eq. (1).

\# Thm. **2.3.3 Reconstruction Error:**

$$\epsilon(k) = \|X - \widetilde{X}\|_F^2 = \text{Tr}\left((X - \widetilde{X})^T(X - \widetilde{X})\right) = \sum_{i=k+1}^{n} \sigma_i^2$$

Remark (Model Selection - How to choose embedding dimension k): knee point

2.4 Stochastic Neighbor Embedding

$*$ Def. **2.4.1 (Shannon) Entropy:** Let X be a discrete random variable which takes values in the set $\mathcal{X}$, with PMF $p : \mathcal{X} \to [0, 1]$ such that $p(x) := \mathbb{P}[X = x]$. Then:

$$H(X) = -\sum_{x\in\mathcal{X}} p(x)\log p(x)$$

$*$ Def. **2.4.2 KL Divergence:** Let p, q be PMF over $\mathcal{X}$. Then:

$$D(p||q) := \sum_{xin\mathcal{X}} p(x)\log\frac{p(x)}{q(x)}$$

$*$ Def. **2.4.3 Jensen-Shannon Divergence:** Symmetric version of the KL-Divergence:

$$D_{JS}(p; q) := \frac{1}{2}\big(D(p||\rho) + D(q||\rho)\big)$$

where:

$$\rho = \frac{1}{2}(p + q)$$

Let $X = \{x_1, \ldots, x_n\} \in \mathbb{R}^D$ and a distance measure $\|x_i - x_k\|$.

∗ Def. **2.4.4 "Conditional Probability":** Define "conditional probability":

$$p_{j|i} := \frac{\exp\left(-\|x_i - x_j\|^2/2\sigma^2\right)}{\sum_{k\neq i}\exp(-\|x_i - x_k\|^2/2\sigma^2)}$$

This is the Heat-Kernel Affinity. We define $p_{i|i} := 0$ (convention). Let E be the symmetric matrix defined by $E_{ij} = e^{-\|x_i - x_j\|^2}$. This gives a non-symmetric matrix $P_{ji} = p_{j|i}$ which is E_{ji} divided by entire i-th row of$E$. Therefore the columns sum to1, but the rows don't.

- Probability Measure This is a probability measure, since $\sum_j p_{j|i} = 1$
- Non-Symmetric $p_{j|i} \neq p_{i|j}$. Imagine three points $a < b < c$ on a line, with b in the middle. Then $p_{b|a} > p_{a|b}$, since b has other close neighbors (more popular) than a.
- Distance $p_{i|j}$ transforms the euclidean distance $\|x_i - x_j\|$ into a probability measure. Low distance = High probability (of being neighbors). This allows us to turn the problem of finding similar embeddings into finding similar probability distributions P and Q.

Thm. **2.4.5 SNE Embedding:** We wish to find low dimensional representation $Y = \{y_1, \dots y_n\} \in \mathbb{R}^d$ with $d << D$, where we measure affinity $q_{i|j}$ by:

$$q_{j|i} := \frac{\exp\left(-\|y_i - y_j\|^2\right)}{\sum_{k\neq i}\exp(-\|y_i - y_k\|^2)}$$

(some omit the σ in q), and:

$$P_i := p_{\cdot|i} = (p_{1|i}, \dots, p_{n|i})$$
$$Q_i := q_{\cdot|i} = (q_{1|i}, \dots, q_{n|i})$$

Thus P_i is the normalized similarity vector for x_i, and defines a conditional probability distribution.

We wish that y_i is similar to x_i.

∗ Def. **2.4.6 Perplexity:** σ_i is chosen using perplexity, with:

$$\text{Perp}(P_i) := 2^{H(P_i)} \quad \text{with } H(P_i) := -\sum_{j\neq i} p_{j|i}\log_2 p_{j|i}$$

$\text{Perp}(P_i)$ is a smooth measure of the effective number of neighbors of x_i.

∗ Def. **2.4.7 SNE Cost Function:** Define the Cost Function, that minimizing it aims to maintain neighborhood similarity:

$$C(Y) = C(\{y_i\}) = \sum_i KL(P_i|Q_i) = \sum_i\sum_{j\neq i} p_{j|i}\log\frac{p_{j|i}}{q_{j|i}}$$

We look for:

$$\min_Y C(Y)$$

- Goal: Goal of SNE is to minimize mismatches between $p_{j|i}$ and $q_{j|i}$.
- Non-Symmetry: For **close** points ($p_{j|i}$ high) there is **high** cost for mapping them **far** (i.e. small $q_{j|i}$). However, for **far** points ($p_{j|i}$ low), there is **small** cost for mapping them close (i.e. assigning high $q_{j|i}$). This leads to asymmetry, where SNE preserves **local** structure, but not **global** structure.

Thm. **2.4.8 Cost Function Gradient:** in Tirgul.

$$\frac{\partial C}{\partial y_i} = 2\sum_j \left(p_{j|i} - q_{j|i} + p_{i|j} - q_{i|j}\right)(y_i - y_j)$$

Thm. **2.4.9 SNE Gradient Descent:**

- We initialize y by sampling map points randomly from an isotropic Gaussian with small variance that is centered around the origin.
- A relatively large momentum is added to gradient to avoid poor local minima. Alternatively, the current gradient is added to an exponentially decaying sum of previous gradients.

  $$\mathcal{Y}^{(t)} = \mathcal{Y}^{(t-1)} + \eta\frac{\partial C}{\partial y} + \alpha(t)(\mathcal{Y}^{(t-1)} + \mathcal{Y}^{(t-2)})$$

  where $\alpha(t)$ is the momentum at iteration t and η is the learning rate.

Remark (SNE Notes):

- t-SNE is good for visualization, not necessarily as input to a downstream . It does not preserve global structure.
- t-SNE is nonlinear.
- t-SNE does not preserve distances. Close points will behave similarly, however, more global structure is not preserved. E.g. let x be our base points, and x_1, x_2 two points with $d(x_1 - x) < d(x_2 - x)$. Then it's not necessarily true that $d(y_1 - y) < d(y_2 - y)$, where y, y_1, y_2 are the t-SNE embeddings. If that's a goal, then use MDS. Intuitively, PCA preserves large pairwise distances, but not small pairwise distances. t-SNE does not aim to preserve any (global) distances.
- T-SNE is designed to preserve local similarities between data points, which means it excels at revealing clusters and local patterns in high-dimensional data. However, this focus on local structure comes at the expense of accurately representing global relationships.
- Probability Distribution: T-SNE uses a heavy-tailed Student t-distribution in the low-dimensional space, which tends to push apart points that are not similar

2.5 t-SNE

Crowding Problem Mapping far x_i points to nearby (crowded) y_i points in the embedding dim is both problematic, and makes gradient descent unstable. Specifically, think of x_i, x_j being two outliers who are near each other. Then $p_{j|i}$ (and $p_{i|j}$) will be very small, and hence the cost function won't take them into consideration.

tSNE solves this by using:

1. Symmetry: uses a symmetrized version of the SNE cost function, with simpler gradients.
2. Student-t distribution: Uses Student-t distribution instead of a Gaussian to compute the *similarity* in the low-dimensional space. A heavy-tailed distribution goes to zero slower than an exponential. This will give outliers high values.

Solving Asymmetry We can do **symmetric SNE**, by taking the denominator with respect to all pairwise possibilities. We write:

$$p_{ij} := \frac{\exp(-\|x_i - x_j\|^2/2\sigma^2)}{\sum_k \sum_{\ell \neq k} \exp(-\|x_k - x_\ell\|^2/2\sigma^2)}$$

Recall that previously, $P_{ji} = p_{j|i}$ was given as E_{ji} divided by the i-th row of E. Now instead, P_{ji} is defined to be E_{ji} divided by the *entire* matrix E (which has a zero main diagonal), and therefore $P_{ji} = P_{ij}$. In the *low-dimensional* map $q_{i|j}$ as the denominator (Z_i) by pairwise similarities:

$$q_{ij} := \frac{e^{-\|y_i - y_j\|^2}}{\sum_k \sum_{\ell \neq k} e^{-\|y_k - y_\ell\|^2}}$$

However, this still does not solve the crowding problem. p_{ij} can still be arbitrarily small, which is problematic. Therefore, tSNE uses a different conditional probability:

$$p_{ij} := \frac{p_{i|j} + p_{j|i}}{2n}$$

This symmetric definition ensures that:

$$\sum_j p_{ij} > \frac{1}{2n}$$

i.e. p_{ij} is bounded below, regardless of how much of an outliers x_i and x_j are, and therefore y_i and y_j will have to be somewhat reasonable.

Heavy-Tail Additionally, to solve the crowding problem, we give more weight to outliers, by using for the mapping q_{ij} a heavy-tailed distribution instead of the Gaussian. Specifically we use the Student-t distribution:

$$q_{ij} := \frac{(1 + \|y_i - y_j\|^2)^{-1}}{\sum_k \sum_{\ell \neq k} (1 + \|y_k - y_j\|^2)^{-1}}$$

The heavy tails "allows" the map to place points further apart.