Dimensionality reduction: PCA and tSNE

David Puelz

Reference: Introduction to Statistical Learning Chapter 10.1-10.2

Outline

1. Introduction to PCA
2. Brief reminder of some linear algebra notation
3. PCA as a projection
4. Generalizing to more than one principal component
5. PCA as an optimization problem
6. tSNE for visualization

Introduction to PCA

The goal of PCA is to find low-dimensional summaries of high-dimensional data sets.

This is useful for compression, for denoising, for plotting, and for making sense of data sets that initially seem too complicated to understand.

It differs from clustering:

- Clustering assumes that each data point is a member of one, and only one, cluster. (Clusters are mutually exclusive.)
- PCA assumes that each data point is like a combination of multiple basic "ingredients." (Ingredients are not mutually exclusive.)

You can fiddle with ingredients continuously, but cluster membership only discretely.

Think about recipes

Nestle Toll House Chocolate-chip cookies:

- 280 grams flour
- 150 grams white sugar
- 165 grams brown sugar
- 225 grams butter
- 2 eggs
- 0 grams water
- ...

Think about recipes

Mary Berry's Victoria sponge cake:

- 225 grams flour
- 225 grams white sugar
- 0 grams brown sugar
- 225 grams butter
- 4 eggs
- 0 grams water
- ...

Think about recipes

seriouseats.com old fashioned flaky pie dough:

- 225 grams flour
- 15 grams white sugar
- 0 grams brown sugar
- 225 grams butter
- 0 eggs
- 115 grams water
- ...

Think about recipes

Each baked good is constructed by following a recipe: a combination of the same basic ingredients.

In our analogy of baking with data:

- Each data point x_i is like a baked good.
- In PCA, the **principal components** are like the ingredients.

The amounts of each ingredient differ between baked goods:

- E.g. 225g sugar for sponge cake, versus 15g for pie dough.
- In PCA, the **scores** are like the amounts of each ingredient in a given baked good.

Think about recipes

Each baked good is constructed by following a recipe: a combination of the same basic ingredients.

In our analogy of baking with data:

- Each data point x_i is like a baked good.
- In PCA, the **principal components** are like the ingredients.

The amounts of each ingredient differ between baked goods:

- E.g. 225g sugar for sponge cake, versus 15g for pie dough.
- In PCA, the **scores** are like the amounts of each ingredient in a given baked good.

In PCA, our goal is to reverse-engineer both the **ingredients** and the **amounts/recipes** from an observed set of "baked goods" (i.e. original data points).

The result of PCA

Let's see a before and after to start building some intuition for what PCA does.

Before: raw survey data on a bunch of TV shows

	Entertaining	Engaged	Original	Confusing	Funny
30 Rock	4.2	3.7	4.0	2.2	3.7
Next Top Model	4.2	3.8	3.8	2.0	3.5
American Chopper	4.2	3.6	3.9	2.1	3.5
Bones	4.3	4.1	3.8	1.9	3.4
Close to Home	4.1	3.8	3.8	1.9	2.9
Cold Case	4.2	3.9	4.0	1.9	3.0

plus 16 more columns.

The result of PCA

After: survey data on a bunch of TV shows run through PCA

	PC1	PC2
30 Rock	-2.64	0.12
Next Top Model	-0.78	-0.19
American Chopper	-2.31	0.68
Bones	3.24	2.22
Close to Home	-0.63	2.86
Cold Case	1.62	2.88

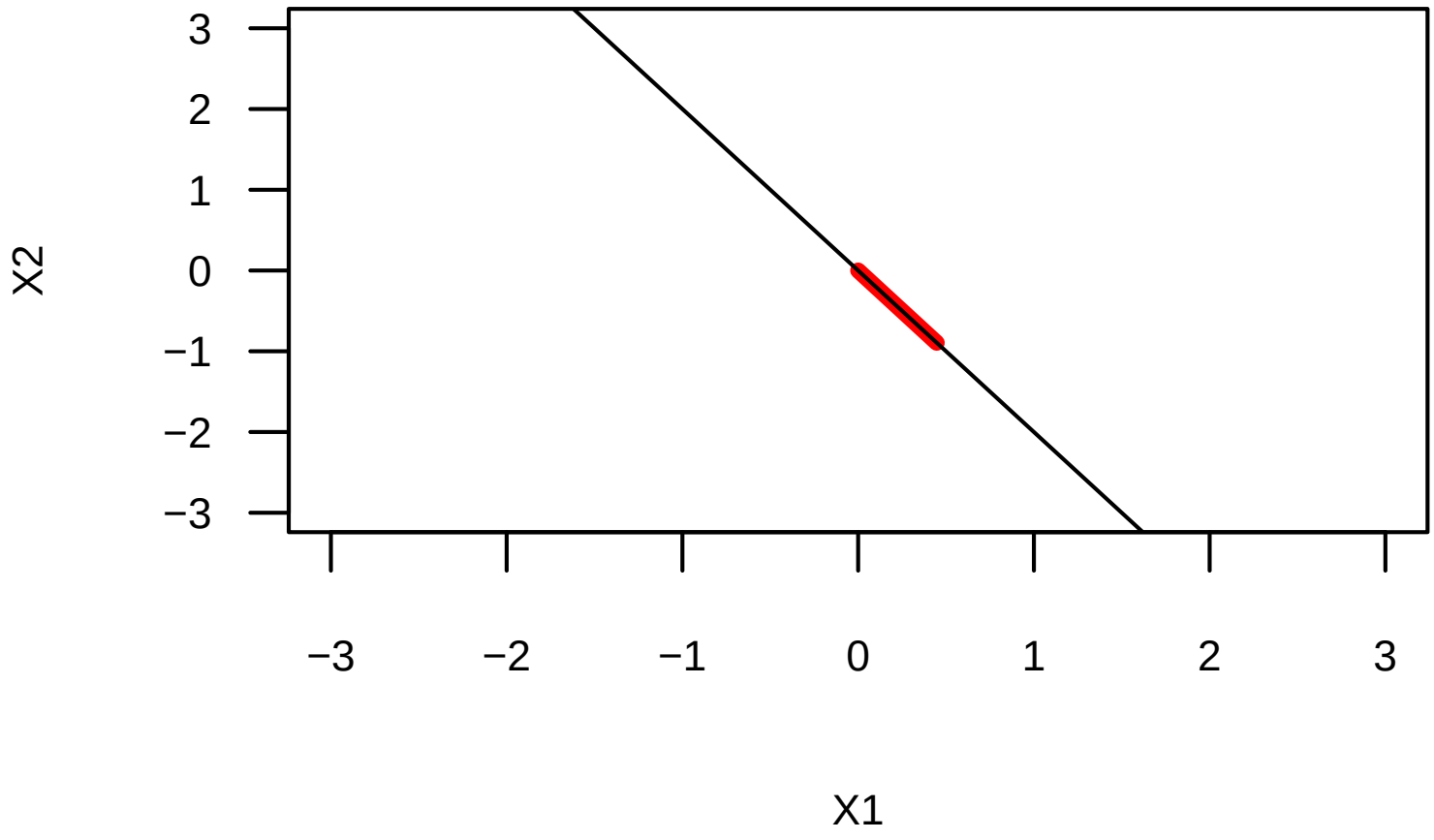
From 20 raw variables to 2 summaries. PC1 and PC2 are like "ingredients." The numbers are "amounts" of each "ingredient."

The trick to PCA is interpreting the summaries!

Some linear algebra reminders

Alas, PCA is less tasty than baking, and uses more linear algebra. Say that $v \in \mathbb{R}^P$ is some vector. This defines a *subspace* of $\mathbb{R}^P$:

$$\mathcal{V} = \{z : z = cv, c \in \mathbb{R}\}$$



Some linear algebra reminders

Let's say that $x_i = (x_{i1}, \ldots, x_{iP})$ represents our data vector for the ith case.

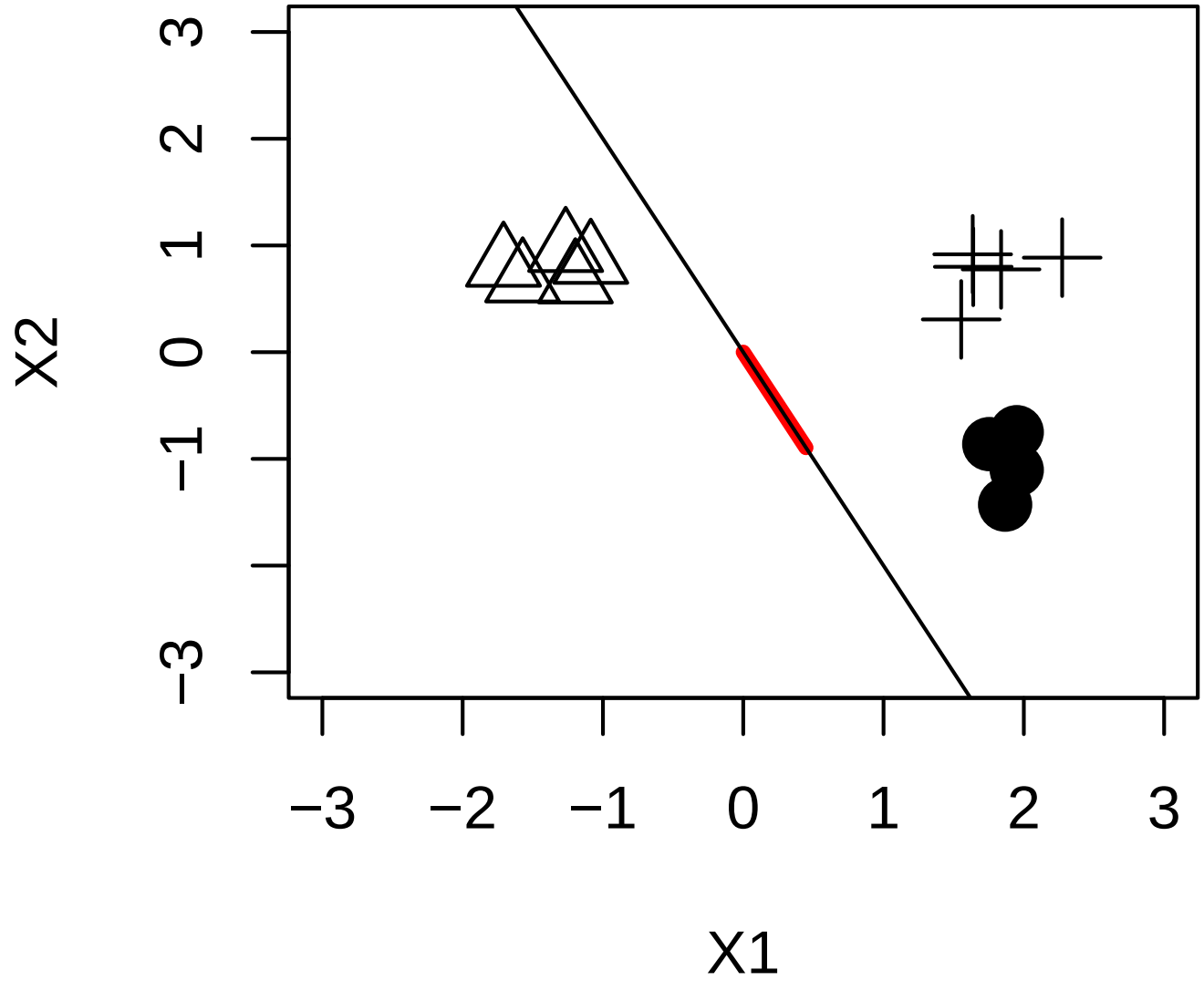
Now we summarize x_i by *projecting* it onto the subspace $\mathcal{V}$. The scalar location of this projected point in this subspace $\mathcal{V}$ is

$$c_i = x_i \cdot v = x_i^T v$$

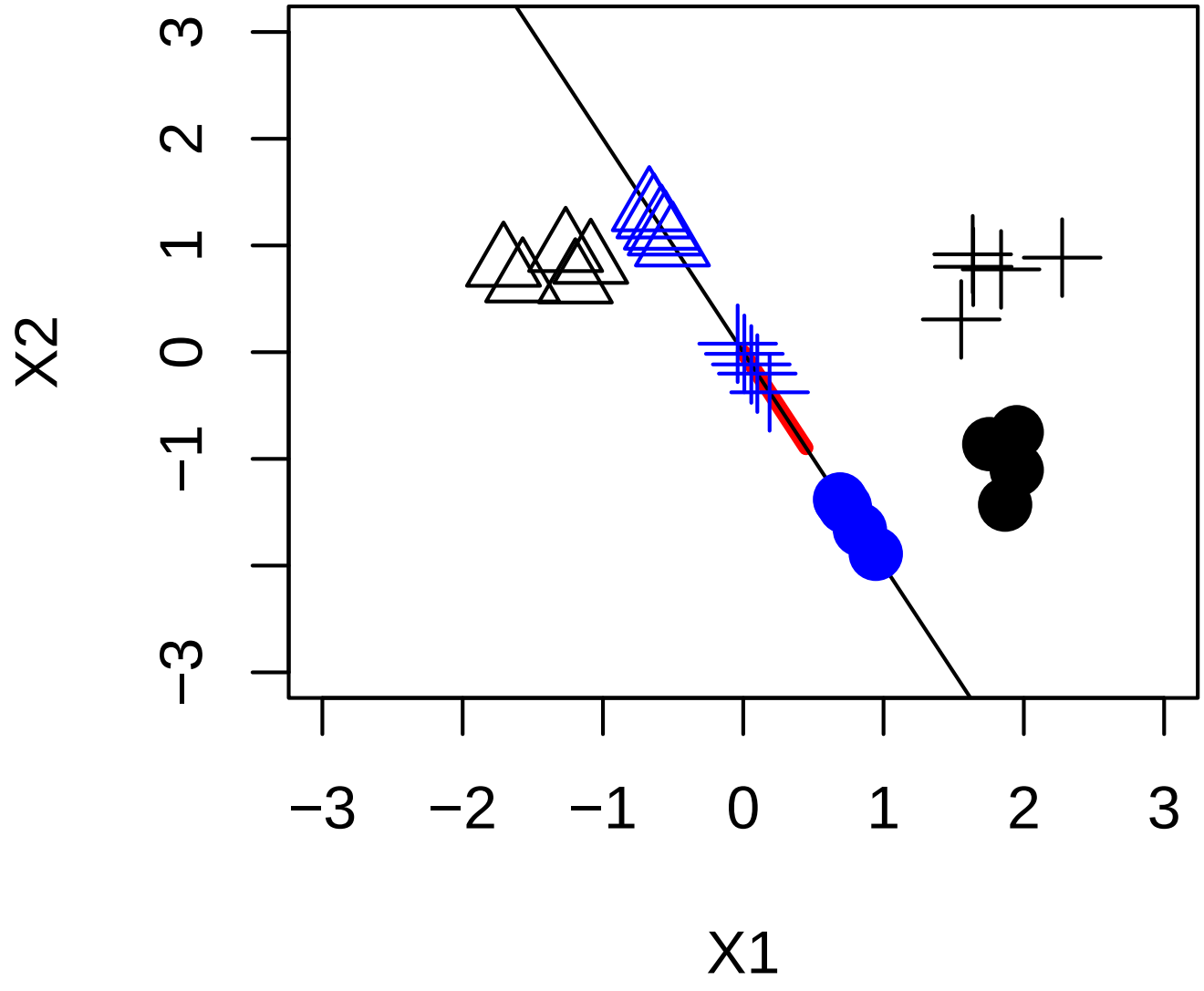
This is a one-number summary of our original point x_i. A different choice of v gives a different summary.

Let's see a picture, then consider why this is true.

The original points



With the projected points



Projection and dot product

The projection of x_i onto $\mathcal{V}$ is **the vector that lies in $\mathcal{V}$ and is closest to x_i.** We can find this projection by minimizing the squared distance between x_i and the projected point $z = c_i v$:

$$\min_{c_i \in \mathcal{R}} \|x_i - c_i v\|^2$$

Take the derivative with respect to c_i and set it to zero:

$$\frac{d}{dc_i} \|x_i - c_i v\|^2 = 0$$

Solving for c_i, we find:

$$c_i = x_i \cdot v = x_i^T v$$

This scalar location represents the position of the projected point in the subspace $\mathcal{V}$.

Key ideas

Key idea 1: projection = summary

- Each point's location along the subspace is a **one-number linear summary** of a P-dimensional data vector:

$$c_i = x_i \cdot v = x_i^T v$$

- The goal of principal components analysis (PCA) is to find the "best" projection, i.e. the best linear summary of the original data points.
- But what does "best" actually mean?

Key ideas

Key idea 2: the "best summary" is the one that preserves as much of the **variance** in the original data points as possible.

- Intuition: we're already trying to crowd these P-dimensional points into 1-D. We should give them "room to breathe" by crowding them on top of each other as little as possible *within* that 1-D space!
- More variance in the scores (c_i) means more "spread out" summaries in 1-D.
- And the more they're spread out, the more we have preserved differences between the projected points that were present in the original points. **See pca_intro.R**.

Finding the first PC vector

Given data points $x_1, \ldots, x_N$, with each $x_i \in \mathbb{R}^P$, and a candidate vector v_1, the variance of the projected points is

$$\text{variance} = \frac{1}{n} \sum_{i=1}^{n} [c_i - \bar{c}]^2$$

where $c_i x_i \cdot v_1$.

So we solve:

$$\underset{v_1 \in \mathbb{R},\, \|v\|_2 = 1}{\text{maximize}} \quad \sum_{i=1}^{n} \left[x_i \cdot v_1 - \left(\frac{1}{n} \sum_{i=1}^{N} x_i \cdot v_1 \right) \right]^2$$

Note: we constrain v_1 to have length 1; otherwise we could blow up the variance of the projected points as large as we wanted to.

Finding the first PC vector

The solution v_1 to this optimization problem:

- is a unit-length vector of dimension P.
- is called the first principal component (synonyms: loading, rotation.)
- represents, via the set of its scalar multiples $\{cv_1 : c \in \mathcal{R}\}$, the one-dimensional subspace capturing as much of the information in the original data matrix as possible.

The projected points $c_i = x_i \cdot v$ are called the *scores* on the first principal component.

Multiple Principal Components

Now let $v_1, v_2, \ldots, v_K$ be a set of K unit-length, orthogonal vectors, each P-dimensional. Consider the set of all linear combinations of these vectors:

$$\mathcal{V} \equiv \text{span}\{\mathbf{v}_1, \ldots, \mathbf{v}_K\} = \left\{ z : z = \sum_{k=1}^{K} c_k \mathbf{v}_k \mid c_1, \ldots, c_K \in \mathbb{R} \right\}$$

Here the v_k's are our "ingredients" and the c_k's are our "amounts."

Each z living in this space is P-dimensional vector whose "real" dimension is only K—potentially much smaller!

Projecting Data onto the Subspace

To summarize our data matrix X, we project each data point x_i onto the subspace $\mathcal{V}$, and the coordinates of the projected points in this subspace are given by:

$$\hat{x}_i = \text{proj}_{\text{span}\{\mathbf{v}_1,\ldots,\mathbf{v}_K\}}(x_i) = \sum_{k=1}^{K} \mathbf{v}_k(\mathbf{v}_k^T x_i)$$

Or, more concisely:

$$\hat{x}_i = \text{proj}_{\text{span}\{\mathbf{v}_1,\ldots,\mathbf{v}_K\}}(x_i) = \mathbf{V}\mathbf{V}^T x_i$$

The "amounts" of each vector v_k are given by the projection of x_i onto v_K (via a dot product; see earlier discussion.)

The Optimization Problem

We want to find the orthonormal basis $V_K = \begin{bmatrix} | & | & & | \\ v_1 & v_2 & \dots & v_K \\ | & | & & | \end{bmatrix}$ that maximizes the projected variance.

Variance of Projected Data. Let X be our data matrix with N samples and P features, and let V_K be the matrix with K orthonormal vectors as its columns. We want to project our data matrix X onto the subspace spanned by the vectors in V_K. The projected data matrix is given by:

$$A = XV_K$$

Where $A \in \mathbb{R}^{N \times K}$ is the "summary" data matrix, i.e. where we've summarized each $x_i \in \mathcal{R}^P$ via its projection "score" $a_i \in \mathcal{R}^K$.

Covariance Matrix of Projected Data

To find the variance of the projected data, we compute the covariance matrix of the projected data matrix A:

$$\Sigma_A = \frac{1}{N-1} A^T A = \frac{1}{N-1} (XV_K)^T (XV_K)$$

By substituting the expression for the covariance matrix of the original data $\Sigma = \frac{1}{N-1} X^T X$, we can simplify the covariance matrix of the projected data:

$$\Sigma_A = V_K^T \Sigma V_K$$

Trace of Covariance Matrix and Sum of Variances

Let's look at Σ_A more closely:

$$\Sigma_A = \begin{bmatrix} \sigma_1^2 & \text{Cov}(a_1, a_2) & \dots & \text{Cov}(a_1, a_K) \\ \text{Cov}(a_2, a_1) & \sigma_2^2 & \dots & \text{Cov}(a_2, a_K) \\ \vdots & \vdots & \ddots & \vdots \\ \text{Cov}(a_K, a_1) & \text{Cov}(a_K, a_2) & \dots & \sigma_K^2 \end{bmatrix}$$

Since we are projecting onto the orthonormal basis $\{\mathbf{v}_1, \dots, \mathbf{v}_K\}$, the projected components are uncorrelated. This means that the off-diagonal elements of Σ_A are zero:

$$\text{Cov}(a_i, a_j) = 0 \quad \forall i \neq j$$

Trace of Covariance Matrix and Sum of Variances

So we have an unambigious optimization problem: maximizing the trace of the covariance matrix Σ_A is equivalent to maximizing the sum of the variances of our "summary" variables.

$$\text{Tr}(\Sigma_A) = \sigma_1^2 + \sigma_2^2 + \cdots + \sigma_K^2$$

So define

$$\max_{V_K} \text{Tr}(V_K^T \Sigma V_K)$$

Subject to the orthonormality constraint:

$$V_K^T V_K = I_K$$

Lagrange Multipliers

To solve this constrained optimization problem, we introduce the Lagrange multipliers $\Lambda = \text{diag}(\lambda_1, \lambda_2, \ldots, \lambda_K)$, and define the Lagrangian function:

$$\mathcal{L}(V_K, \Lambda) = \text{Tr}(V_K^T \Sigma V_K) - \text{Tr}(\Lambda^T (V_K^T V_K - I_K))$$

To find the optimal V_K, we take the gradient of the Lagrangian with respect to V_K and set it to zero:

$$\nabla_{V_K} \mathcal{L}(V_K, \Lambda) = 2\Sigma V_K - 2V_K \Lambda = 0$$

This leads to the eigenvalue equation:

$$\Sigma V_K = V_K \Lambda$$

Solution via Eigenvalue Decomposition

This is how the PC vectors are actually found:

1. Perform the eigenvalue decomposition of the covariance matrix Σ to obtain the eigenvector matrix V and the eigenvalue matrix Λ.
2. Select the first K eigenvectors corresponding to the K largest eigenvalues to form the matrix V_K.

The matrix V_K represents the orthonormal basis that maximizes the variance of the projected data, and its columns are the principal component vectors.

Examples

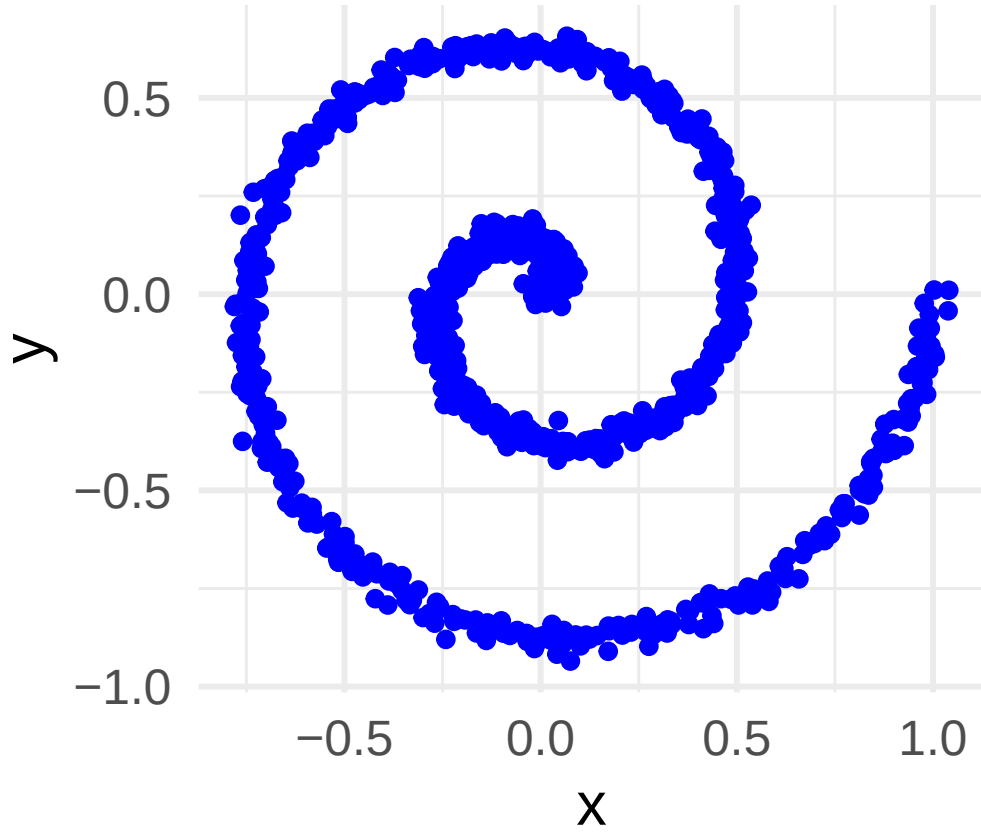
Let's see the examples in:

- `nbc.R`
- `congress109.R`
- `ercot_PCA.R`

These examples will help us learn to answer the million-dollar question in PCA: what do the summaries mean?

Introduction to tSNE

Recall that PCA provides a *linear* summary of a high-dimensional set of input features. But what if a linear summary just isn't a good summary?



tSNE

There are *a lot* of ways to do nonlinear dimensionality reduction.

Today we'll talk about tSNE (t-Distributed Stochastic Neighbor Embedding):

- the most popular non-linear dimensionality reduction technique
- particularly well suited for embedding high-dimensional data into a space of *two* (or maybe *three*) dimensions, which can then be visualized in a scatter plot.
- It's often (though not always) combined with other dimensionality reduction techniques like PCA.

It's my favorite trick for exploratory plots of high-D data.

Why tSNE?

- tSNE preserves the local structure of the data. Data points that are close together in the high-dimensional space remain close together in the low-dimensional space.
- The visualizations produced by t-SNE are easy to interpret. Each instance corresponds to a point in the scatterplot, and similar instances are modeled by nearby points.
- tSNE is generally faster and more computationally efficient than other nonlinear dimensionality reduction techniques. **Few hyperparameters, no neural network training loop.**

Working principle of tSNE

1. tSNE calculates two probabilities for each pair of points:

- the probability of similarity of the points in the original, high-D space
- the probability of similarity of the points in the corresponding low-D space (typically in the 2D plane).

2. It then tries to minimize the difference between these probabilities by finding an optimal location for each point in the low-D space.

Understanding "Probability of Similarity" in t-SNE

In t-SNE, the "probability of similarity" is a way of quantifying the closeness of a pair of points.

- **In High-D Space:** t-SNE models similarity by imagining a Gaussian distribution centered at this point and then measuring the density of nearby points.
- **In Low-D Space:** t-SNE models similarity using a t-distribution, which has heavier tails than a Gaussian distribution. (Why? No deep theory; the plots are just prettier.)
- **Minimizing the Difference**: t-SNE then tries to make the low-dimensional similarities as close as possible to the high-dimensional ones by minimizing the divergence between the two distributions.

t-SNE Hyperparameters

t-SNE has a few key hyperparameters that can significantly impact the results.

- Number of Components (`n_components`): the dimensionality of the embedded space (usually set to 2 for visualization purposes).
- `perplexity`: hard to interpret exactly, but can be thought of as a dial controlling the balance between maintaining the local and global structure of the data. Typical values: 5 to 50.
- `early_exaggeration`: controls how much space will be between natural clusters in the embedded space; default in `sklearn` is 12.0. High values tend to separate clusters better but can also "hallucinate" clusters.

Remember that every dataset is unique, so the "best" hyperparameters can differ. Experimentation helps!

Examples

Let's see the examples in `tSNE.ipynb`.