Dimensionality reduction

https://github.com/Maarten-vd-Sande/Teaching

Maarten van der Sande

Content

- Principal Component Analysis (PCA)
  - K-means clustering
- t-distributed Stochastic Neighbour Embedding (t-SNE)
  - Louvain clustering

Disclaimers

- I have never actually *really* used the methods

Disclaimers

- I have never actually *really* used the methods
- I'm going to make untrue simplifications

Disclaimers

- I have never actually *really* used the methods
- I'm going to make untrue simplifications
- Please stop me when something is unclear

Why dimensionality reduction?

Why dimensionality reduction?

- Visualization

Why dimensionality reduction?

- Visualization
- Computational cost (storage, memory, processors, etc.)

Why dimensionality reduction?

- Visualization
- Computational cost (storage, memory, processors, etc.)
- Reduce chance of over fitting

PCA

run:

```
python3 manual_PCA.py
```

K-means clustering I

How to analyze this data? e.g.

- We can do regression (fit a line)
- Cluster the data

K-means clustering I

How to analyze this data? e.g.

- We can do regression (fit a line)
- Cluster the data

Simple cluster algorithm: K-means!

K-means clustering II

Algorithm 1 K-means clustering

1: Initialize K random cluster centroids
2: **repeat**
    Group each point to the nearest centroid
    Move each centroid to the middle of all its points
3: **until** convergence

K-means clustering III

```
run:
python3 k-means.py
```

t-SNE I

- t-SNE fundamentally different from PCA.

t-SNE I

- t-SNE fundamentally different from PCA.
- t-SNE has the *"objective"* to project:
  - points that are close in high-dimensional space close in low-dimensional space
  - points that are far in high-dimensional space far in low-dimensional space

t-SNE I

- t-SNE fundamentally different from PCA.
- t-SNE has the *"objective"* to project:
  - points that are close in high-dimensional space close in low-dimensional space
  - points that are far in high-dimensional space far in low-dimensional space
- It does so by moving points in the low-dimensional space so that its *"objective"* is maximized.

t-SNE I

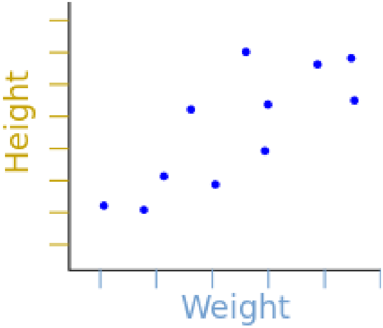
- t-SNE fundamentally different from PCA.
- t-SNE has the *"objective"* to project:
  - points that are close in high-dimensional space close in low-dimensional space
  - points that are far in high-dimensional space far in low-dimensional space
- It does so by moving points in the low-dimensional space so that its *"objective"* is maximized.
- **First we need a metric for closeness and farness!**

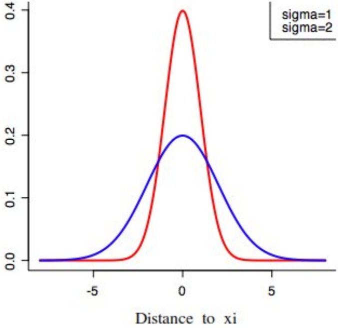
t-SNE II

Closeness (measured in chance) between point x_i and point x_j:

$$p_{ji} = \exp\left(\frac{-||x_i - x_j||^2}{\sigma_i^2}\right) \tag{1}$$

this is just a normal distribution! With $||x_i - x_j||$ the distance between x_i and x_j, and σ^2 as the variance.





t-SNE III

Closeness (measured in chance) between point x_i and point x_j:

$$p_{ji} = \exp\left(\frac{-||x_i - x_j||^2}{\sigma_i^2}\right) \tag{2}$$

this is just a normal distribution! With $||x_i - x_j||$ the distance between x_i and x_j, and σ^2 as the variance.

Now normalize (divide by the sum of all chances):

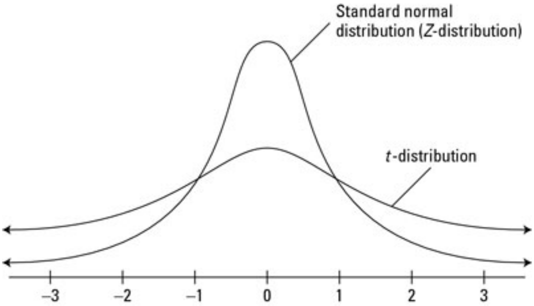
$$p_{j|i} = \frac{p_{ji}}{\sum_k p_{ki}} \tag{3}$$

t-SNE IV

Our measure for distance in low dimensional space:

$$q_{ji} = 1 + ||x_i - x_j||^{-1} \tag{4}$$

this is a t-distribution! With $||x_i - x_j||$ the distance between x_i and x_j. Luckily we do not have to do any difficult tricks with perplexity here.



t-SNE IV

Our measure for distance in low dimensional space:

$$q_{ji} = 1 + ||x_i - x_j||^{-1} \tag{5}$$

this is a t-distribution! With $||x_i - x_j||$ the distance between x_i and x_j. Luckily we do not have to do any difficult tricks with perplexity here.

Now normalize (divide by the sum of all chances):

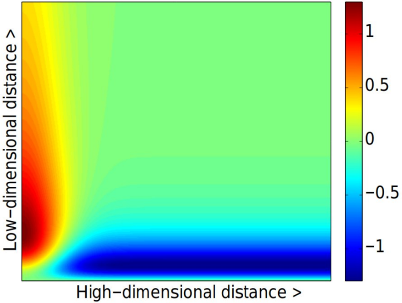
$$q_{j|i} = \frac{q_{ji}}{\Sigma_k q_{ki}} \tag{6}$$

t-SNE V

The *"objective"* is to maximize is the cost function:

$$C = \Sigma_i \Sigma_j p_{i|j} \log \frac{p_{i|j}}{q_{i|j}} \tag{7}$$

t-SNE VI



(c) Gradient of t-SNE.

t-SNE VII

Now all we have to do is set the variance σ_i for each dot. This is done with the perplexity (hyper)parameter:

t-SNE VII

Now all we have to do is set the variance σ_i for each dot. This is done with the perplexity (hyper)parameter:

$$Perp(P_i) = 2^{-\sum_j p_{j|i} \log_2 p_{j|i}} \tag{8}$$

t-SNE VII

Now all we have to do is set the variance σ_i for each dot. This is done with the perplexity (hyper)parameter:

$$Perp(P_i) = 2^{-\Sigma_j p_{j|i} \log_2 p_{j|i}} \tag{8}$$

All you need to remember:

A higher perplexity value results in a higher variance

t-SNE VII

Now all we have to do is set the variance σ_i for each dot. This is done with the perplexity (hyper)parameter:

$$Perp(P_i) = 2^{-\Sigma_j p_{j|i} \log_2 p_{j|i}} \tag{8}$$

All you need to remember:

A higher perplexity value results in a higher variance

```
python3 perplexity.py
```

t-SNE VIII

```
python3 tSNE.py
```

Louvain clustering I

Let's imagine a magical function (Q) exists that tells us how well our clusters fit on the $p_j|i$ matrix.

Algorithm 2 Louvain clustering

```
Each datapoint gets its own cluster. c={0, 1,..., n}
repeat
    for each datapoint i do
        for each cluster j do
            if Q(c[j]) > Q(c[i]) then
                c[i] = c[j]
            end if
        end for
    end for
until convergence
```

Louvain clustering II

Luckily for us, this magical function exists!

Louvain clustering II

Luckily for us, this magical function exists!

$$Q = \frac{1}{2m} \Sigma_{ij} (A_{ij} - \frac{k_i k_j}{2m}) \delta(c_i, c_j) \tag{9}$$

where A_{ij} represents edge weight between nodes i and j,
k_i the sum of the weights attached to node i,
$2m$ is the sum of all the weights in the graph,
c_i is the cluster i belongs to, and
$\delta()$ is one if $c_i == c_j$ and zero otherwise.

Louvain clustering II

Luckily for us, this magical function exists!

$$Q = \frac{1}{2m}\Sigma_{ij}(A_{ij} - \frac{k_i k_j}{2m})\delta(c_i, c_j) \tag{9}$$

where A_{ij} represents edge weight between nodes i and j,
k_i the sum of the weights attached to node i,
$2m$ is the sum of all the weights in the graph,
c_i is the cluster i belongs to, and
$\delta()$ is one if $c_i == c_j$ and zero otherwise.

```
python3 louvain.py
```