DIMENSIONALITY REDUCTION

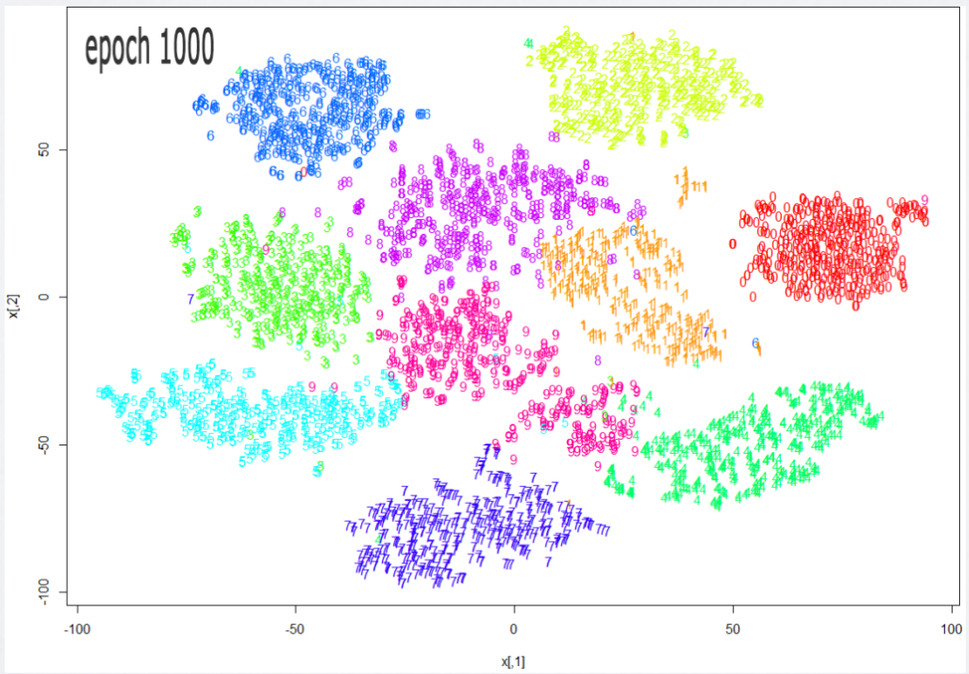
Low dimensionality embedding

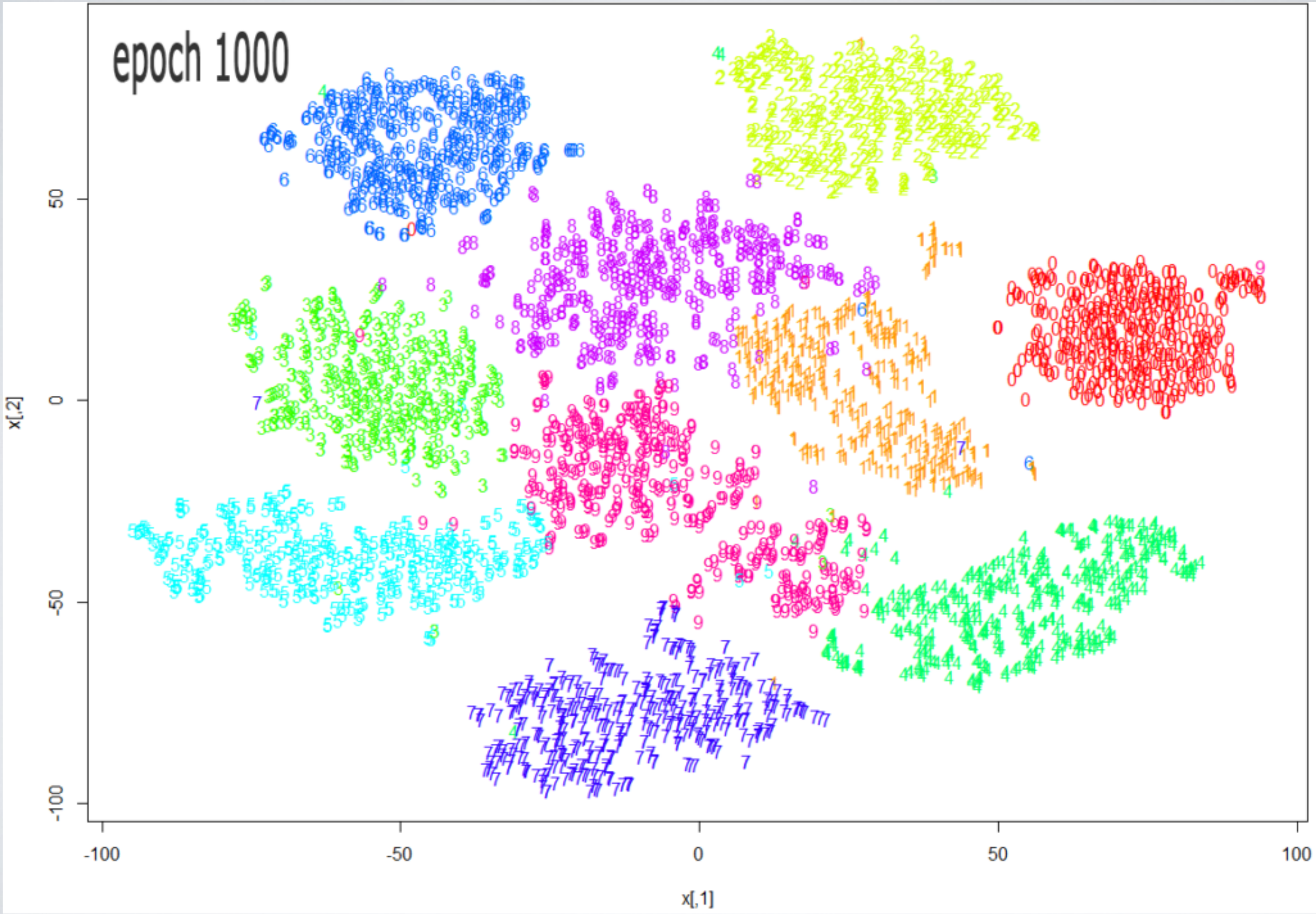
DIMENSIONALITY REDUCTION

- Multiple reasons to do it
  - 2D/3D visualisation
  - Curse of Dimensionality / Scalability
  - Correlated input features

VISUALIZATION

- Your data is perfectly fine, but you want to intuitively understand how it is organized
  - Are there groups of similar objects?
  - Are my clusters meaningful?
  - Is my classification/clustering on some types of elements and not others.




epoch 1000
x[,2]
50
0
-50
-100
-100
-50
0
50
100
x[,1]

CURSE OF DIMENSIONALITY

- Having hundreds/thousands of attributes is a problem for data analysis.
  - e.g.: medicine: blood analysis, genomics….
  - e.g.: cooking recipes: each column an ingredient…
- We want to reduce number of attributes while keeping most of the information
- Scalability

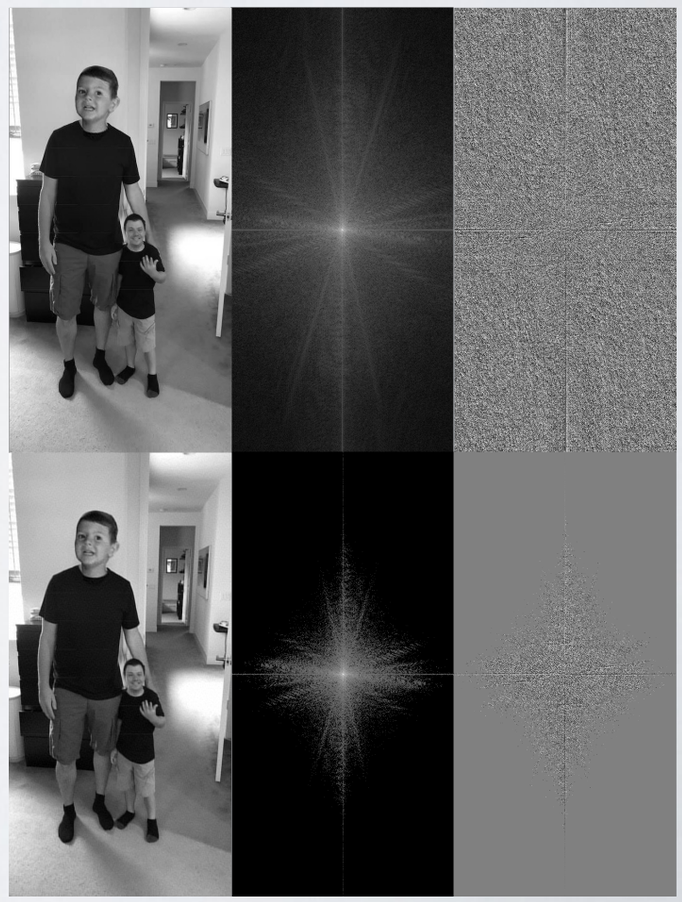
CORRELATION

- Assume that you have correlated features such as age, height and weight.
  - Linear regression will attribute the coefficients somewhat randomly between them
  - Decision tree will spend a lot of time choosing between them for no reason
- Dimensionality reduction can create a single variable to capture what is common
  - The rest can be lost or captured by another feature,

PCA

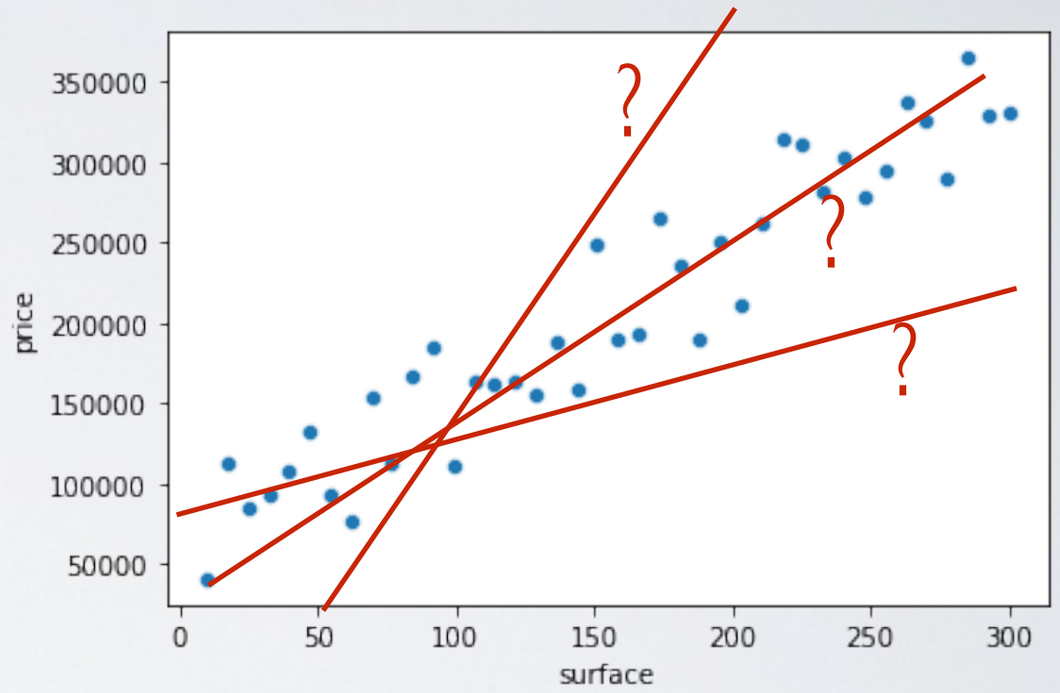
PCA

- PCA: Principal Component Analysis
- Defines new dimensions that are linear combinations of initial dimensions
  - Objective: concentrate the **variance** on some dimensions
    - So that we can keep only these ones.
    - Those we remove contain low variance, thus low information
- Similar principle than the Fourier transform technique for image compression

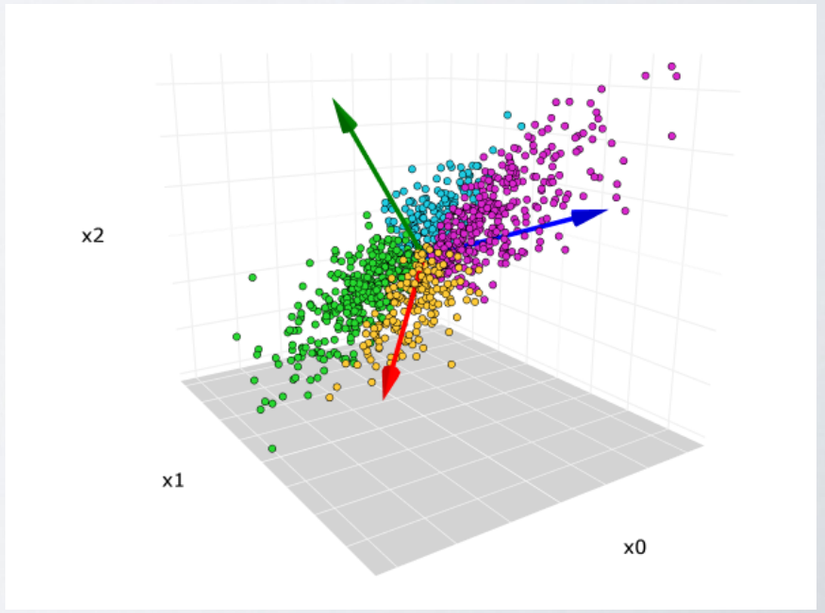


PCA

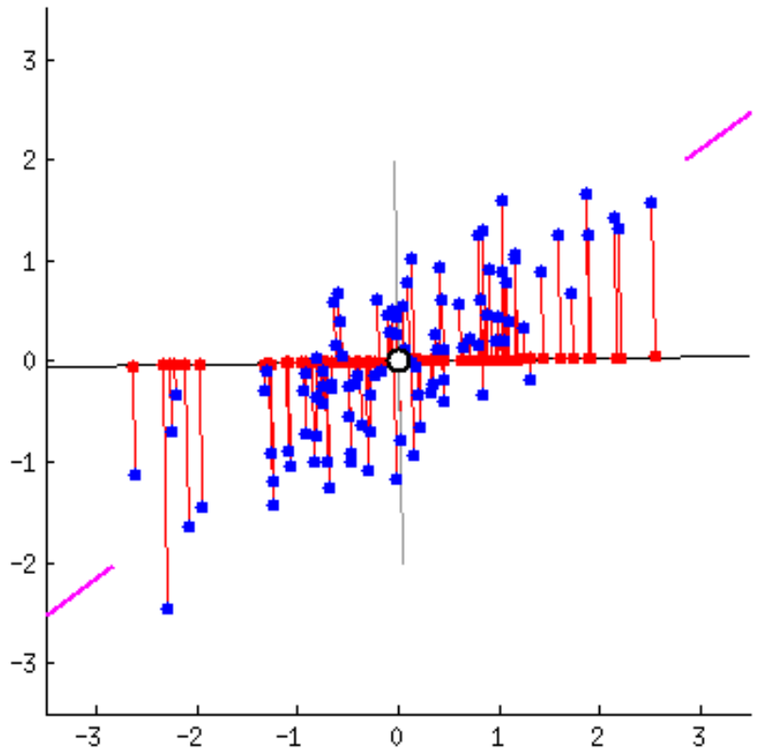
- Algorithm:
  - 1)Find an "axis", a unit vector defining a line in the space
    - That minimizes the variance=>the squared distance from all points to that line



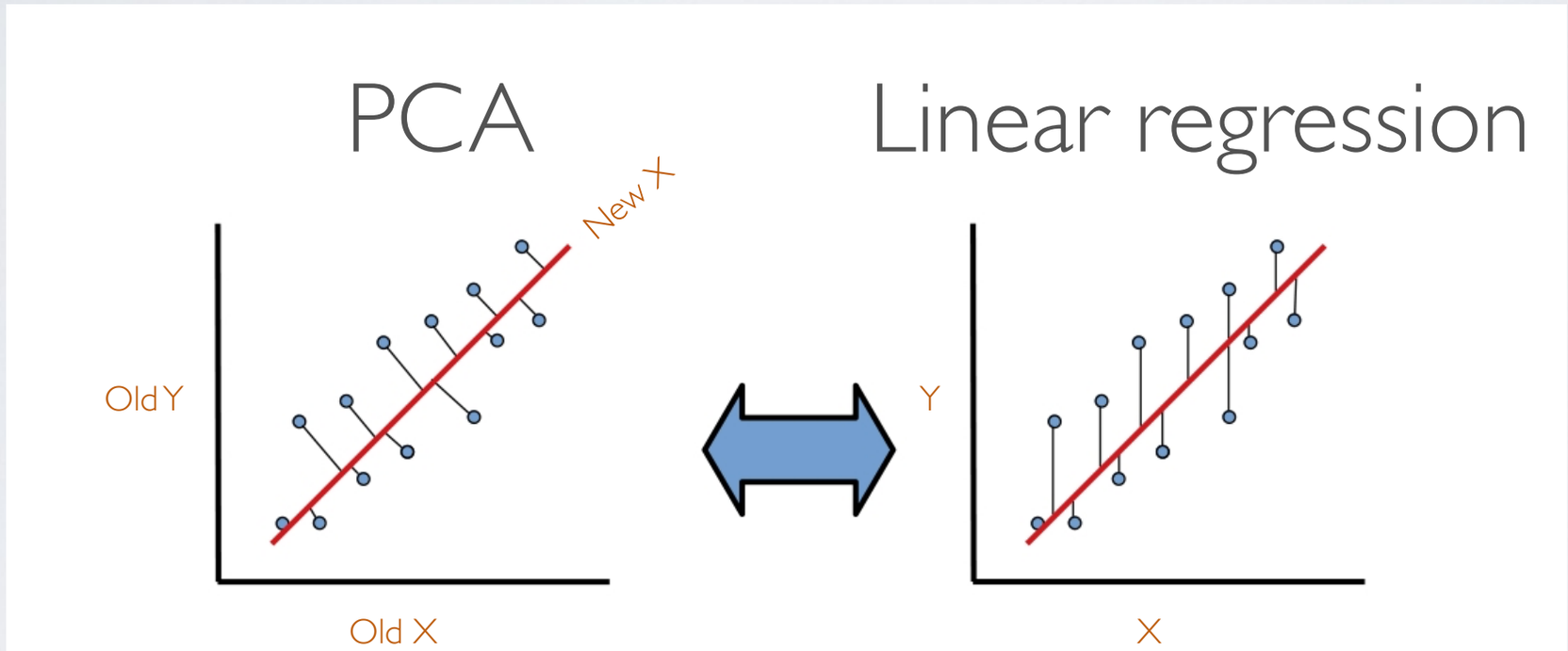
- 2)For d in (initial_d-1)
  - Find another axis, with two constraints:
    - Orthogonal to all previous axis
    - Among those, minimize the variance



- 3)At the end, keep the first k dimensions/axis
  - Some information is lost


3
2
1
0
−1
−2
−3
−3
−2
−1
0
1
2
3

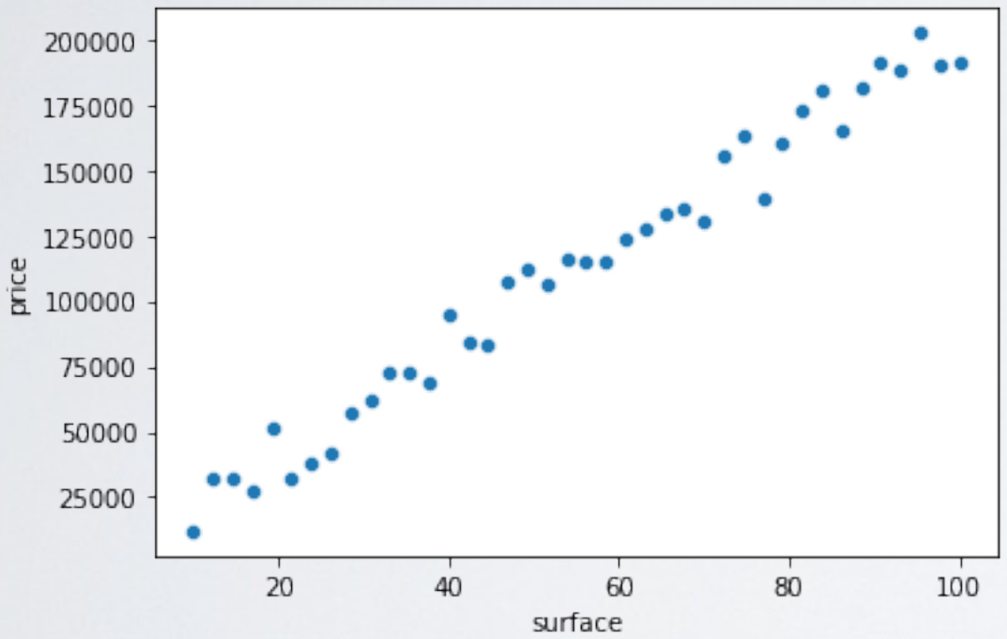
PCA VS LINEAR REGRESSION



Variance,
i.e., squared error to the mean
on a chosen axis

Squared error,
minimizes distance to a line,
on a particular axis (target)

EXAMPLE PCA 2D

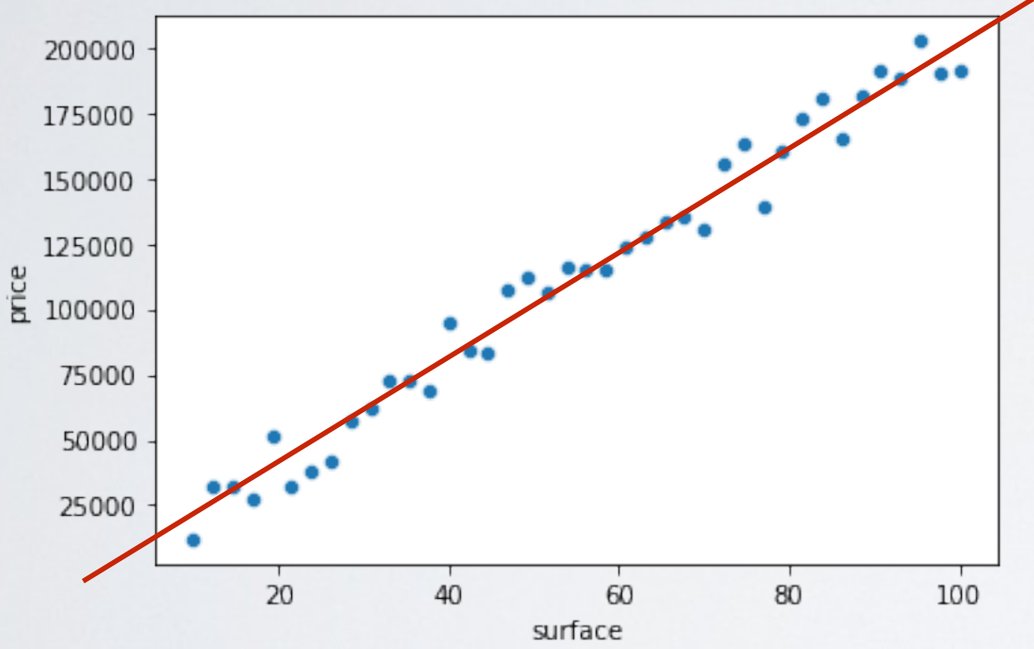


Old axis:

[0,1]

[1,0]

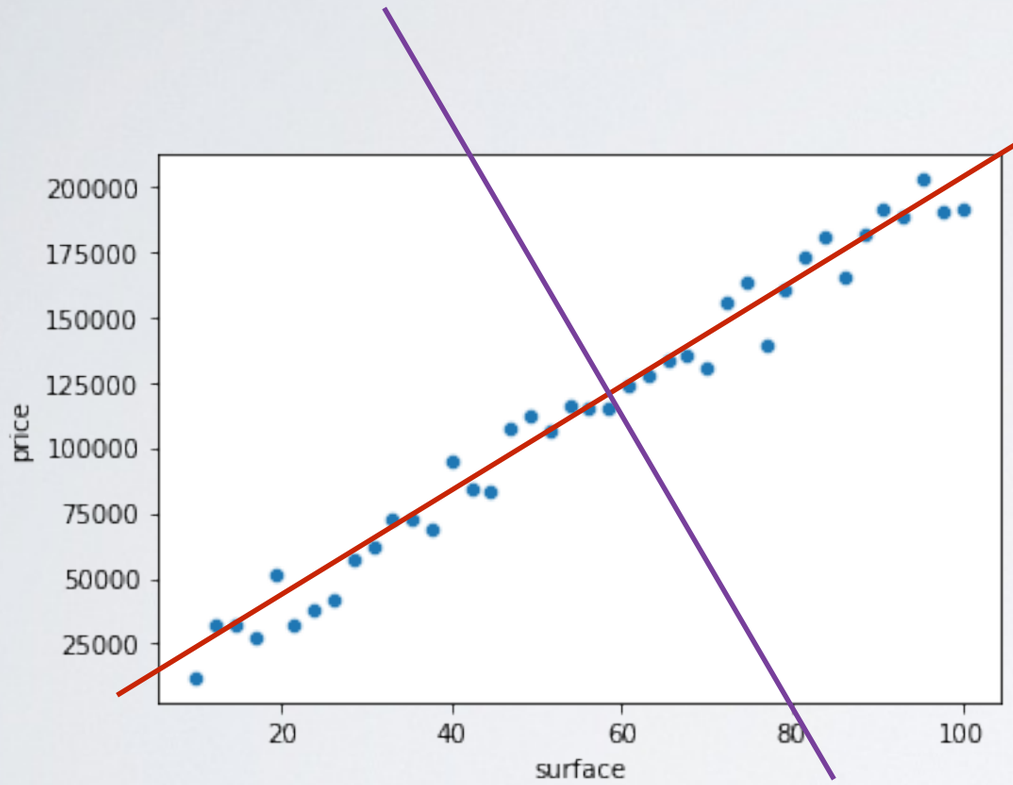
EXAMPLE PCA 2D



New axis 1
[50,100 000]
=>[1,20 000]

Old axis:
[0,1]
[1,0]

EXAMPLE PCA 2D



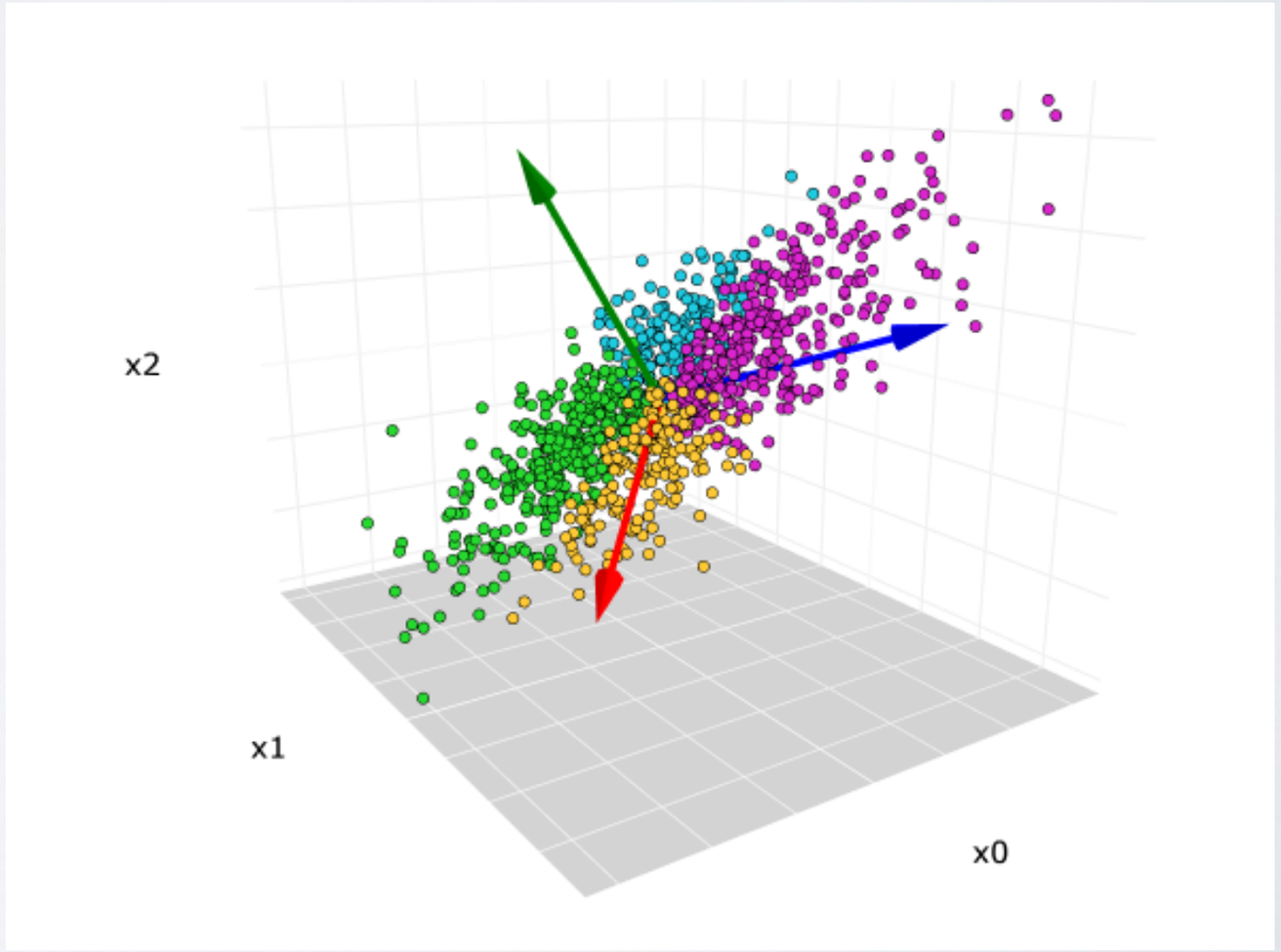
New axis 1
[50,100 000]
=>[1,20 000]

Old axis:
[0,1]
[1,0]

New axis 2
No choice:
orthogonal to axis 1

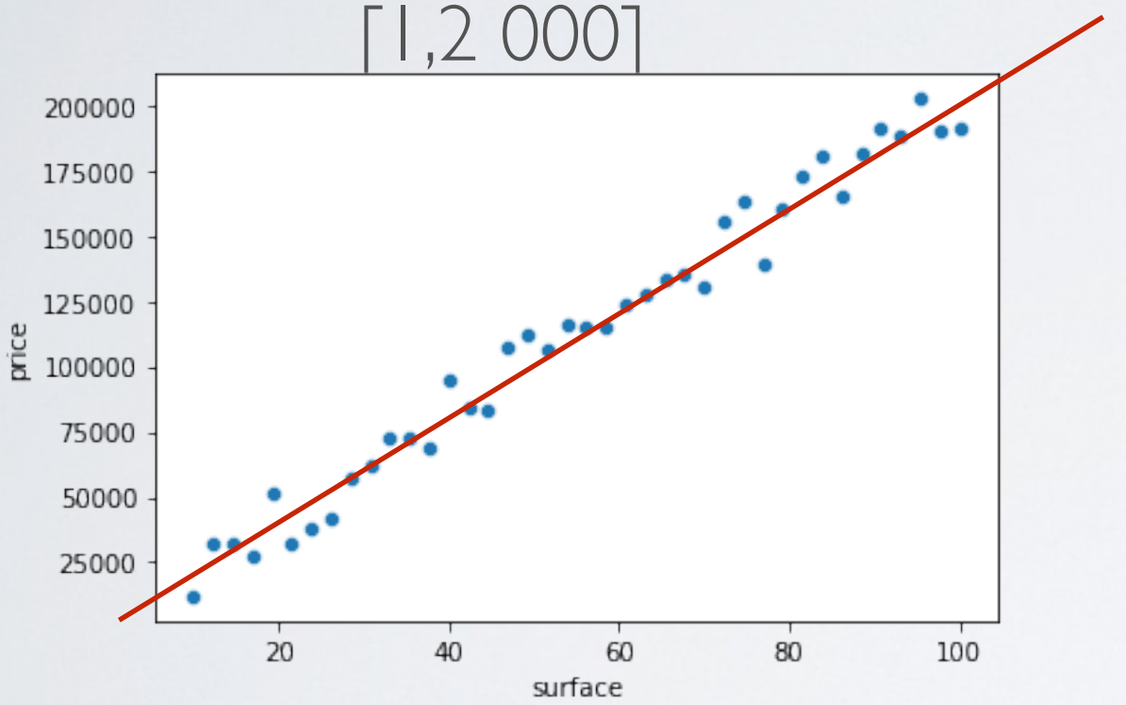
EXAMPLE PCA 2D

In 3D:
Second choice in a plan
Third choice imposed by orthogonality



EXAMPLE PCA 2D

New axis 1
[50,100 000]
[1,2 000]

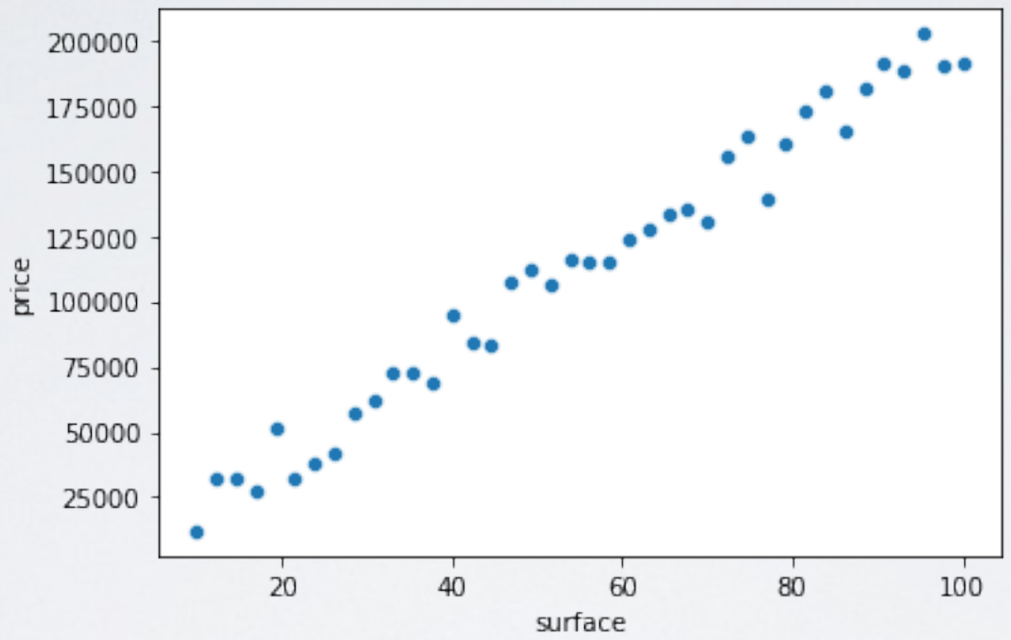


pca.components_

```
[-4.79586975e+01, -9.99999885e+04]
[ 9.99999885e+04, -4.79586975e+01]
```

=>[-48 , 99 999]
=>[1 , 2 083]

EXAMPLE PCA 2D



Covariance matrix (original)

```
[7.27810651e+02, 1.48478888e+06],
[1.48478888e+06, 3.09597381e+09]
```

Sum of variance

2897097325.718247

Variance by dimension

3.09597381e+09 727.810

Covariance matrix (pca)

```
[2.89709731e+09, 0]
[0, 1.75019564e+01]
```

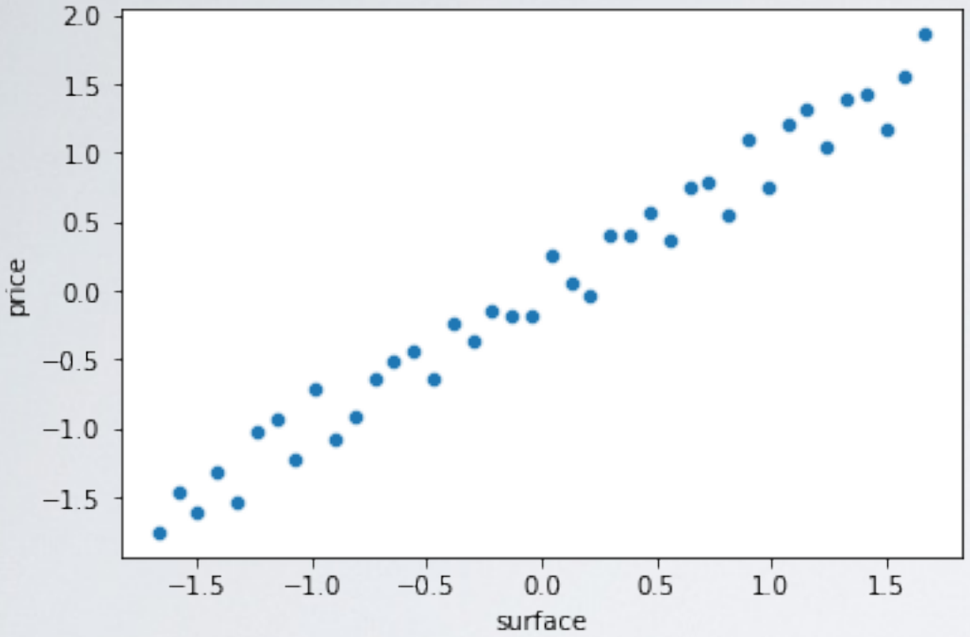
Sum of variance

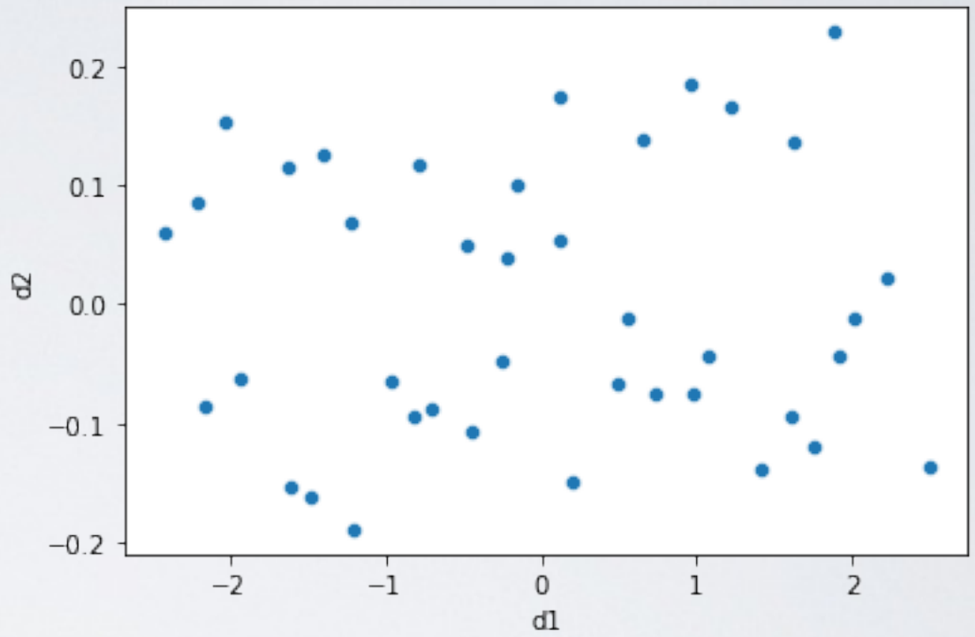
2897097325.718247

Variance by dimension

2.89709731e+09 17.501

AFTER STANDARDIZATION





Covariance matrix (original)

```
[1.        , 0.98675899],
[0.98675899, 1.        ]
```

Sum of variance

2

Variance by dimension

1 1

Covariance matrix (pca)

```
[ 1.98675899e+00, 0],
[0,  1.32410092e-02]
```

Sum of variance

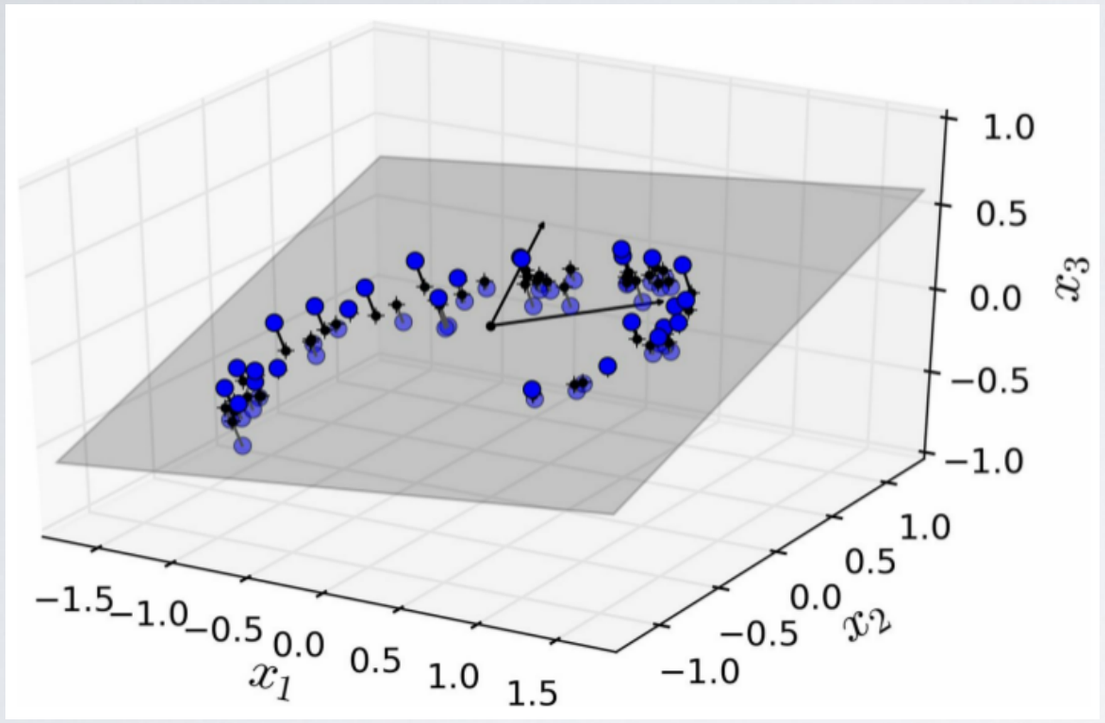
2

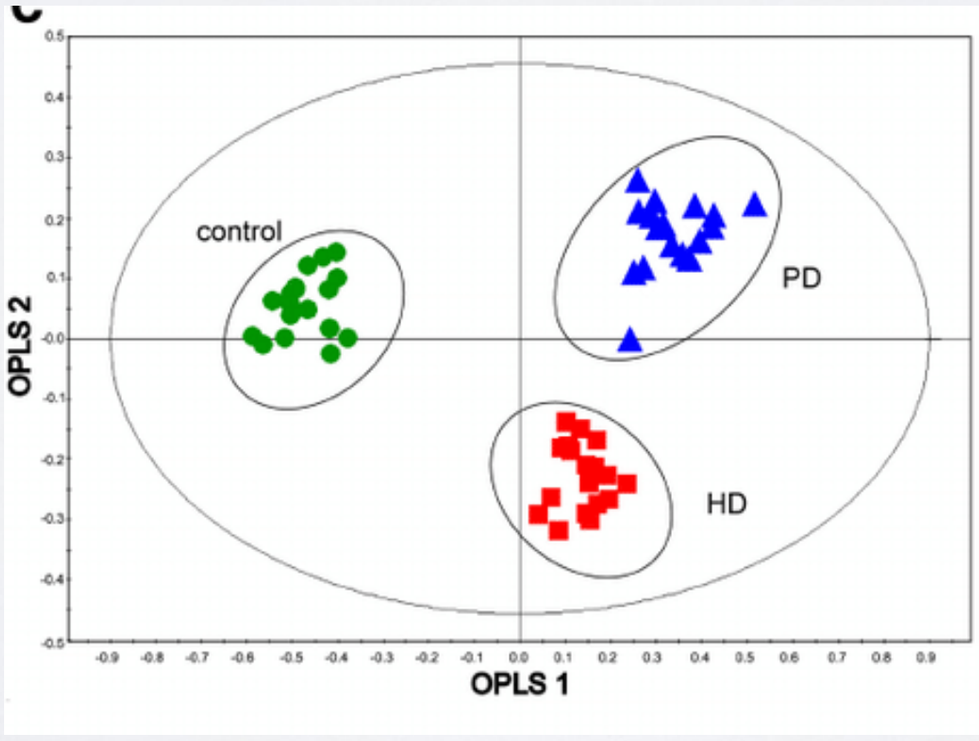
Variance by dimension

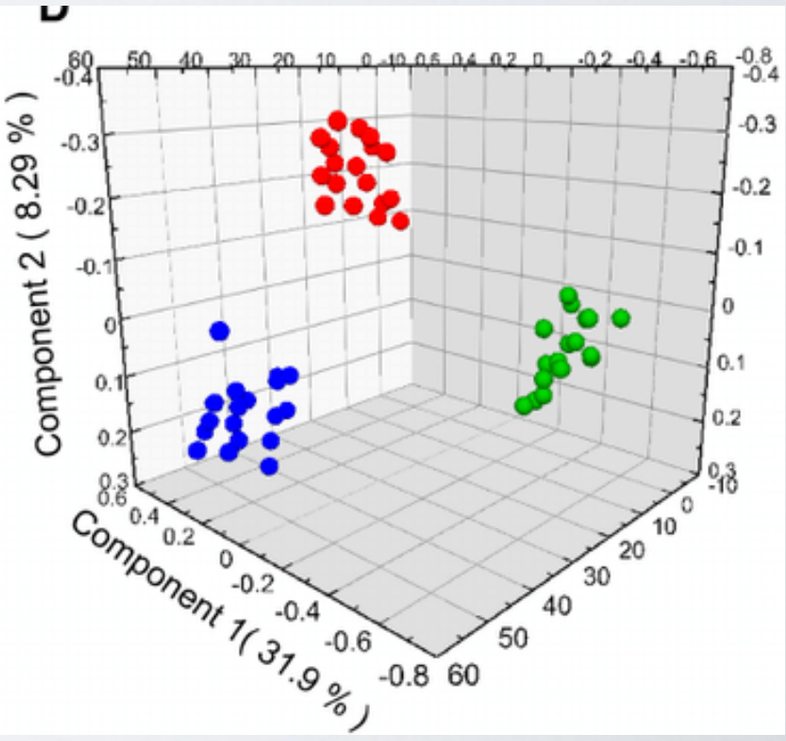
1.98675899 0.01324101

Explained variance(ratio) [0.9933795, 0.0066205]

3D=>2D


1.0
0.5
0.0
−0.5
−1.0
x_3
1.0
0.5
0.0
−0.5
−1.0
x_2
−1.5
−1.0
−0.5
0.0
0.5
1.0
1.5
x_1

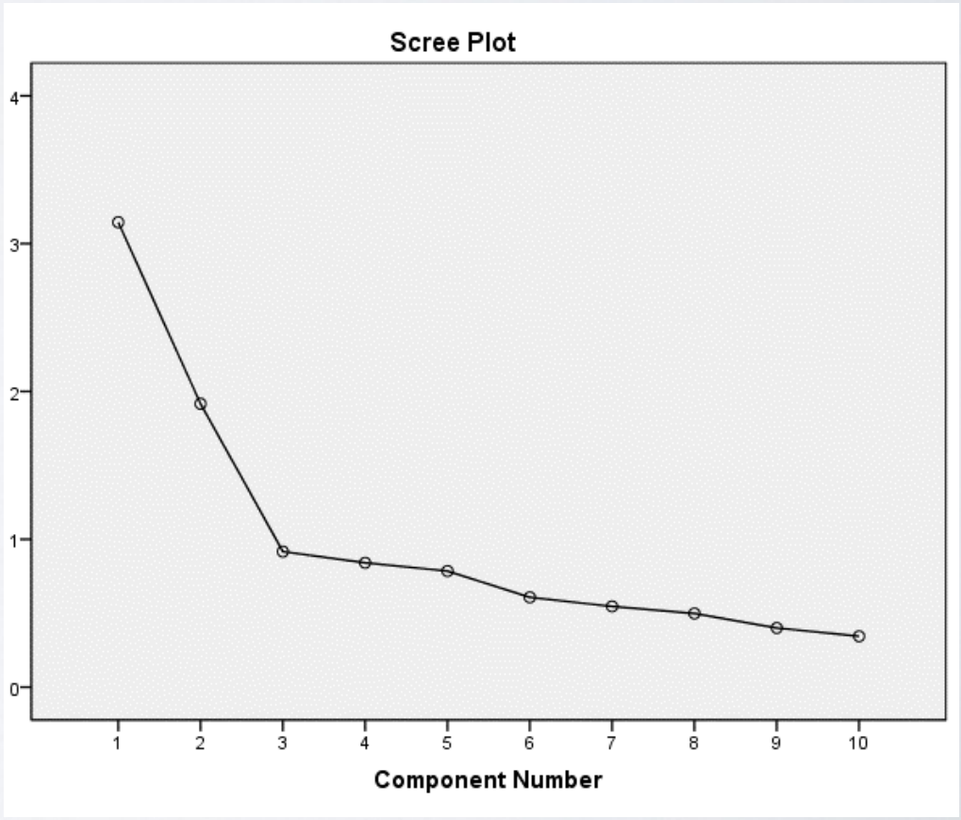

control
PD
HD
OPLS 2
OPLS 1


Component 2 (8.29 %)
Component 1(31.9 %)

CHOOSING COMPONENTS

- How to choose k?
  - Elbow method
  - OR fix beforehand a min threshold of explained variance, e.g.: 80%
    - We are fine with losing 20% of information

Explained variance



COMPUTATION IN PRACTICE

- Find the eigenvectors of the covariance matrix of centered data

- If you want k new dimensions, pick the k eigenvectors associated with the k largest eigenvalues
  - Eigenvalues = explained variance

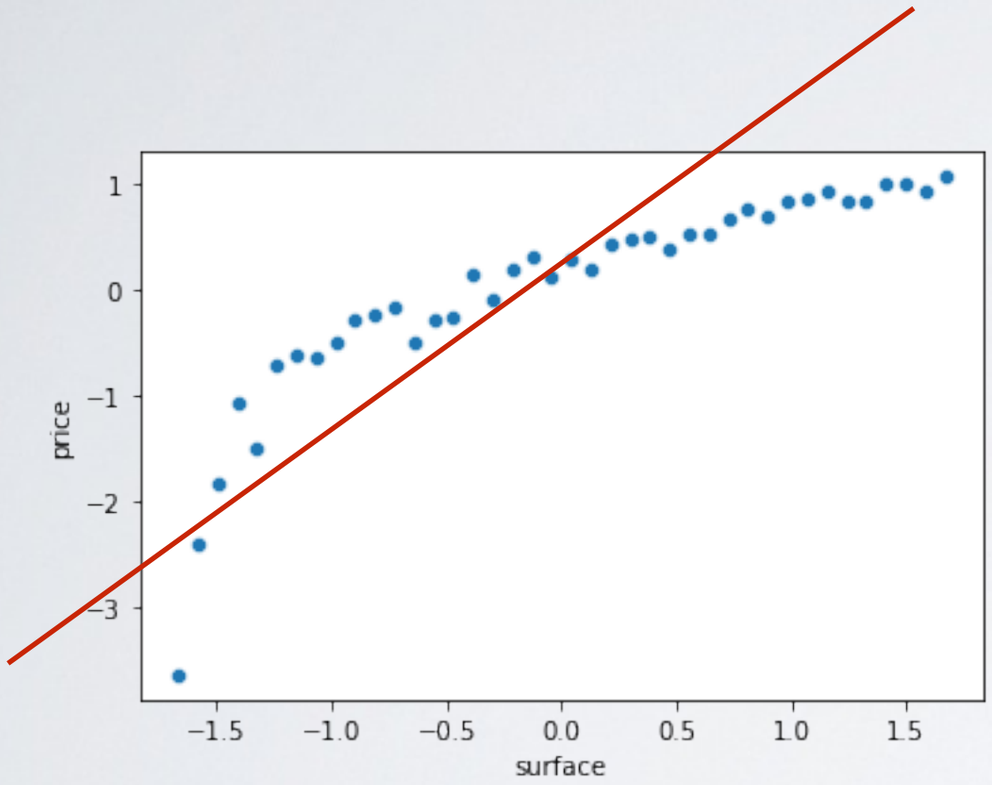
- The eigenvectors corresponding to the top eigenvalues are coefficients of the linear transformation

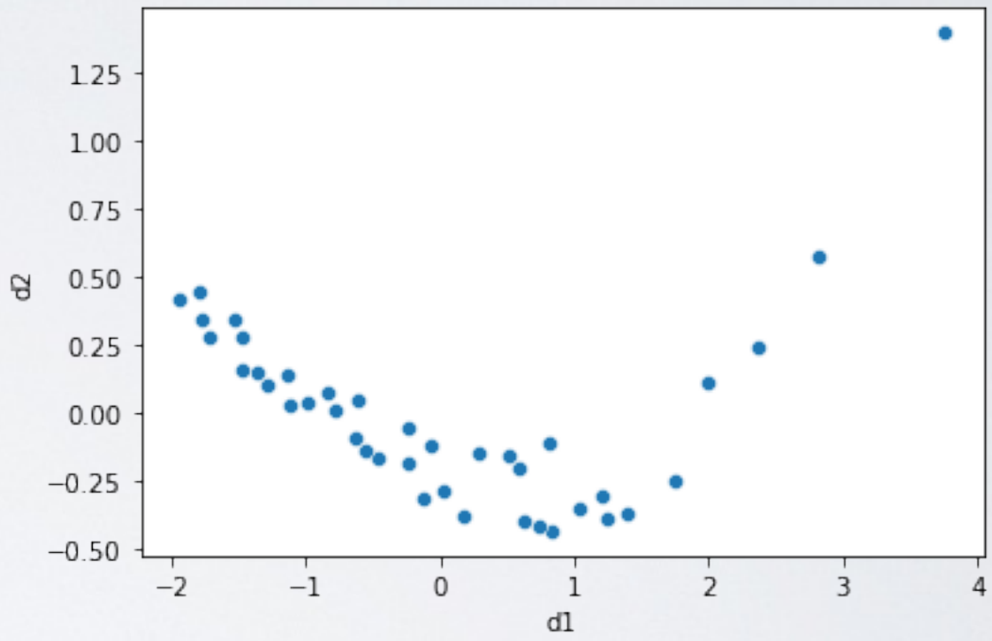
PCA POPULARITY

- Why is PCA popular?
- Similar reasons than linear regression:
  - Historically important
  - Analytical solutions
    - Guarantee to find the global minimum of the objective
    - Could be done before modern computers
  - Interpretable solution
  - Intuitively pleasant

- No reason to consider it "better" than other methods…

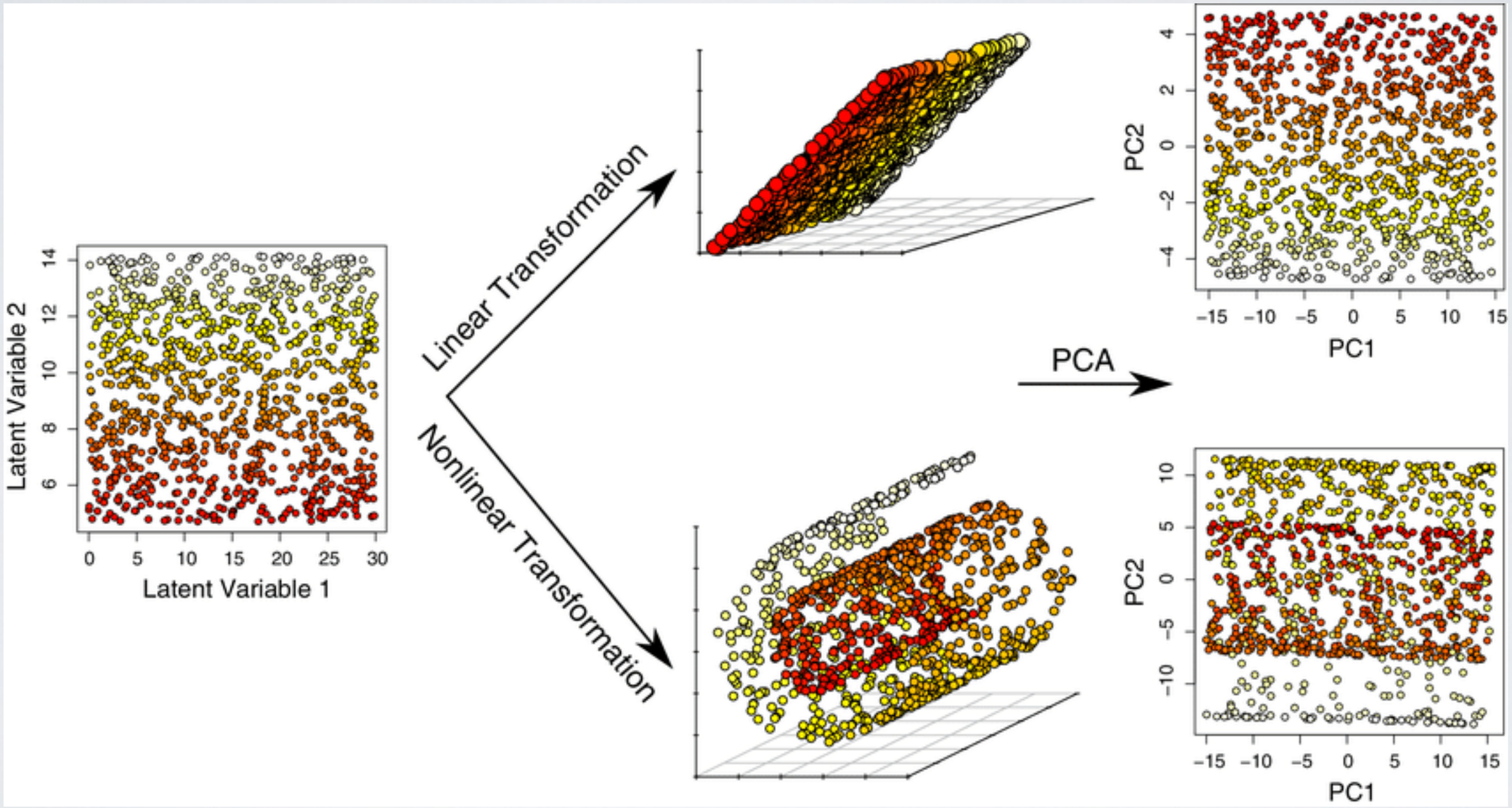
NON-LINEAR SITUATIONS






```
Pearson correlation(d1,d2): 0
```

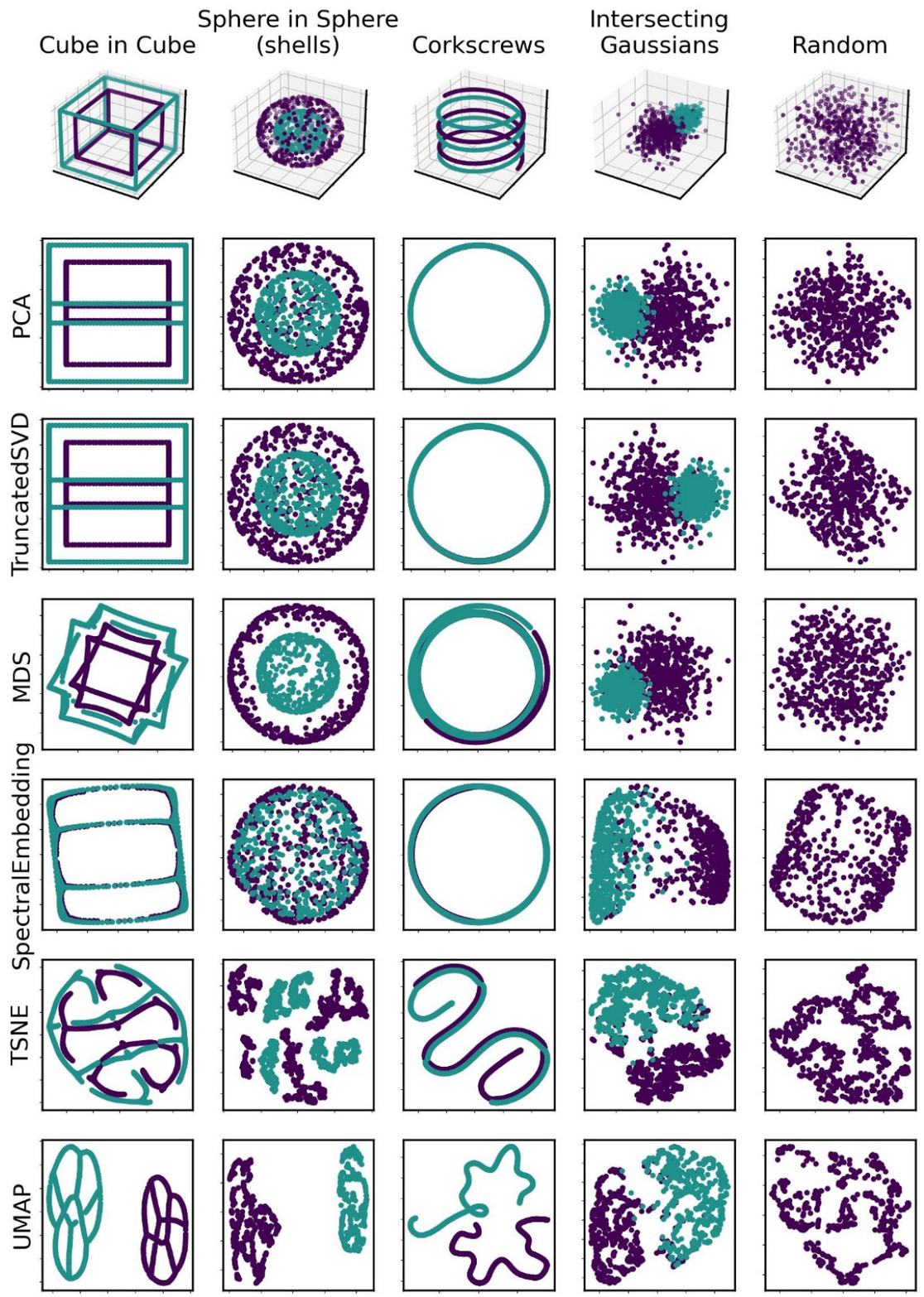
NONLINEAR DATA



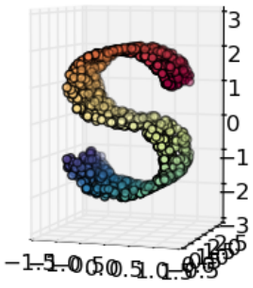
MANIFOLDS

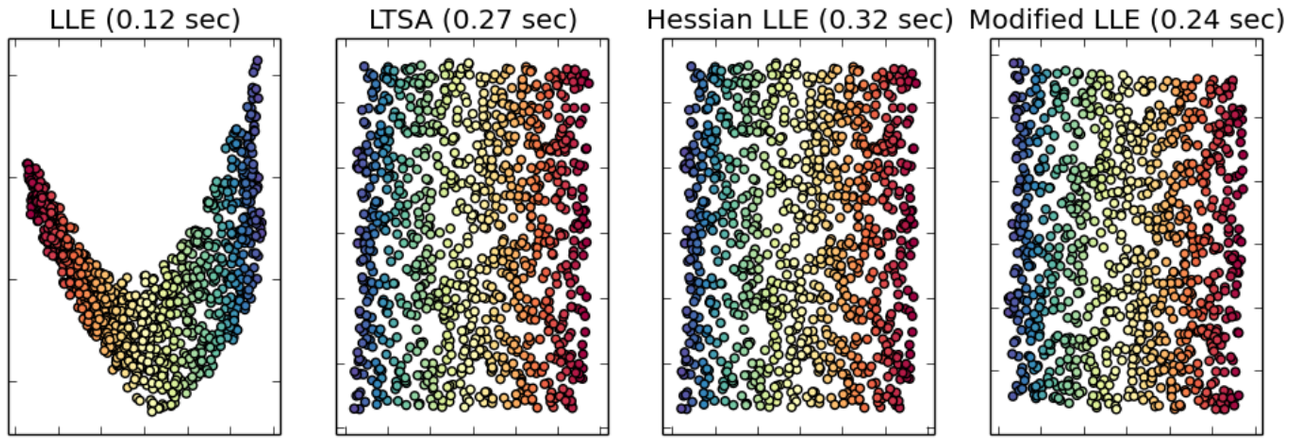
MANIFOLDS

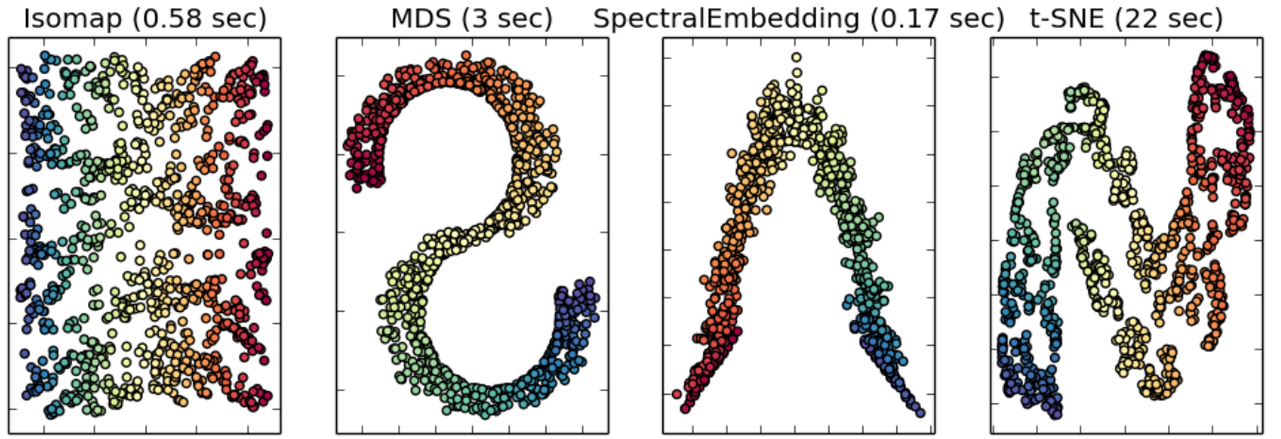
- Manifolds are another approach to dimensionality reduction
- The general principle is to
  - 1)Define a notion of distance between elements in the original space
  - 2)Define a notion of distance between elements in a reduced, target space
  - 3)Minimize the difference between distances in original and target space
- In many cases, the process is nonlinear, i.e., we choose distances such as
  - We care more about preserving close proximity than exact distance for nodes that are "far" from each other


Cube in Cube
Sphere in Sphere (shells)
Corkscrews
Intersecting Gaussians
Random
PCA
TruncatedSVD
MDS
SpectralEmbedding
TSNE
UMAP

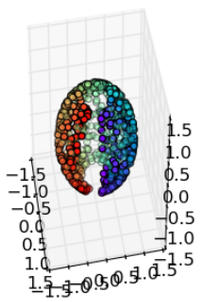
Manifold Learning with 1000 points, 10 neighbors

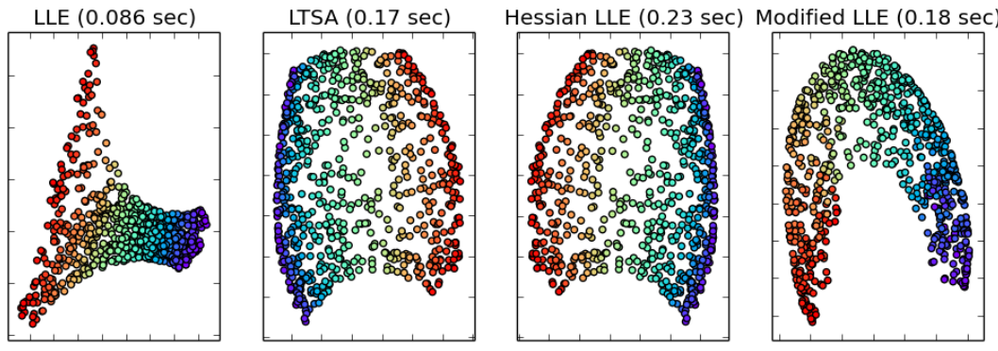


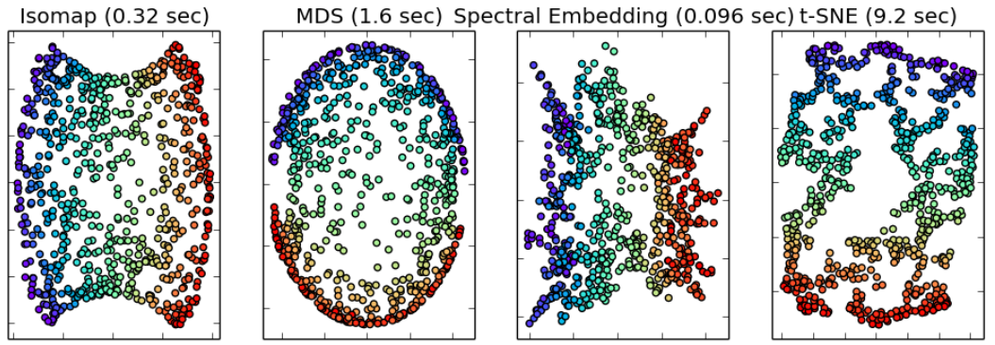


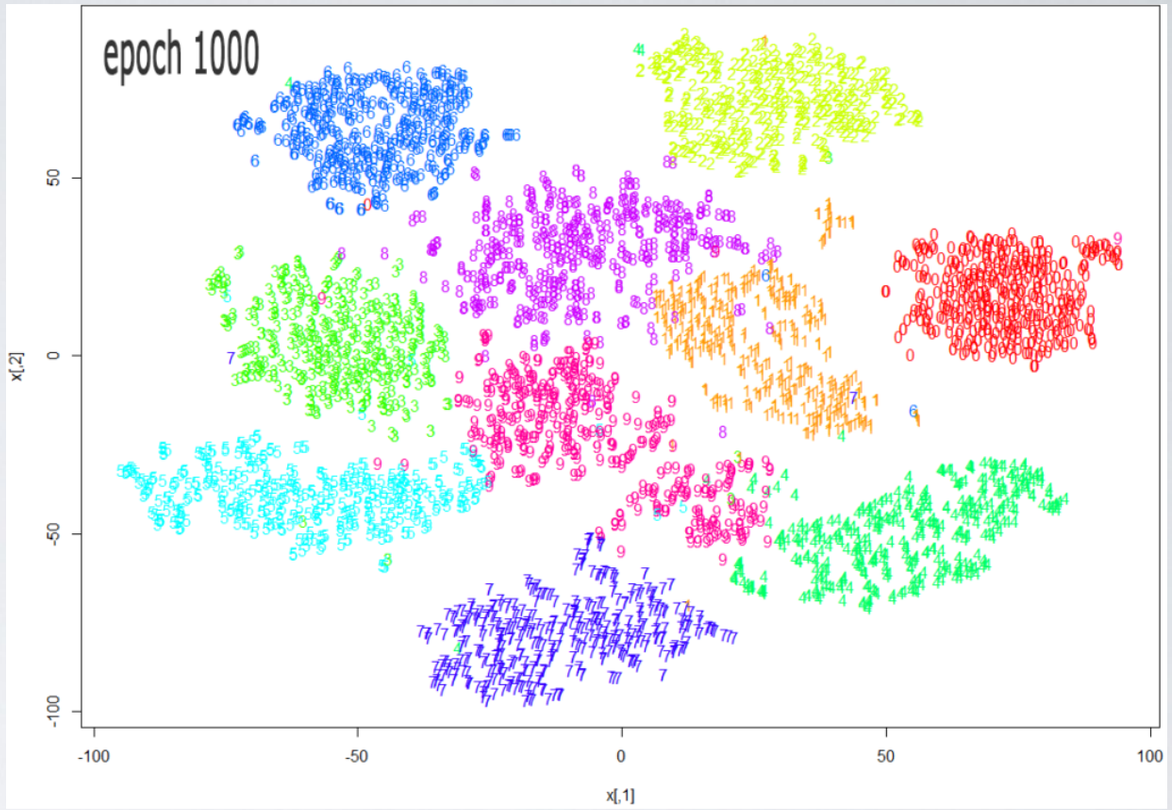


Manifold Learning with 1000 points, 10 neighbors

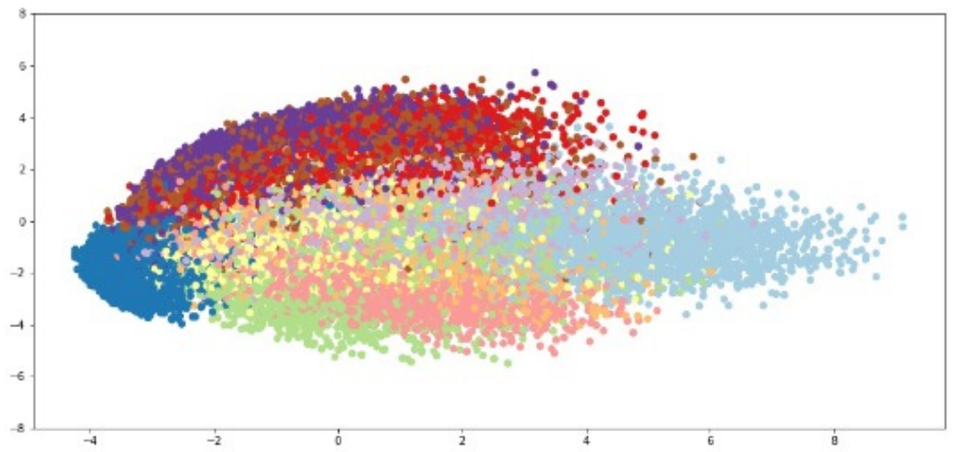




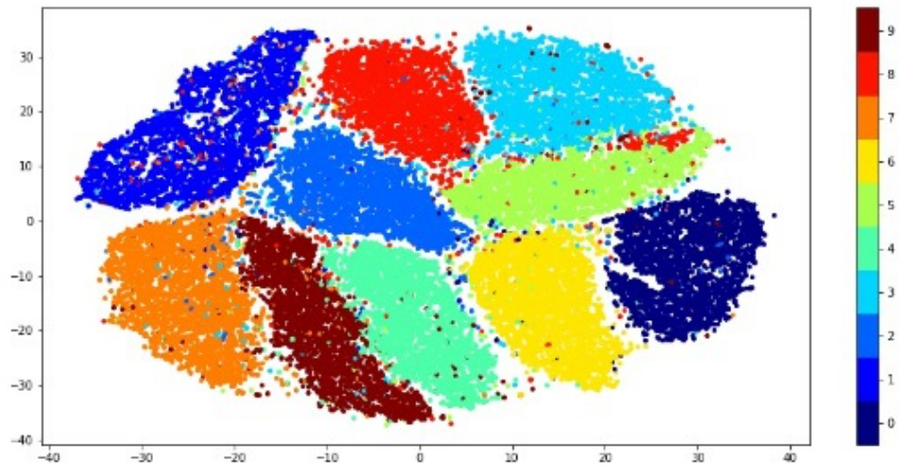




MNIST - PCA



MNIST - TSNE



MDS

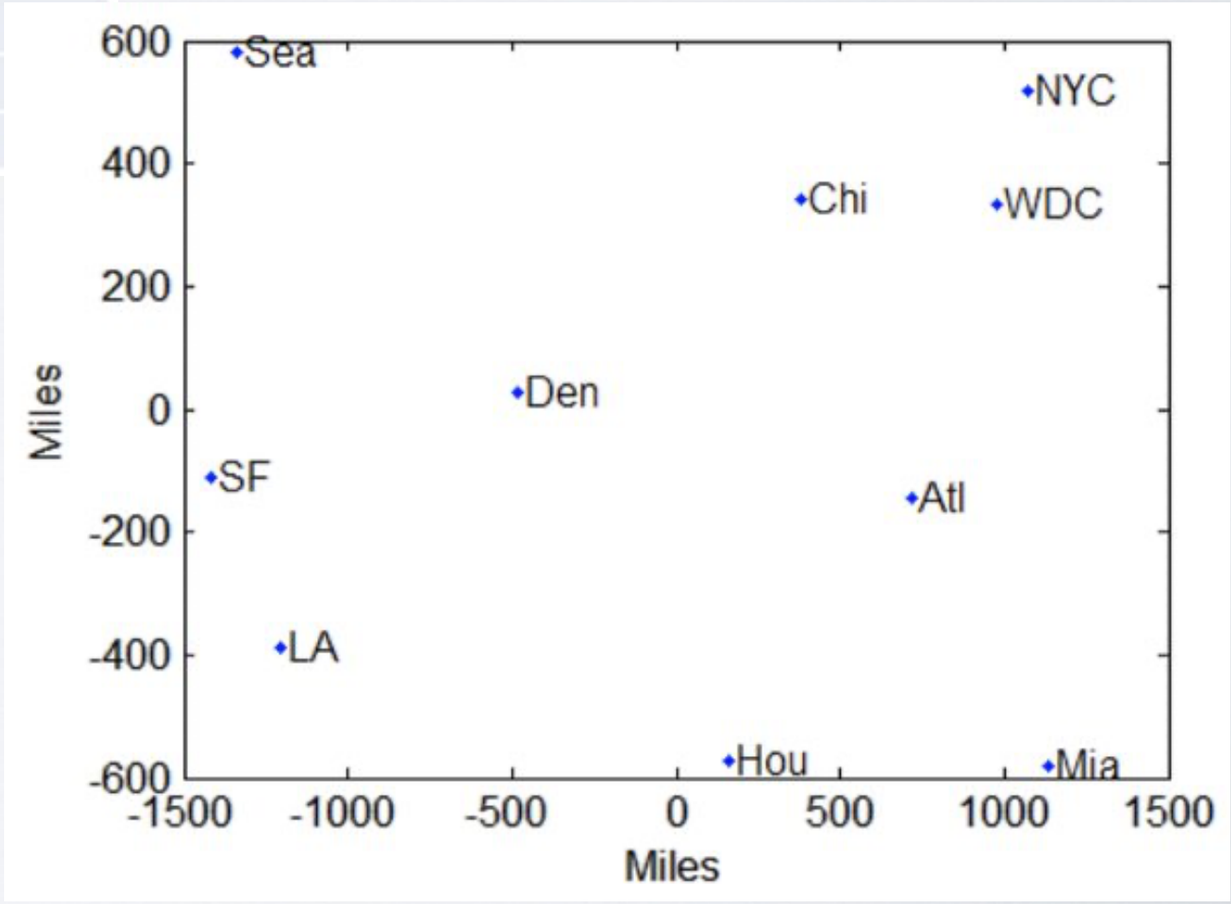
- MDS: Multi-dimensional Scaling:
  - Simply minimize distance between original space and target space
    - e.g., d-dimensional forced to 2-dimensional

- How to do it?
  - 1)Compute all pairwise distances between Objects=>similarity matrix
    - n x f matrix => n x n matrix
  - 2)Compute PCA on this similarity matrix
    - PCA preserves columns information => preserve distance on a similarity matrix

- Problems:
  - Very costly (nb features=nb elements), n^2
  - Try to preserve all distances, therefore extremely constrained

MDS

	Atl	Chi	Den	Hou	LA	Mia	NYC	SF	Sea	WDC
Atl	0	587	1212	701	1936	604	748	2139	2182	543
Chi	587	0	920	940	1745	1188	713	1858	1737	597
Den	1212	920	0	879	831	1726	1631	949	1021	1494
Hou	701	940	879	0	1374	968	1420	1645	1891	1220
LA	1936	1745	831	1374	0	2339	2451	347	959	2300
Mia	604	1188	1726	968	2339	0	1092	2594	2734	923
NYC	748	713	1631	1420	2451	1092	0	2571	2408	205
SF	2139	1858	949	1645	347	2594	2571	0	678	
Sea	2182	1737	1021	1891	959	2734	2408	678	0	
WDC	543	597	1494	1220	2300	923	205	2442	2329	

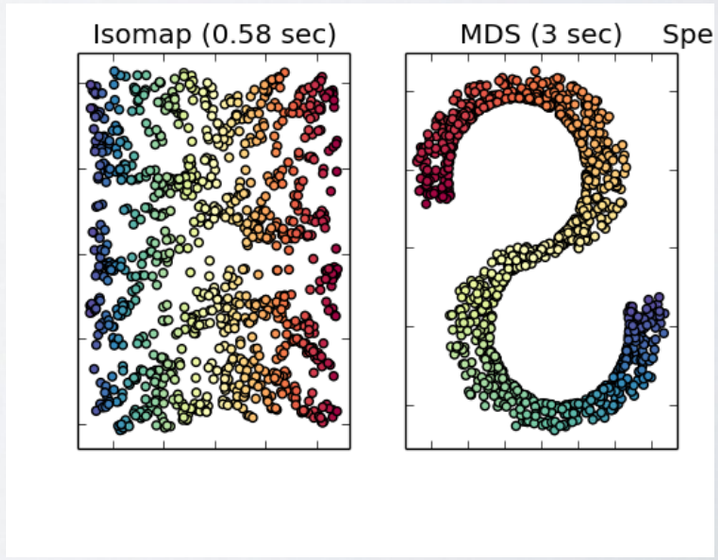


ISOMAP

- Variation of MDS
  - 1)We define a graph such as two elements are connected if they are at distance<threshold. (Alternative: fixed number of neighbors)
    - Put a weight on edges=euclidean distance
  - 2)Compute a similarity matrix, such as distance= weighted shortest path distance
  - 3)Apply PCA on it

- Computing shortest paths on a graph is fast
  - Floyd–Warshall algorithm

- Much less constraints



T-SNE

T-SNE

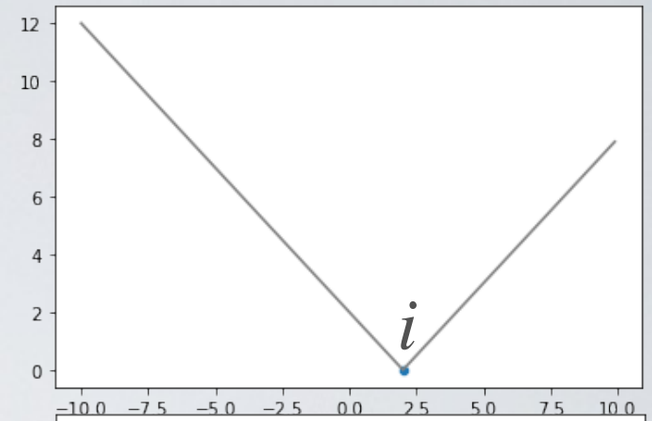
- t-SNE : t-distributed stochastic neighbor embedding
- Non-linear dimensionality reduction
- Currently the most popular method for visualizing data in low dimensions

T-SNE

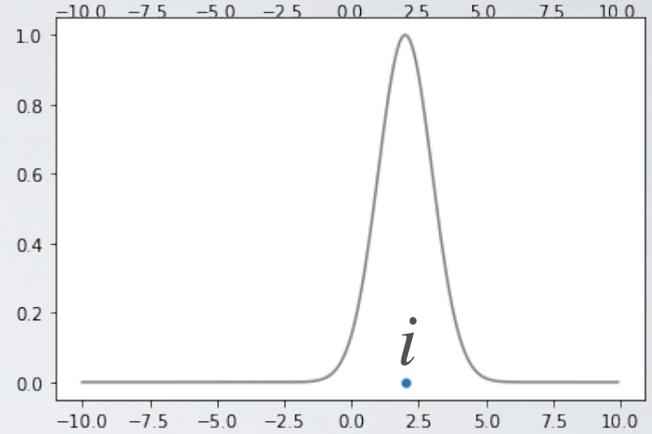
- General principle:
  - Define a notion of similarity $p_{j|i}$ in the high dimensional space P
  - Define a notion of similarity $q_{j|i}$ in the low dimensional space Q
  - For each point of initial coordinates x_i, find a new coordinate y_i in the lower dimensional space, such as to minimize the difference between P and Q
    - $\forall_{i,j} p_{j|i} \approx q_{j|i}$

SNE

Euclidean



Normal



- Distance in the original space $\boldsymbol{P}$
  - To compute how far j is from i, consider a normal distribution centered in j with variance σ
  - Mathematically: the raw distance is given as $s^P_{j|i} = e^{-\frac{\|x_i - x_j\|^2}{2\sigma^2}}$
  - $p_{j|i} = \frac{s^P_{j|i}}{\sum_{k \neq i} s^P_{j|k}}$
    - Normalizes the similarity by sum of similarity to all other points.
    - With proper σ, local definition of similarity

T-SNE

- Previous method, SNE, defines similarity in Q in the same way, setting for convenience $\sigma = \frac{1}{\sqrt{2}}$, thus
  - $s^Q_{j|i} = e^{\|y_i - y_j\|^2}$
    - y are *computed* new features
  - With $q_{j|i} = \frac{s^Q_{j|i}}{\sum_{k \neq i} s^Q_{j|k}}$

SNE

- Define the similarity between P and Q
  - Minimize the KL-divergence (Kullback-Leibler divergence)
    - $C = \mathrm{KL}\left(P \parallel Q\right) = \sum_{i \neq j} p_{ij} \log \frac{p_{ij}}{q_{ij}}$
  - KL: Information theory:
    - Average additional number of bits required to encode a sample from P using the optimal code to encode samples of Q
    - Non-symmetric

- Solved by gradient descent
  - $\frac{\partial C}{\partial y_i} = 2 \sum_j (p_{j|i} - q_{j|i} + p_{i|j} - q_{i|j})(y_i - y_j)$
    - y_i is the vector of i in the new space

SNE

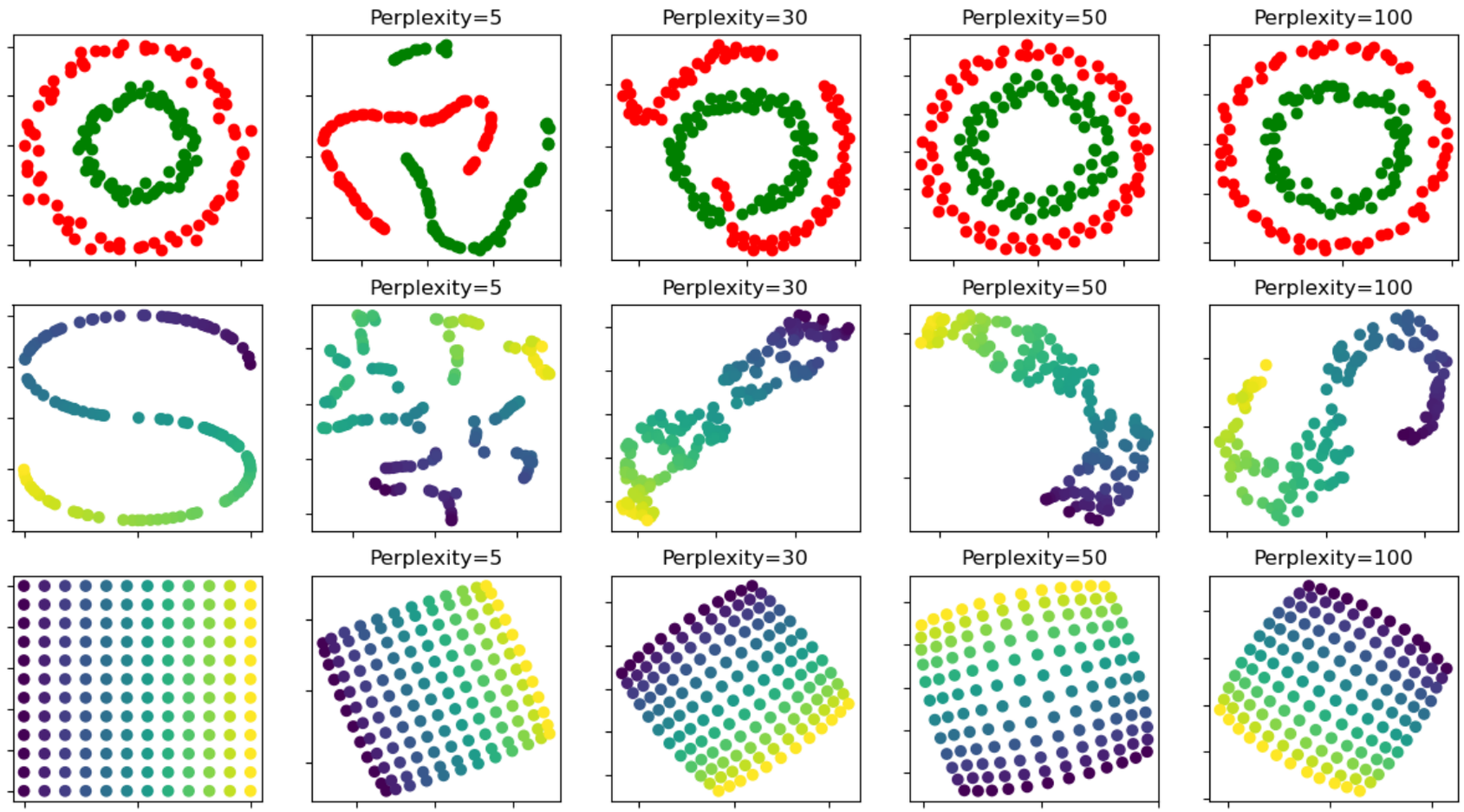
- $\frac{\partial C}{\partial y_i} = 2\sum_j (p_{j|i} - q_{j|i} + p_{i|j} - q_{i|j})(y_i - y_j)$

- Intuitive interpretation:
  - Imagine a system of attraction/springs between points
  - If the distance is right, left=0, don't move
  - If i is "too far", left is positive, we go in the direction of j
  - If i is "too close", left is negative, we move away from j

SNE

- Missing part: how to fix σ for similarity in original space?
- Density of points is assumed heterogeneous, σ is chosen independently for each point.
  - User fix a parameter of ***perplexity***, $perp(p_i) = 2^{H(p_i)}$
    - With $H(P_i) = -\sum_j p_{j|i} log_2 p_{j|i}$ Shannon Entropy
    - Search for σ_i to obtain the right perplexity.
  - Idea similar to imposing a fix number of neighbors
    - But you can have few points at very short distance and many at long distance
    - Or many at medium distance
    - As long as the overall distribution of distance is respected

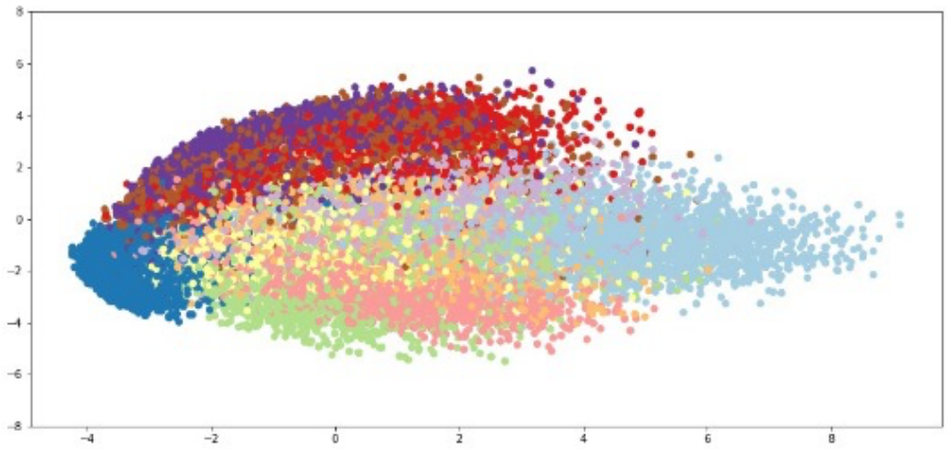
INFLUENCE OF PERPLEXITY



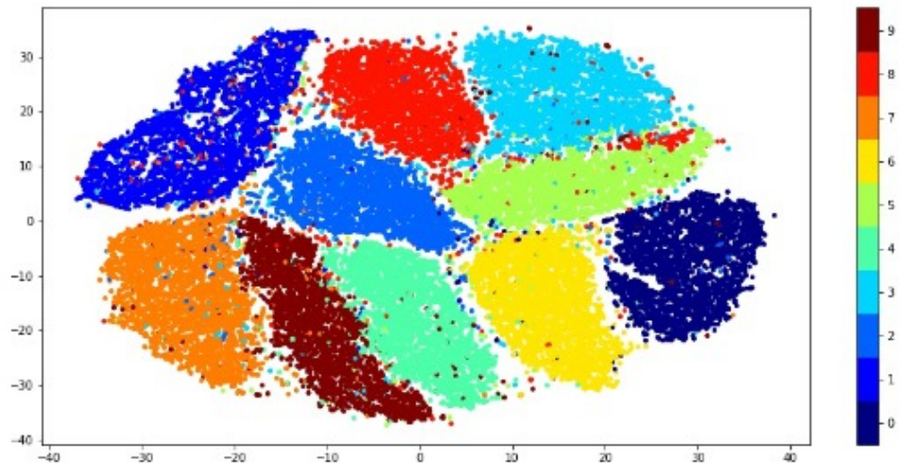
T-SNE

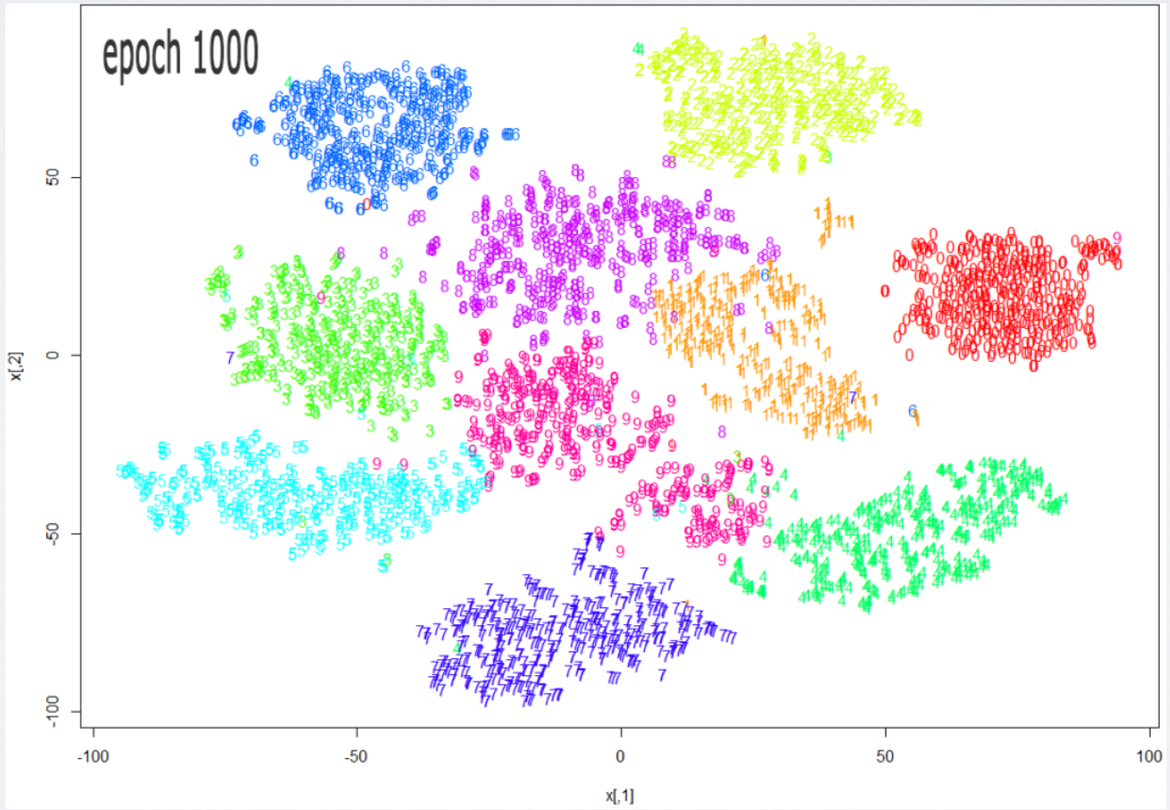
- T-SNE modifies the distance in the lower dimensional space
- It uses a Student-t distribution, which leads to several advantages
  - Makes optimisation easier and faster
  - Tends to "exaggerate" similarities/dissimilarities
    - Emphasizes clusters, i.e., groups of nodes all close together

MNIST - PCA



MNIST - TSNE





LOW DIMENSIONAL EMBEDDINGS

EMBEDDINGS

- A recent usage of low dimensional embeddings is to encode complex objects as vectors
  - Words as Vector => Word2Vec
  - Nodes (of graph) as Vectors => Node2Vec
  - Documents as Vectors => Doc2Vec
  - ....

WORD EMBEDDING

WORD EMBEDDING

- Words can be understood as a (very) high dimensional space
  - Using One Hot encoding: vocabulary of 1000 words=>1000 columns

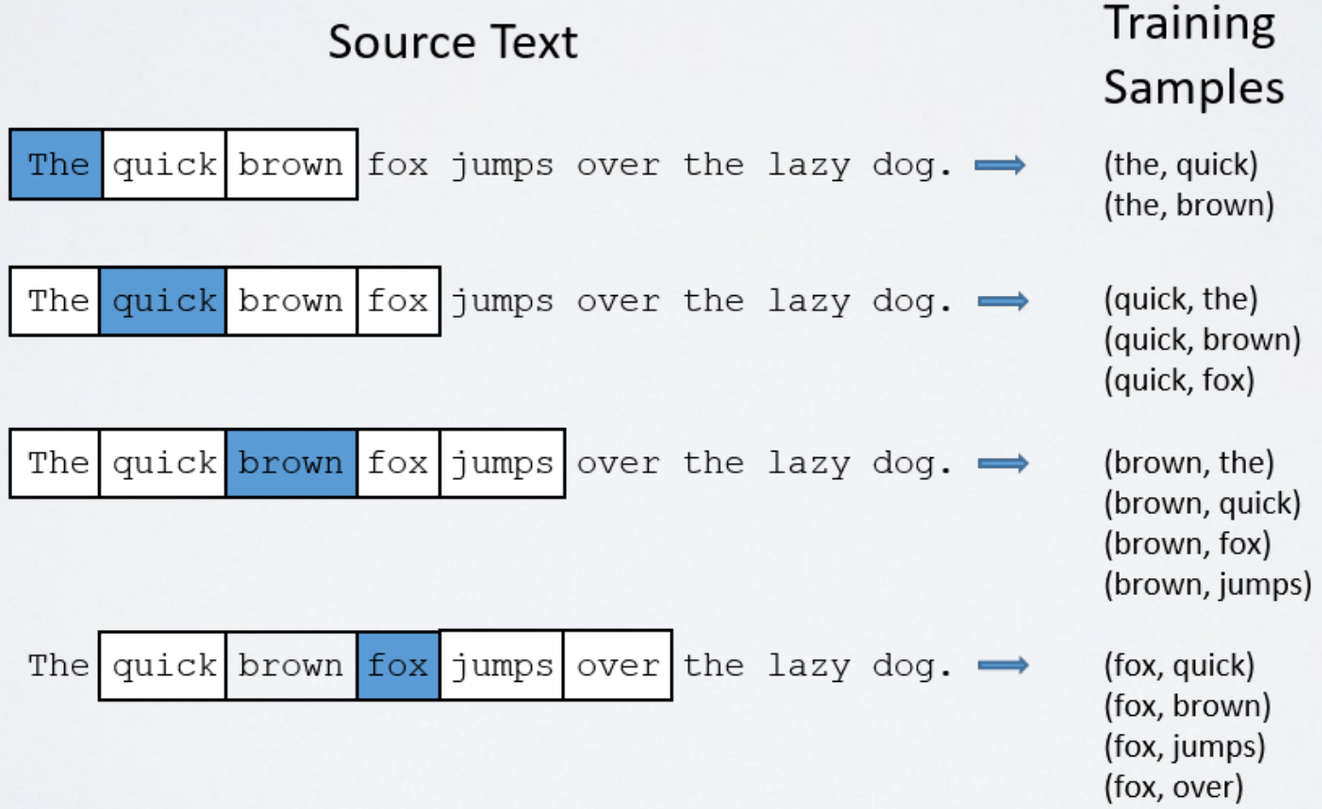
- Could we assign a vector in "low dimension", encoding the "semantic" of a word?
  - Two words with similar meanings should be close

SKIPGRAM

Word embedding

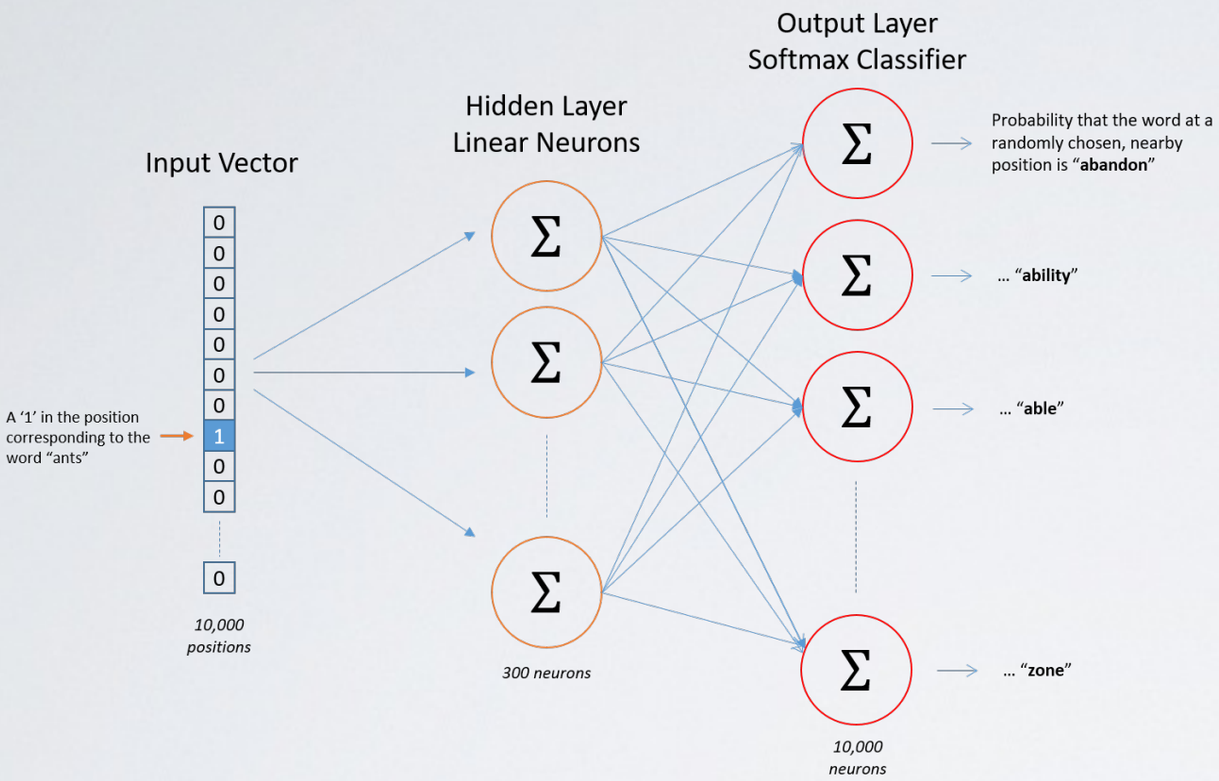
Corpus => Word = vectors

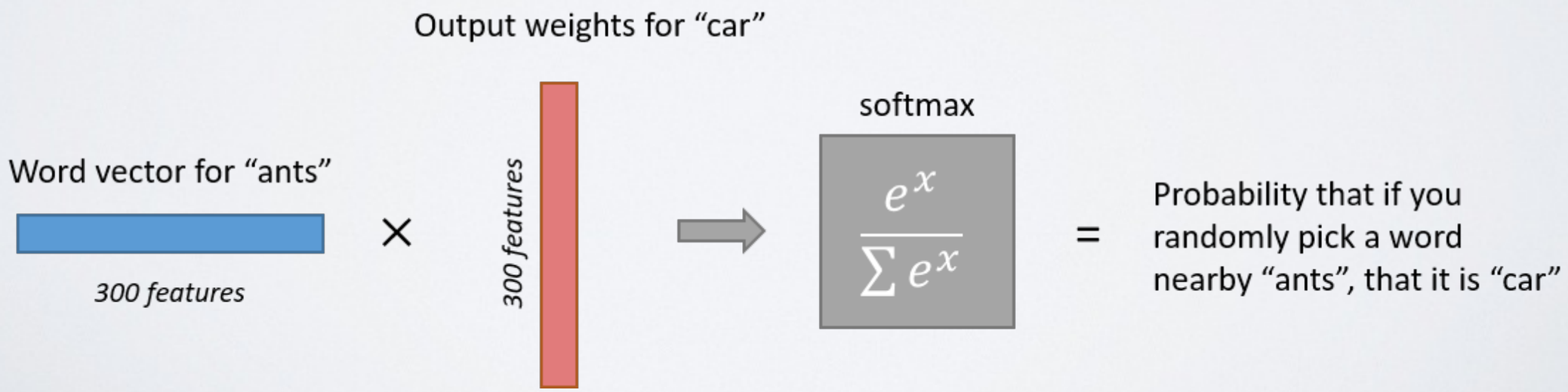
Similar embedding= similar **context**



[http://mccormickml.com/2016/04/19/word2vec-tutorial-the-skip-gram-model/]

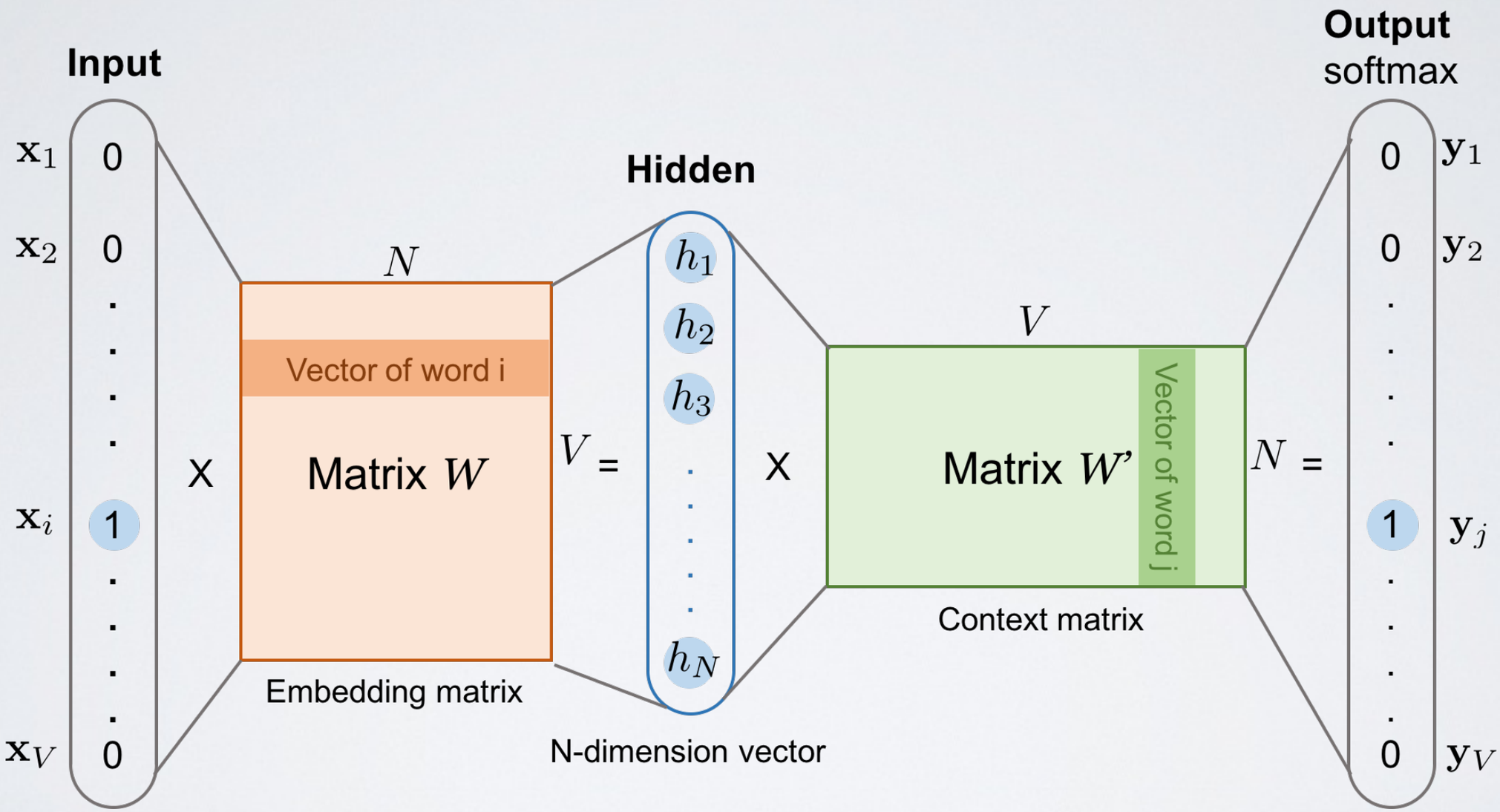
SKIPGRAM





https://towardsdatascience.com/word2vec-skip-gram-model-part-1-intuition-78614e4d6e0b

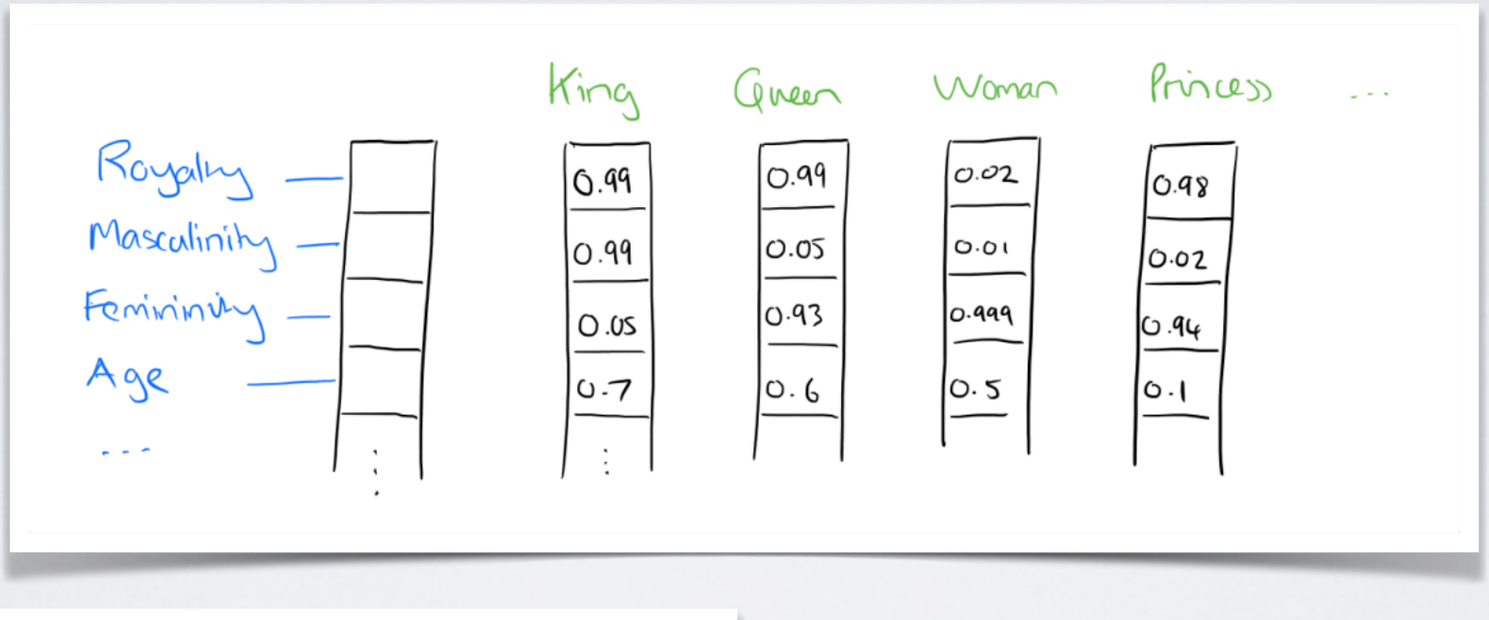
SKIPGRAM

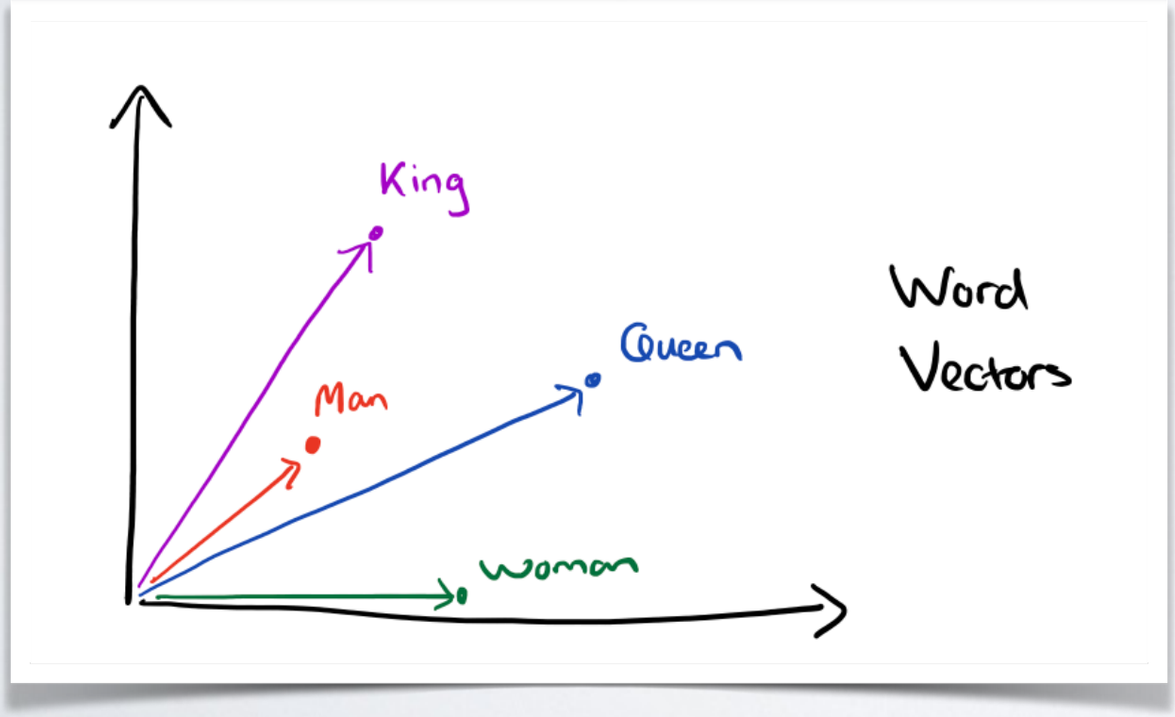


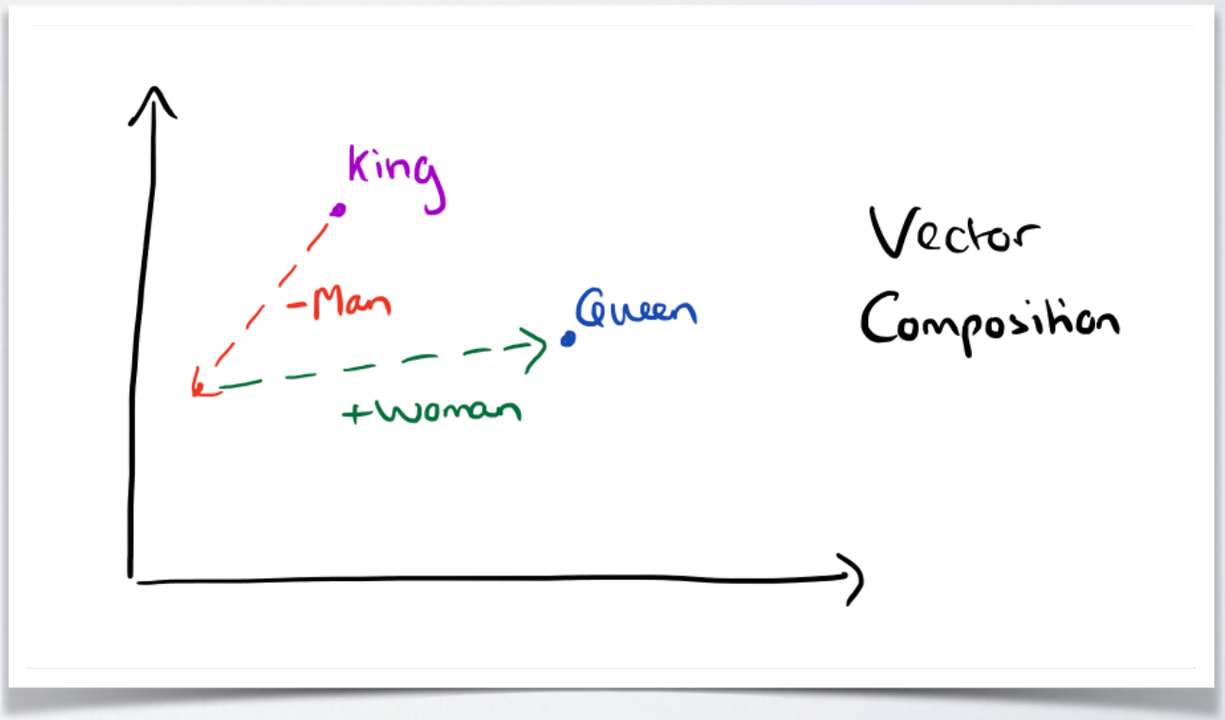
N=embedding size. **V**=vocabulary size

https://towardsdatascience.com/word2vec-skip-gram-model-part-1-intuition-78614e4d6e0b

SKIPGRAM







[https://blog.acolyer.org/2016/04/21/the-amazing-power-of-word-vectors/]

SKIPGRAM

Table 8: *Examples of the word pair relationships, using the best word vectors from Table 4 (Skip-gram model trained on 783M words with 300 dimensionality).*

Relationship	Example 1	Example 2	Example 3
France - Paris	Italy: Rome	Japan: Tokyo	Florida: Tallahassee
big - bigger	small: larger	cold: colder	quick: quicker
Miami - Florida	Baltimore: Maryland	Dallas: Texas	Kona: Hawaii
Einstein - scientist	Messi: midfielder	Mozart: violinist	Picasso: painter
Sarkozy - France	Berlusconi: Italy	Merkel: Germany	Koizumi: Japan
copper - Cu	zinc: Zn	gold: Au	uranium: plutonium
Berlusconi - Silvio	Sarkozy: Nicolas	Putin: Medvedev	Obama: Barack
Microsoft - Windows	Google: Android	IBM: Linux	Apple: iPhone
Microsoft - Ballmer	Google: Yahoo	IBM: McNealy	Apple: Jobs
Japan - sushi	Germany: bratwurst	France: tapas	USA: pizza

[https://blog.acolyer.org/2016/04/21/the-amazing-power-of-word-vectors/]

PRE-TRAINED

- You can easily train word2vec on your own dataset, but it needs to be large enough
  - https://radimrehurek.com/gensim/models/word2vec.html
- You can use pre-trained embeddings, trained on enormous corpus (Twitter, Wikipedia…)
  - e.g., Glove: https://nlp.stanford.edu/projects/glove/

USAGE

- Single words=> Use directly vectors

- Short texts=> Weighted average vectors (more weights to more important words, e.g., rare words: TF-IDF…)

- Long texts=> More tricky. Need other approaches (Doc2vec, RNN)

USAGE

- Parameters:
  - Embedding dimensions *d*
  - Context size

GRAPH EMBEDDING

GENERIC "SKIPGRAM"

- Algorithm that takes an input:
  - The element to embed
  - A list of "context" elements
- Provide as output:
  - An embedding with interesting properties
    - Works well for machine learning
    - Similar elements are close in the embedding
    - Somewhat preserves the overall structure

DEEPWALK

- Skipgram for graphs:
  - 1)Generate "sentences" using random walks
  - 2)Apply Skipgram

- Parameters:
  - Same as Skipgram
    - Embedding dimensions *d*
    - Context size
  - Parameters for "sentence" generation: length of random walks, number of walks starting from each node, etc.

Perozzi, B., Al-Rfou, R., & Skiena, S. (2014, August). Deepwalk: Online learning of social representations. In *Proceedings of the 20th ACM SIGKDD international conference on Knowledge discovery and data mining* (pp. 701-710). ACM.

NODE2VEC

- Use biased random walk to tune the context to capture *what we want*
  - "Breadth first" like RW => local neighborhood (edge probability ?)
  - "Depth-first" like RW => global structure ? (Communities ?)
  - 2 parameters to tune:
    - **p**: bias towards revisiting the previous node
    - **q**: bias towards exploring undiscovered parts of the network

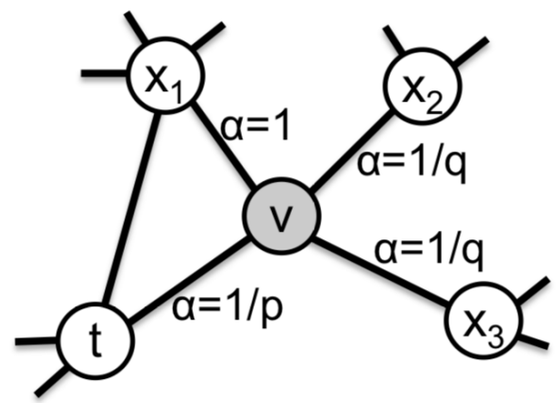


Figure 2: Illustration of the random walk procedure in *node2vec*. The walk just transitioned from t to v and is now evaluating its next step out of node v. Edge labels indicate search biases α.

Grover, A., & Leskovec, J. (2016, August). node2vec: Scalable feature learning for networks. In *Proceedings of the 22nd ACM SIGKDD international conference on Knowledge discovery and data mining* (pp. 855-864). ACM.