# Applications of unsupervised learning to accelerators

**Presenter:** R. Lehe

**Day 8**

# Supervised vs unsupervised machine learning

**Supervised:**

- The training data contains inputs $\boldsymbol{x}$ and labels $y$
- The aim is to **predict the label** $y$ for **new inputs** $\boldsymbol{x}$

Example: spam detection

$\boldsymbol{x}$: representation of the text of an email
$y$: 0 = valid email
1 = spam

**Unsupervised:**

- The data only contains inputs $\boldsymbol{x}$
- The aim is to find **"interesting patterns"** in the data

Example: group a large number of recent news articles into several "trending topics"

$\boldsymbol{x}$: representation of the text of a news article

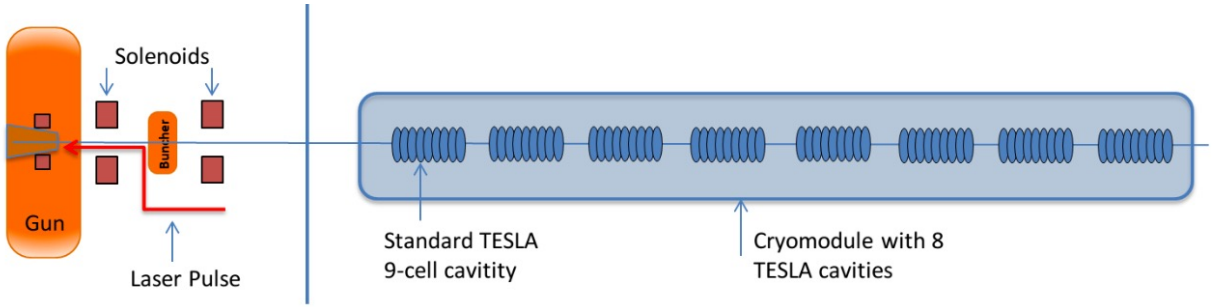
More vague ; oftentimes, no **ground truth** or **quantitative measure** of how well the algorithm performs

# Outline

- Dimensionality reduction
  - PCA
  - Auto-encoders
- Clustering
  - K-means
  - DBSCAN
- Anomaly detection

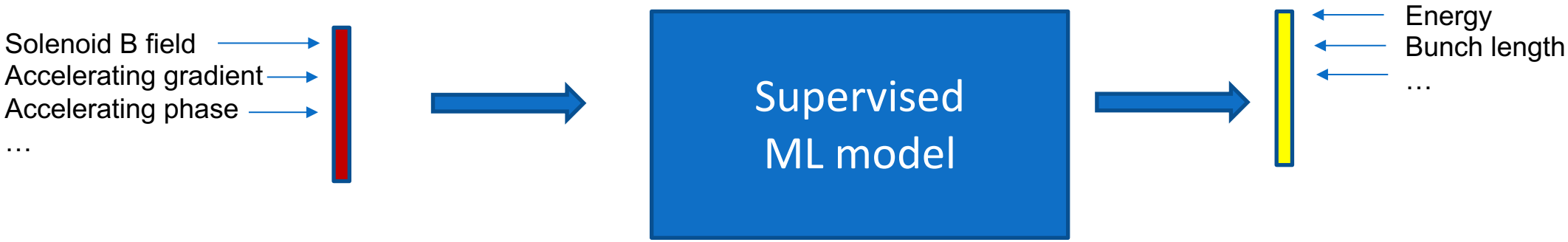
# Dimensionality reduction: overview

**Example:** prediction of accelerator properties



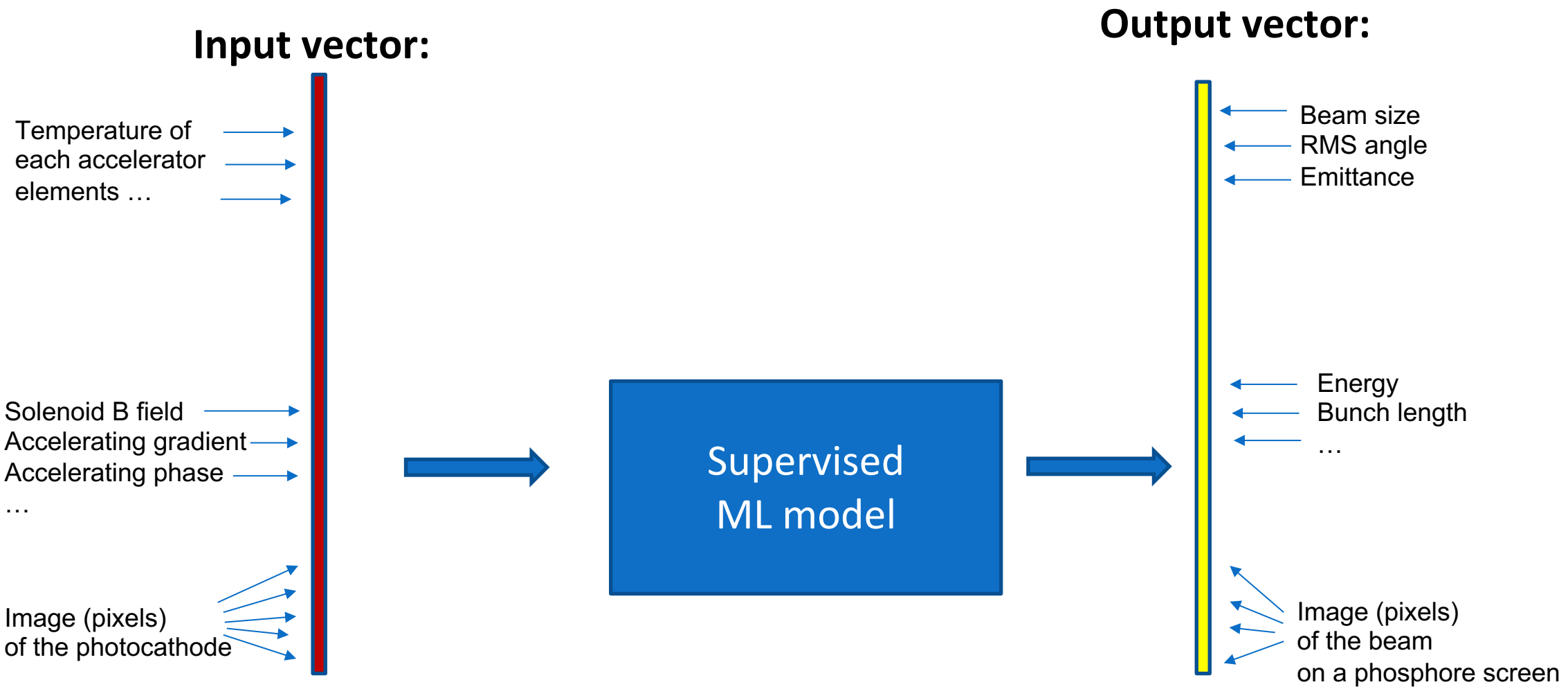

**Input vector:**
accelerator parameters

**Output vector:**
final beam properties



# Dimensionality reduction: overview

Input/output vectors can often become **very large** (i.e. high-dimensional).
Some of that data can be **redundant**.



# Issues with ML using high-dimensional vectors

- In principle, ML methods can deal with **high-dimensional** and **redundant** data.

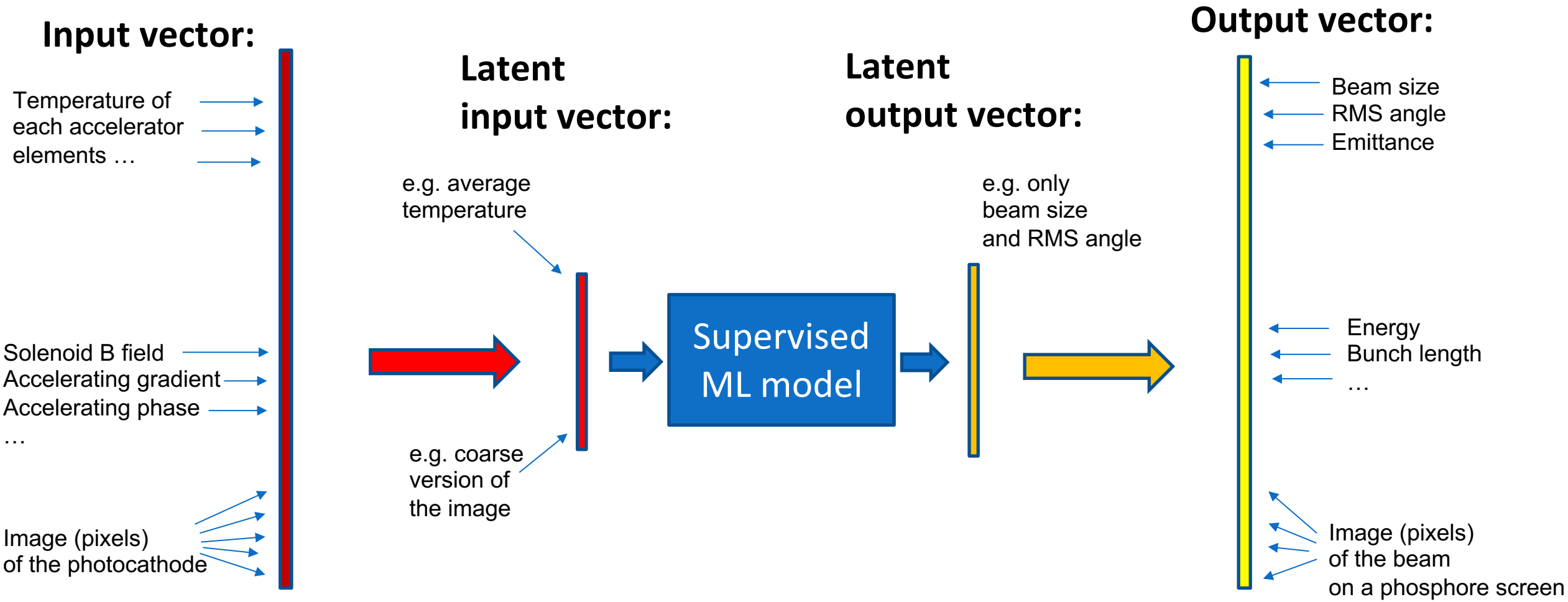
But:

- Training is **more expensive** (esp. Gaussian process)
- More likely to **overfit** the training data
- Harder for a human to **interpret**

# Dimensionality reduction: overview

**Idea:** reduce the **size** of the input/output that goes into the ML model.

Use **compressed ("latent")** input/output vector that **eliminates some of the redundant information** and capture the **essence of the data.**

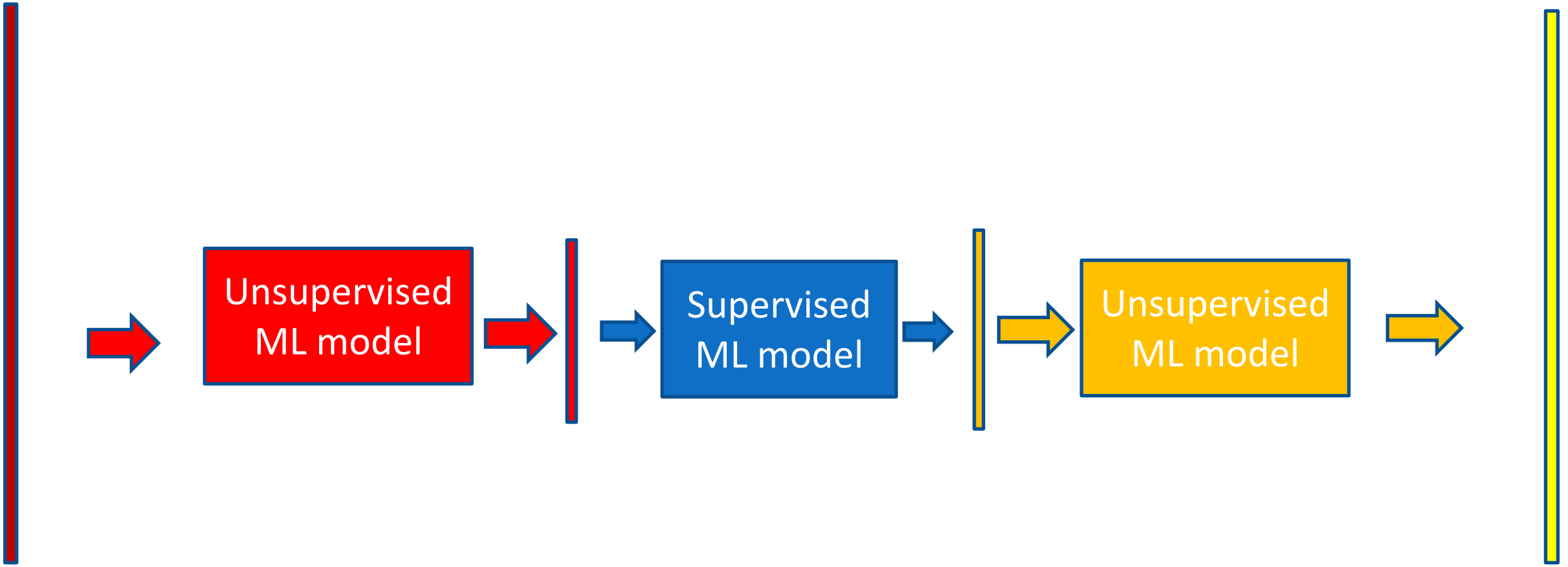


# Dimensionality reduction: overview

The discovery of the relevant “compressed features” need to be **automated**.

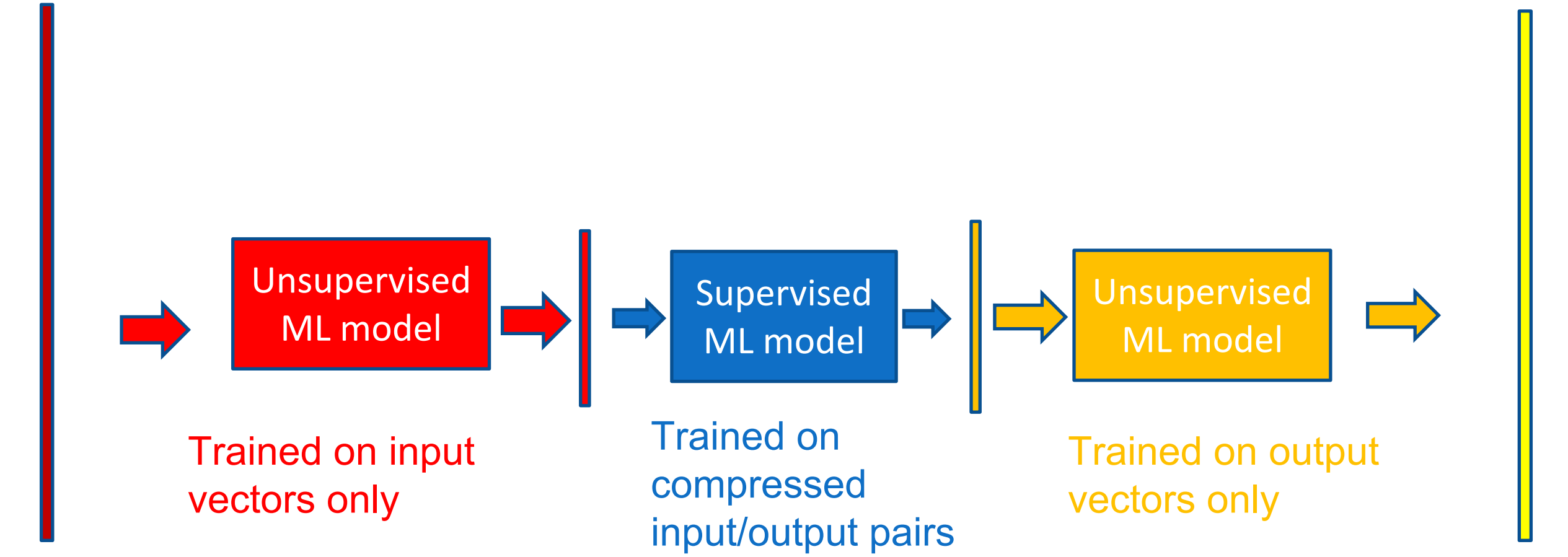
**Input vector:**

**Output vector:**



# Dimensionality reduction: overview

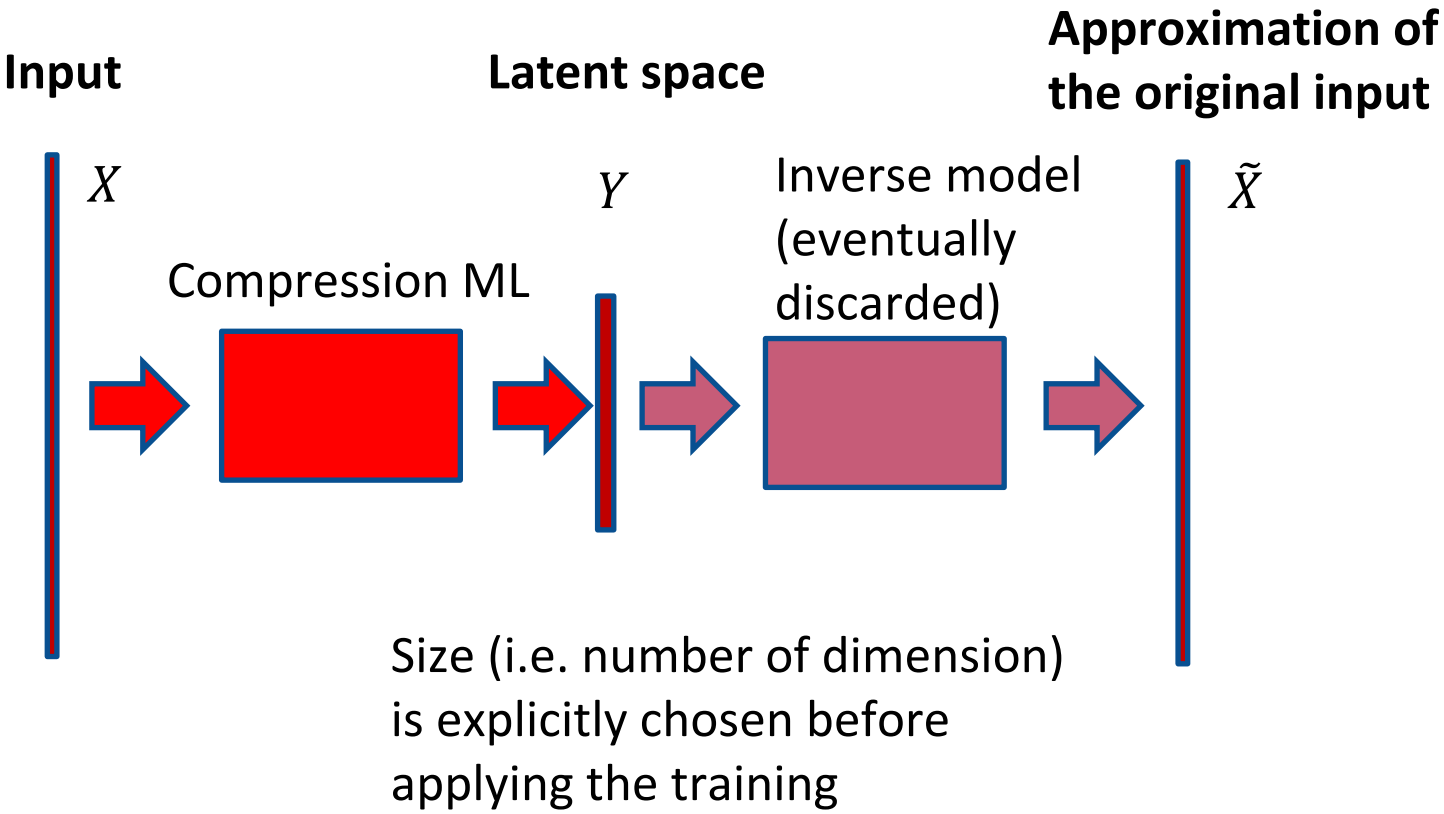
**Key point:** the compression ML model (which eliminates some of the redundant parameters) can be trained on **input/output only,**
whereas supervised ML needs input/output pairs and may **overfit**.



# Dimensionality reduction: overview

**Dimensionality reduction models are trained by reproducing their input vector.**

Step 1: train compression ML models




Size (i.e. number of dimension) is explicitly chosen before applying the training

Step 2: train supervised model on compressed vector

# Other uses of dimensionality reduction

- **Visualization:**
  Reduce the input space to a **two-dimensional** latent space, and plot the data in 2D

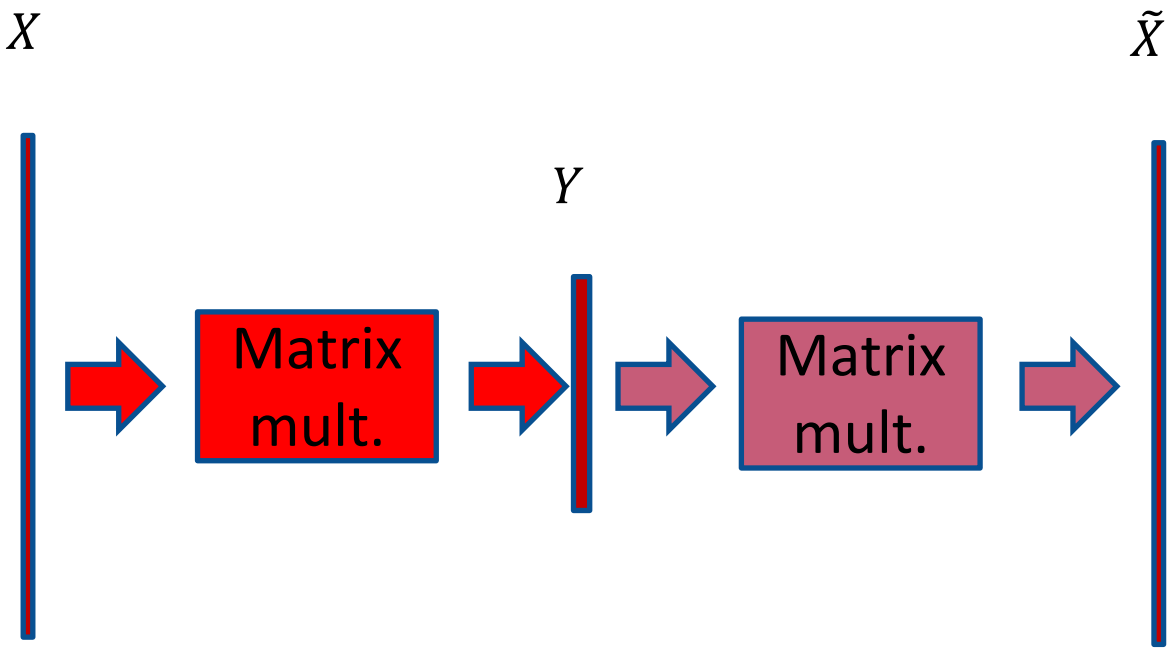
- **Optimization/control** on reduced number of parameters

Algorithms for dimensionality reduction:
**PCA**, ICA, NMF, **auto-encoders**, …

# Principal component analysis (PCA)

- In PCA, the components of the reduced vector *Y* are obtained by a **linear combination** of the original components *X* (i.e. a matrix multiplication)

- The matrix is chosen so as to **minimize the reconstruction error.**




**Reconstruction error:**

$$\mathcal{E} = \sum_{i=1}^{N} \sum_{j=1}^{D} (X_{ij} - \tilde{X}_{ij})^2$$

Sum over examples in the dataset

Sum over features

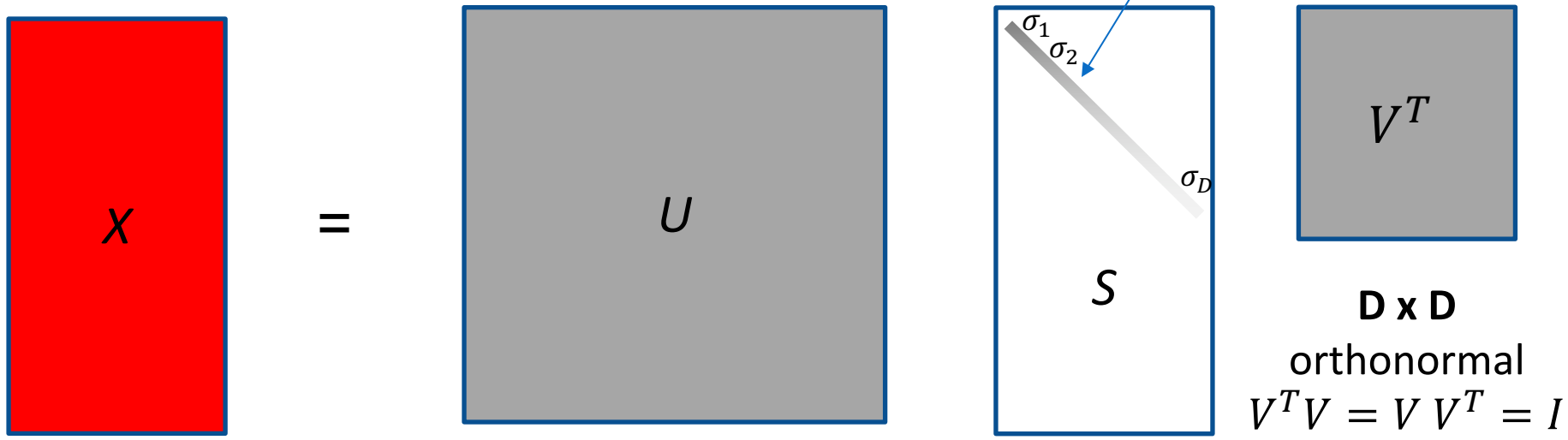
# Singular value decomposition (SVD)

Any matrix X can be decomposed as $X = U\,S\,V^T$

Singular values $\sigma_i \geq 0$
sorted in decreasing order

D: number of features

N: number of examples in the dataset




**N x D**

**N x N**
orthonormal
$U^T U = I$

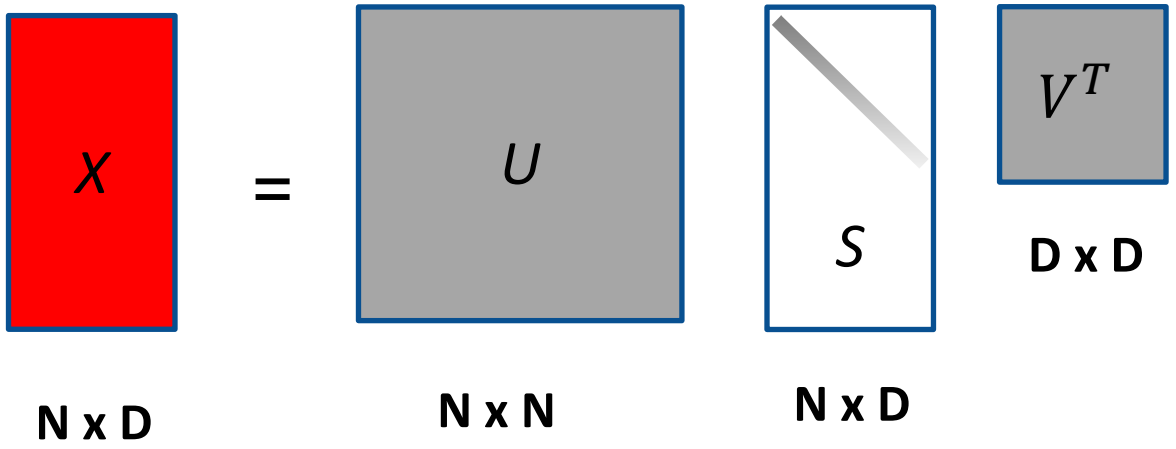
**N x D**
zero, except
on the "diagonal"

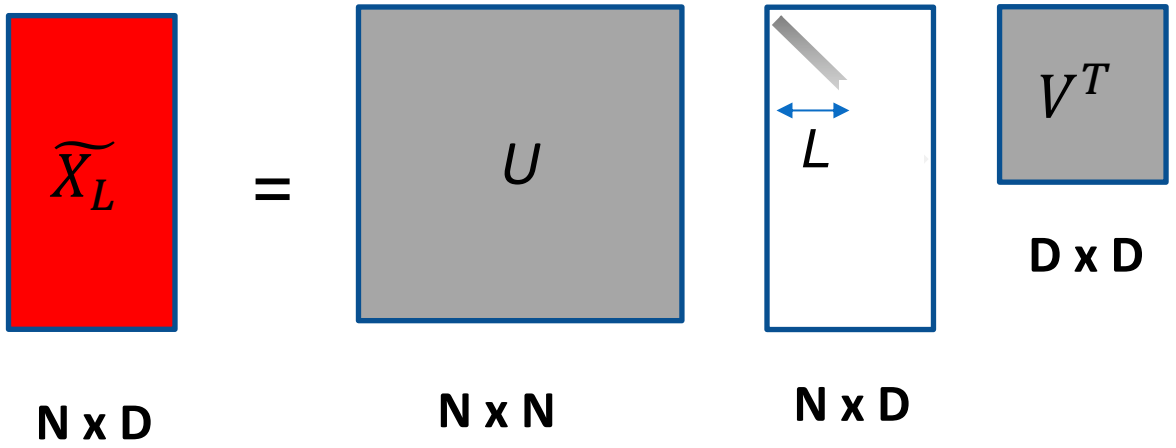
**D x D**
orthonormal
$V^T V = V\,V^T = I$

# Truncated SVD

**Approximation:**
only retain the L
highest singular values
(with L < D)

(Set the other ones to zero)

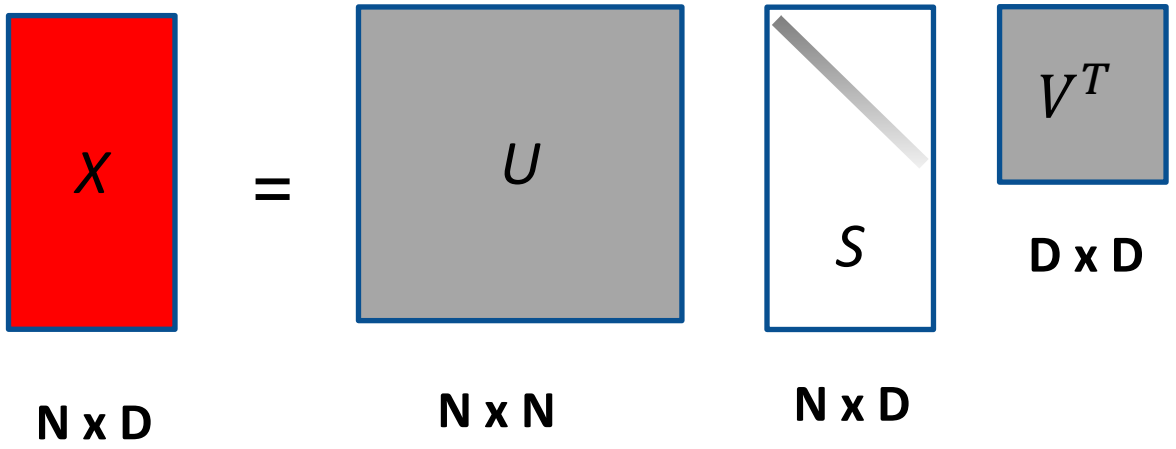


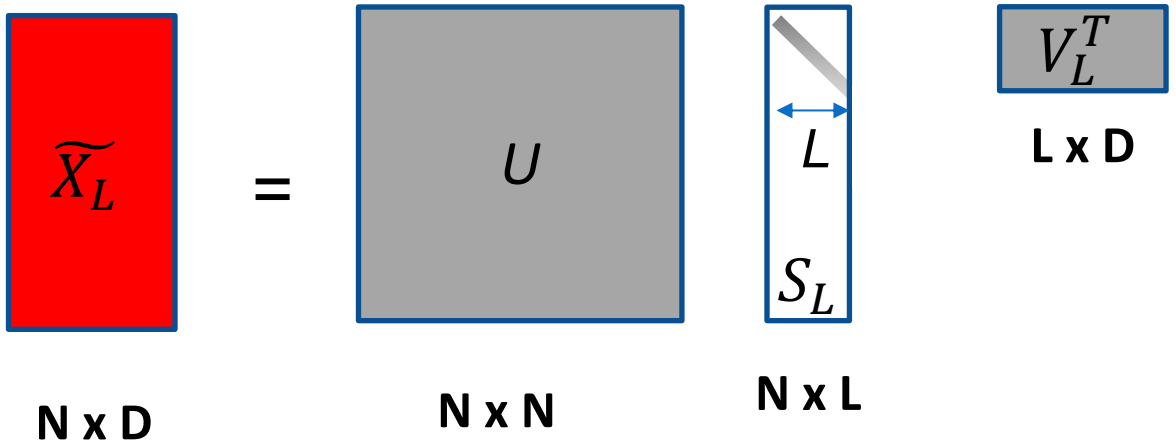
# Truncated SVD

**Approximation:**
only retain the L
highest singular values
(with L < D)

$$X = U\,S\,V^T$$




$$\widetilde{X_L} = U\,S_L\,V_L^T$$

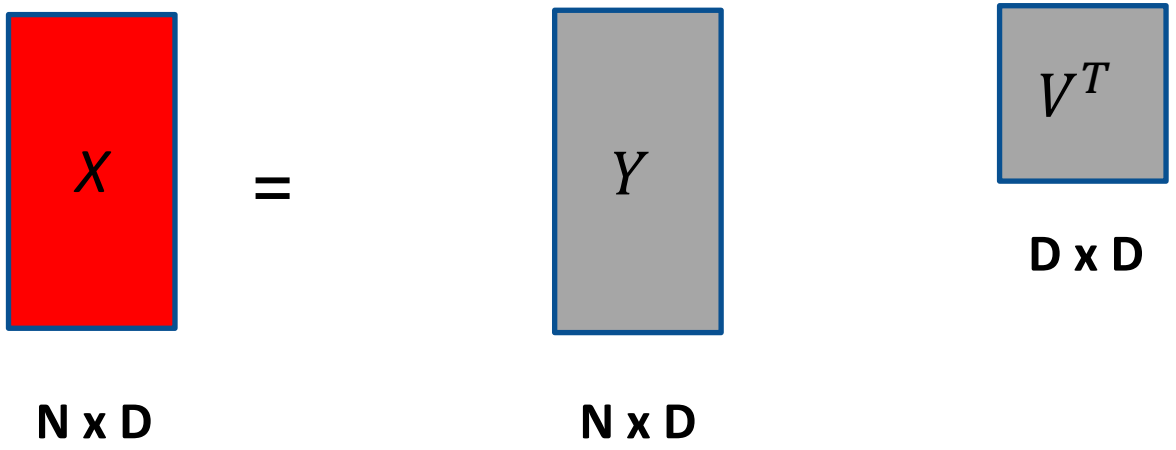



Contains the $L$ first lines of $V^T$

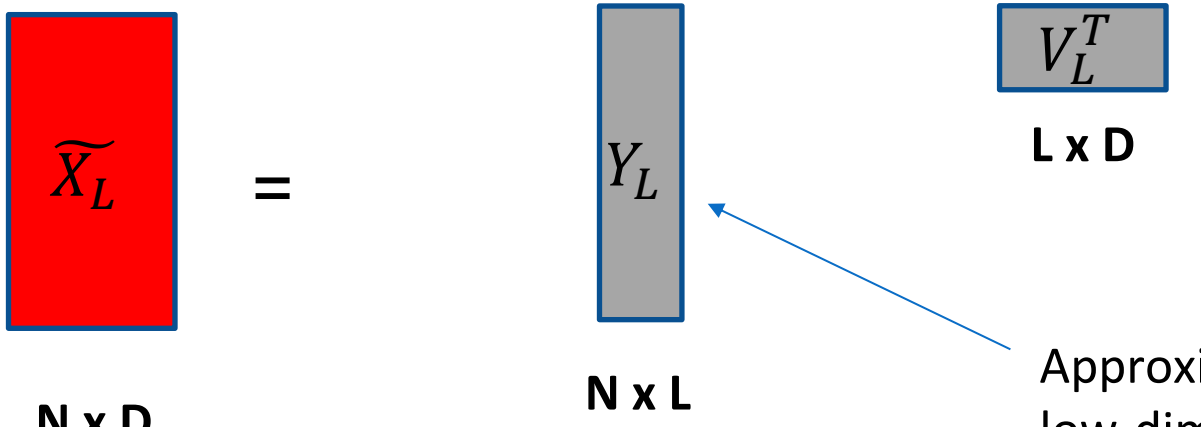
# Truncated SVD

**Approximation:**
only retain the L
highest singular values
(with L < D)

$$X = Y\,V^T$$




$$\widetilde{X_L} = Y_L\,V_L^T$$




Contains the $L$ first lines of $V^T$

It can be shown that:

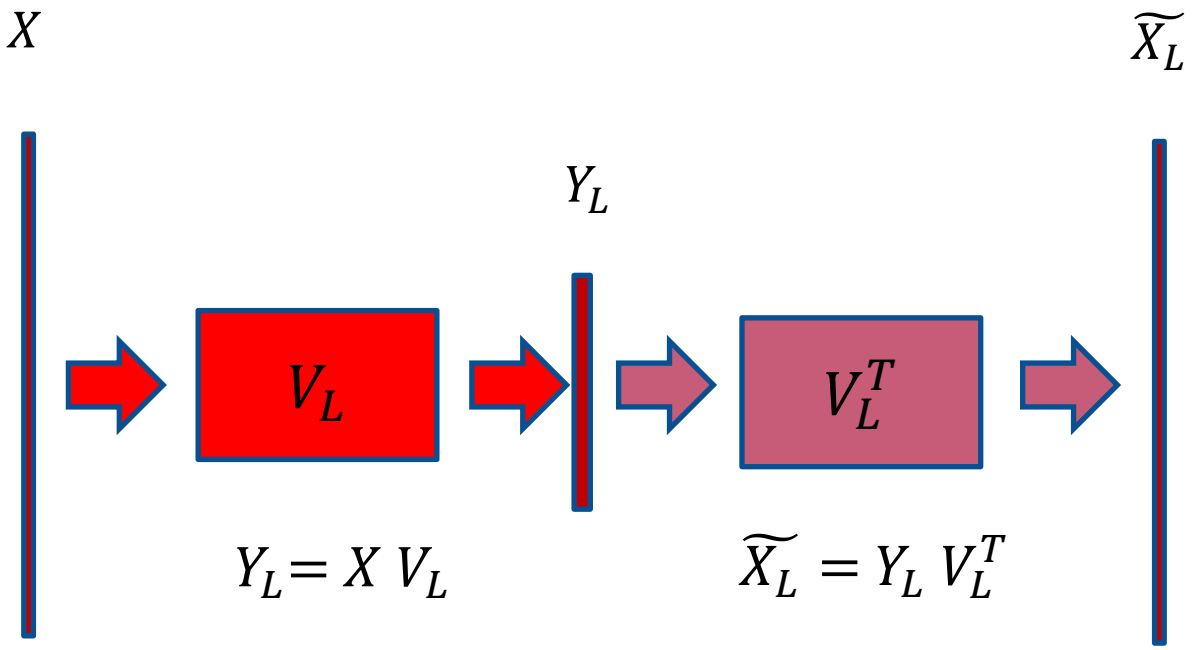
$$Y_L = X\,V_L$$

Contains the $L$ first rows of $Y$

Approximate low-dimensional representation of X

# Projection onto principal components using truncated SVD

The matrix that we were looking for is $V_L$.




$V_L$ contains the $L$ first rows of $V$, which is given by the SVD

# Quantifying PCA: the explained variance

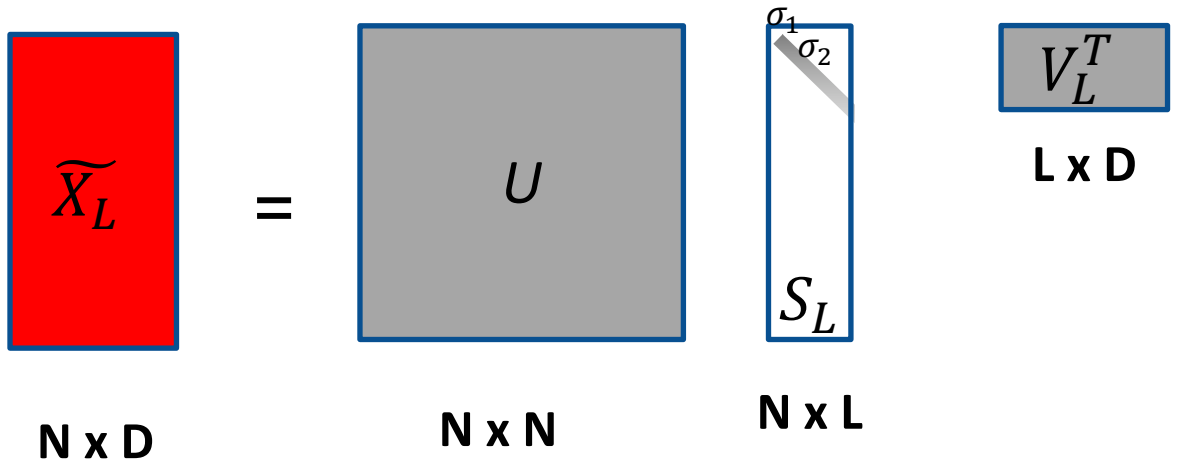
Variance of the reconstructed vector:
(assuming the mean of each component has been subtracted)

$$Var(\tilde{X}) = \frac{1}{N}\sum_{i=1}^{N} \tilde{X}_{i,1}^2 + \ldots + \frac{1}{N}\sum_{i=1}^{N} \tilde{X}_{i,D}$$

Variance of first component

Variance of last component

$$\begin{aligned} Var(\tilde{X}_L) &= Tr(\tilde{X}_L^T \tilde{X}_L) \\ &= Tr(U S_L V_L^T V_L S_L^T U^T) \\ &= Tr(U S_L S_L^T U^T) \\ &= Tr(S_L S_L^T U^T U) \\ &= Tr(S_L S_L^T) \\ &= \sum_{i=1}^{L} \sigma_i^2 \end{aligned}$$



# Quantifying PCA: the explained variance

Variance of the original vector: $$Var(X) = \sum_{i=1}^{D} \sigma_i^2$$

Explained variance: $$Var(\tilde{X}_L) = \sum_{i=1}^{L} \sigma_i^2$$

L < D

- PCA attempts to find the reconstructed vector that **recovers as much as possible of initial variance.**

(i.e. it tries to project the initial vector onto the direction where the data varies the most)

- Because the $\sigma_i$ are sorted in decreasing order, the first few terms in the sum may already recover most of the initial variance.

# How to use PCA in practice: scikit-learn

## Initialization

*class* `sklearn.decomposition.` **`PCA`**(*n_components=None*, ***, *copy=True*, *whiten=False*, *svd_solver='auto'*, *tol=0.0*, *iterated_power='auto'*, *random_state=None*) ¶ [source]

**n_components : *int, float or 'mle', default=None***

Number of components to keep. if n_components is not set all components are kept:

```
n_components == min(n_samples, n_features)
```

## Usage

**`fit_transform`**(*X*, *y=None*) [source]

Fit the model with X and apply the dimensionality reduction on X.

| | |
|---|---|
| **Parameters:** | **X : *array-like of shape (n_samples, n_features)***<br>Training data, where n_samples is the number of samples and n_features is the number of features.<br><br>**y : *Ignored*** |
| **Returns:** | **X_new : *ndarray of shape (n_samples, n_components)***<br>Transformed values. |

# How to use PCA in practice: pytorch

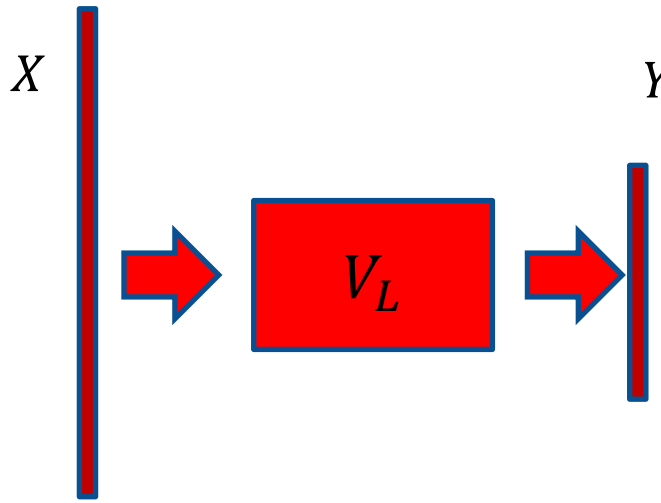
`torch.pca_lowrank(A, q=None, center=True, niter=2)` [SOURCE]

Performs linear Principal Component Analysis (PCA) on a low-rank matrix, batches of such matrices, or sparse matrix.

This function returns a namedtuple `(U, S, V)` which is the nearly optimal approximation of a singular value decomposition of a centered matrix $A$ such that $A = Udiag(S)V^T$.

- $U$ is m x q matrix
- $S$ is q-vector
- $V$ is n x q matrix

- `matmul(A, V[:, :k])` projects data to the first k principal components




$Y_L = X\,V_L$

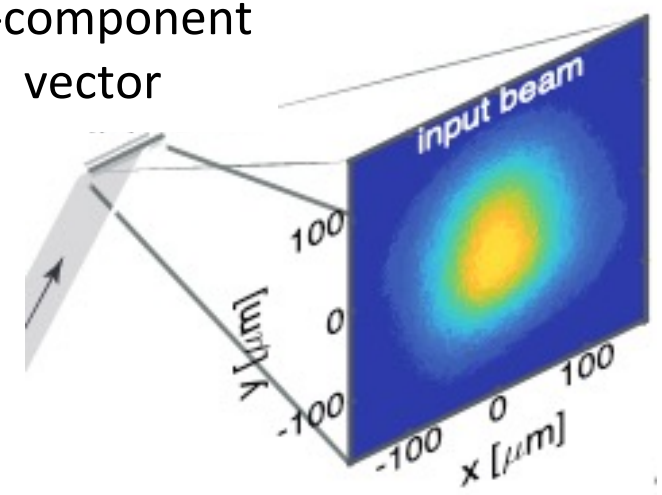
$V_L$ contains the $L$ first rows of $V$, which is given by the SVD

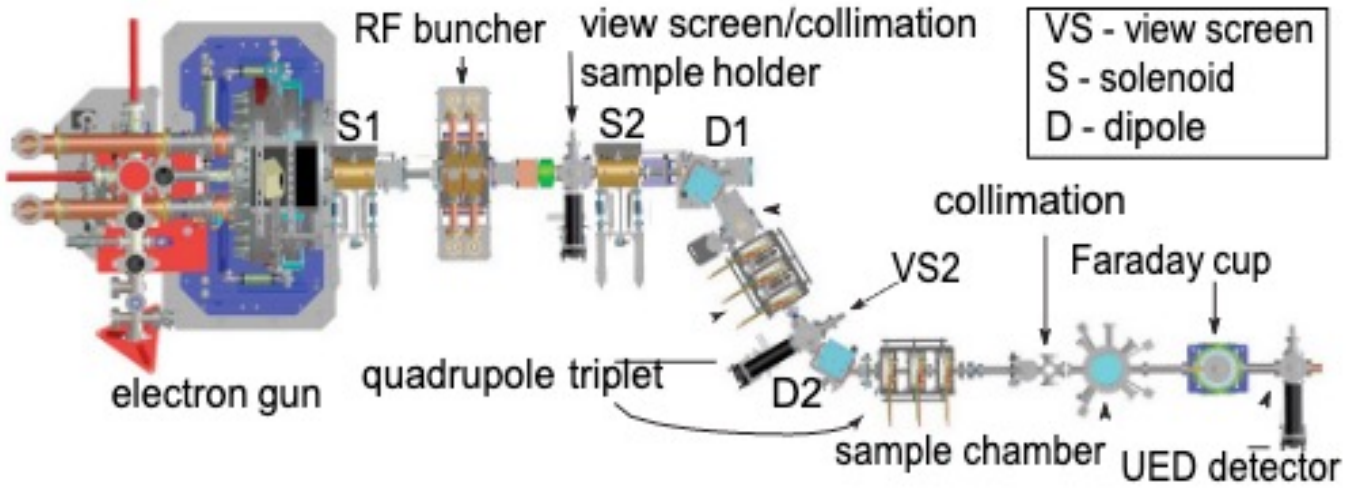
# Example: PCA in accelerator physics

*A. Scheinker et al., arxiv:2102.10510 (2021)*

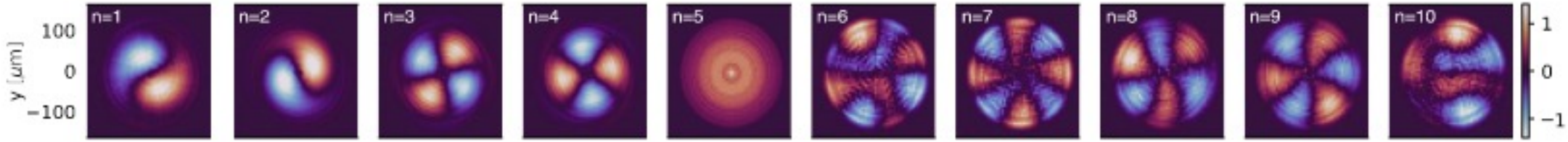
Parameterization of the 2D transverse profile of a beam
(for the purpose of reconstructing the unknown input profile)

15-component vector



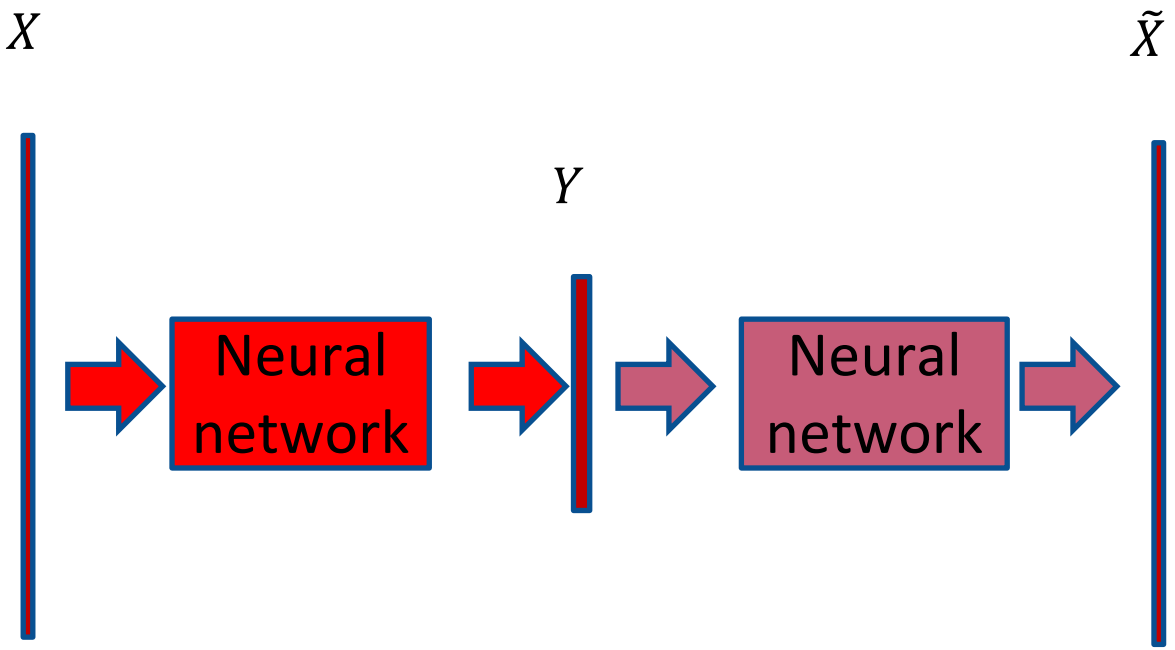

The first 10 components:



# Auto-encoders

- Instead of using matrix multiplications (like PCA), auto-encoders use **neural networks**.

- The neural networks are trained (with backprop) to reduce the reconstruction error. (i.e. the reconstruction error is the loss function)



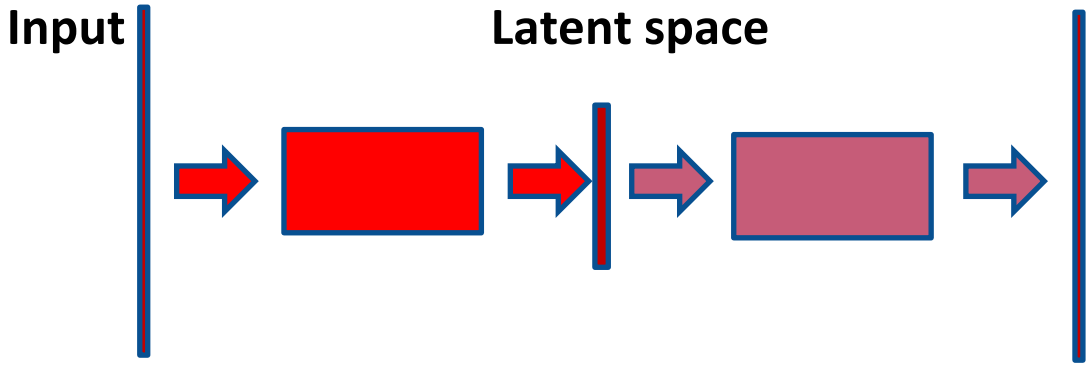

**Reconstruction error:**

$$\mathcal{E} = \sum_{i=1}^{N} \sum_{j=1}^{D} (X_{ij} - \tilde{X}_{ij})^2$$

Sum over examples in the dataset

Sum over features

# Comparison between auto-encoders and PCA




**PCA:**

- The latent features can only be **linear combinations** of input features.
- Only one tuning parameter: dimension of the latent space

**Auto-encoder:**

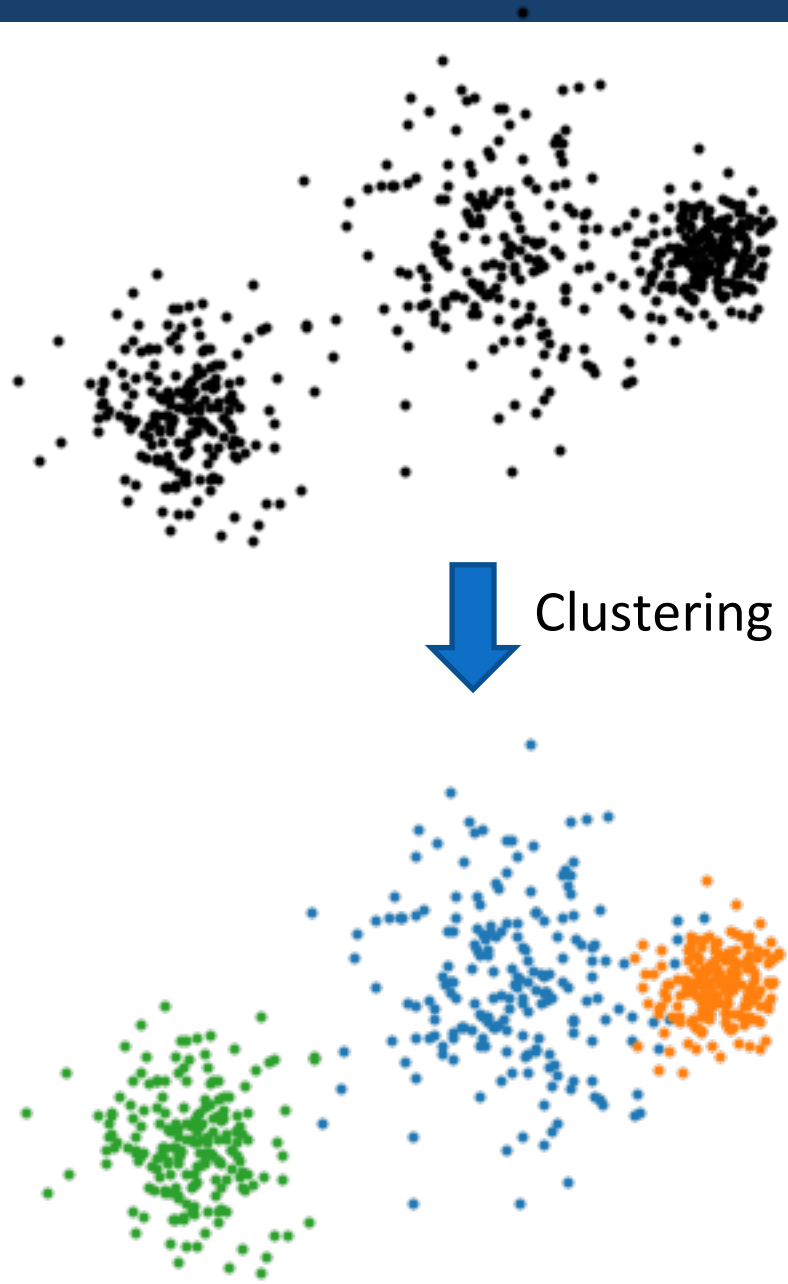
- The latent features can be **complex non-linear combinations** of input features.
- Many tuning parameters:
  - Architecture: fully-connected, CNN, etc.
  - Number of layers, neurons, etc.
  - Training parameters: epochs, learning rate, etc.

# Outline

- Dimensionality reduction
  - PCA
  - Auto-encoders
- Clustering
  - K-means
  - DBSCAN
- Anomaly detection

# Clustering

- Aim: Find **clusters of “similar” data points**, within the dataset
  - 2D/3D: can be done by eye, by a human
  - N-D: impossible for a human
- Examples:
  - News: group news articles into ”trending topics”
  - Physics: group physical observations into different “physical regimes”
- The clusters are **not known in advance**.
  This can be a tool for **discovery**.



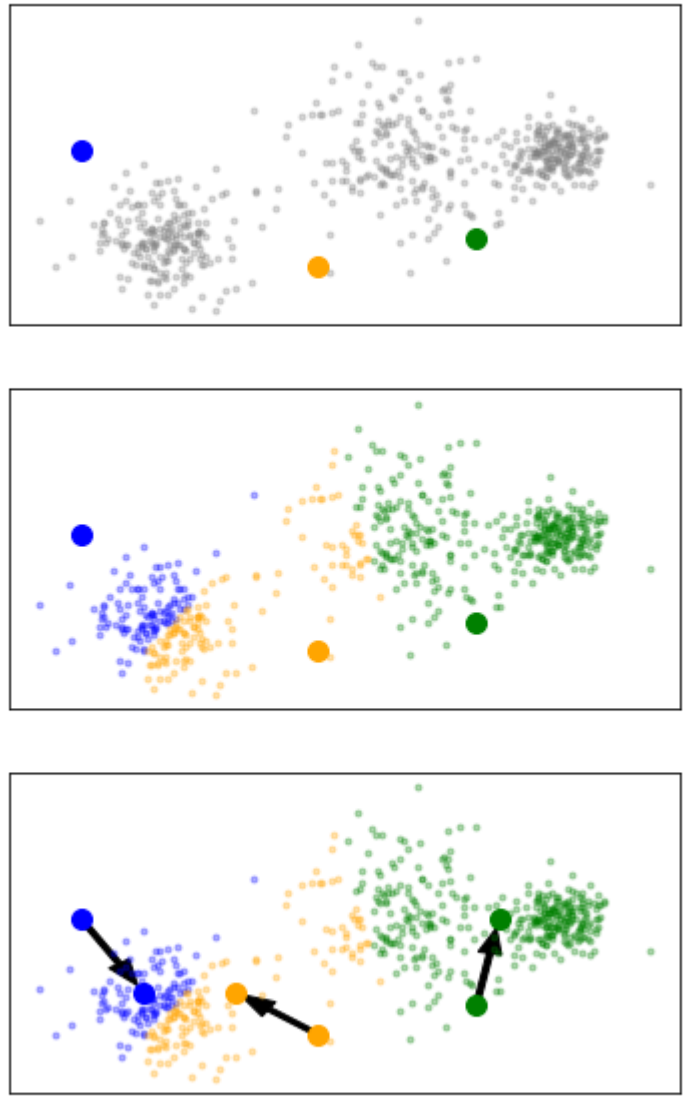
# K-means: algorithm

Choose *k* initial points (e.g. randomly), called "centroids"

Then perform several iterations of the following steps:

- Attribute each data points to the nearest centroid
- Move the centroid to the barycenter (or "mean") of its data points

k=3, iteration **#1**



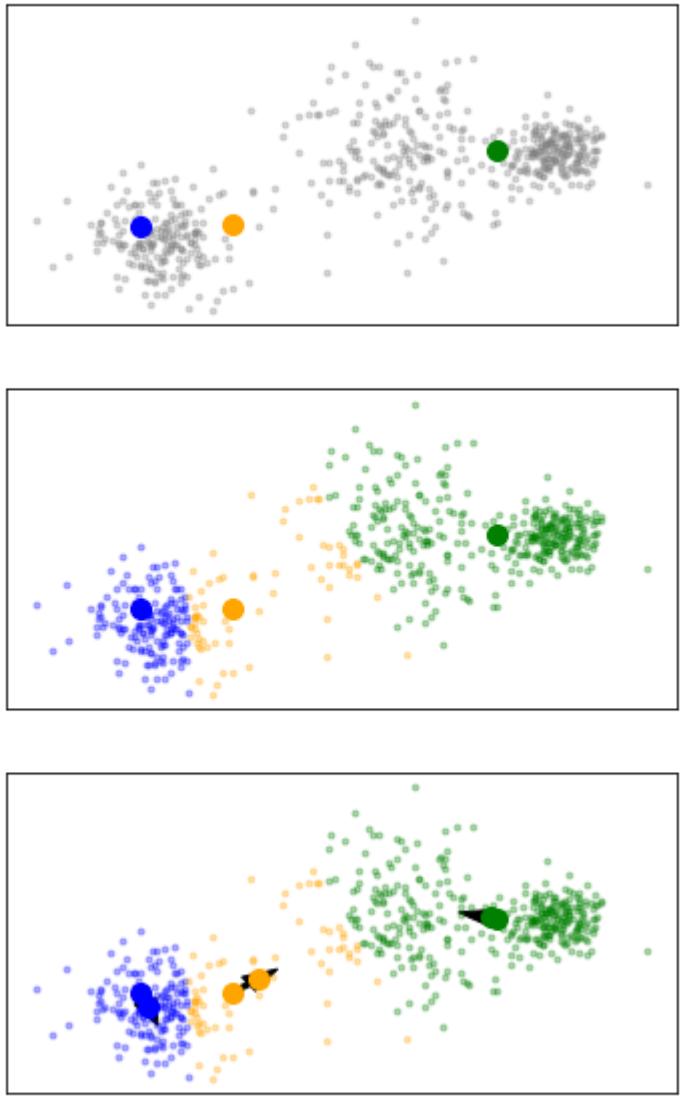
# K-means: algorithm

Choose *k* initial points (e.g. randomly), called "centroids"

Then perform several iterations of the following steps:

- Attribute each data points to the nearest centroid
- Move the centroid to the barycenter (or "mean") of its data points

k=3, iteration #2



# K-means: algorithm

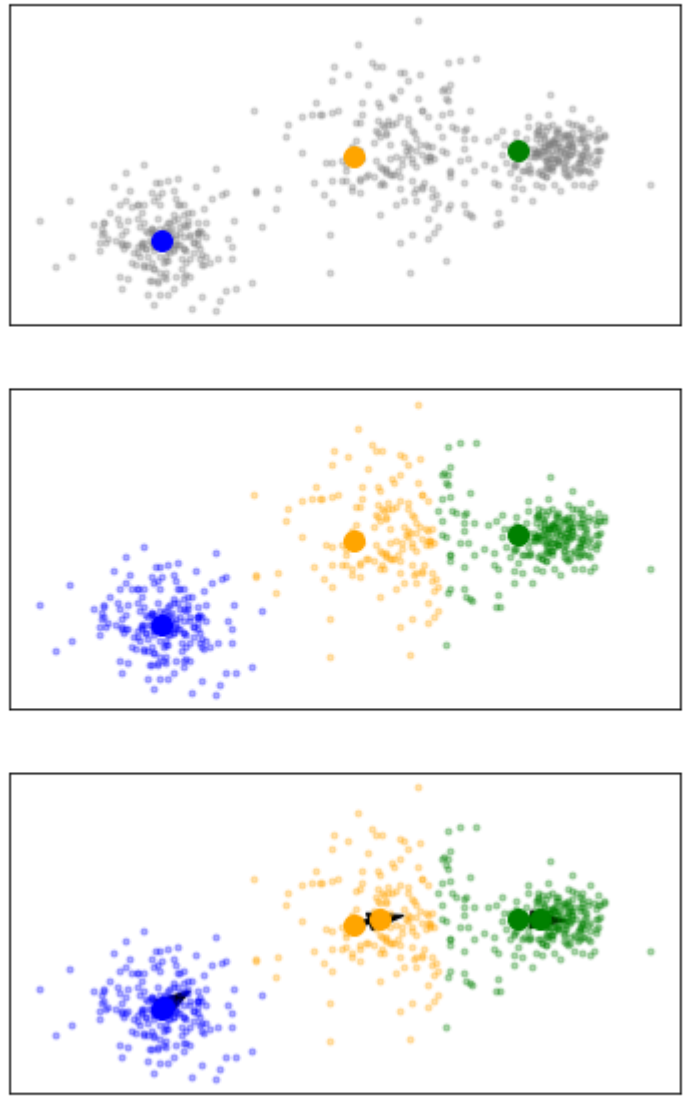
Choose *k* initial points (e.g. randomly), called "centroids"

Then perform several iterations of the following steps:

- Attribute each data points to the nearest centroid
- Move the centroid to the barycenter (or "mean") of its data points

The algorithm stops when the centroids do not move significantly from one iteration to the next.

k=3, iteration **#5**



# K-means: practical consideration

- User needs to **pick the number of clusters** *k* in advance
- By construction, k-means can only find **convex** clusters
- Can be sensitive to **outliers**
  Results may depend on the positions of the initial centroids

A few k-means prediction:
(from scikit-learn.org)

# K-means: scikit-learn interface

*class* `sklearn.cluster.`**`KMeans`**(*n_clusters=8, *, init='k-means++', n_init=10, max_iter=300, tol=0.0001, precompute_distances='deprecated', verbose=0, random_state=None, copy_x=True, n_jobs='deprecated', algorithm='auto'*)

[source]

K-Means clustering.

Read more in the User Guide.

**Parameters:**

**n_clusters : *int, default=8***

The number of clusters to form as well as the number of centroids to generate.

**init : *{'k-means++', 'random'}, callable or array-like of shape (n_clusters, n_features), default='k-means++'***

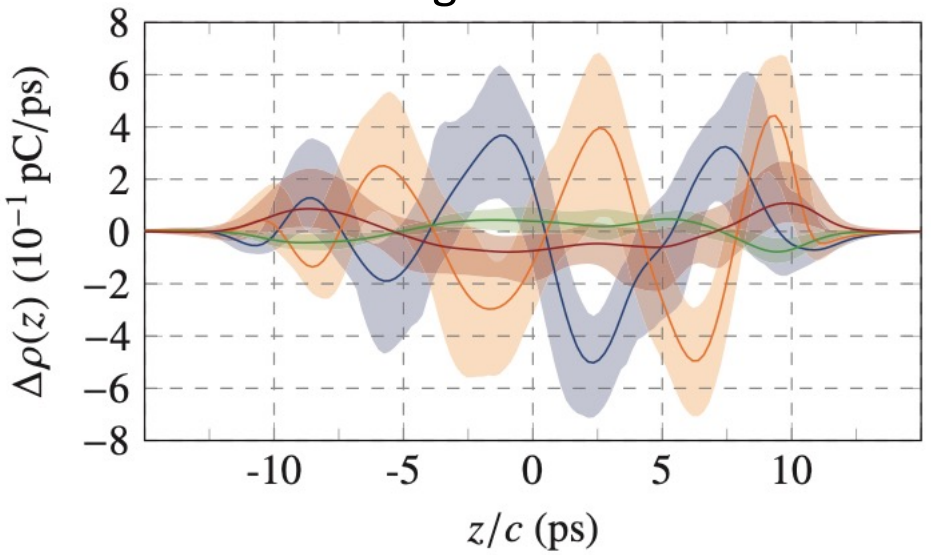
Method for initialization:

# K-means: example in accelerator physics

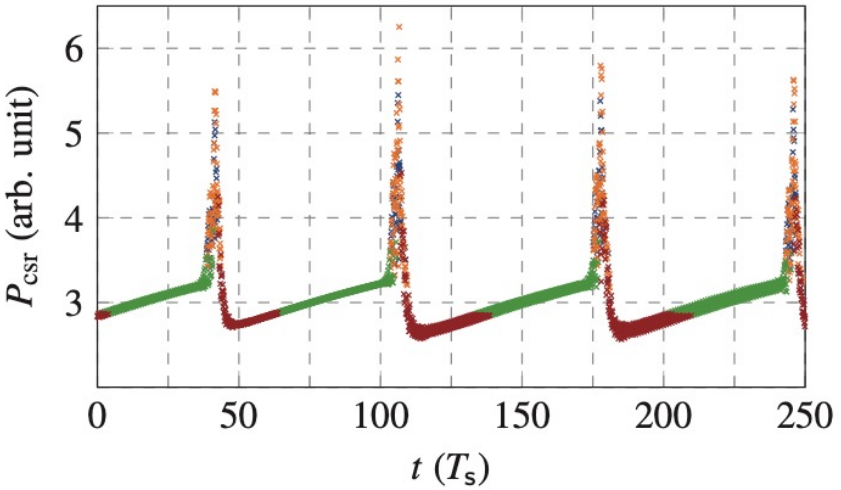
T. Boltz et al., “Studies of longitudinal dynamics in the micro-bunching instability using machine learning”, IPAC2018, Vancouver, doi:10.18429/JACoW-IPAC2018-THPAK030

- Simulations of the KARA storage ring (Karlsruhe Research Accelerator)
- The beam can interact with its own CSR radiation and develop micro-bunching instability
- Input to k-means (k=4): discretized longitudinal bunch profile
- k-means extracts 4 regimes

Typical beam density fluctuations for each regime




Transition between different regimes during a simulation:

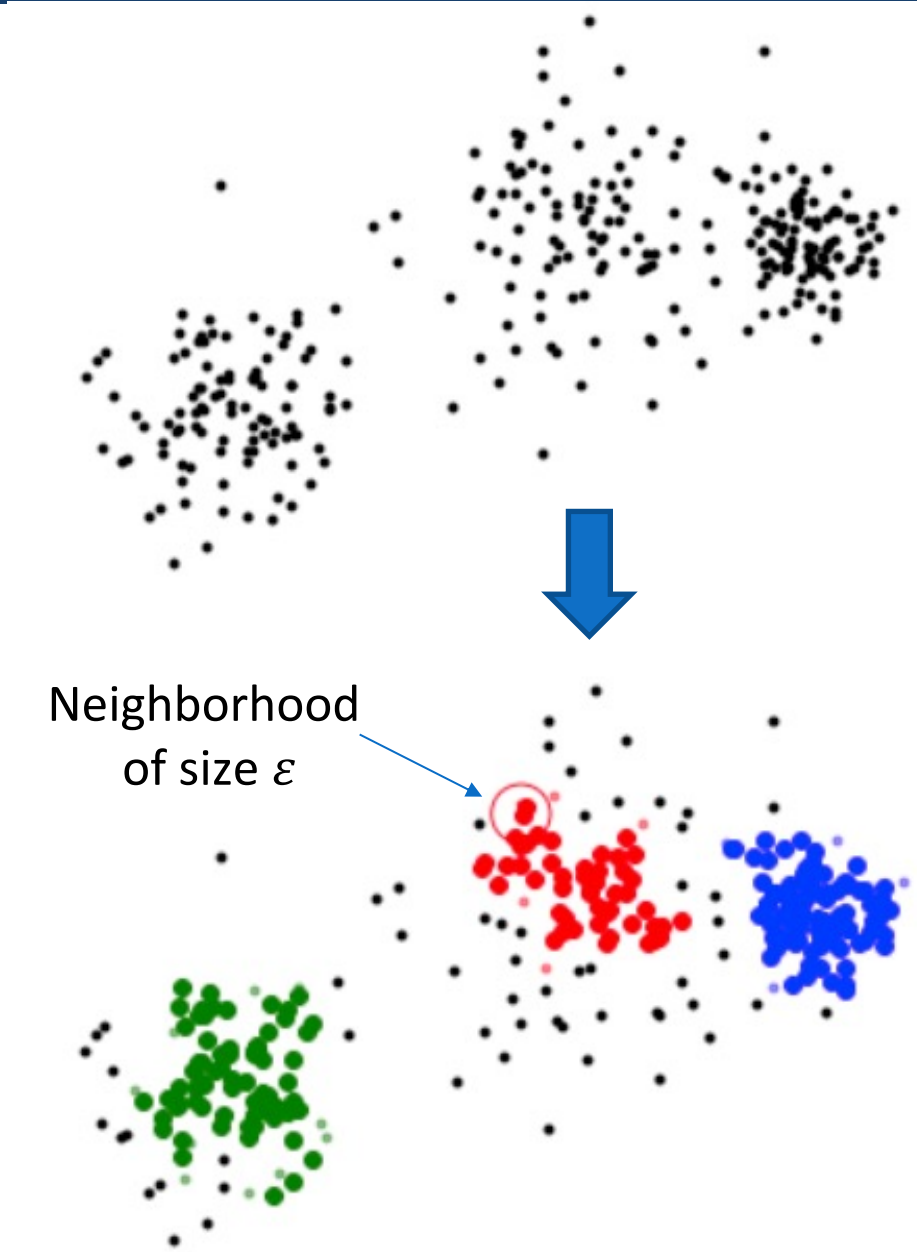


# DBSCAN

Pick a **distance** $\varepsilon$ and a minimum **number of neighbors** $n_{min}$

- Points that have at least $n_{min}$ neighbors within a distance $\varepsilon$ are considered **"core points"**.
- Core points that are within a distance $\varepsilon$ of each other are part of the same cluster.
- None-core points that are within a distance $\varepsilon$ of a a core points are also in the same cluster.
- Other points are considered **outliers**.

Intuitively: find **areas of high density** separated by areas of low density.



# DBSCAN: practical considerations

- Users does not need to specify the number of cluster *k* in advance **but** it is crucial to specify a proper $\varepsilon$ and/or $n_{min}$
- Clusters can be **any shape** ; not necessarily convex
- Ability to predict **outliers** (i.e. do not belong to any cluster)

A few k-means prediction:
(from scikit-learn.org)

# DBSCAN: scikit-learn interface

*class* `sklearn.cluster.` **`DBSCAN`**(*eps=0.5, *, min_samples=5, metric='euclidean', metric_params=None, algorithm='auto', leaf_size=30, p=None, n_jobs=None*) [source]

Perform DBSCAN clustering from vector array or distance matrix.

DBSCAN - Density-Based Spatial Clustering of Applications with Noise. Finds core samples of high density and expands clusters from them. Good for data which contains clusters of similar density.

Read more in the User Guide.

**Parameters:**

**eps : *float, default=0.5***

The maximum distance between two samples for one to be considered as in the neighborhood of the other. This is not a maximum bound on the distances of points within a cluster. This is the most important DBSCAN parameter to choose appropriately for your data set and distance function.

**min_samples : *int, default=5***

The number of samples (or total weight) in a neighborhood for a point to be considered as a core point. This includes the point itself.

# Outline

- Dimensionality reduction
  - PCA
  - Auto-encoders
- Clustering
  - K-means
  - DBSCAN
- Anomaly detection

# Anomaly detection: overview

**Idea:** automatically detect **“outliers”** in the data.
(i.e. data points that do not “look like” the rest of the data)

Applications:

- **For data analysis:**
  Automatically **clean-up** large amount of data (too large to be feasible by hand) e.g. faulty measurement devices in a large experimental campaign.

- **During operation:**
  Detect anomalous accelerator behavior, in order to raise an early warning (e.g. danger of damaging equipment, risk of having to stop the operation)

# Anomaly detection: definition

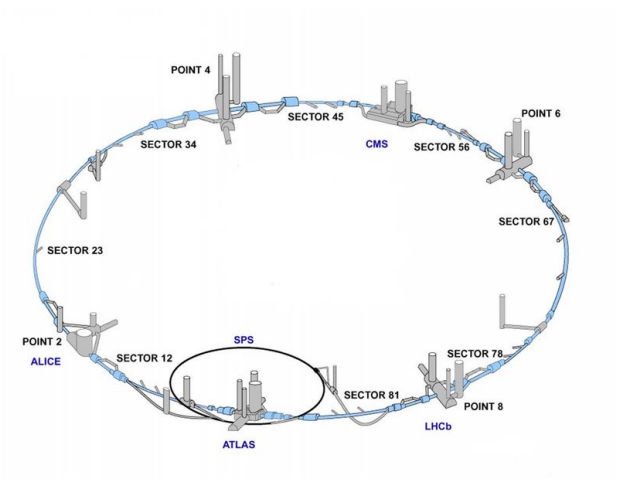
In ML terminology, “anomaly detection” is an **unsupervised** methods

- Applies when it is **difficult** to label anomalous behavior.
  (e.g. too time-consuming for a human to go through the whole data)

- If **most of the data** can be **labeled** (”normal” vs “anomalous”):
  It is better to use **supervised learning** (classification) than anomaly detection methods. (e.g. logistic regression, decision trees)

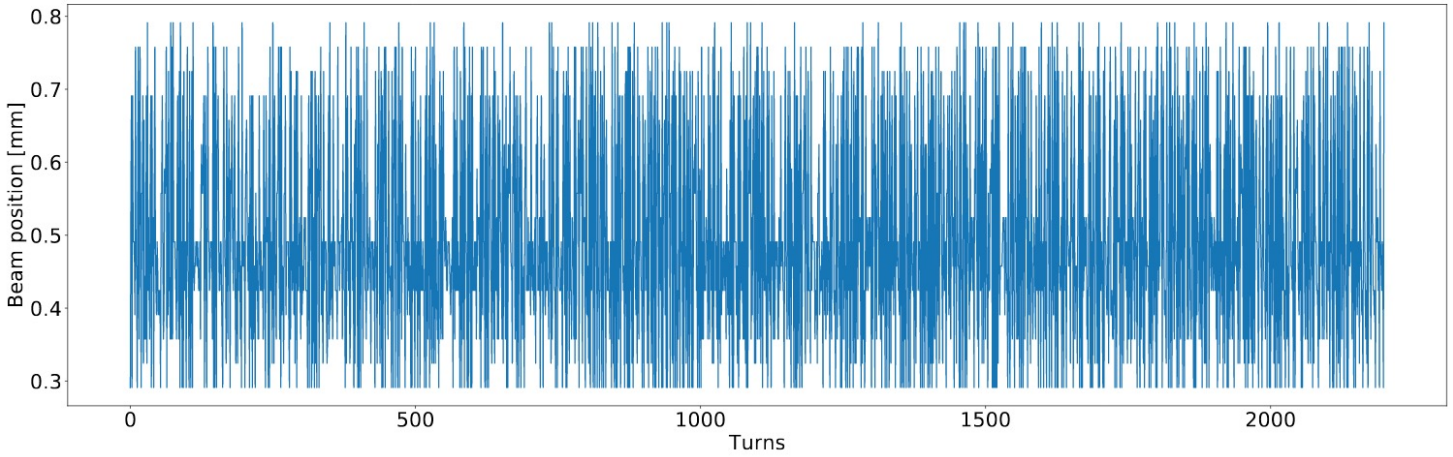
# Anomaly detection example: detecting faulty BPM in LHC

E. Fol, CERN thesis 2017-336 (2017)




Beam Position Monitors (BPM) record the position of the beam at each turn, for 512 different locations on the ring

→ Timeseries data at each of the 512 locations



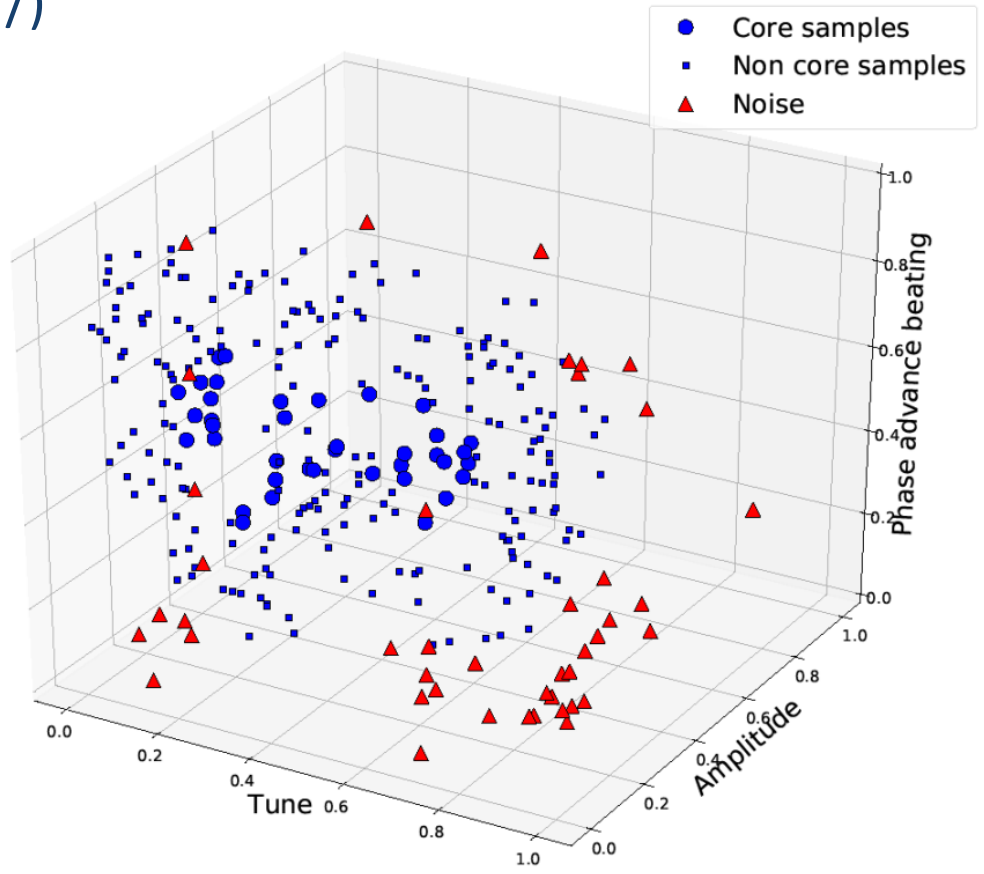

However, BPMs sometimes produce **incorrect readings.**
Faulty readings are **identifiable by a human**. Various occurences:
- high noise
- unphysical spikes
- incorrect tune frequency

but it is tedious to verify every single BPM by eye.

# DBSCAN

E. Fol, “Detection of faulty Beam Position Monitors”, ICFA Beam Dynamics Mini-Workshop: Machine Learning Applications for Particle Accelerators, 2018 (https://indico.fnal.gov/event/16327/)

- Extracted 3 representative features from each BPM timeseries
- Ran DBSCAN on this 3-dimentional input to detect **outliers**

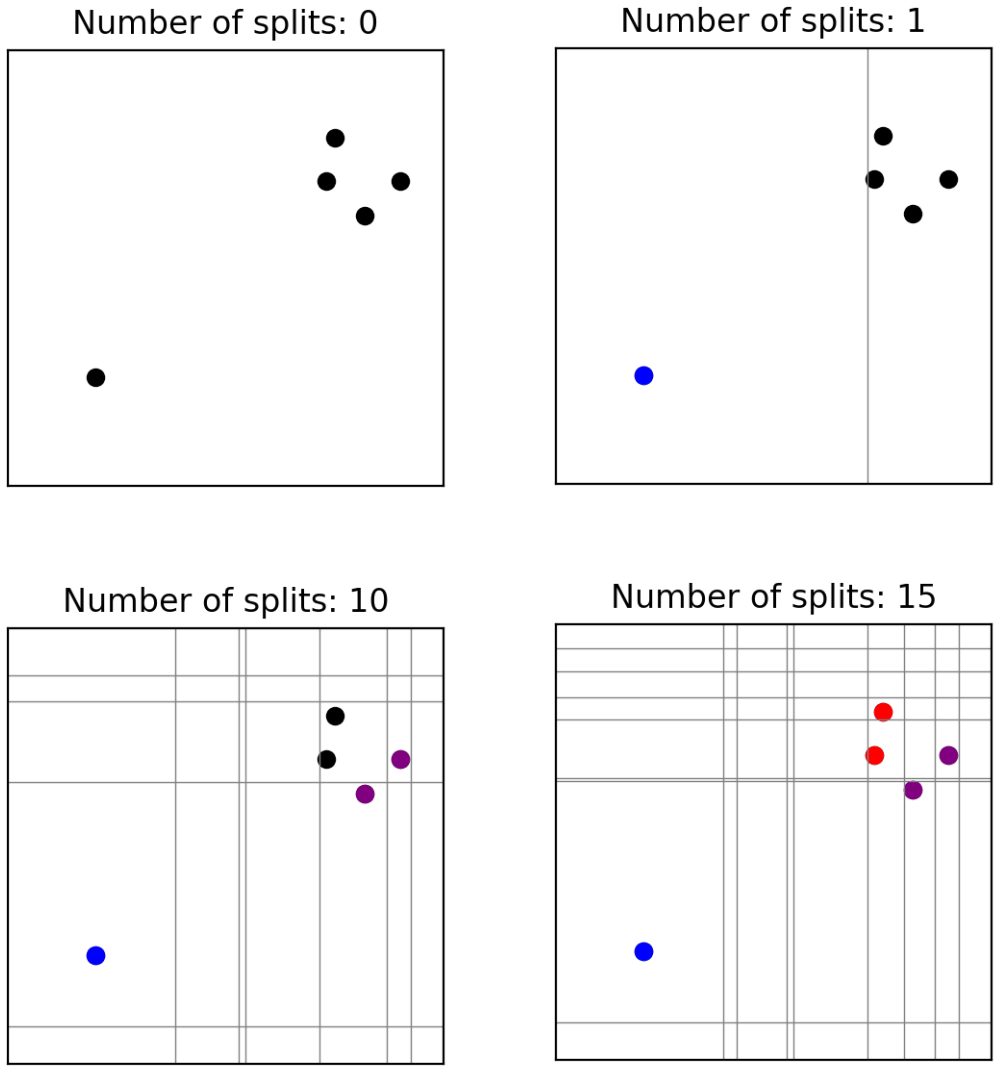



ε= 0.3, MinPts = 80

# Isolation forest

Iteratively:

- Randomly select a **feature**
- Randomly select a **split value** between the min and max of that feature




For each data point, record the **number of splits** that were need to isolated it from the others.

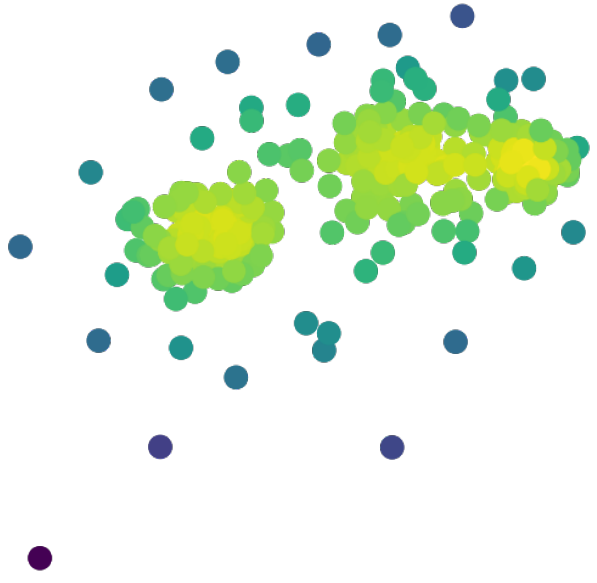
(Outliers typically need fewer splits.)

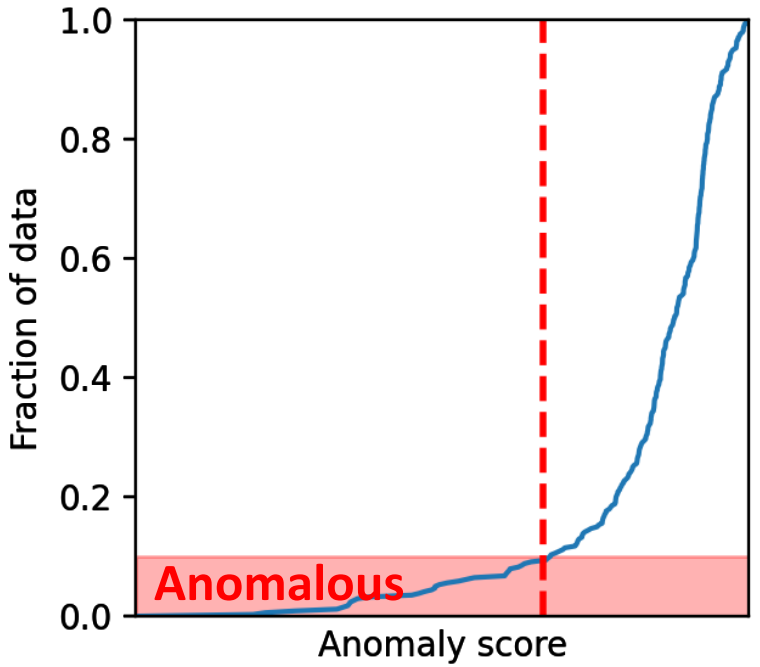
+ **Repeat from scratch many times.**

Each data point's **score** is the **average** number of splits needed to isolate it.

# Isolation forest: setting the decision threshold

- The anomaly score (average number of splits to isolate a point) can have arbitrary values
- One needs to set a **threshold** that determines which points are labeled as **anomalous**.
- This is often set by imposing that a given **fraction of the data** should be considered anomalous (The user needs to provide a "contamination" fraction.)





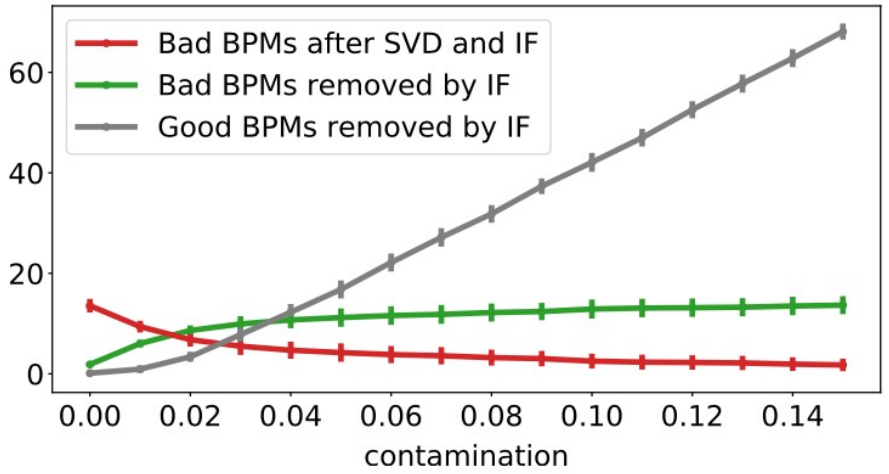
# Isolation forest: accelerator example

E. Fol et al. ”Detection of faulty beam position monitors using unsupervised learning”, PRSAB 23, 102805 (2020)

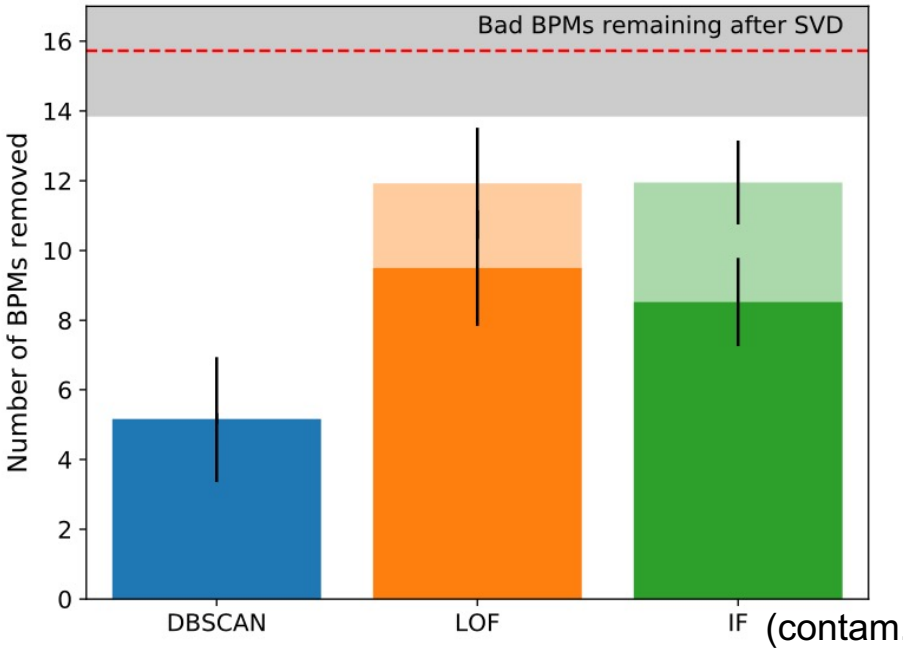
- In order to **benchmark** the method: generated **simulated** BPMs signals and **added anomalies on purpose** to some of them. (i.e. ground truth is known in this case)

- Pre-cleaned the data with an SVD-based method and extracted 3 features from each BPM signal

- Ran Isolation Forest for several values of the contamination parameter: **Trade-off** between removing good BPMs and missing bad BPMs



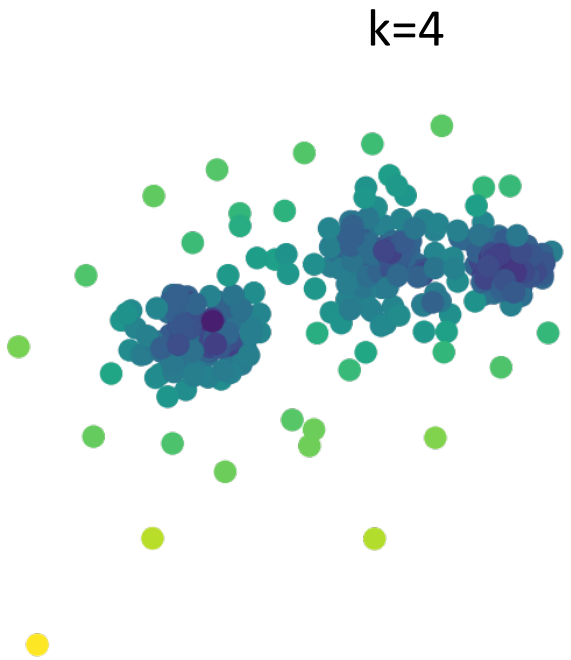

IF = Isolation Forest

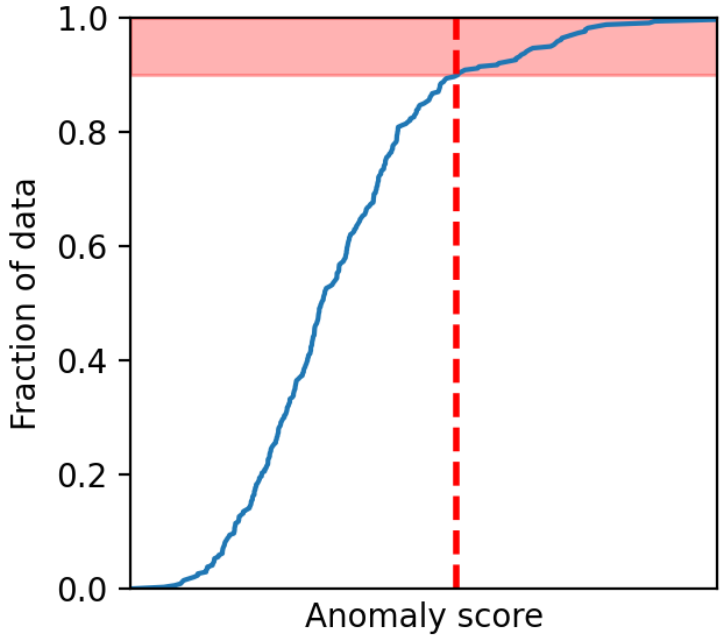



(contam. = 0.01)

# K-NN distance

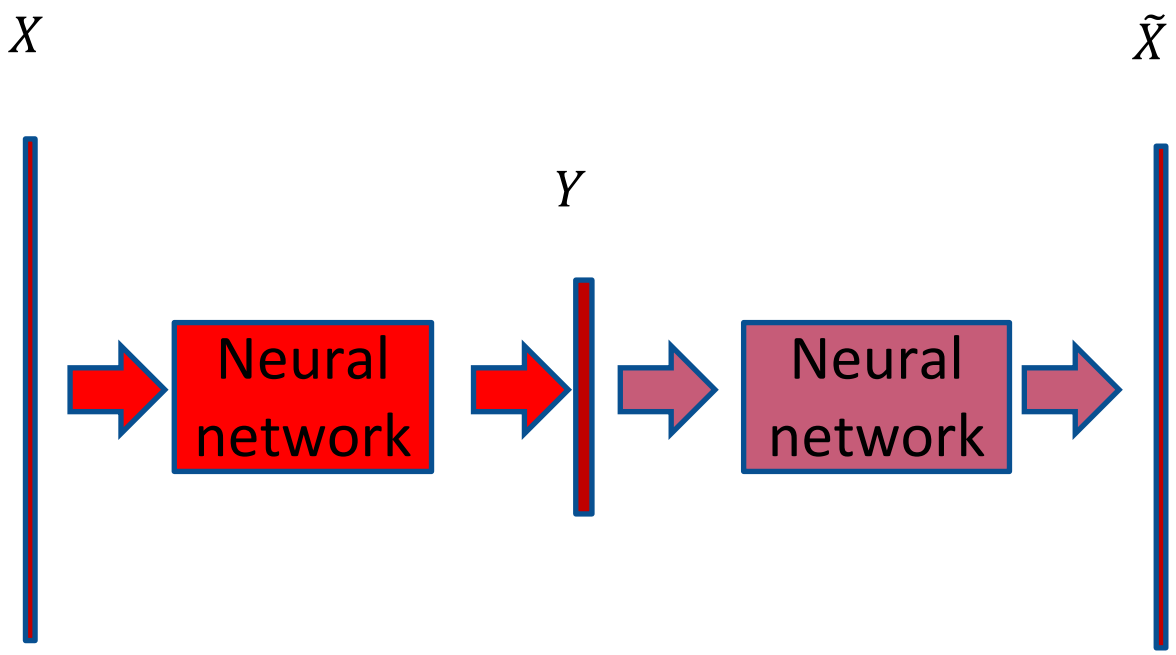
- Anomality score = distance from *k*th nearest neighbor (in this case, a **high** score indicates a likely outlier)
- Motivation for using $k > 1$:
  an outlier may occasionally be close to another outlier by chance.
- Again, the user needs to set a **threshold** on the anomaly score (e.g. based on the expected contamination fraction)




Other method based on distances to neighbors: Local Outlier Factor (LOF)

# Auto-encoders for anomaly classification




Evaluate the **reconstruction error** for each data point

$$\mathcal{E}_i = \sum_{j=1}^{D} (X_{ij} - \tilde{X}_{ij})^2$$

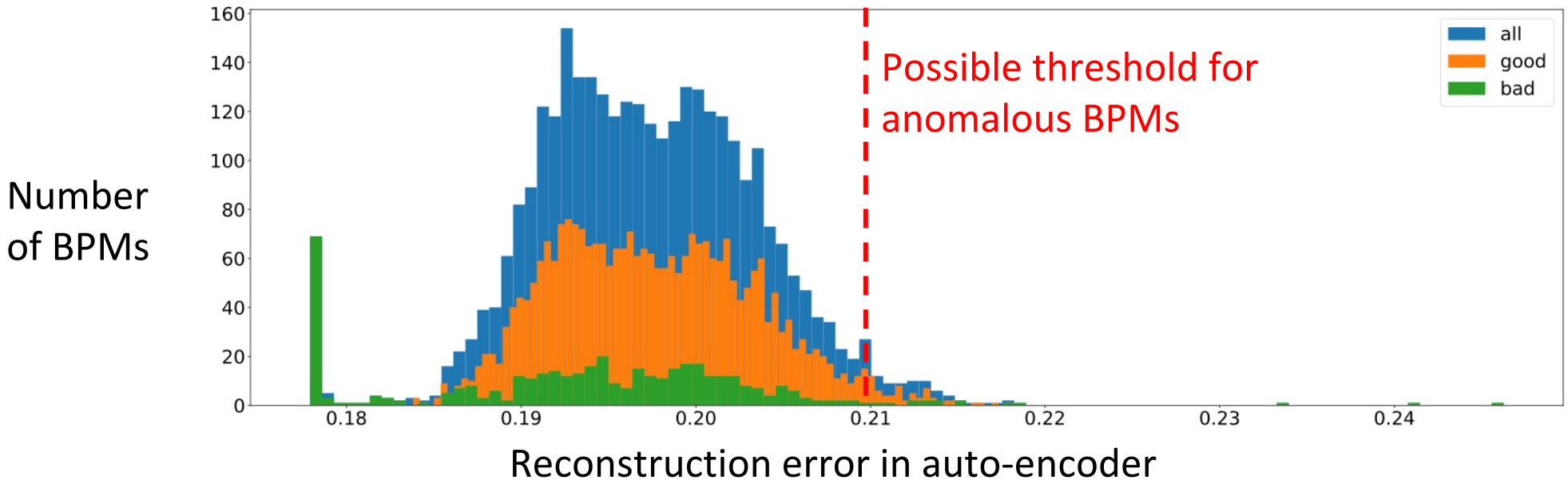
use it as the data points's **anomaly score** + set a **threshold** to determine which point should be considered anomalous

**Intuition:** Outliers are rare/inexistent in the dataset that was used to train the auto-encoder. Therefore, the auto-encoder has lower accuracy when reconstructing outliers.

# Auto-encoders: accelerator examples

## E. Fol, CERN thesis 2017-336 (2017)

- Evaluation on real, experimental data (no ground truth available)
  BPMs were labeled as anomalous (”bad”) or normal (“good”) by another method (SVD clean)
- Mixed results:
  - Some anomalous BPMs give low reconstruction error
  - But manually inspecting the BPMs that have high reconstruction error reveals anomalous BPMs that were not detected by SVD clean.



# Any questions?

- Dimensionality reduction
  - PCA
  - Auto-encoders
- Clustering
  - K-means
  - DBSCAN
- Anomaly detection