# LESSON 18

## Feature engineering Part 2: Dimensionality reduction. Feature Selection and Extraction

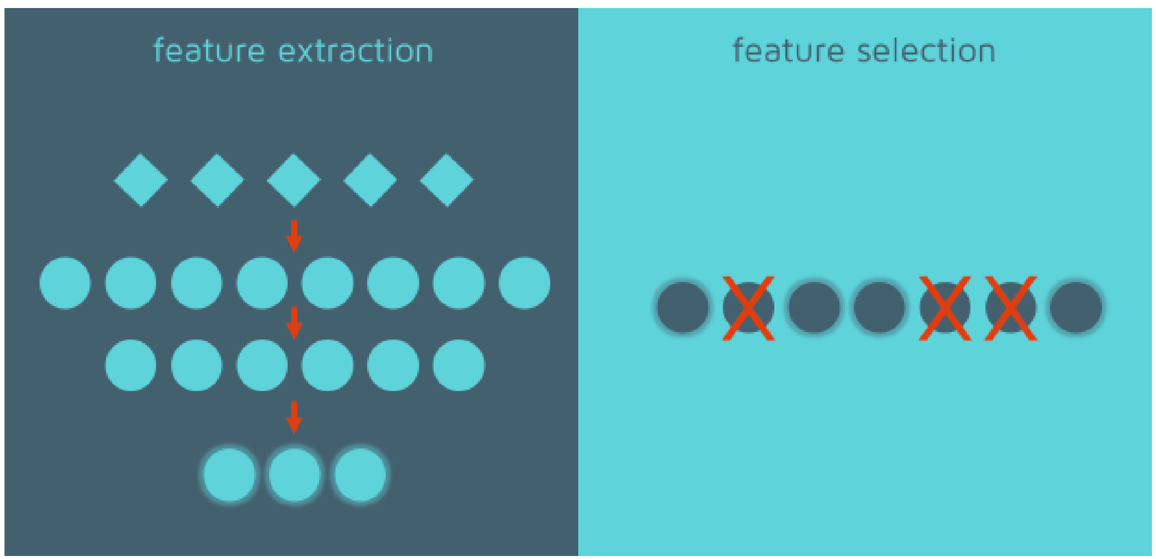



# Outline

- Feature engineering Part 2: Dimensionality reduction
  - Feature Selection
    - Filter methods
    - Wrappers
    - Embedded methods
  - Feature Extraction
    - PCA
    - Linear Discriminant Analysis
    - t-Distributed Stochastic Neighbor Embedding
    - Independent Component Analysis
- Coding Feature Engineering
- Main points

# Feature Extraction /Feature Selection

- **Extraction**: Getting useful features from existing data.
- **Selection**: Choosing a subset of the original pool of features



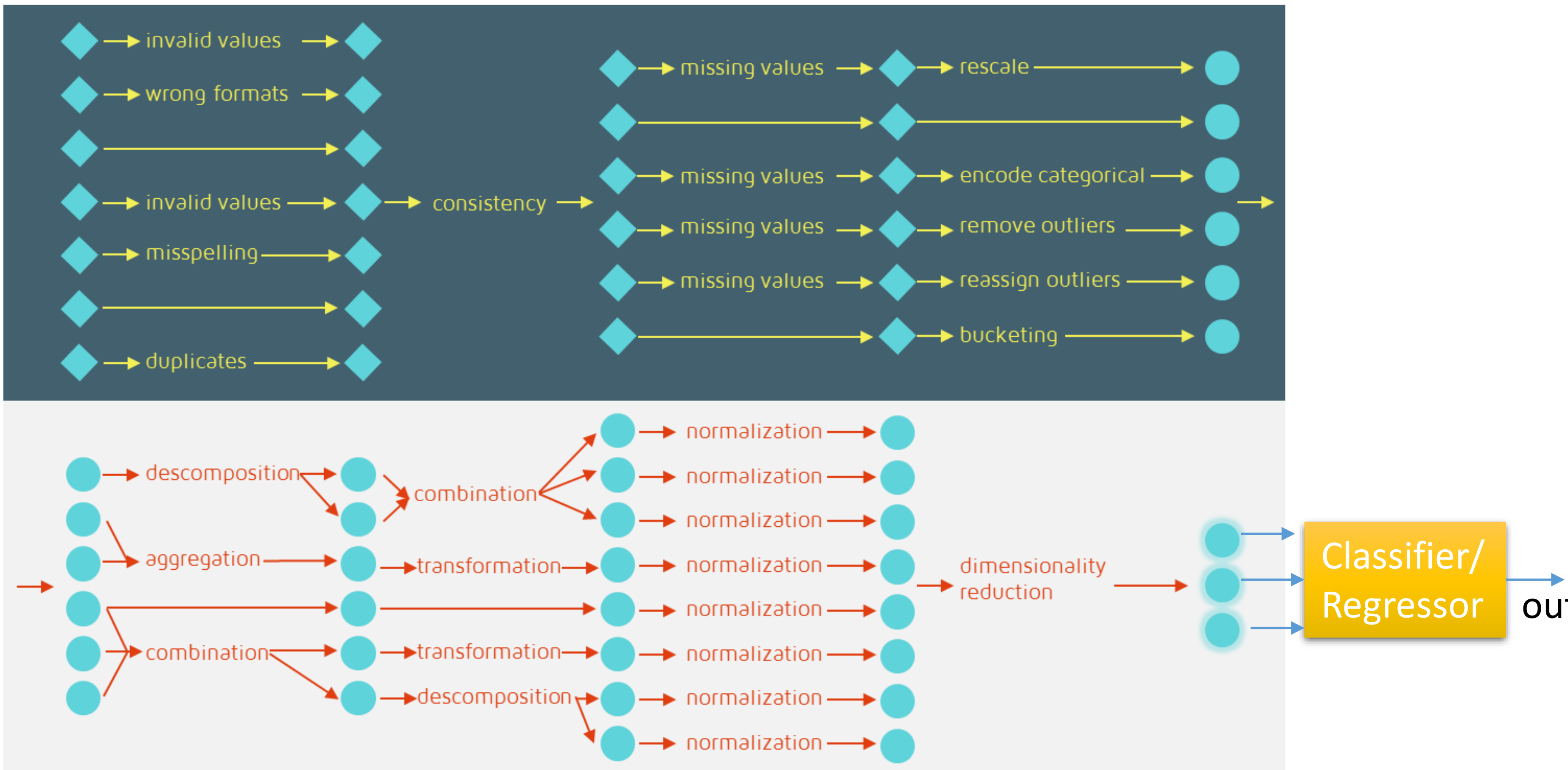

PCA/CNN.. Next lessons

Wrappers….

Plot from https://quantdare.com/

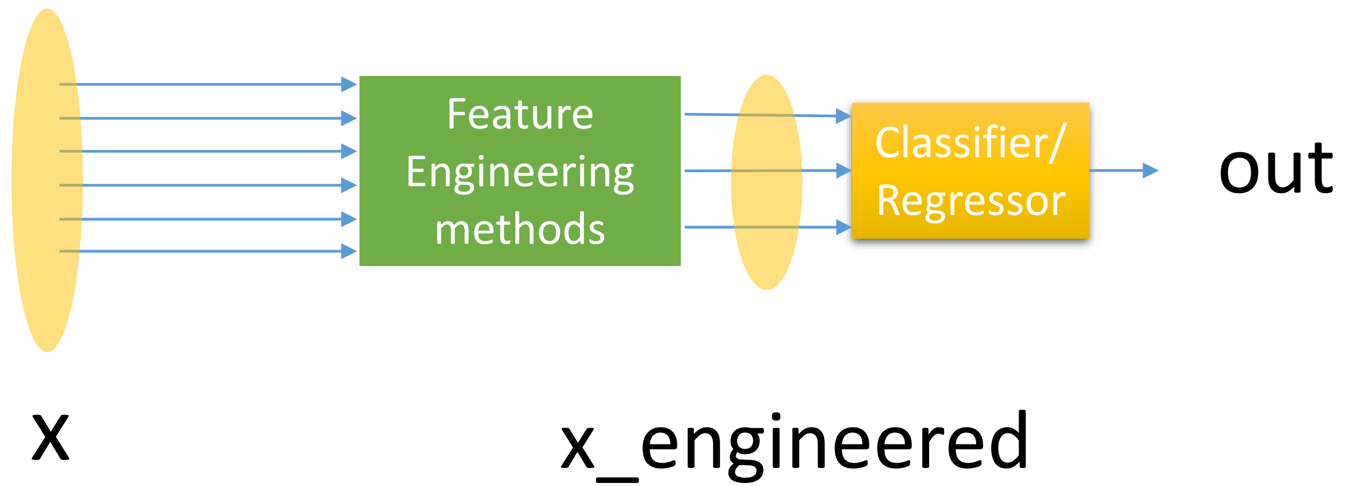
# Feature engineering

Homogenization → Invalid values → Missing values → Consistency → Encoding categorical → Outliers removal → Decomposition → Aggregation → Transformation → Normalization → Dimensionality reduction (for example feature selection)



# Feature engineering

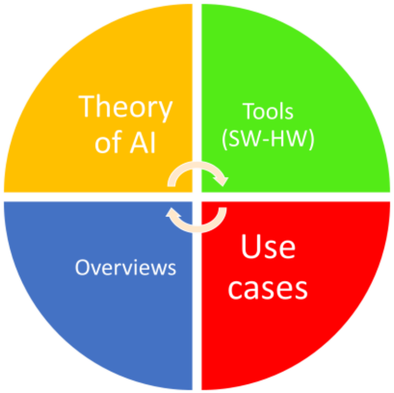
One typical application



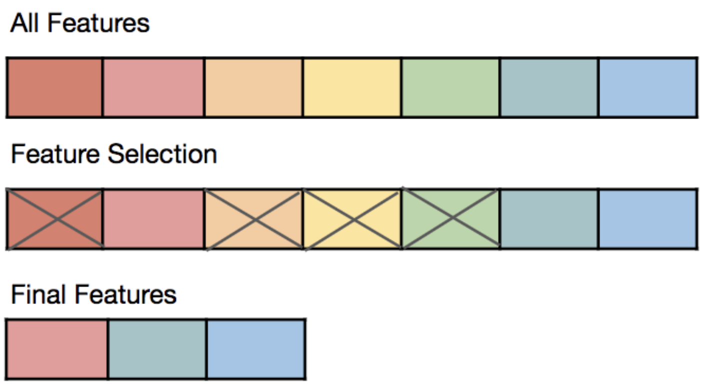
# THEORY

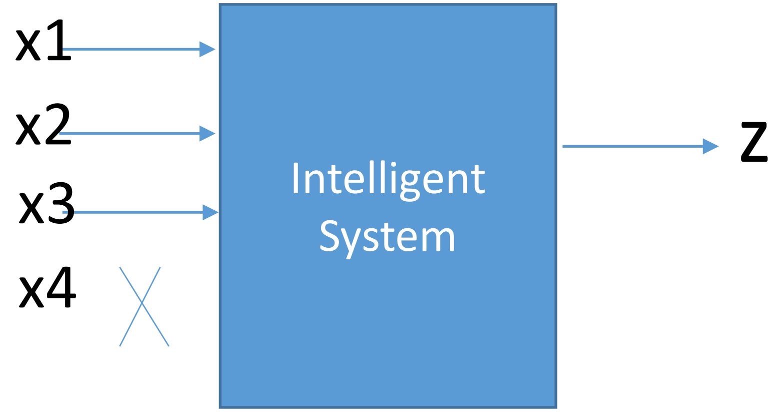
# Feature engineering: Feature selection

What is the importance of my single features?



# Feature selection: the general idea



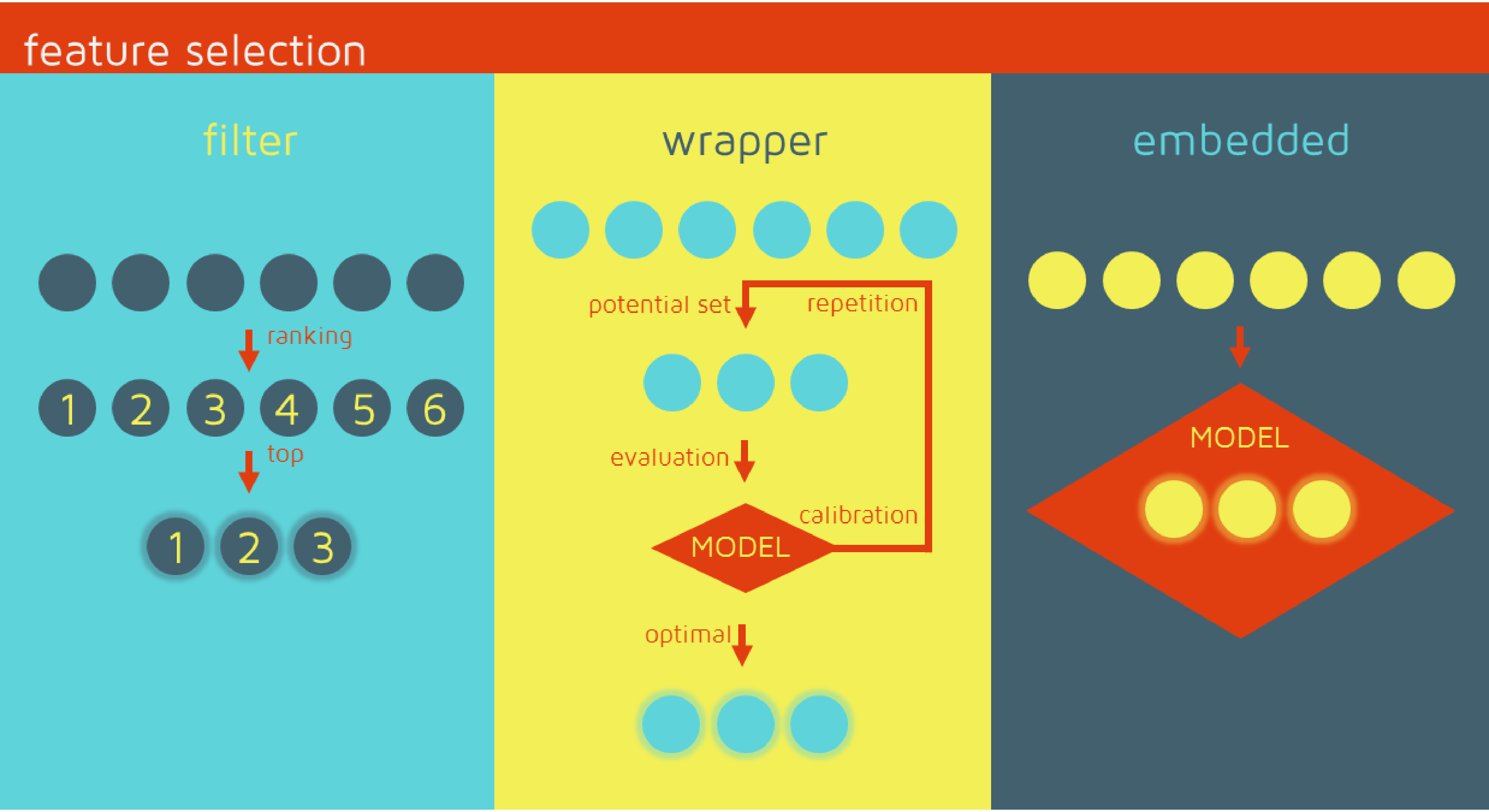




Irrelevant or partially relevant features can negatively impact model performance.

Plus:

- Reduces overfitting due to noise.
- Can improve accuracy
- Reduces Training Time (matrix X is smaller)

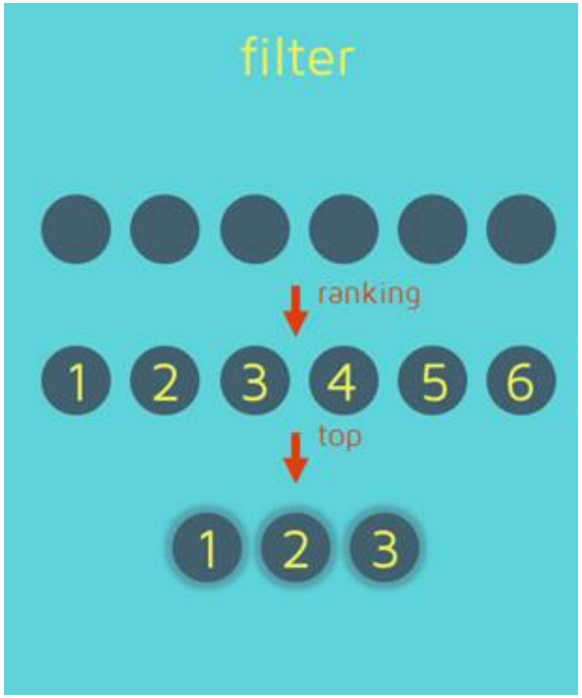
# Filters, wrappers, embedded methods




Plot from https://quantdare.com/

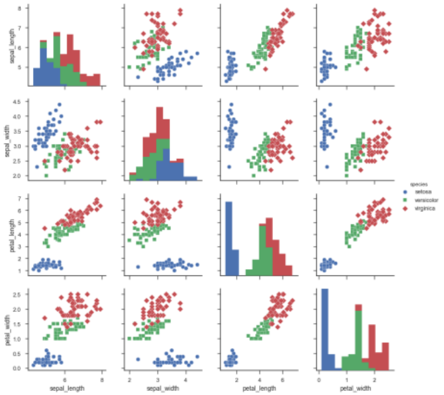
# Filters

- Methods
  - **Criterion**:
    Measure feature/feature subset "relevance"
  - **Search**:
    Usually order features (individual feature ranking or nested subsets of features)
  - **Assessment**:
    Use statistical tests
- Results
  - Are (relatively) robust against overfitting
  - May fail to select the most "useful" features




Examples

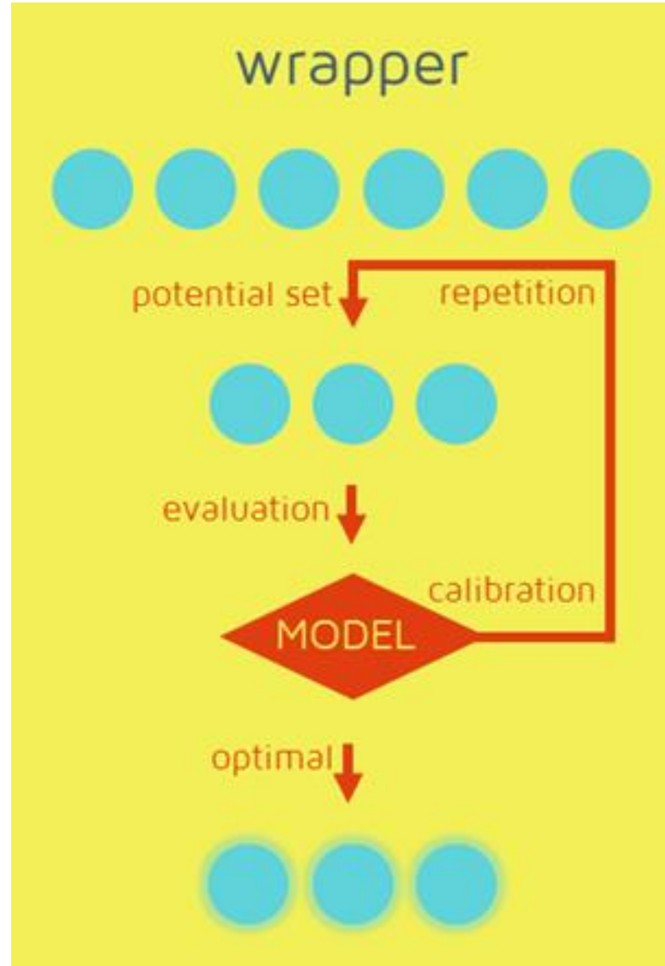
- Prefer feature with high variance
- Prefer feature with low correlation
  (Similarity → cross. correlation)



N very correlated features
→keep one

# Wrappers

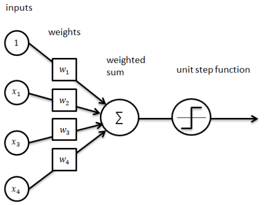
- Methods
  - **Criterion**:
    Measure feature subset "usefulness"
  - **Search**:
    Search the space of all feature subsets
  - **Assessment**:
    Model+cross-validation
- Results
  - Can in principle find the most "useful" features, but
  - Are prone to overfitting

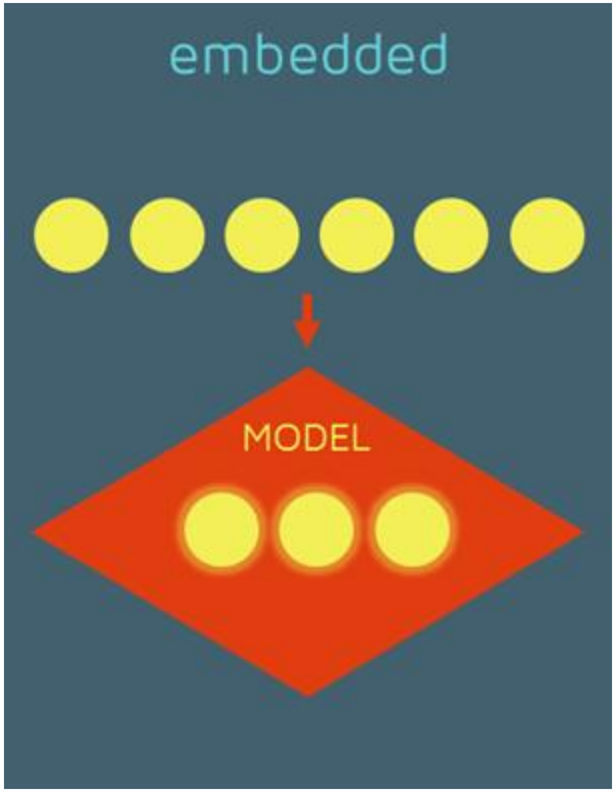



Typical models are kNN or NN

# Embedded methods

- Methods
  - **Criterion**:
    Measure feature subset "usefulness"
  - **Search**:
    Search guided by the learning process
  - **Assessment**:
    Use cross-validation
- Results
  - Similar to wrappers
  - Less computationally expensive
  - Less prone to overfitting



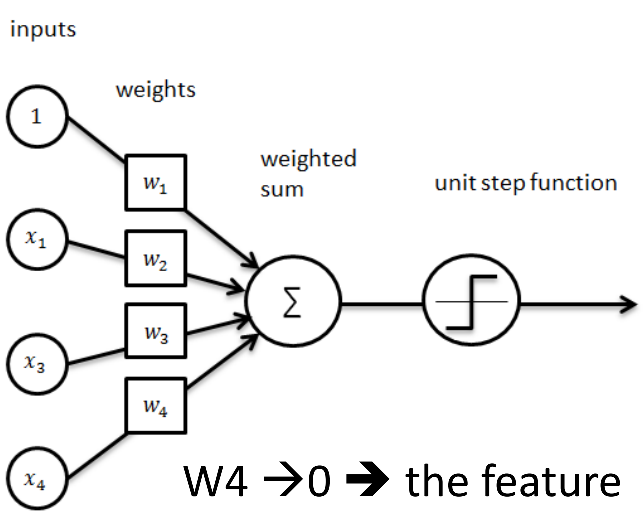

Embedded Methods

- are specific to a given learning machine
- Performs variable selection (implicitly) in the process of training

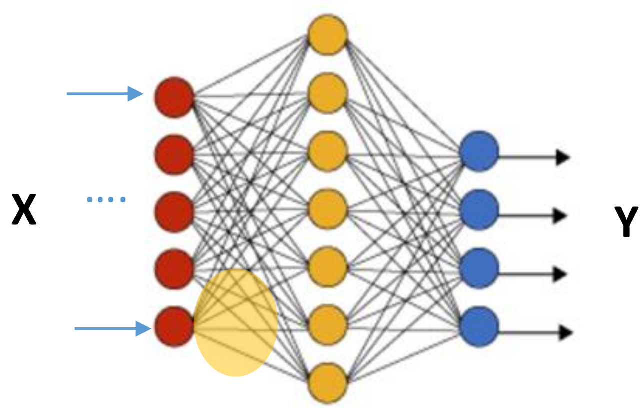
# FS: Embedded methods Neural networks example

- In case of normalized input (same range), if you find one of the weights of the input layer → 0

➔ The training algorithm considered it as limited relevance to the output




W4 →0 ➔ the feature is not «working»

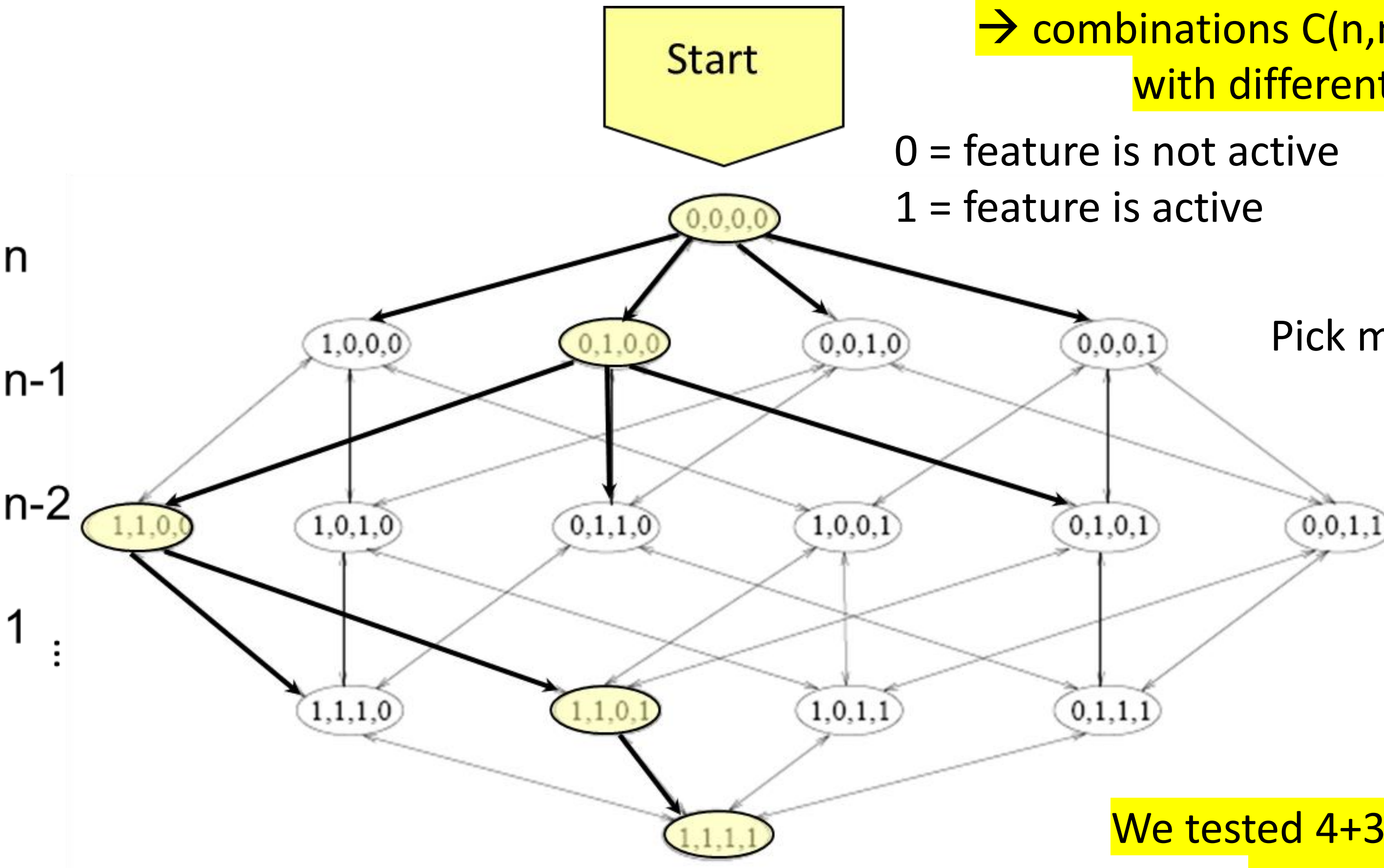



$W_{5i}$ →0 ➔ the feature is not «working»

# Forward selection (wrapper): adding features

Also referred to as SFS: Sequential Forward Selection

All possible configurations
→ combinations C(n,m) or = (n!)/k!(n-k!)
with different m=1,2,3,4



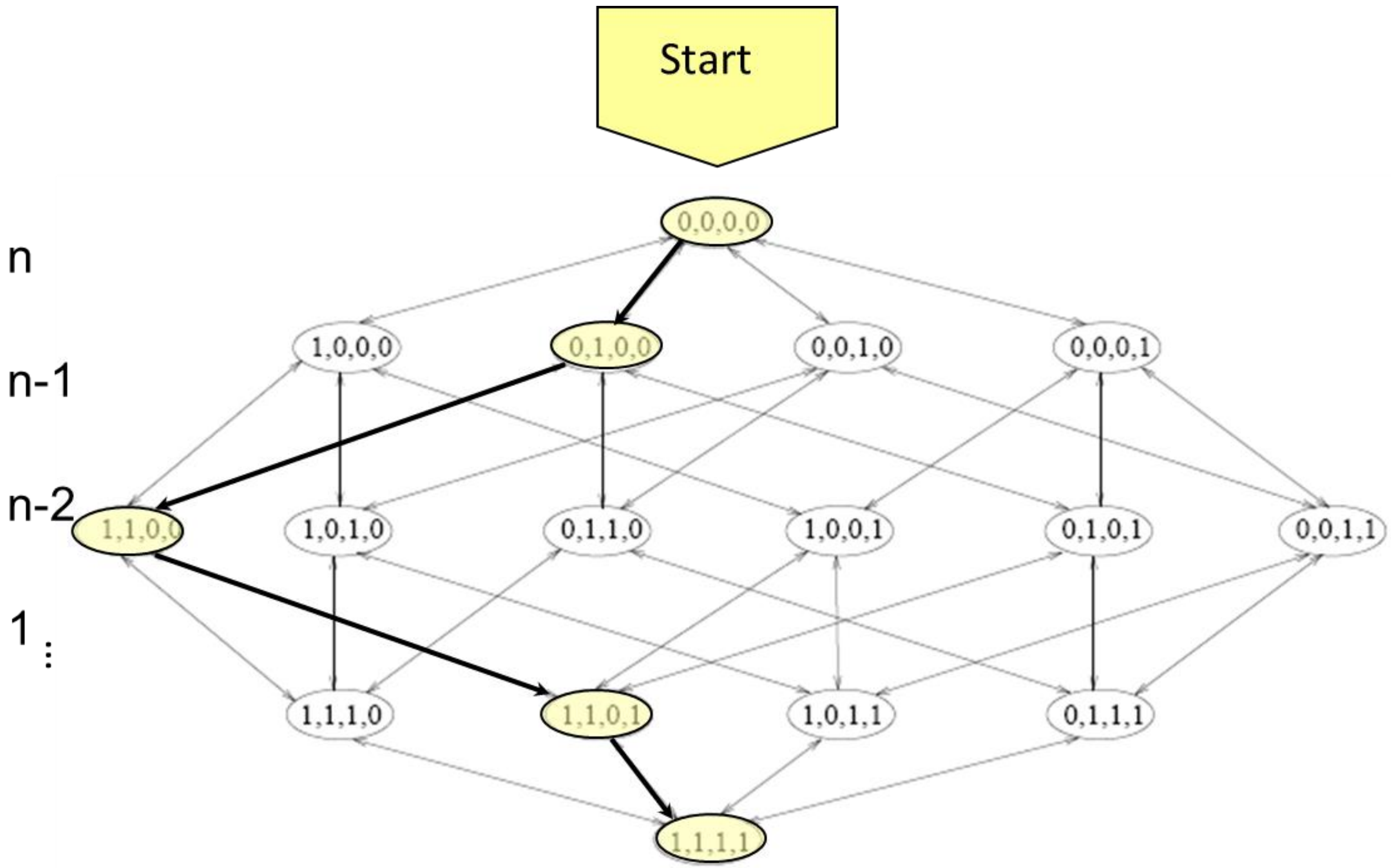

0 = feature is not active
1 = feature is active

Pick most salient feature

Pick most salient
sub group of features

We tested 4+3+2+1 configurations
Now select the best

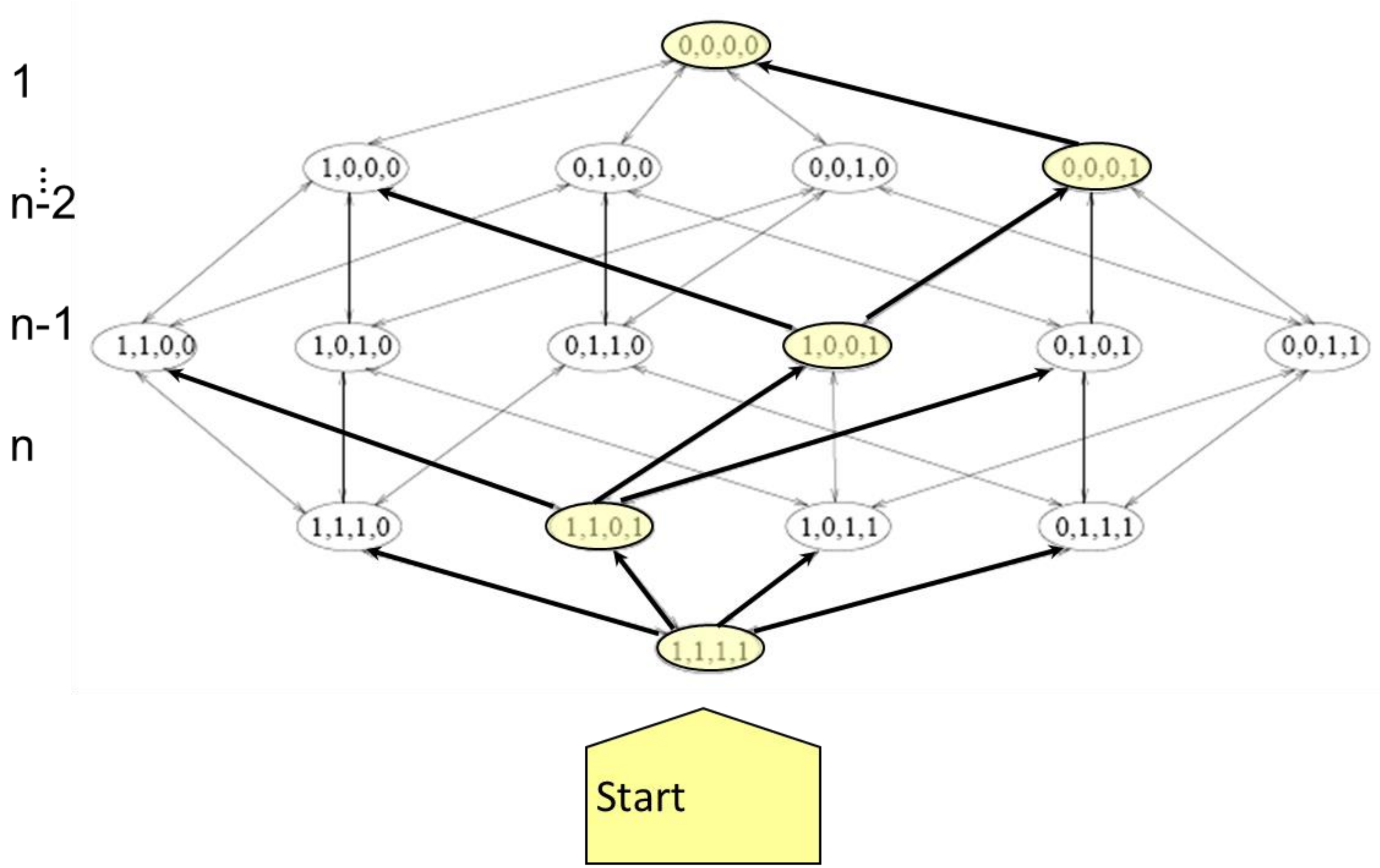
# Forward selection (embedded)




Guided search: we do not consider alternative paths

We tested 4 configurations
Select now the best

# Backward elimination (wrapper)




Also referred to as SBS: Sequential Backward Selection

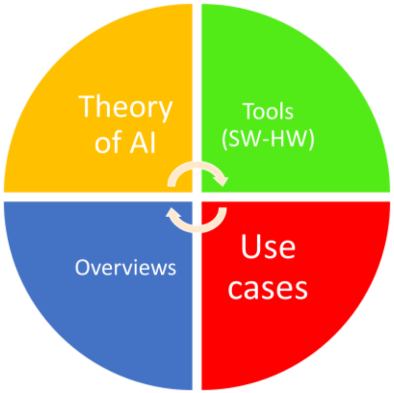



# THEORY

## Feature engineering: Feature extraction

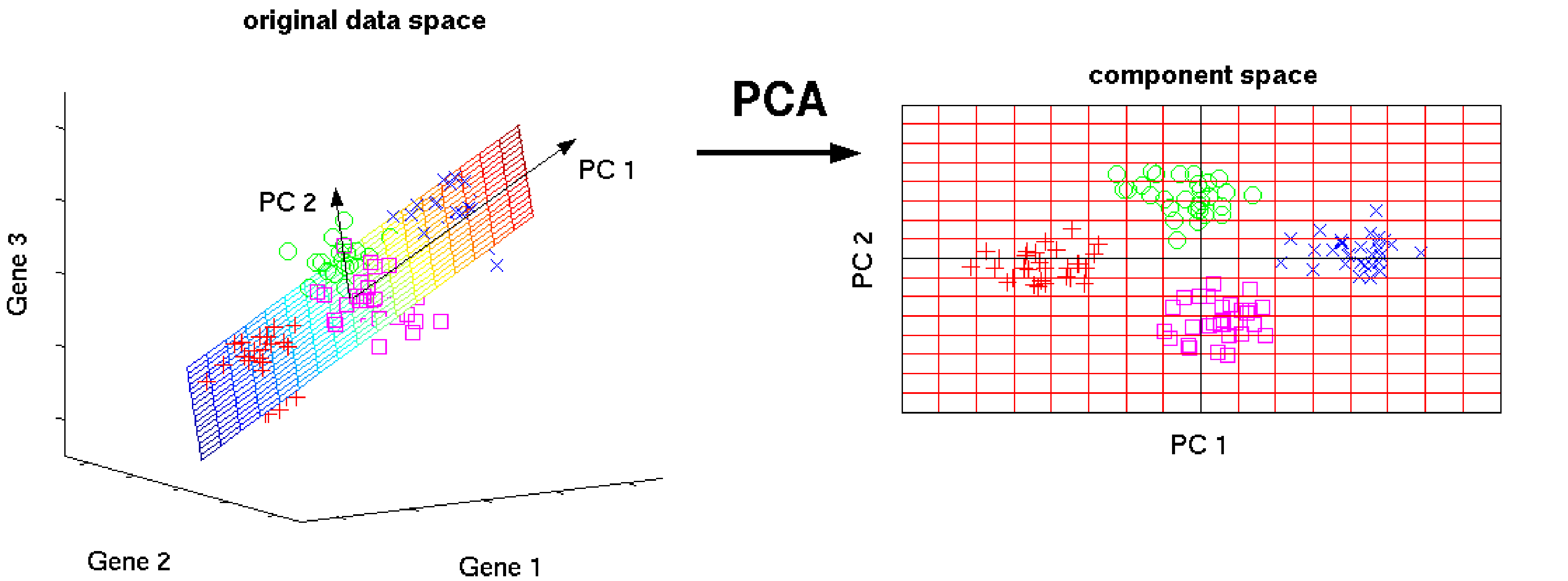
# Principal Component Analysis (PCA)

The most important unsupervised (using no label) dimensionality reduction technique



# Principal Component Analysis

It's all about finding a proper shifted, rotated, sub space describing the data




3 features: Gene1, Gene2, Gene 3
4 classes: '+', '[]', 'x', 'o'

2 features (principal components): PC1, PC2
4 classes: '+', '[]', 'x', 'o'

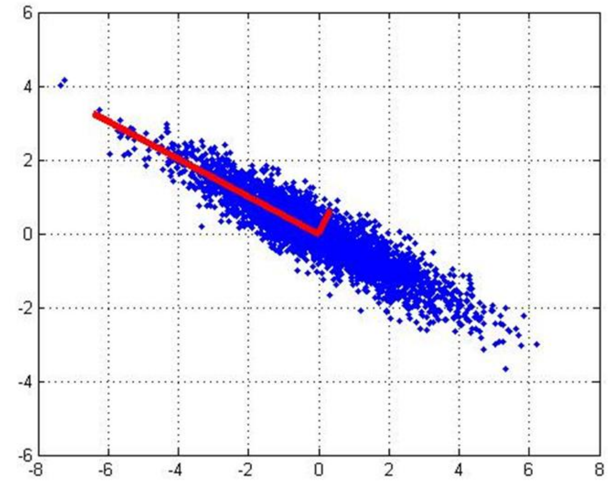
# PCA applications

!

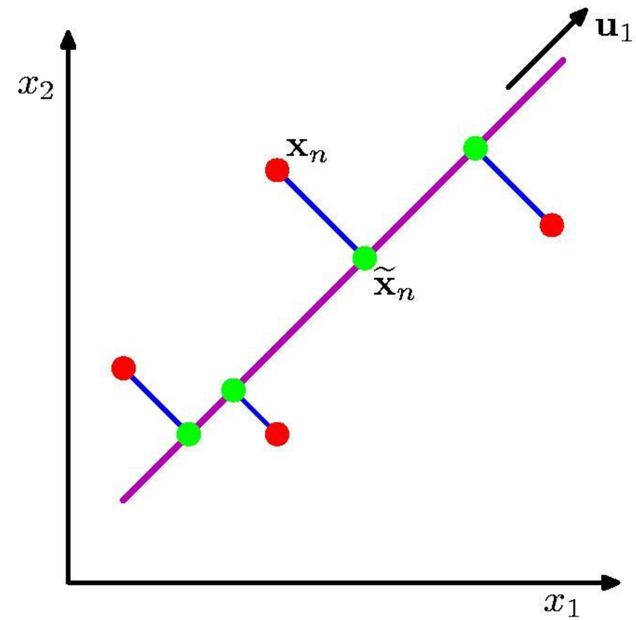
- Data Visualization
- Data Compression/Dimensionality reduction
- Noise Reduction
- Data Classification
- Trend Analysis
- Factor Analysis
- Feature extraction
- Inside neural network
  - Deep learning models
  - Automatic feature creation!

# PCA idea

- It is UNSUPERVISED → using no labels
- Given data points in a d-dimensional space, project into lower dimensional space while preserving as much information as possible
  - E.g., find best planar approximation to 3D data
  - E.g., find best 12-D approximation to 10000-D data
- In particular, choose projection that minimizes *squared error* in reconstructing original data

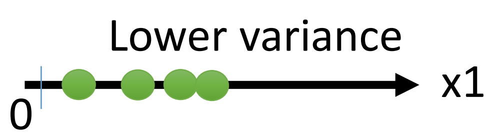


# PCA idea (2)




Orthogonal projection of data onto lower-dimension linear space that:

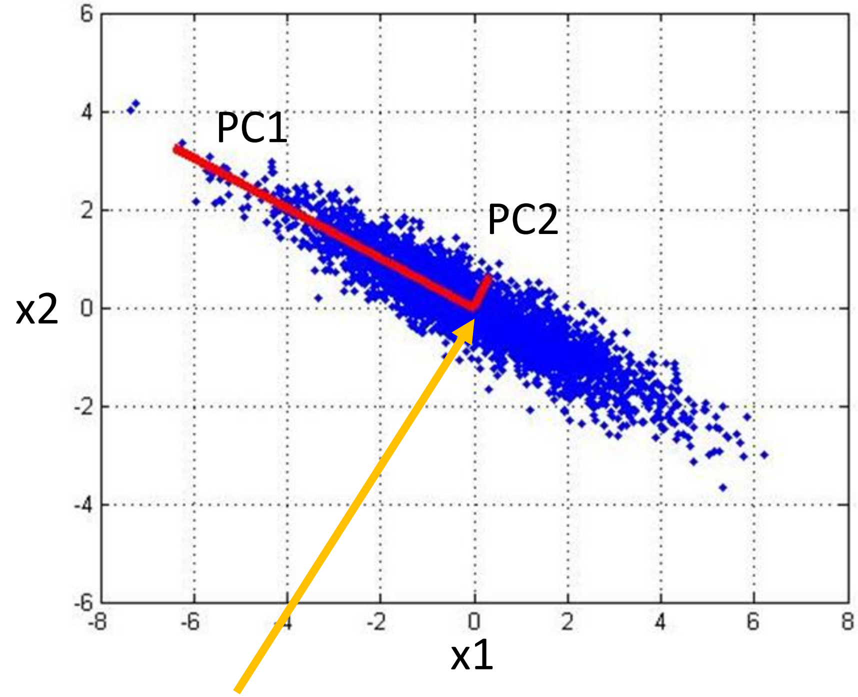
- maximizes variance of projected data (**purple line**)





- minimizes mean squared distance between data point and projections (sum of **blue lines**)

# PCA idea (3)




- **Vectors** originating from the center of mass
- Principal component #1 (PC1) points in the direction of the **largest variance**.
- Each subsequent principal component…
  - is **orthogonal** to the previous ones, and
  - points in the directions of the **largest variance of the <u>residual</u> subspace**

# PCA algorithm (sample covariance matrix)

- Given data $\{\mathbf{x}_1, \ldots, \mathbf{x}_m\}$, compute covariance matrix $\Sigma$

$$\boxed{\Sigma = \frac{1}{m}\sum_{i=1}^{m}(\mathbf{x}_i - \overline{\mathbf{x}})(\mathbf{x} - \overline{\mathbf{x}})^T}$$ where $$\boxed{\overline{\mathbf{x}} = \frac{1}{m}\sum_{i=1}^{m}\mathbf{x}_i}$$

**PCA** basis vectors = the eigenvectors of $\Sigma$

- Larger eigenvalue $\Rightarrow$ more important eigenvectors

**x_reduced** = PCA( **x**, k)     %k=3

# PCA algorithm: main steps

PCA algorithm($\mathbf{X}$, $k$): top $k$ eigenvalues/eigenvectors

% $\underline{\mathbf{X}}$ = N × m data matrix,
% … each data point $\mathbf{x}_i$ = column vector, i=1..m

1. $\mathbf{X} \leftarrow$ subtract mean $\underline{\mathbf{x}}$ from each column vector $\mathbf{x}_i$ in $\underline{\mathbf{X}}$
2. $\mathbf{\Sigma} \leftarrow \mathbf{X}\mathbf{X}^T$ … covariance matrix of $\mathbf{X}$
3. $\{ \lambda_i, \mathbf{u}_i \}_{i=1..N}$ = eigenvectors/eigenvalues of $\mathbf{\Sigma}$
   ... $\lambda_1 \geq \lambda_2 \geq \ldots \geq \lambda_N$
4. Return $\{ \lambda_i, \mathbf{u}_i \}_{i=1..k}$
   % top $k$ principle components




# THEORY

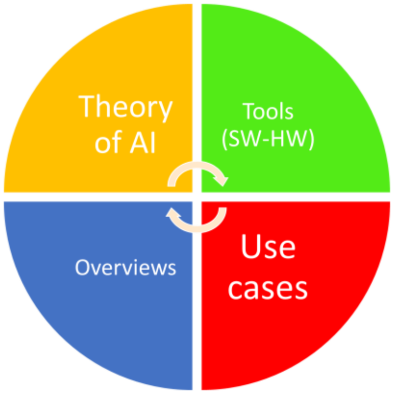
## Feature engineering: Feature extraction

# Dimensionality Reduction via LDA, t-SNE, ICA

Linear Discriminant Analysis

t-Distributed Stochastic Neighbor Embedding

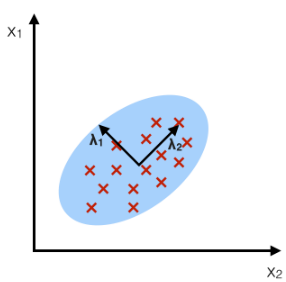
Indipendent Component Analysis



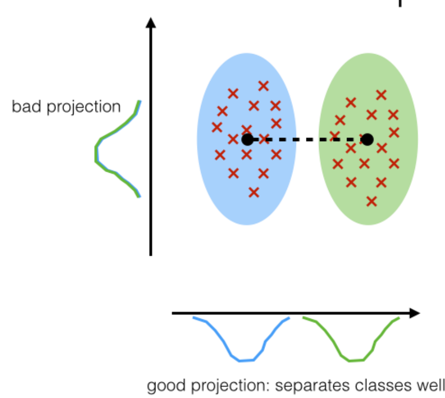
# Dimensionalty reduction: Linear Discriminant Analysis (**LDA**)

- Find a linear combination of features that characterizes or separates the classes
- The resulting combination may be used as a linear classifier or dimensionality reduction before a classifier

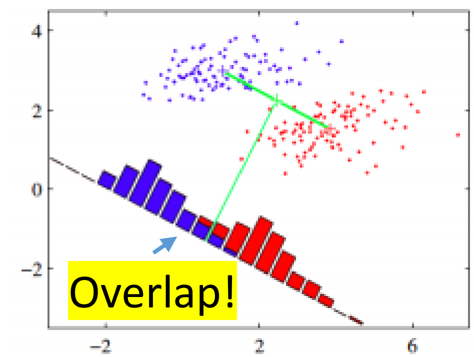
**PCA:**
component axes that maximize the variance



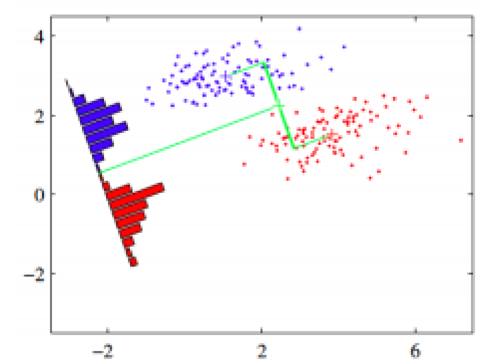

**LDA:**
maximizing the component axes for class-separation




Wrong rotation



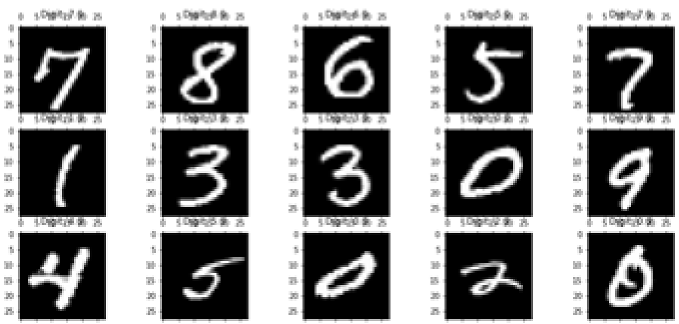

Best rotation



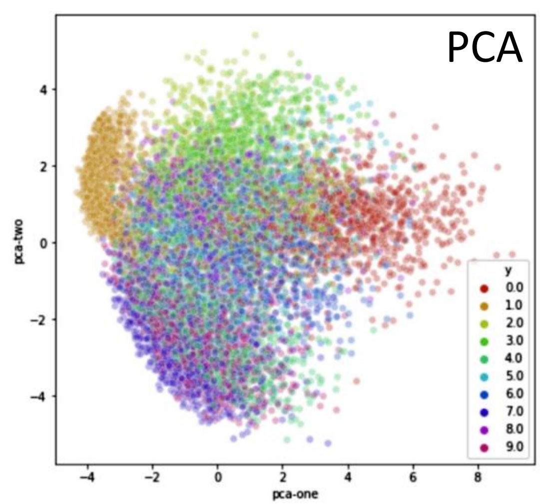

The code and usage
of the Fischer LDA classifier
is presented Lesson 14

# Dimensionalty reduction: t-SNE

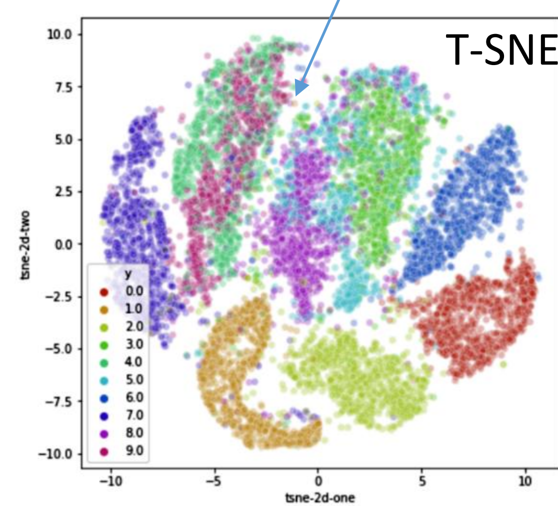
- t-Distributed Stochastic Neighbor Embedding
- Is a non-linear technique for dimensionality reduction that is particularly well suited for the visualization/engineering of high-dimensional datasets
- **x_reduced** = tSNE( **x** , k)  % k = 2 or 3 max



**MNIST dataset:**
60,000 train images
28x28 = <u>784 features</u>
10 classes: ’0’, ‘1’,… , ‘9’



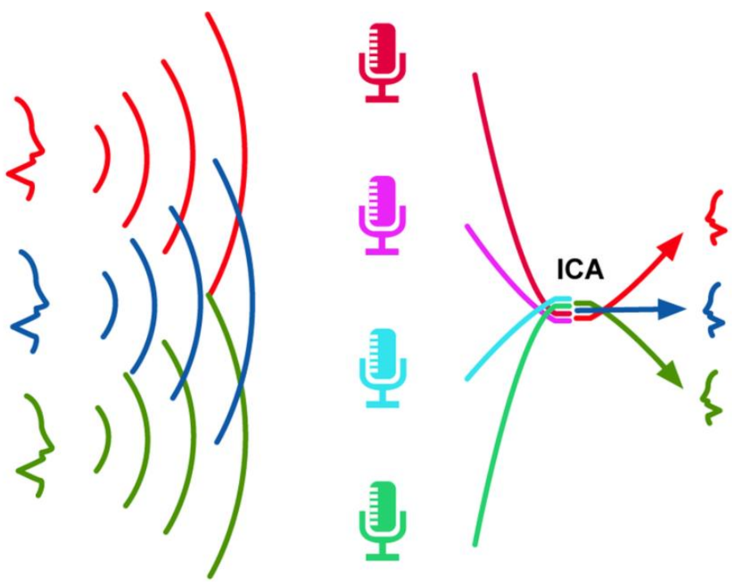

Improved separation of classes



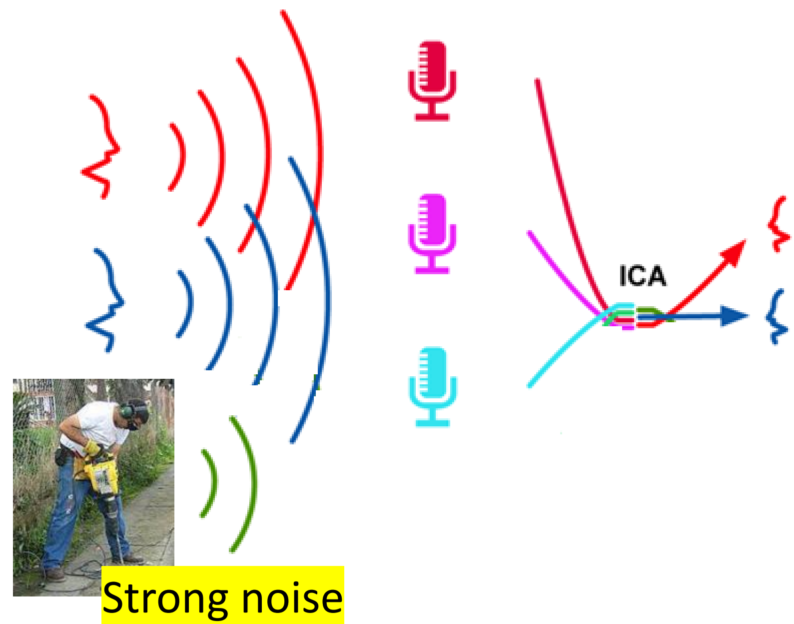
# Dimensionalty reduction:
## Indipendent Component Analysis (ICA)

Often used in signal processing and industrial application

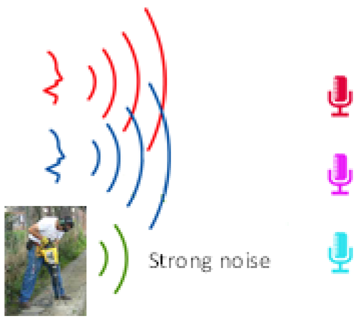
**Example 1**:
3 speakers, 4 mics

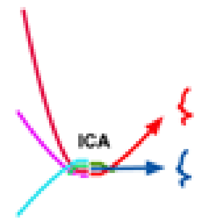



**Example 2**
2 speakers + noise, 3 mics



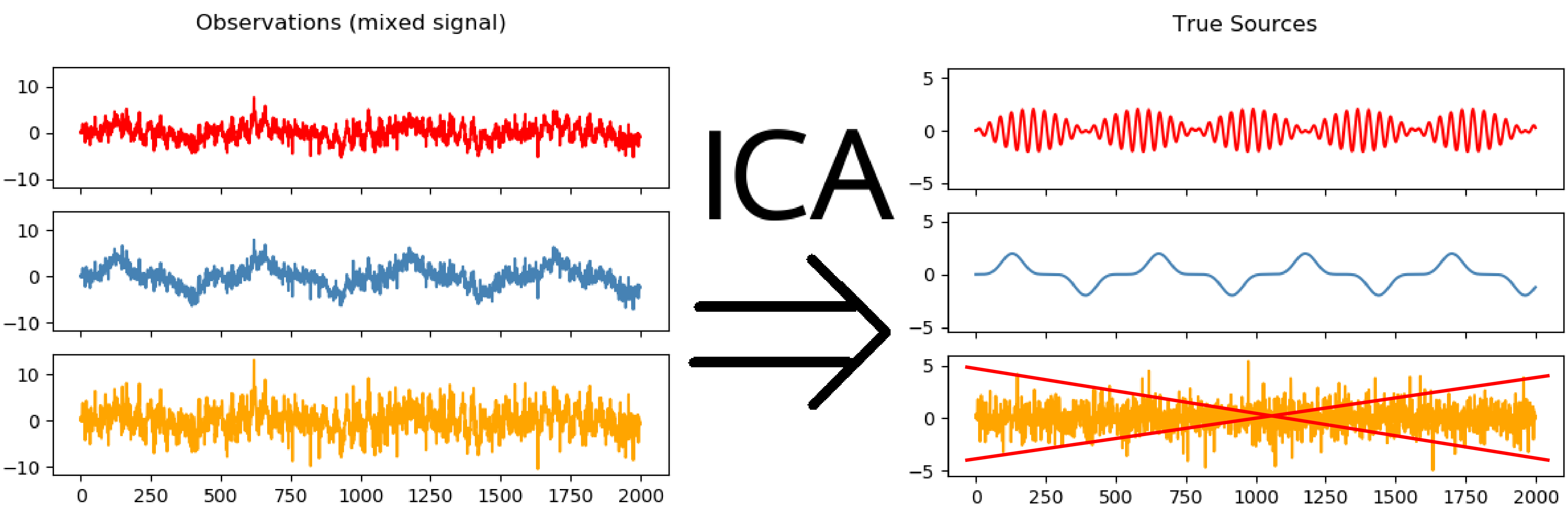
# Dimensionalty reduction:
## Indipendent Component Analysis (ICA)




3 feature/signals
(coming from 2 independent sources + noise)

We can drop the third transformed signal

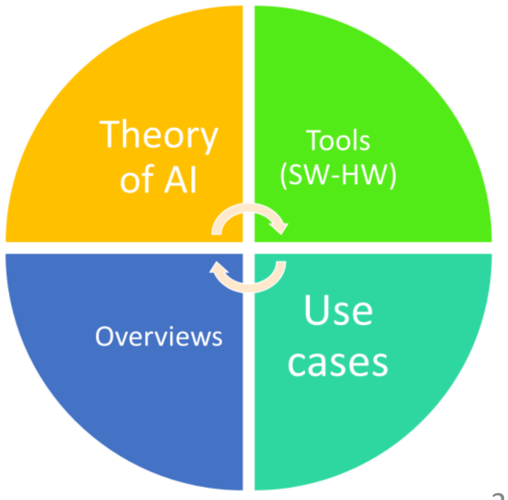





# Toolboxes

## Coding
## Feature Extraction and Selection

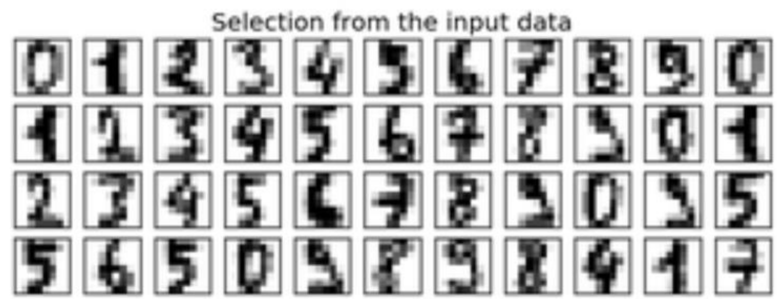
### In Python-Colab



# Dataset used in the scripts

- sklearn.datasets.load_digits
- Each datapoint is a 8x8 image of a digit.

| | |
|---|---|
| Classes | 10 |
| Samples per class | ~180 |
| Samples total | 1797 |
| Dimensionality | 64 |
| Features | integers 0-16 |




1797 samples!
<< MNIST DB

# Dimensionality Reduction via PCA

Feature extraction!

```
# General imports
import numpy as np
import matplotlib.pyplot as plt
```

Load the digits dataset (classification).

Each datapoint is a 8x8 image of a digit

```
[2]  from sklearn.datasets import load_digits
     digits = load_digits()
     digits.data.shape
```

The **shape** is a tuple that gives you an indication of the no. of dimensions in the array.

```python
print( digits.data[0].shape )
```

```
(64,)
```

```python
print(digits.data[0])
```

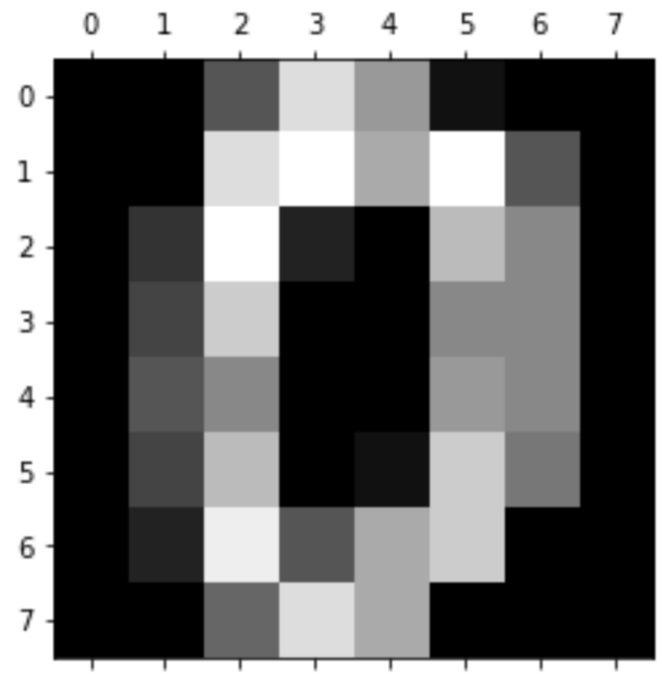
```
[ 0.   0.   5. 13.   9.   1.   0.   0.   0.   0. 13. 15. 10. 15.   5.   0.   0.   3.
 15.   2.   0. 11.   8.   0.   0.   4. 12.   0.   0.   8.   8.   0.   0.   5.   8.   0.
  0.   9.   8.   0.   0.   4. 11.   0.   1. 12.   7.   0.   0.   2. 14.   5. 10. 12.
  0.   0.   0.   0.   6. 13. 10.   0.   0.   0.]
```

```python
print(digits.target[0])
```

```
0
```

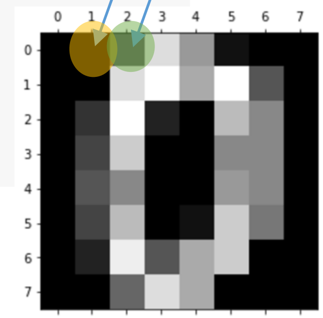
```python
import matplotlib.pyplot as plt
plt.gray()
plt.matshow(digits.images[0])
plt.show()
```

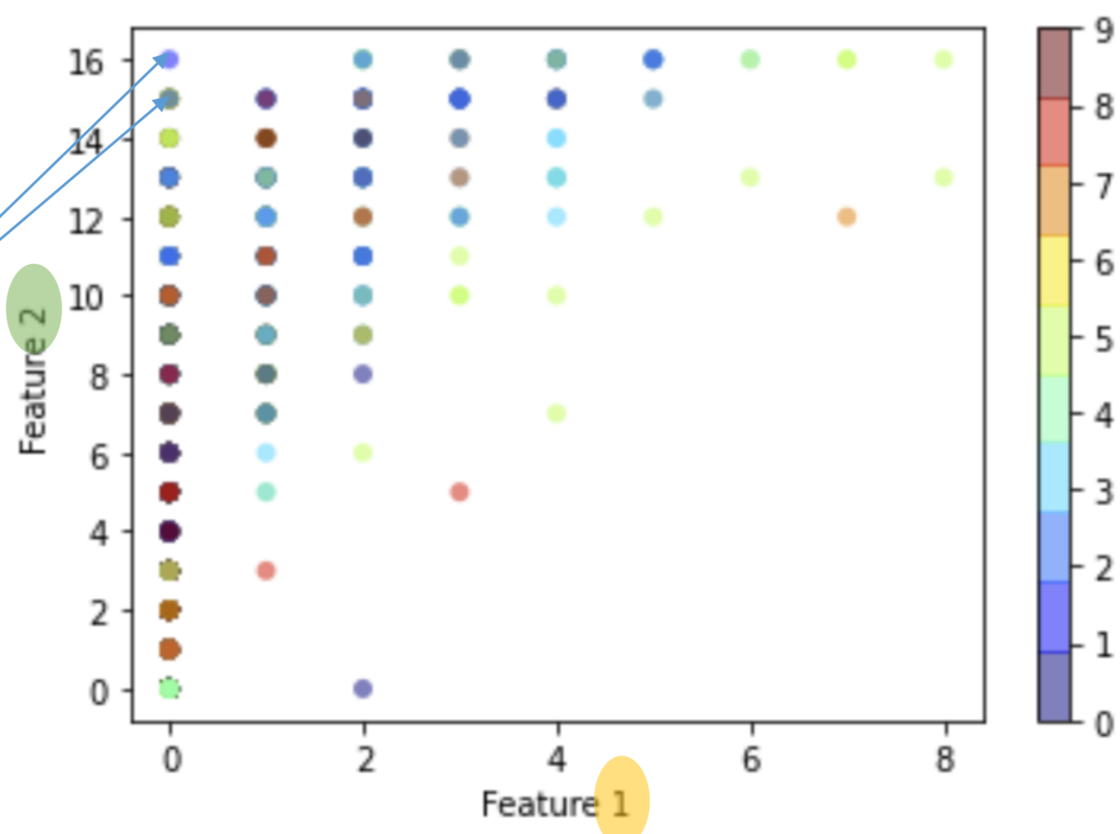
```
<Figure size 432x288 with 0 Axes>
```



Let's plot just two features to see the correlation

```
[3] plt.scatter(digits.data[:, 1], digits.data[:, 2],
                c=digits.target, edgecolor='none', alpha=0.5,
                cmap=plt.cm.get_cmap('jet', 10))
    plt.xlabel('Feature 1')
    plt.ylabel('Feature 2')
    plt.colorbar();
```



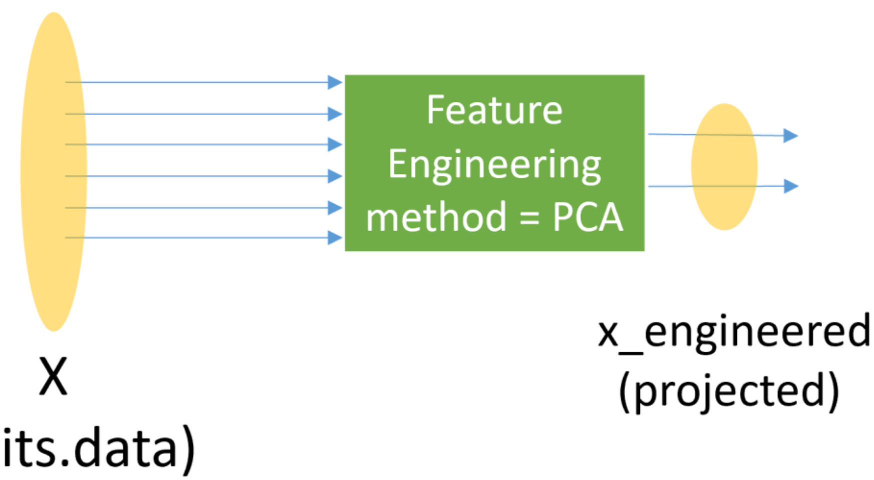




The effects of the quantization of the features (gray level 0-16) is evident.

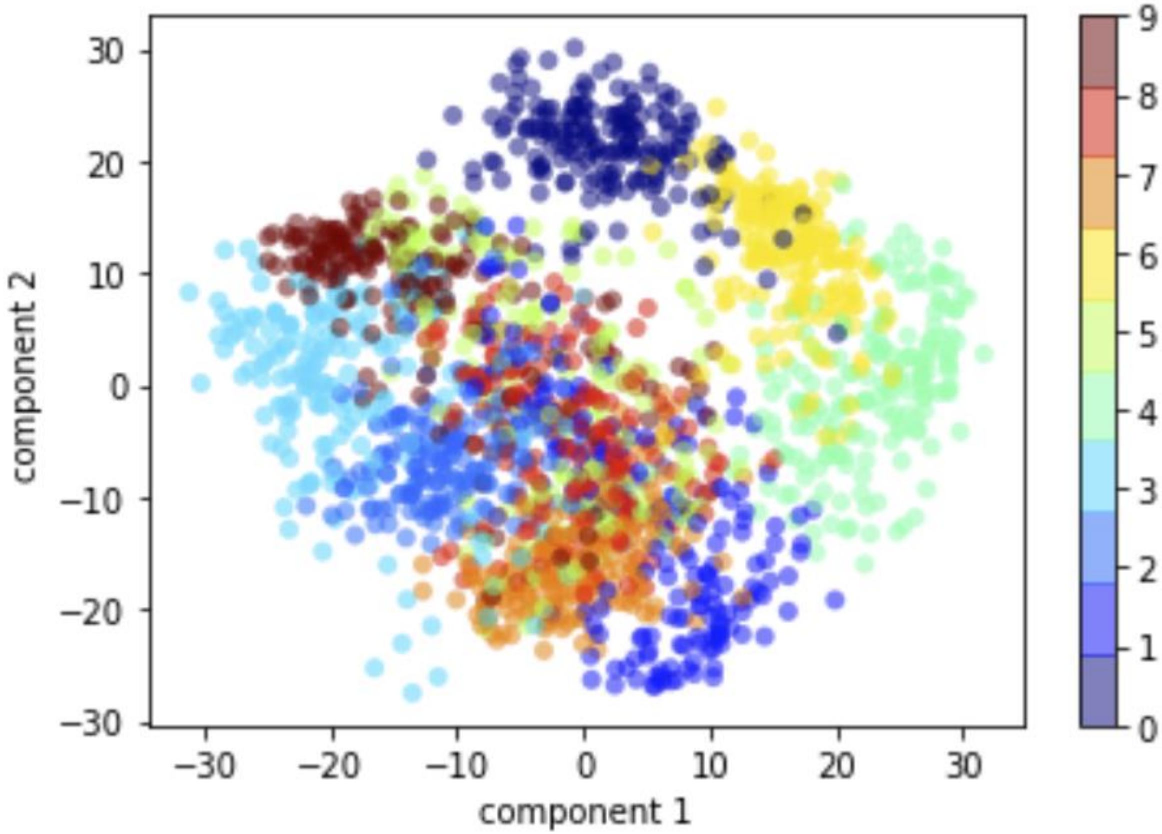
+ Bad clustering (no clusters..)

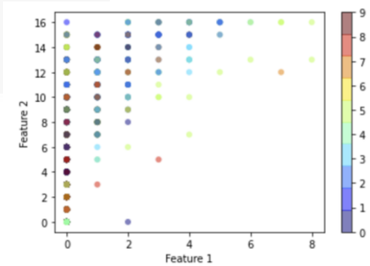
```python
from sklearn.decomposition import PCA

pca = PCA(2)  # project from 64 to 2 dimensions
projected = pca.fit_transform(digits.data)
print(digits.data.shape)
print(projected.shape)
```

```
(1797, 64)
(1797, 2)
```



```
plt.scatter(projected[:, 0], projected[:, 1],
            c=digits.target, edgecolor='none', alpha=0.5,
            cmap=plt.cm.get_cmap('jet', 10))
plt.xlabel('component 1')
plt.ylabel('component 2')
plt.colorbar();
```



# Dimensionality Reduction via t-Distributed Stochastic Neighbor Embedding (t-SNE)

## Feature extraction with t-SNE

Feature extraction!

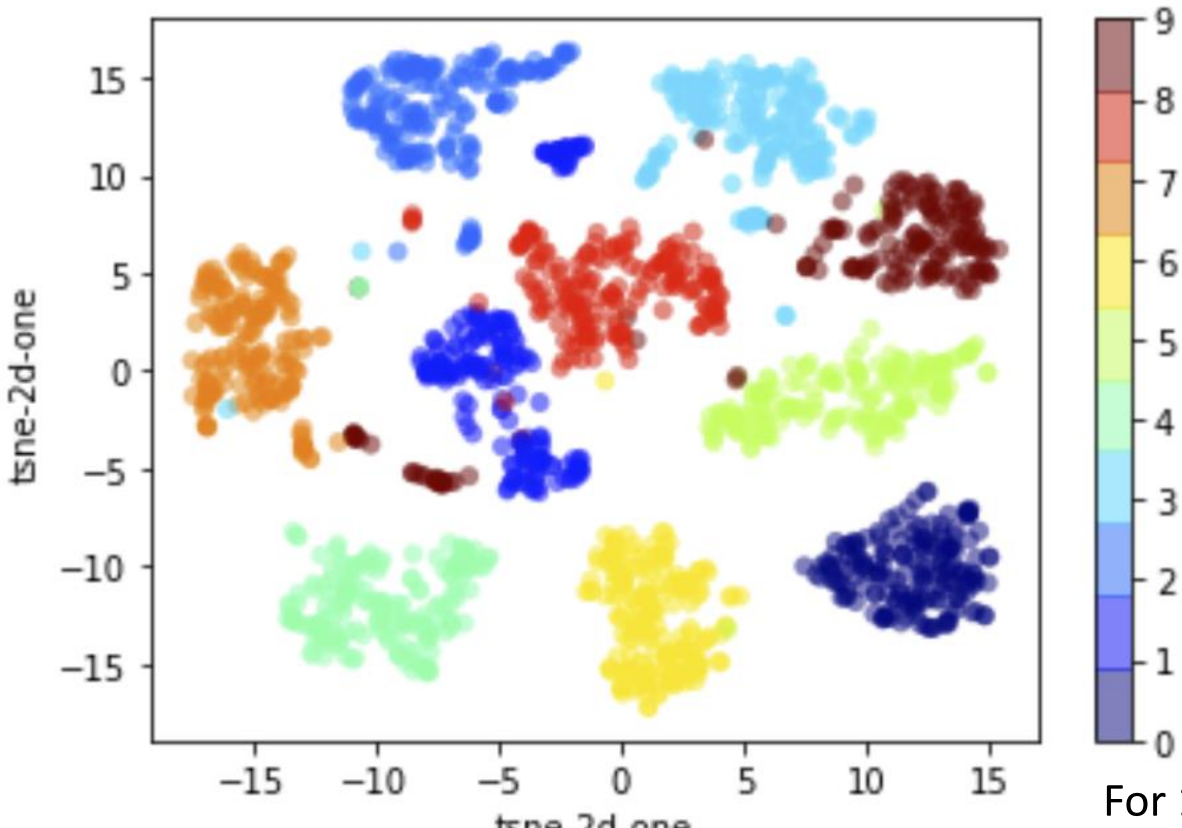
```
[6] from sklearn.manifold import TSNE

    tsne = TSNE(n_components=2, verbose=1, perplexity=40, n_iter=300)
    tsne_results = tsne.fit_transform(digits.data)  # no y labels
```

```
[t-SNE] Computing 121 nearest neighbors...
[t-SNE] Indexed 1797 samples in 0.010s...
[t-SNE] Computed neighbors for 1797 samples in 0.445s...
[t-SNE] Computed conditional probabilities for sample 1000 / 1797
[t-SNE] Computed conditional probabilities for sample 1797 / 1797
[t-SNE] Mean sigma: 8.394135
[t-SNE] KL divergence after 250 iterations with early exaggeration: 61.611313
[t-SNE] KL divergence after 300 iterations: 0.958217
```

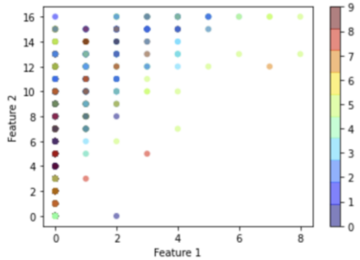
```python
plt.scatter(tsne_results[:,0], tsne_results[:,1] ,
            c=digits.target, edgecolor='none', alpha=0.5,
            cmap=plt.cm.get_cmap('jet', 10))
plt.xlabel('tsne-2d-one')
plt.ylabel('tsne-2d-one')
plt.colorbar();
```

Same plotting as PCA

```python
plt.scatter(projected[:, 0], projected[:, 1],
            c=digits.target, edgecolor='none', alpha=0.5,
            cmap=plt.cm.get_cmap('jet', 10))
```

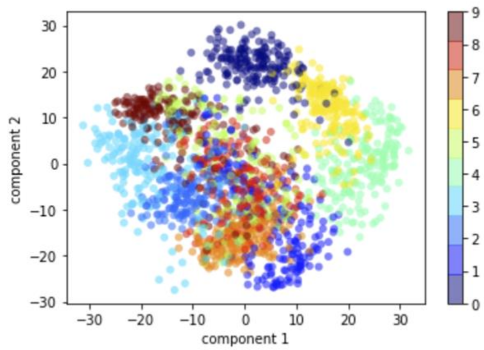



For 1797 samples
is not so bad …

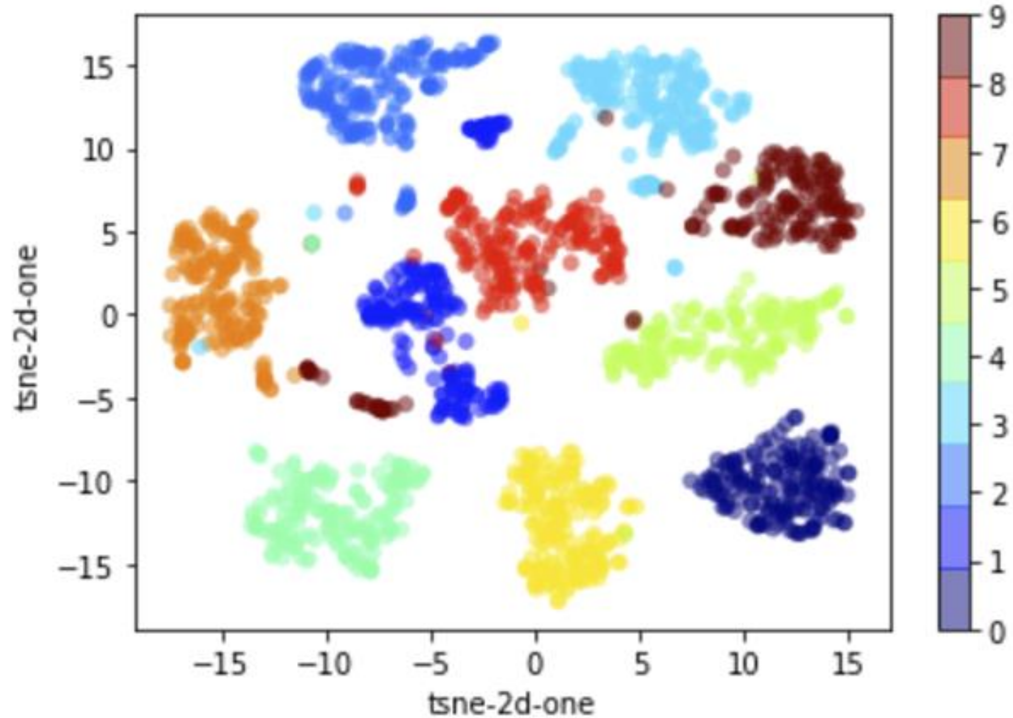



Feature Selection
(manual….. )

PCA




T-SNE



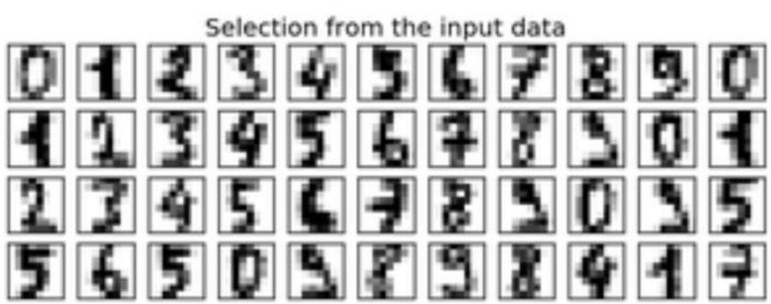

For 1797 samples
is not so bad …

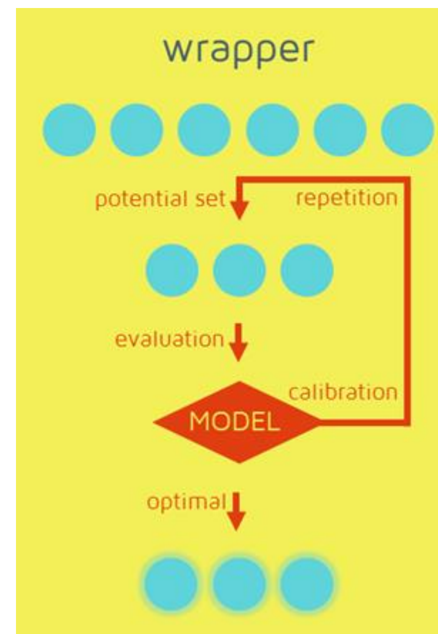
# Dimensionality Reduction
# Feature Selection

**Feature selection!**

Recursive Feature Elimination
wrapper: SVM
(you can try with kNN, LDA, NN, etc.)

| | |
|---|---|
| Classes | 10 |
| Samples per class | ~180 |
| Samples total | 1797 |
| Dimensionality | 64 |
| Features | integers 0-16 |




Which pixel is more significative to induce the class?

```python
from sklearn.svm import SVC
from sklearn.feature_selection import RFE
```

Feature ranking with Recursive Feature Elimination

```python
# Load the digits dataset #digits = load_digits()
X = digits.images.reshape((len(digits.images), -1))
y = digits.target
```

The unwrapped image of the digit (range of gray 0-16)

```python
print(X[1])
print(y[1])
```

```
[ 0.  0.  0. 12. 13.  5.  0.  0.  0.  0.  0. 11. 16.  9.  0.  0.  0.  0.
  3. 15. 16.  6.  0.  0.  0.  7. 15. 16. 16.  2.  0.  0.  0.  0.  1. 16.
 16.  3.  0.  0.  0.  0.  1. 16. 16.  6.  0.  0.  0.  0.  1. 16. 16.  6.
  0.  0.  0.  0.  0. 11. 16. 10.  0.  0.]
1
```

X[1]

Defining the wrapper classifier (SVM)

```
# Create the RFE object and rank each pixel
svc = SVC(kernel="linear", C=1)
rfe = RFE(estimator=svc, n_features_to_select=1, step=1)
rfe.fit(X, y)
ranking = rfe.ranking_.reshape(digits.images[0].shape)
```

Just to reshape the results as the digit images

```python
# Create the RFE object and rank each pixel
svc = SVC(kernel="linear", C=1)
rfe = RFE(estimator=svc, n_features_to_select=1, step=1)
rfe.fit(X, y)
ranking = rfe.ranking_.reshape(digits.images[0].shape)
```

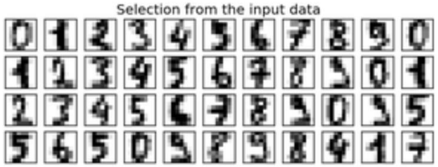
```python
# Plotting the results
print("ranking of the pixel w.r.t the classification capability")
print(ranking)
```

Less important pixel

```
ranking of the pixel w.r.t the classification capability
[[64 50 31 23 10 17 34 51]
 [57 37 30 43 14 32 44 52]
 [54 41 19 15 28  8 39 53]
 [55 45  9 18 20 38  1 59]
 [63 42 25 35 29 16  2 62]
 [61 40  5 11 13  6  4 58]
 [56 47 26 36 24  3 22 48]
 [60 49  7 27 33 21 12 46]]
```

Most important pixel for classification

(since we used as wrapper a SVM to classify the class == the digit)

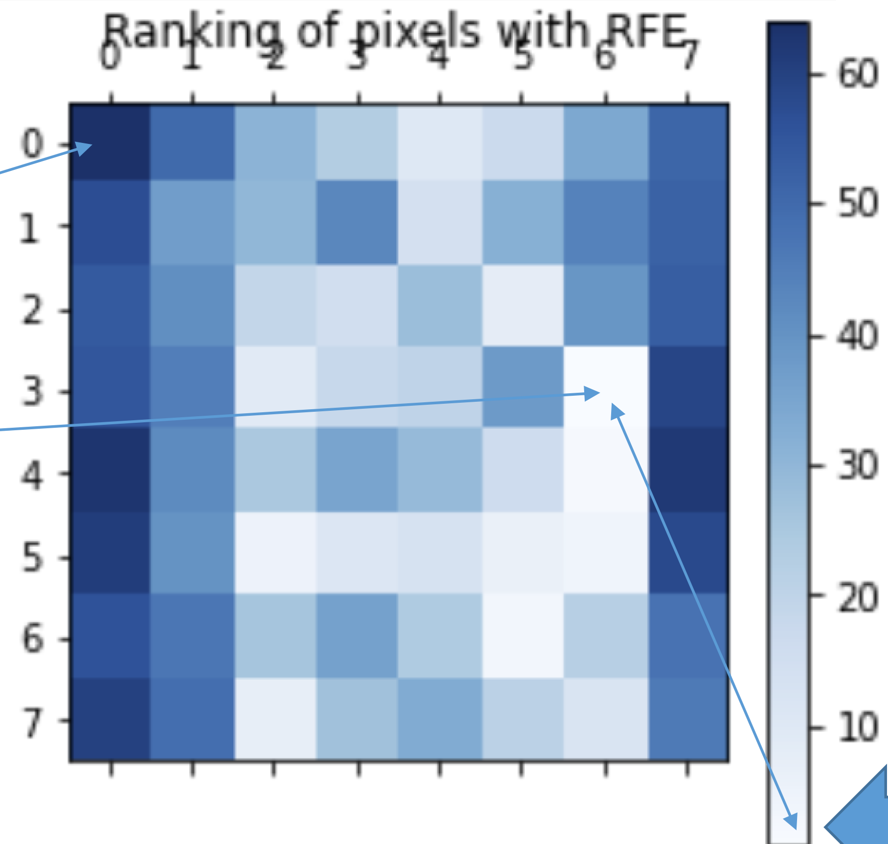


```python
# Plot pixel ranking
plt.matshow(ranking, cmap=plt.cm.Blues)
plt.colorbar()
plt.title("Ranking of pixels with RFE")
plt.show()
```

Less important pixel

```
ranking of the pixel w.r.t the classification capability
[[64 50 31 23 10 17 34 51]
 [57 37 30 43 14 32 44 52]
 [54 41 19 15 28  8 39 53]
 [55 45  9 18 20 38  1 59]
 [63 42 25 35 29 16  2 62]
 [61 40  5 11 13  6  4 58]
 [56 47 26 36 24  3 22 48]
 [60 49  7 27 33 21 12 46]]
```

Most important pixel for classification




Rank = 1

# Main points

- Feature engineering Part 2: Dimensionality reduction
  - Feature Selection
    - Filter methods
    - Wrappers
    - Embedded methods
  - Feature Extraction
    - PCA
    - Linear Discriminant Analysis
    - t-Distributed Stochastic Neighbor Embedding
    - Independent Component Analysis
- Coding Dimensionality Reduction in Python-Colab