Les données numériques: mean, variance, standardisation and PCA

Diane Lingrand

UNIVERSITÉ
CÔTE D'AZUR

SI3

2022 - 2023

Outline

- Mean, Variance
- Normalisation, Standardisation
- PCA (Principal Component Analysis)

Outline

Mean and variance for a 1D vector

- arithmetic mean : $\mu = \frac{1}{n}\sum_{i=1}^{n} x_i$
- variance : $\sigma^2 = \frac{1}{n}\sum_{i=1}^{n}(x_i - \mu)^2$
  - standard deviation (*écart type*) : $\sigma = \sqrt{\sigma^2}$
- mean absolute deviation (mad) : $s = \frac{1}{n}\sum_{i=1}^{n}|x_i - \mu|$

Mean and variance for a 1D vector

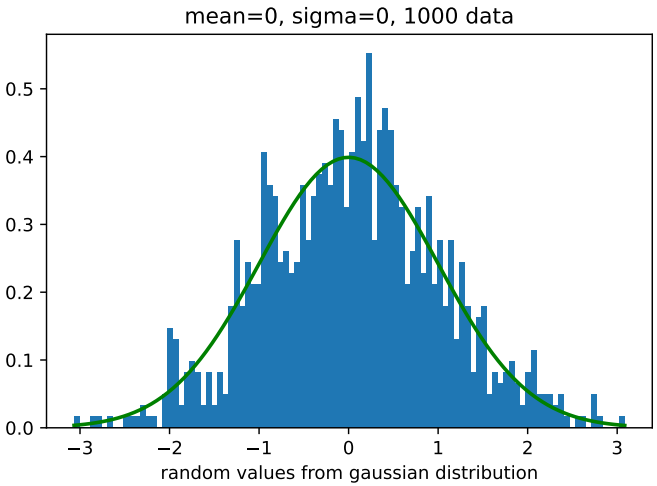
- arithmetic mean : $\mu = \frac{1}{n}\sum_{i=1}^{n} x_i$
- variance : $\sigma^2 = \frac{1}{n}\sum_{i=1}^{n}(x_i - \mu)^2$
  - standard deviation (*écart type*) : $\sigma = \sqrt{\sigma^2}$
- mean absolute deviation (mad) : $s = \frac{1}{n}\sum_{i=1}^{n}|x_i - \mu|$

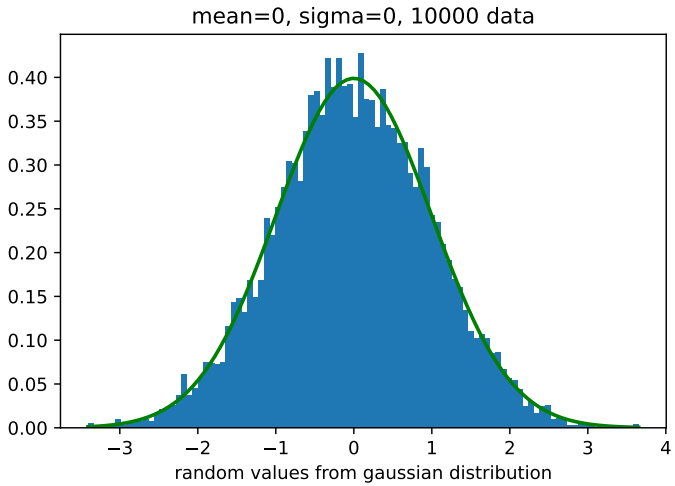
```
a = np.random.randint(low=0, high=20, size=10)
mu = np.mean(a)
print(a)
print("mean = \t %.2f" %mu)
print("std  =\t %.2f" %np.std(a))
print("mad  =\t %.2f" %np.mean(np.abs(a-mu)))
```

```
[16 10 19  4  0  1  5 11 18 16]
mean =   10.00
std  =   6.78
mad  =   6.00
```

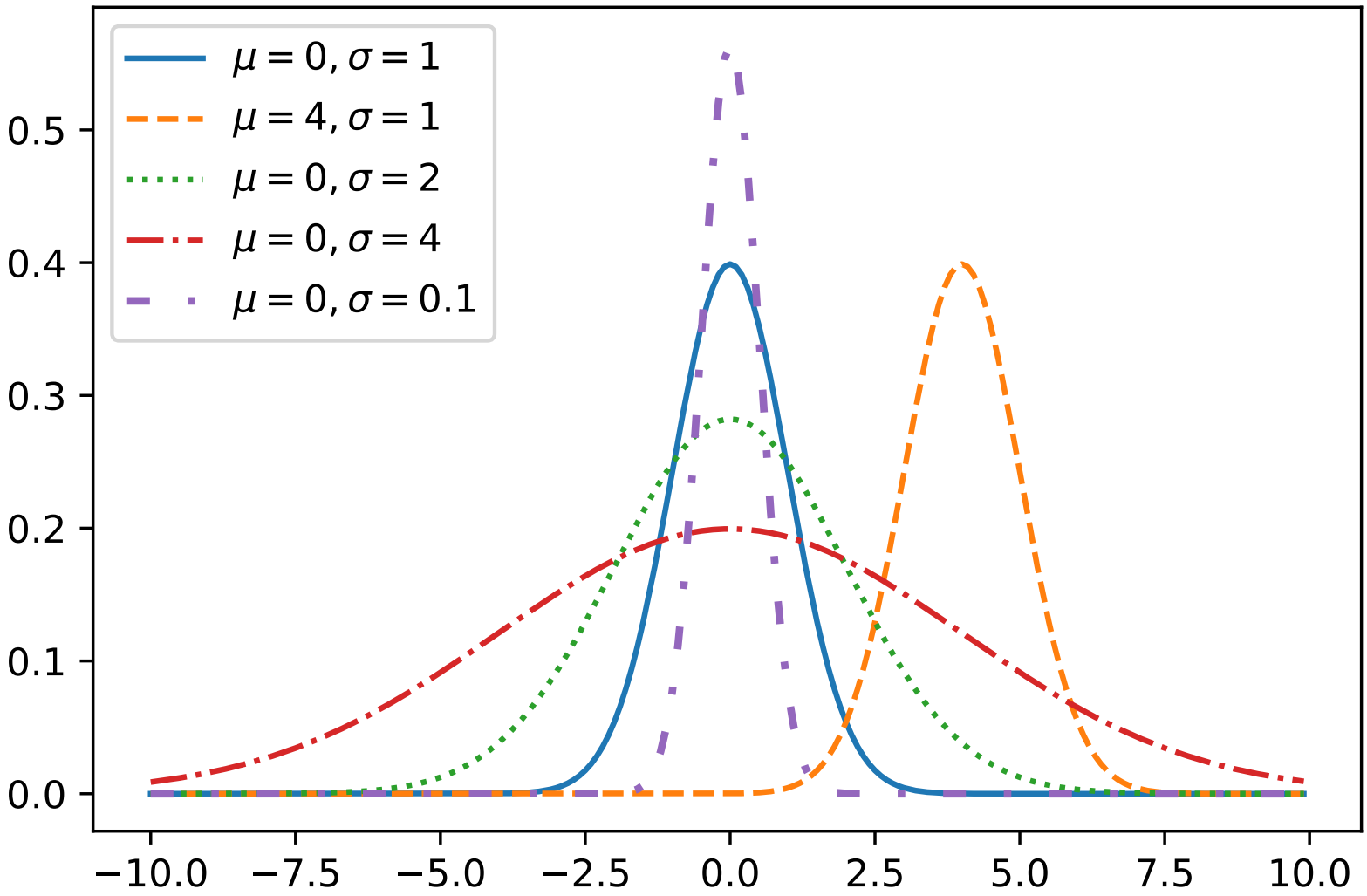
Gaussian distribution

- density probability function : $f(x) = \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{1}{2}(\frac{x-\mu}{\sigma})^2}$
  - for $\mu = 0$ and $\sigma = 1$: $f(x) = \frac{1}{\sqrt{2\pi}} e^{-\frac{x^2}{2}}$





Gaussian distribution



Mean and variance for a data set (n samples of dim. d)

- feature (or component) number i for every samples, $0 \leq i < d$
- sample number j : $x^j \in \mathcal{R}^d$
- mean : $\mu_i = \frac{1}{n} \sum_{j=1}^{n} x_i^j$
- variance : $\sigma_i^2 = \frac{1}{n} \sum_{j=1}^{n} (x_i^j - \mu_i)^2$
- mean absolute deviation : $s_i = \frac{1}{n} \sum_{j=1}^{n} |x_i^j - \mu_i|$

Means and numpy

```
Xtrain = np.random.randint(10, size=(8,3))
print(Xtrain)
print("shape = ",Xtrain.shape)
```

```
[[2 1 0]
 [2 8 6]
 [9 3 1]
 [6 0 3]
 [3 3 1]
 [2 9 1]
 [9 6 2]
 [8 1 4]]
shape =  (8, 3)
```

```
print("global mean: \t\t", np.mean(Xtrain))
np.set_printoptions(precision=2)
print("mean with axis 0:\t", np.mean(Xtrain,axis=0))
print("mean with axis 1:\t", np.mean(Xtrain,axis=1))
```

```
global mean:            4.75
mean with axis 0:       [5.5  4.88 3.88]
mean with axis 1:       [5.67 5.67 4.33 1.   7.67 3.   5.33 5.33]
```

Variance and mean absolute deviation in python

```
Xtrain = np.random.randint(10, size=(8,3))
print(Xtrain)
print("shape = ",Xtrain.shape)
```

```
[[2 1 0]
 [2 8 6]
 [9 3 1]
 [6 0 3]
 [3 3 1]
 [2 9 1]
 [9 6 2]
 [8 1 4]]
shape =  (8, 3)
```

```
n = Xtrain.shape[0]
mu = np.mean(Xtrain,axis=0)
var = np.sum((Xtrain-mu)*(Xtrain-mu), axis=0)/n
sigma = np.sqrt(var)
mad = np.sum(np.abs(Xtrain-mu),axis=0)/n
print('mu = ', mu, '; sigma = ', sigma, '; mad = ', mad)
```

```
mu =  [5.12 3.88 2.25] ; sigma =  [3.02 3.18 1.85] ; mad =  [2.88 2.84 1.56]
```

Outline

Normalisation and standardisation

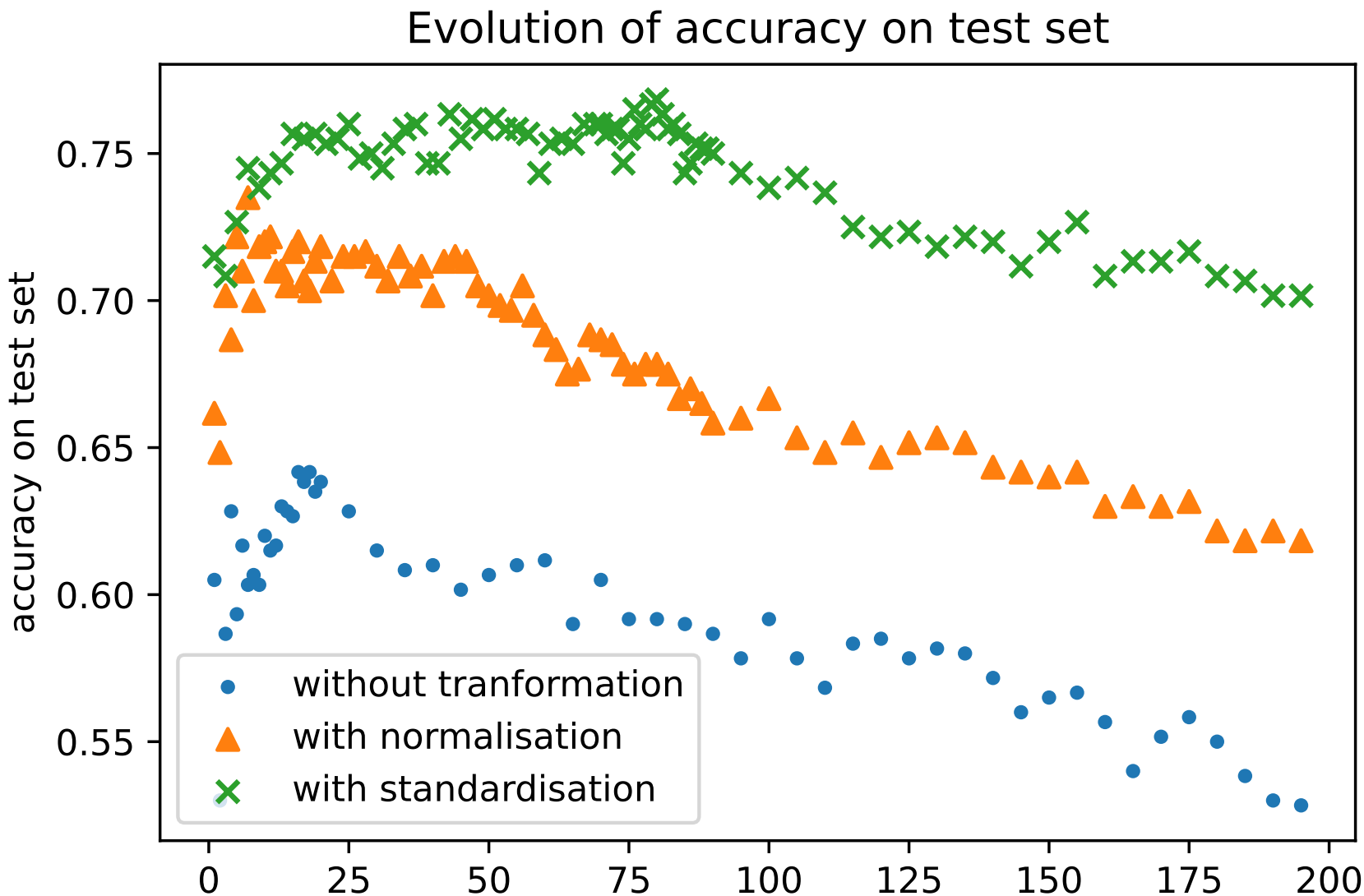
- Normalisation :
  - $x \leftarrow \frac{x - \mu}{\sigma}$
  - new mean : 0 ; new standard deviation : 1
- Standardisation :
  - $x \leftarrow \frac{x - \mu}{s}$
  - new mean : 0 ; new mad : 1

Normalisation and standardisation

- Normalisation :
  - $x \leftarrow \dfrac{x - \mu}{\sigma}$
  - new mean : 0 ; new standard deviation : 1
- Standardisation :
  - $x \leftarrow \dfrac{x - \mu}{s}$
  - new mean : 0 ; new mad : 1
- For the dataset Xtrain :
  - normalisation : $\forall i \; x_i = \dfrac{x_i - \mu_i}{\sigma_i}$
  - standardisation : $\forall i \; x_i = \dfrac{x_i - \mu_i}{s_i}$

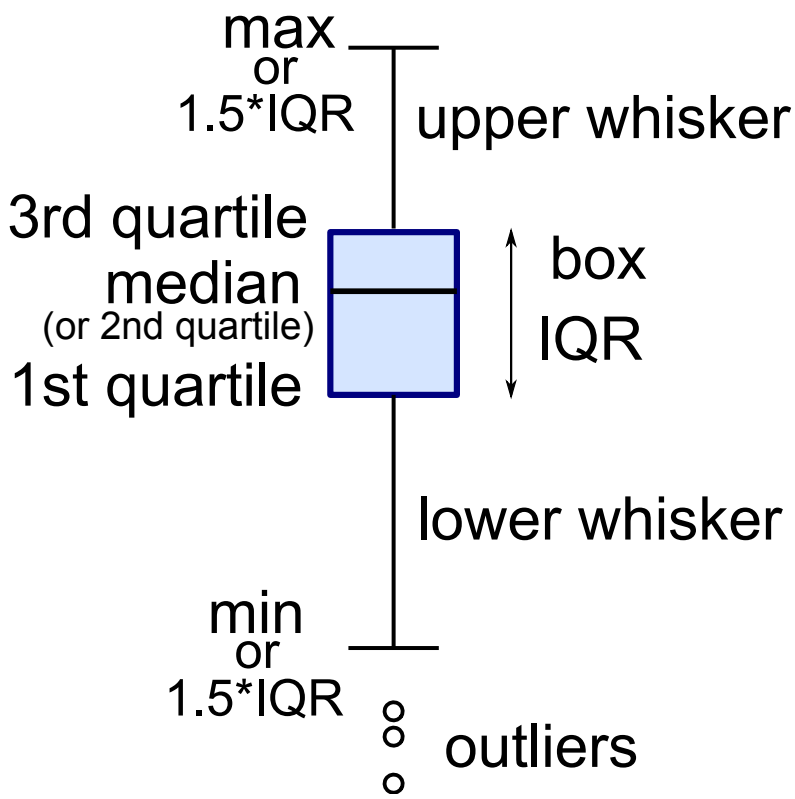
Impact on the accuracy for test set

Experiment using 200 training samples, 200 test samples, dataset Speech Commands, classes 'cat', 'dog' and 'bird'.

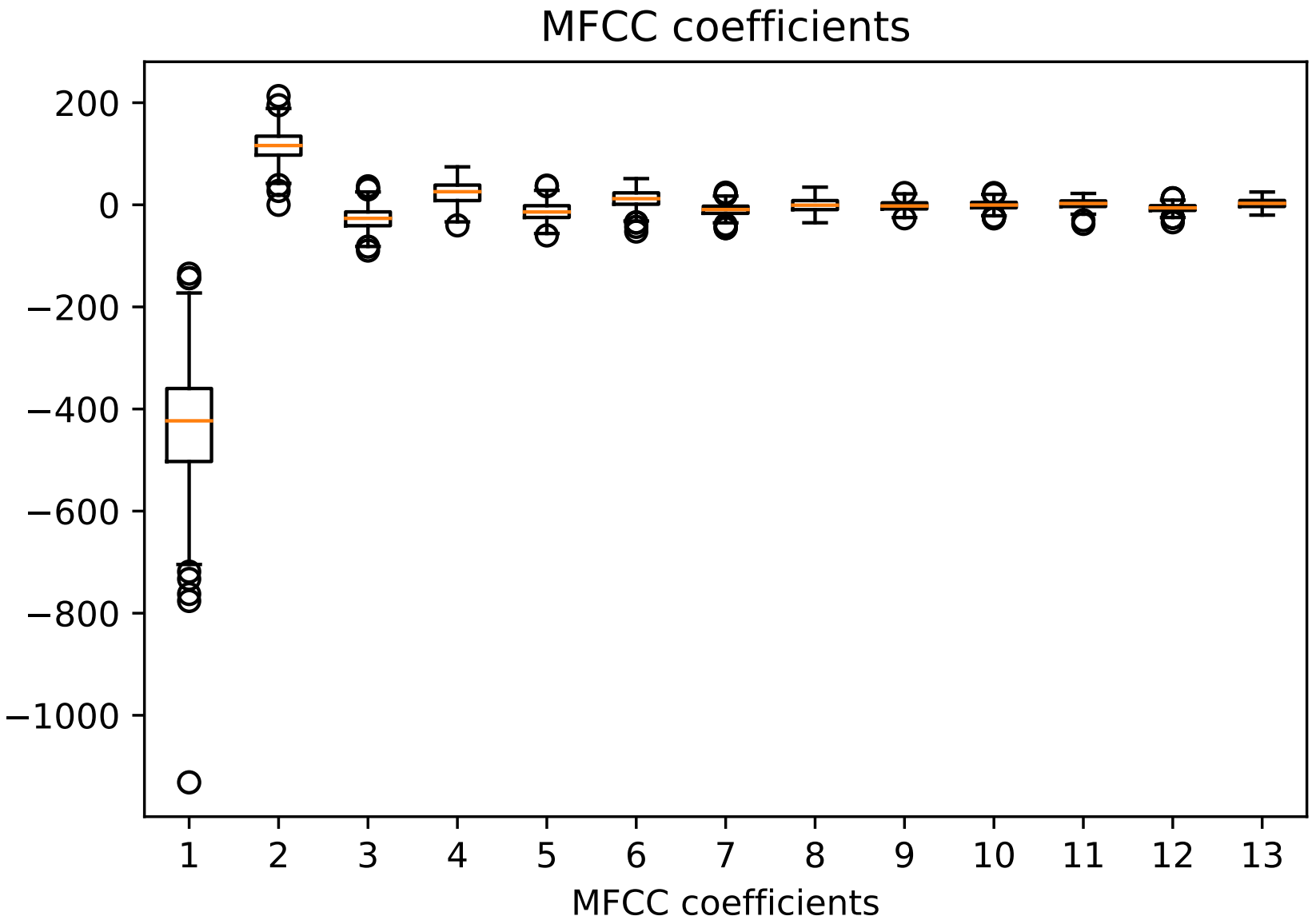


How to read a box plot ?

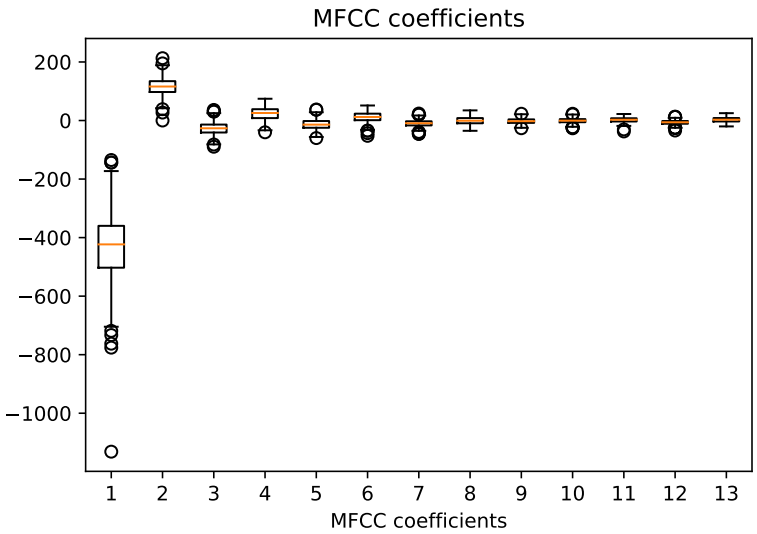
Also called *boîte à moustaches*

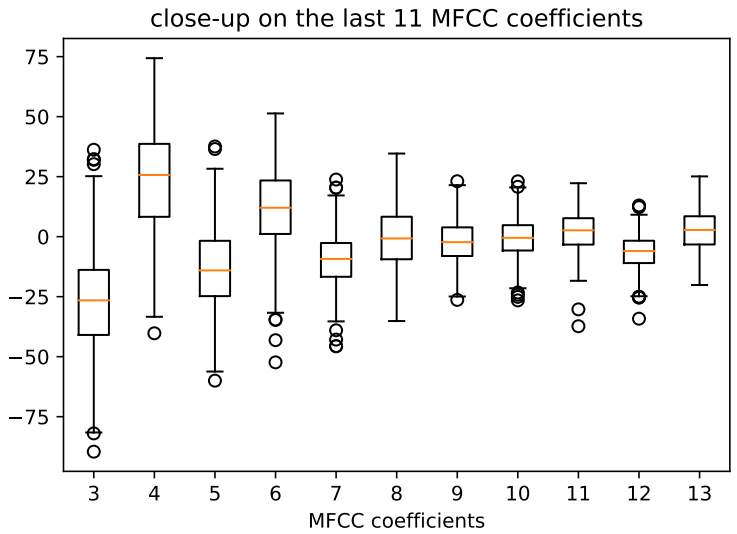


Why such differences ?

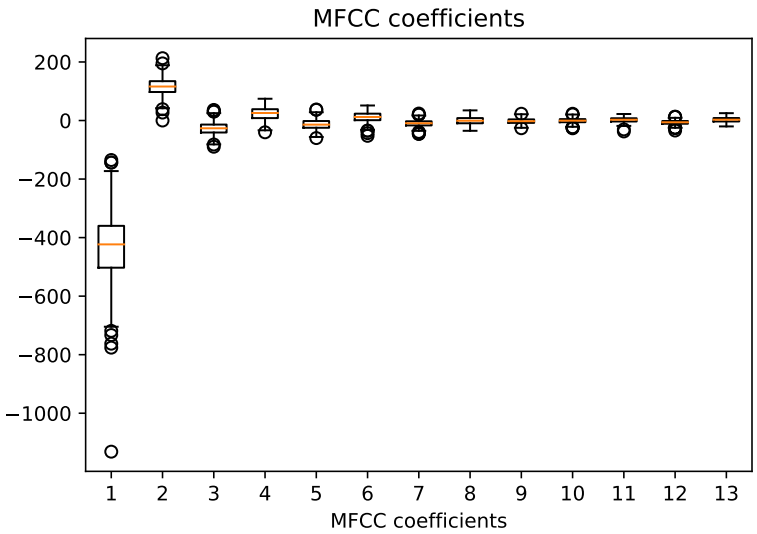


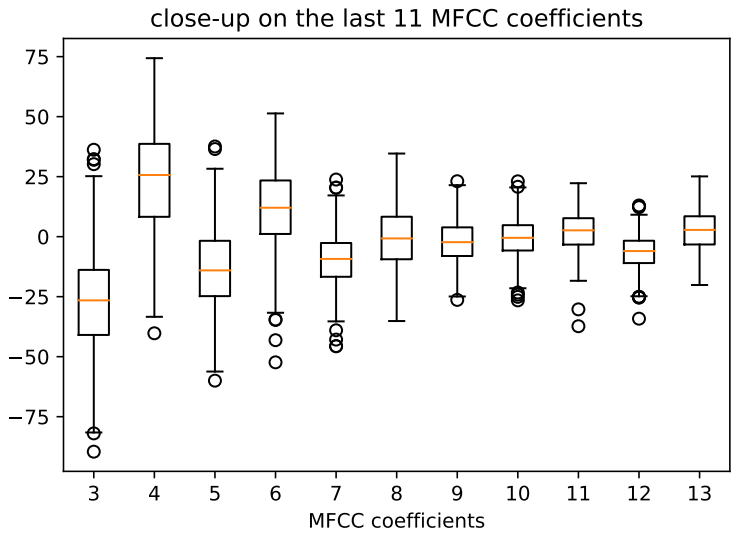
Why such differences ?

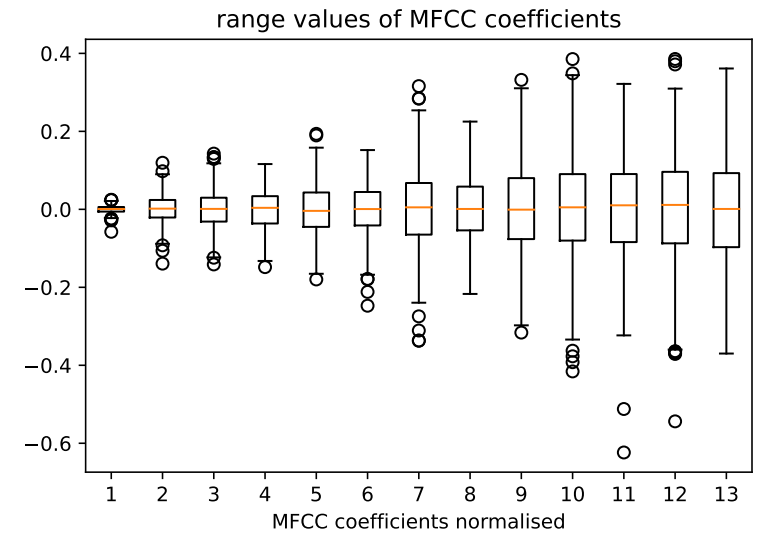


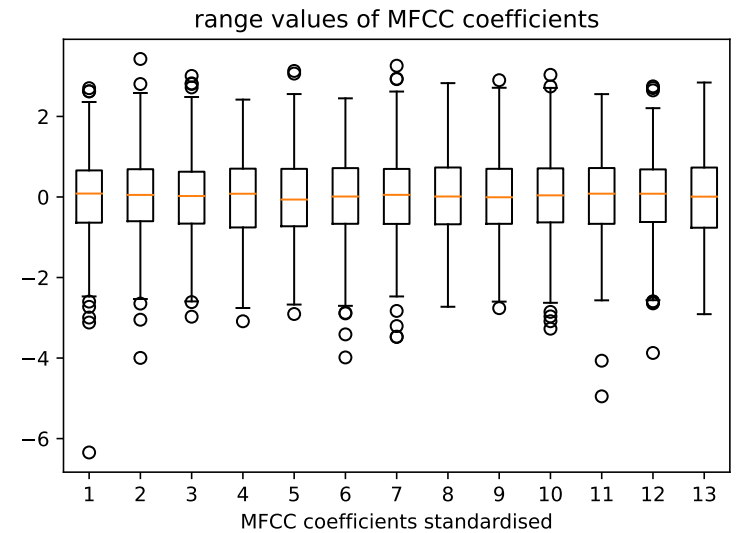


Why such differences ?

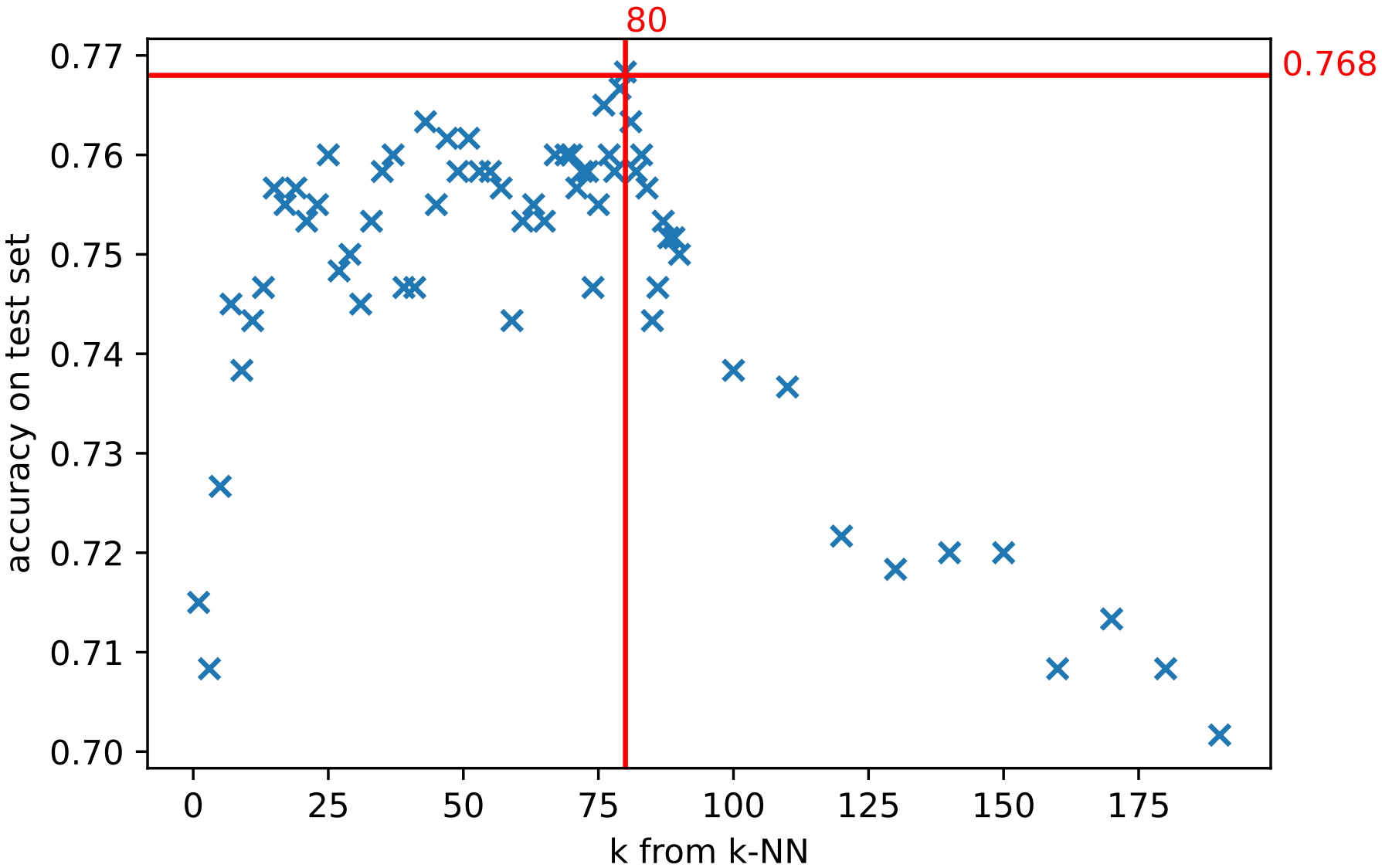






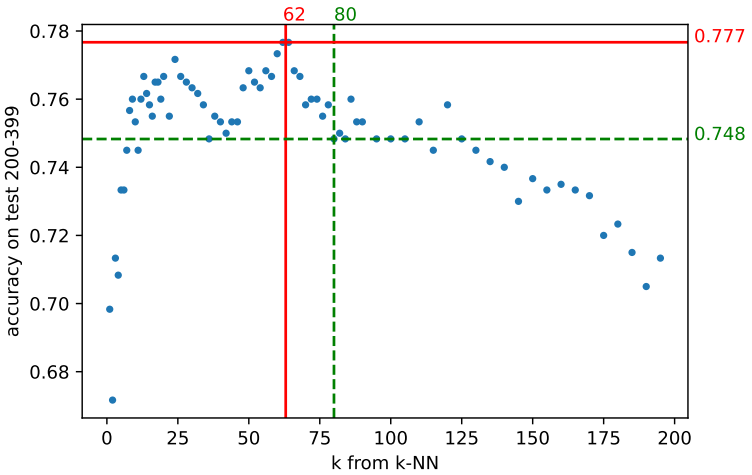


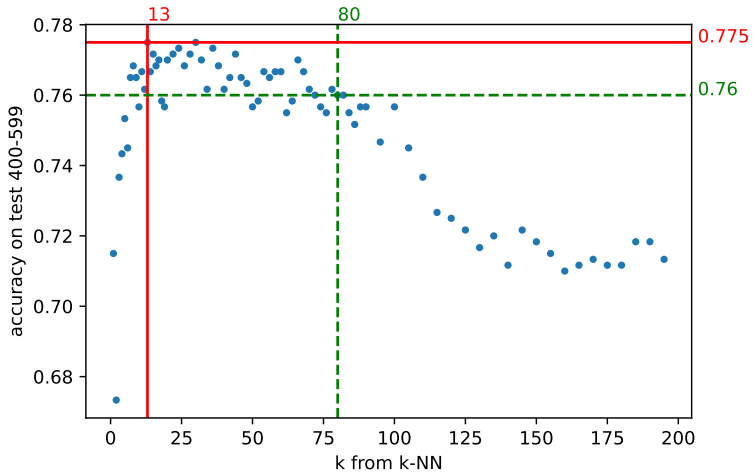
Best k value


80
0.768
0.77
0.76
0.75
0.74
0.73
0.72
0.71
0.70
accuracy on test set
0
25
50
75
100
125
150
175
k from k-NN

Best k value

- For standardised data : best k value is 80 for an accuracy of 76.8%
  - was computed from test data (last 200 samples)
  - train data where data from 0 to 199
- Let's test on other data (not training data, not used for computing k)





- Necessity of splitting data into : train/valid/test
  - otherwise, metrics are over-estimated
    - train for training
    - valid for hyperparameters estimation
    - test for evaluation of metrics

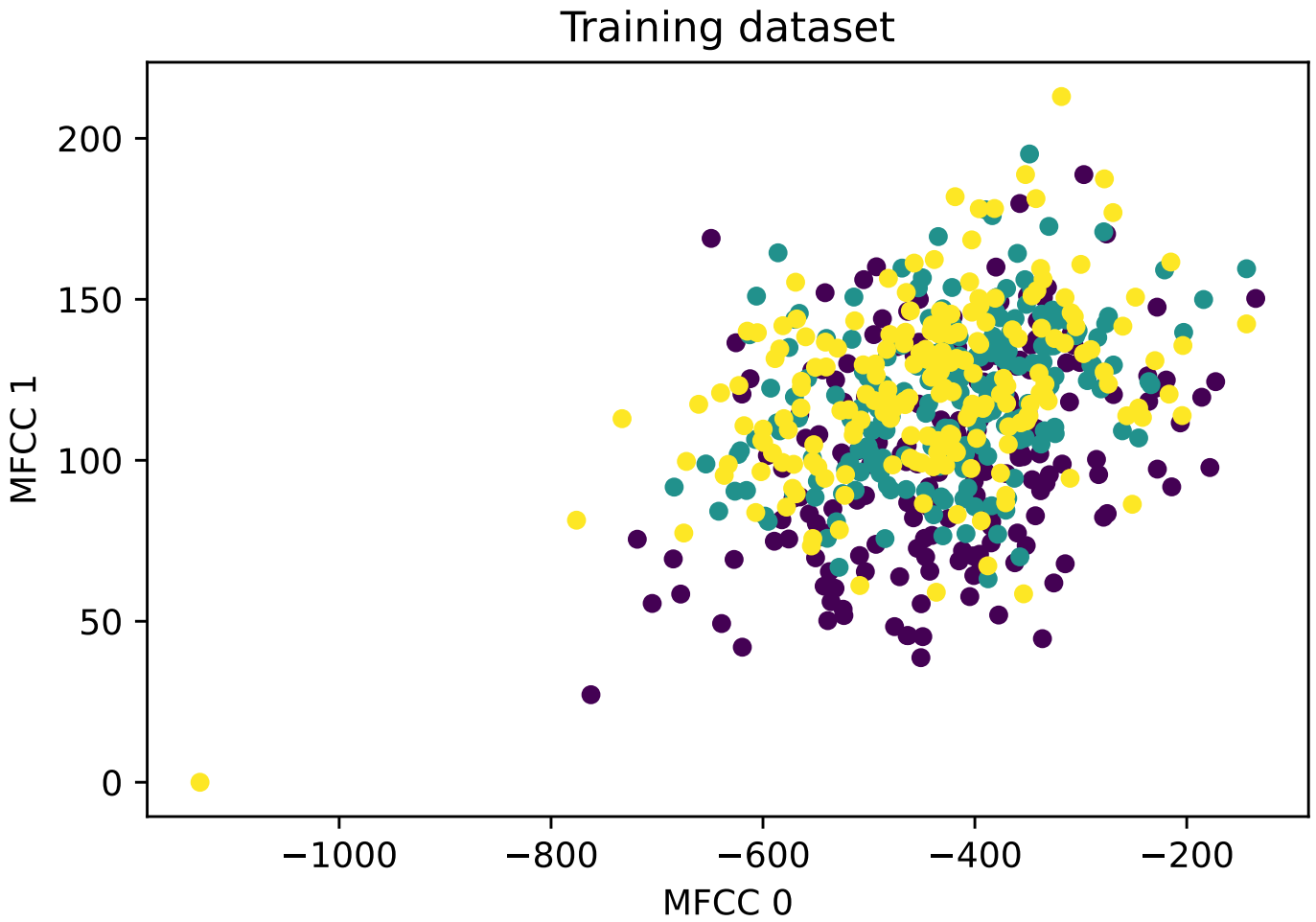
To remember

- It is important to **standardise** the data
- It is important to split dataset into :
  - **train** for training
  - **valid** for hyperparameters estimation
  - **test** for evaluation of metrics

Outline

Vu en TP

```
plt.scatter(Xtrain[:, 0], Xtrain[:, 1], marker="o", c=y)
```



Can we plot it better ... on a screen ?

Covariance

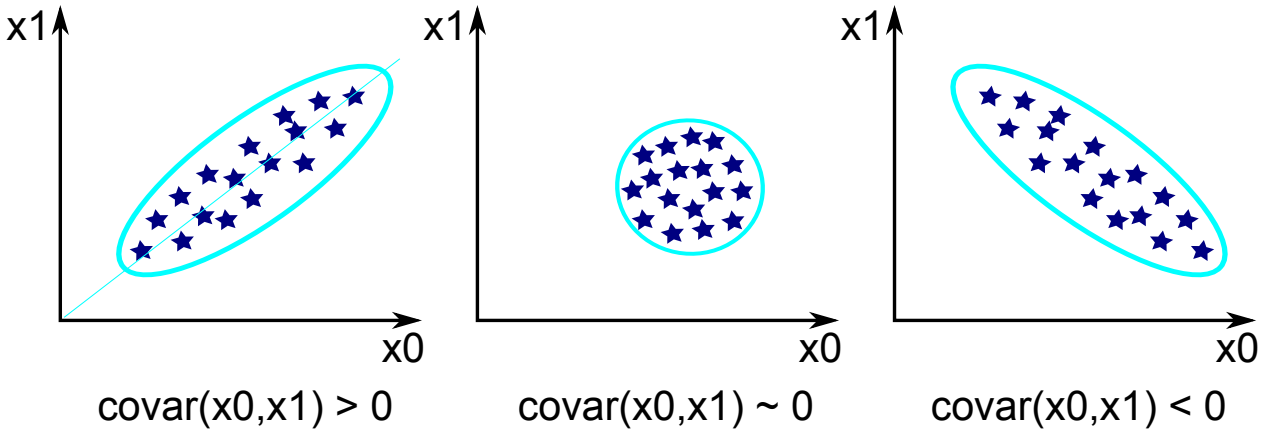
- covariance between 2 variables x_0 and x_1 :
  - measure of the linear relationship between two random variables
  - $\text{covar}(x_0, x_1) = \text{E}[(x_0 - \mu_0)(x_1 - \mu_1)] = \frac{1}{n} \sum_j (x_0^j - \mu_0)(x_1^j - \mu_1)$

Covariance

- covariance between 2 variables x_0 and x_1 :
  - measure of the linear relationship between two random variables
  - $\text{covar}(x_0, x_1) = \text{E}[(x_0 - \mu_0)(x_1 - \mu_1)] = \frac{1}{n}\sum_j (x_0^j - \mu_0)(x_1^j - \mu_1)$
- Example of linearly correlated variables : $x_1^j = \lambda x_0^j$
  - $\text{covar}(x_0, x_1) = \sum_j (x_0^j - \mu_0)(x_1^j - \mu_1) = \lambda \, \text{var}(x_0)$
    - high value : means correlation between the 2 variables

Covariance

- covariance between 2 variables x_0 and x_1 :
  - measure of the linear relationship between two random variables
  - $\text{covar}(x_0, x_1) = \text{E}[(x_0 - \mu_0)(x_1 - \mu_1)] = \frac{1}{n}\sum_j (x_0^j - \mu_0)(x_1^j - \mu_1)$
- Example of linearly correlated variables : $x_1^j = \lambda x_0^j$
  - $\text{covar}(x_0, x_1) = \sum_j (x_0^j - \mu_0)(x_1^j - \mu_1) = \lambda \, \text{var}(x_0)$
    - high value : means correlation between the 2 variables
- Example of non correlated variables : $E[x_0 * x_1] = E[x_0] * E[x_1]$
  - thus $\text{covar}(x_0, x_1) = 0$



Mean, variance and covariance : dim 2

- n samples of dimension 2 : $\boldsymbol{x}^j = [x_0^j, x_1^j]$ with $0 \leq j < n$

Mean, variance and covariance : dim 2

- n samples of dimension 2 : $\boldsymbol{x}^j = [x_0^j, x_1^j]$ with $0 \leq j < n$
- mean of samples : $\boldsymbol{\mu} = 0.5\,[x_0^0 + x_0^1, x_1^0 + x_1^1]$
- variances :

Mean, variance and covariance : dim 2

- n samples of dimension 2 : $\boldsymbol{x}^j = [x_0^j, x_1^j]$ with $0 \leq j < n$
- mean of samples : $\boldsymbol{\mu} = 0.5\,[x_0^0 + x_0^1, x_1^0 + x_1^1]$
- variances :
  - $\text{var}(x_0) = \text{covar}(x_0, x_0) = \sigma_0^2 = \frac{1}{n}\sum_j (x_0^j - \mu_0)^2$
  - $\text{var}(x_1) = \text{covar}(x_1, x_1) = \sigma_1^2 = \frac{1}{n}\sum_j (x_1^j - \mu_1)^2$
- covariance matrix :

$$\Sigma = \begin{pmatrix} \sigma_0^2 & \text{covar}(x_0, x_1) \\ \text{covar}(x_0, x_1) & \sigma_1^2 \end{pmatrix}$$

- variance : $\text{var}(\boldsymbol{x}) = \text{tr}(\Sigma) = \sigma_0^2 + \sigma_1^2$

Variance-covariance matrix

- original variables of dimension $p \geq 2$: $\mathrm{X^j} = [x_0^j \dots x_{p-1}^j]$
- variance-covariance matrix : symmetric matrix of dim $p \times p$:

$$\Sigma = \begin{pmatrix} \mathrm{var}(x_0) & \mathrm{covar}(x_0, x_1) & \dots & \mathrm{covar}(x_0, x_{p-1}) \\ \mathrm{covar}(x_0, x_1) & \mathrm{var}(x_1) & \dots & \mathrm{covar}(x_0, x_{p-1}) \\ \vdots & \vdots & \ddots & \vdots \\ \mathrm{covar}(x_0, x_{p-1p}) & \mathrm{covar}(x_1, x_{p-1}) & \dots & \mathrm{var}(x_{p-1}) \end{pmatrix}$$

- variance : $\mathrm{var}(\boldsymbol{x}) = \mathrm{tr}(\Sigma) = \sum_i \sigma_i^2$

PCA (1901 Karl Pearson, 1936 H. Hotelling)

- Unsupervised
- Analysis of variance-covariance matrix
- Reducing the dimension of data
- Visualisation of data if the reduced dimension is 2 or 3
- Interpretation : dependance between variables
- PCA : often as pre-processing

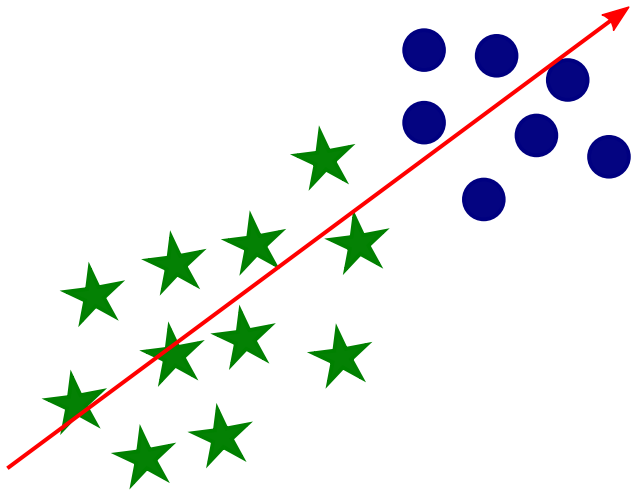
Explained variance

$$\Sigma = \begin{pmatrix} \text{var}(x_0) & \text{covar}(x_0, x_1) & \dots & \text{covar}(x_0, x_{p-1}) \\ \text{covar}(x_0, x_1) & \text{var}(x_1) & \dots & \text{covar}(x_0, x_{p-1}) \\ \vdots & \vdots & \ddots & \vdots \\ \text{covar}(x_0, x_{p-1p}) & \text{covar}(x_1, x_{p-1}) & \dots & \text{var}(x_{p-1}) \end{pmatrix}$$

- variance $= \text{tr}(\Sigma) = \sum_i \sigma_i^2$
  - symmetric squared matrix : diagonalization is possible !
  - there exists a basis of orthogonal vectors where the covariance matrix is diagonal
    - these vectors are eigenvectors of Σ
    - elements on the diagonal are eigenvalues
    - variance $= \sum_k \lambda_k$
- Idea of PCA
  - diagonalisation of Σ
    - order eigenvalues by decreasing order
  - if 0 is an eigenvalue : the corresponding dimensions can be removed
  - the lower eigenvalues do not contribute a lot to the variance

Geometrical interpretation

- original variables : $x_1, x_2, .., x_p$
- principal components : $c_1, c_2, ...c_k, ..., c_q$ with $q \leq p$
- $c_k = \sum_j a_{jk} x_j$ with :
  - c_k and c_j not correlated
  - maximum variance and
  - decreasing importance

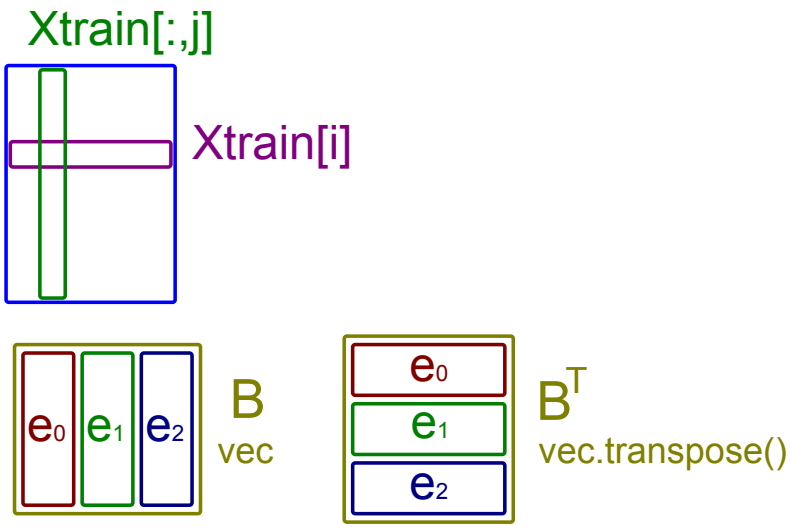


PCA : python code

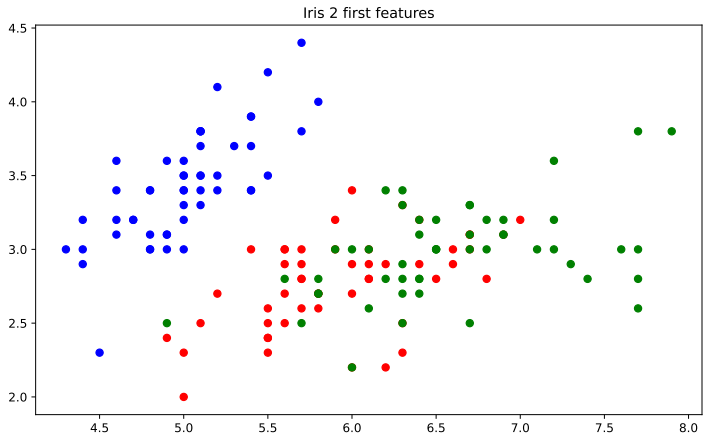
- standardise the data
- compute the covariance matrix (numpy.cov). Caution : read the doc to adapt to Xtrain shape/order.
  - compute it's eigenvalues and eigenvectors (numpy.linalg.eig)
  - sort the eigenvalues in decreasing order (numpy.argsort)
  - select some of the eigenvalues (2 for visualisation on your screen)
  - use the corresponding eigenvectors for transforming the data (numpy.matmult or @)
  - compare the obtained variance with the original variance
- for the validation or test dataset
  - apply the exact same transformation
  - do not recompute covariance or eigenvalues and eigenvectors

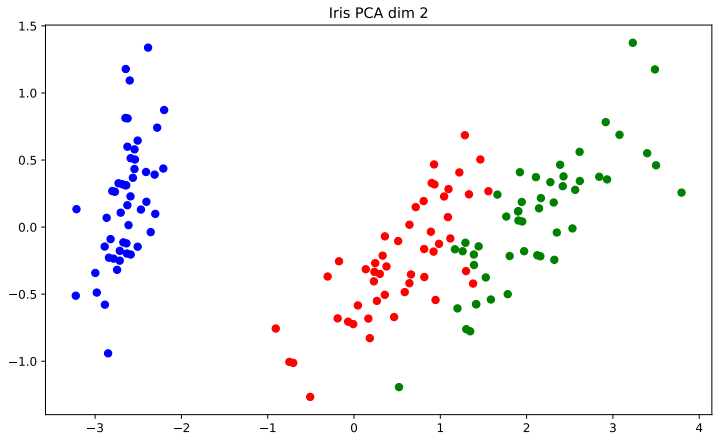
Details on `numpy.linalg.eig`

- the function returns :
  - `val, vec = np.linalg.eig(myCovMatrix)`
  - an array of eigen values : `val[i]` is the i^{th} eigen value
  - an array of eigen vectors : `vec[:,i]` is the i^{th} eigen vector corresponding to the ith eigen value
- after ordering both eigenvals and eigenvectors, we can reduce the target space to a smaller dimension (equal dimension is also possible)
- in order to express a sample s in the new basis : Bs
  - B^T is the matrix for basis change
  - we need to transpose `vec`
  - we need to transpose `Xtrain`
  - and transpose the result :
    - `Xtrain@vec`

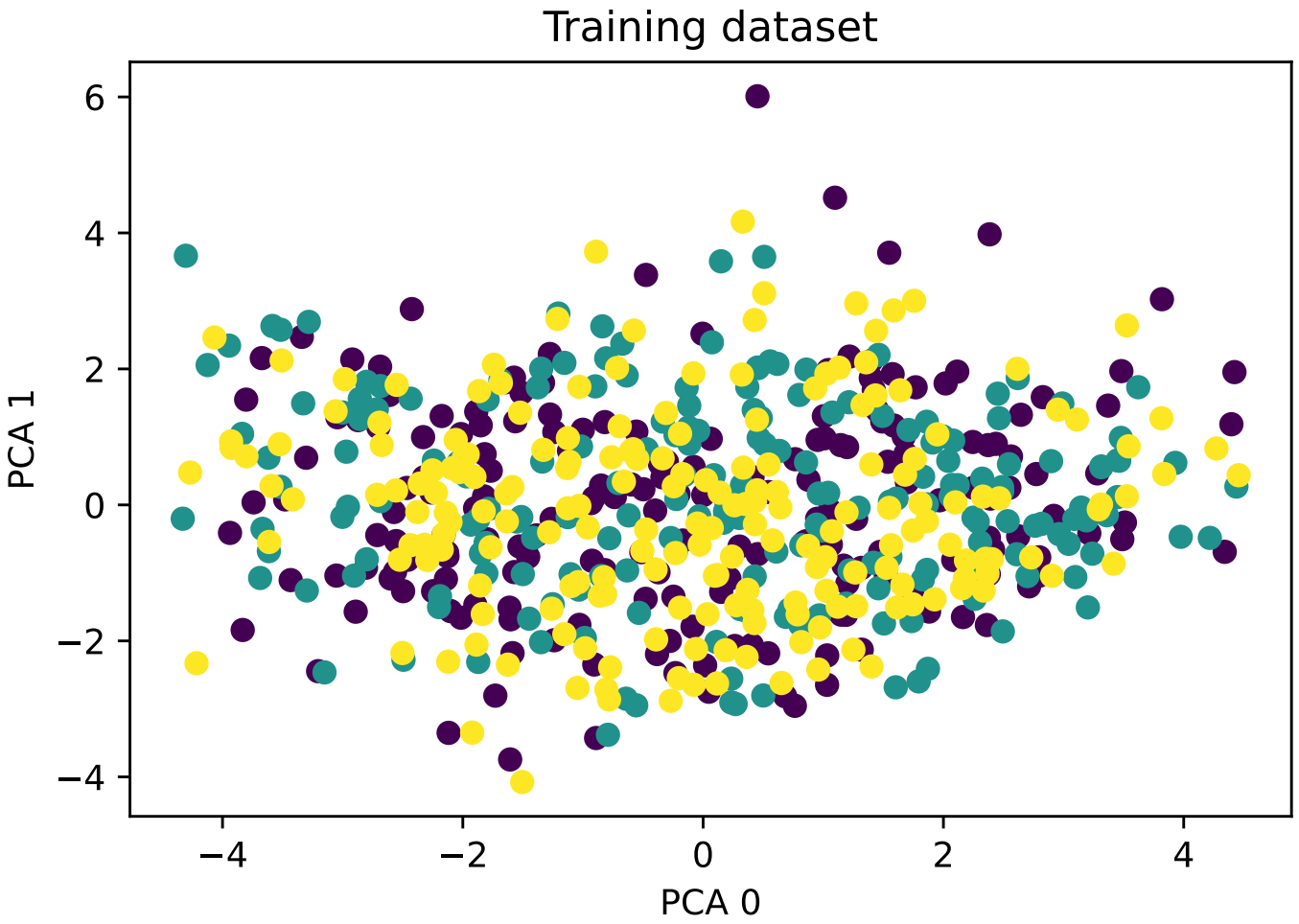


Example : Iris dataset





Example : sounds 'cat', 'dog', 'bird'



Examples : variance explained

- PCA on Iris dataset (4 features) :
  - if we perform the PCA on dimension 4 (not very useful) :
  - with 4 components : 100% of the variance is explained
  - with 3 components : 99.5%
  - with 2 components : 97.8%
- PCA on 'cat', 'dog', 'bird' (13 features) :
  - with 13 components : 100%
  - with 11 components : 96%
  - with 2 components : 47%

Conclusion

- PCA is for dimension reduction
  - in dimension 2 for visualisation on screen
  - or any other smaller dimension than the original one
  - simple and quite fast
- PCA is not the best dimension reduction algorithm
  - only linear transformation
  - do not guaranty that close (resp. distant) samples in the original space will be close (resp. distant) in the projected space.
  - we will study tSNE next year