VSB TECHNICAL UNIVERSITY OF OSTRAVA | IT4INNOVATIONS NATIONAL SUPERCOMPUTING CENTER

Introduction to scikit-learn

Georg Zitzlsberger georg.zitzlsberger@vsb.cz

26-03-2021

EUROPEAN UNION
European Structural and Investment Funds
Operational Programme Research,
Development and Education

MINISTRY OF EDUCATION, YOUTH AND SPORTS

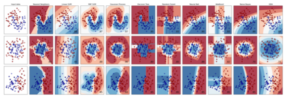
Agenda

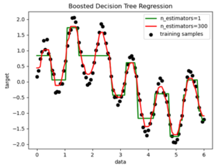
What is scikit-learn?

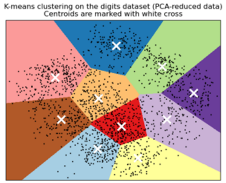
- Simple and efficient tools for predictive data analysis
  - Machine Learning methods
  - Data processing
  - Visualization
- Accessible to everybody, and reusable in various contexts
  - Documented API with lot's of examples
  - Not bound to Training frameworks (e.g. Tensorflow, Pytorch)
  - Building blocks for your data analysis
- Built on *NumPy*, *SciPy*, and *matplotlib*
  - No own data types (unlike Pandas)
  - Benefit from NumPy and SciPy optimizations
  - Extends the most common visualisation tool
- Open source, commercially usable - BSD license

Tools of scikit-learn

- **Classification:**
  Categorizing objects to one or more classes.
  - Support Vector Machines (SVM)
  - Nearest Neighbors
  - Random Forest
  - . . .
- **Regression:**
  Prediction of one (uni-) or more (multi-variate) continuous-valued attributes.
  - Support Vector Regression (SVR)
  - Nearest Neighbors
  - Random Forest
  - . . .
- **Clustering:**
  Group objects of a set.
  - k-Means
  - Spectral Clustering
  - Mean-Shift
  - . . .

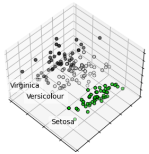


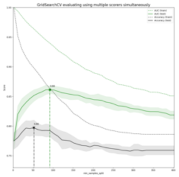


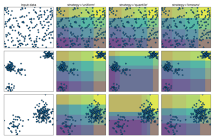


Tools of scikit-learn - cont'd

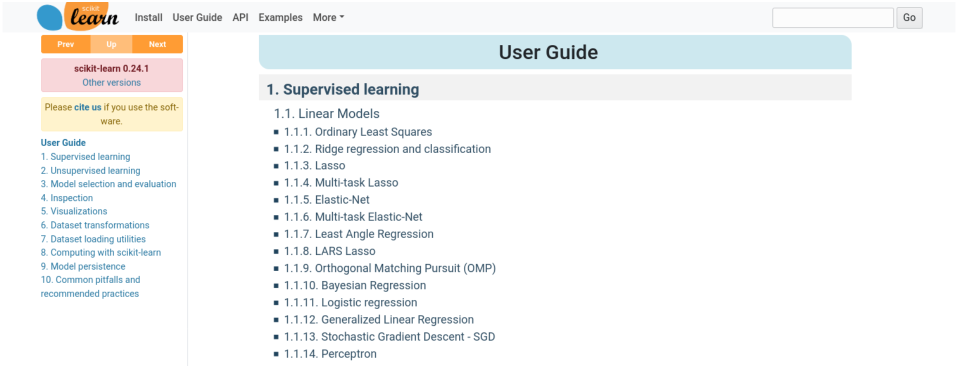
- **Dimensionality reduction:**
  Reducing the number of random variables.
  - Principal Component Analysis (PCA)
  - Feature Selection
  - non-Negative Matrix Factorization
  - ...
- **Model selection:**
  Compare, validate and choose parameters/models.
  - Grid Search
  - Cross Validation
  - ...
- **Pre-Processing:**
  Prepare/transform data before training models.
  - Conversion
  - Normalization
  - Feature Extraction
  - ...







User Guide



scikit-learn 0.24.1
Other versions

Please **cite us** if you use the software.

User Guide
1. Supervised learning
2. Unsupervised learning
3. Model selection and evaluation
4. Inspection
5. Visualizations
6. Dataset transformations
7. Dataset loading utilities
8. Computing with scikit-learn
9. Model persistence
10. Common pitfalls and recommended practices

User Guide

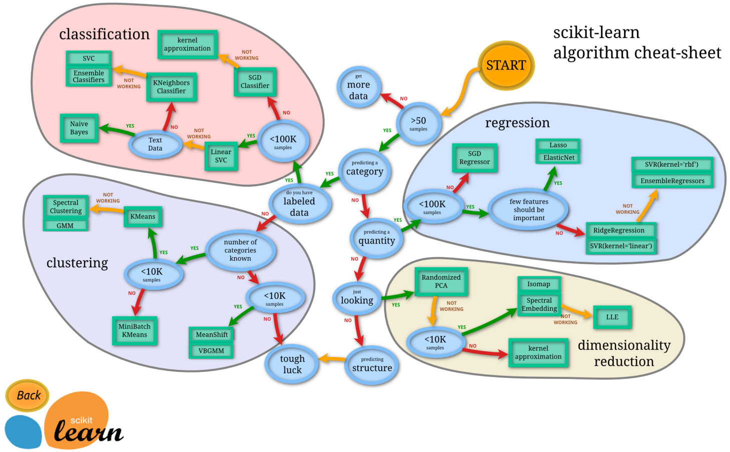
1. Supervised learning

1.1. Linear Models
- 1.1.1. Ordinary Least Squares
- 1.1.2. Ridge regression and classification
- 1.1.3. Lasso
- 1.1.4. Multi-task Lasso
- 1.1.5. Elastic-Net
- 1.1.6. Multi-task Elastic-Net
- 1.1.7. Least Angle Regression
- 1.1.8. LARS Lasso
- 1.1.9. Orthogonal Matching Pursuit (OMP)
- 1.1.10. Bayesian Regression
- 1.1.11. Logistic regression
- 1.1.12. Generalized Linear Regression
- 1.1.13. Stochastic Gradient Descent - SGD
- 1.1.14. Perceptron

The User Guide can be found ▸ here

VSB TECHNICAL UNIVERSITY OF OSTRAVA | IT4INNOVATIONS NATIONAL SUPERCOMPUTING CENTER

Choosing the Right Estimator



(Image: scikit-learn.org)

Linked map can be found ▸ here

How to Get scikit-learn

- Open Source (BSD License) available on ▸ Github
- Current version: *0.24.2*
- Easy install via PIP or Conda for Windows, macOS and Linux, e.g:
  `$ pip install scikit-learn` or
  `$ conda install -c intel scikit-learn`

Programming Model

- Builds on *NumPy*, *SciPy* and *matplotlib*:
  - Avoids conversion of data types
  - Can be integrated seamlessly, even with Tensorflow and Pytorch
  - Benefits from performance optimizations of BLAS, FFT, etc. optimizations
- scikit-learn available as Python module:

```
import sklearn
sklearn.show_versions()

System:
    python: 3.8.6 | packaged by conda-forge | (default, Dec 26 2020, 05:05:16)  [GCC 9.3.0]
executable: /opt/conda/bin/python
   machine: Linux-3.10.0-1127.13.1.el7.x86_64-x86_64-with-glibc2.10

Python dependencies:
          pip: 20.3.3
   setuptools: 49.6.0.post20201009
      sklearn: 0.24.0
        numpy: 1.19.5
        scipy: 1.5.3
       Cython: 0.29.21
       pandas: 1.1.5
   matplotlib: 3.3.3
       joblib: 1.0.0
threadpoolctl: 2.1.0

Built with OpenMP: True
```

- Typical input `(n_samples, n_features)`, but others are also possible

Example Datasets

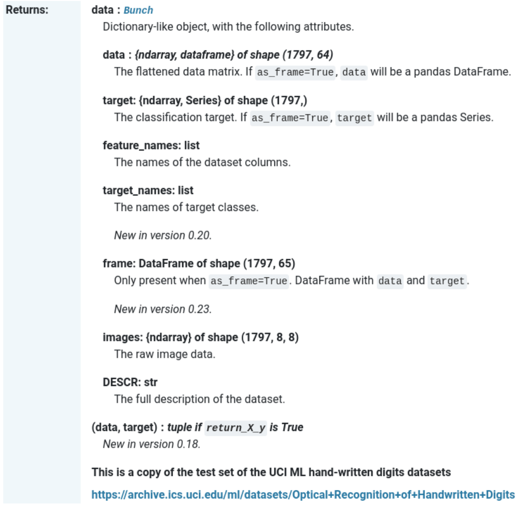
- Easy access to ▸ "toy" datasets via `sklearn.datasets`:
  - Boston house prices dataset
  - Iris plants dataset
  - Diabetes dataset
  - Optical recognition of handwritten digits dataset
  - Linnerrud dataset
  - Wine recognition dataset
  - Breast cancer wisconsin (diagnostic) dataset
- Loading via:

Function	**Description**
`load_boston(*[, return_X_y])`	Load and return the boston house-prices dataset (regression).
`load_iris(*[, return_X_y, as_frame])`	Load and return the iris dataset (classification).
`load_diabetes(*[, return_X_y, as_frame])`	Load and return the diabetes dataset (regression).
`load_digits(*[, n_class, return_X_y, as_frame])`	Load and return the digits dataset (classification).
`load_linnerud(*[, return_X_y, as_frame])`	Load and return the physical excercise linnerud dataset.
`load_wine(*[, return_X_y, as_frame])`	Load and return the wine dataset (classification).
`load_breast_cancer(*[, return_X_y, as_frame])`	Load and return the breast cancer wisconsin dataset (classification).

Using Example Datasets

- Convention:
  - X: Data for training/prediction
  - y: Label in case of supervised learning (aka. target)
  - n_class: How many classes from the set should be retrieved
  - return_X_y: Boolean whether tuple of data and label is desired
  - as_frame: Boolean whether data should be a Pandas DataFrame
- Example:

```python
import sklearn.datasets

sk_digits = sklearn.datasets.load_digits(n_class=2,
                                         return_X_y=True,
                                         as_frame=False)
print(sk_digits)

(array([[ 0.,  0.,  5., ...,  0.,  0.,  0.],
        ...,
        [ 0.,  0.,  6., ...,  6.,  0.,  0.]]),
 array([0, 1, 0, 1, 0, 1, 0, 0,
        ...,
        1, 1, 1, 1, 1, 0, 1, 0]))
```



Returns:

data : *Bunch*
Dictionary-like object, with the following attributes.

data : *{ndarray, dataframe} of shape (1797, 64)*
The flattened data matrix. If `as_frame=True`, `data` will be a pandas DataFrame.

target: {ndarray, Series} of shape (1797,)
The classification target. If `as_frame=True`, `target` will be a pandas Series.

feature_names: list
The names of the dataset columns.

target_names: list
The names of target classes.

New in version 0.20.

frame: DataFrame of shape (1797, 65)
Only present when `as_frame=True`. DataFrame with `data` and `target`.

New in version 0.23.

images: {ndarray} of shape (1797, 8, 8)
The raw image data.

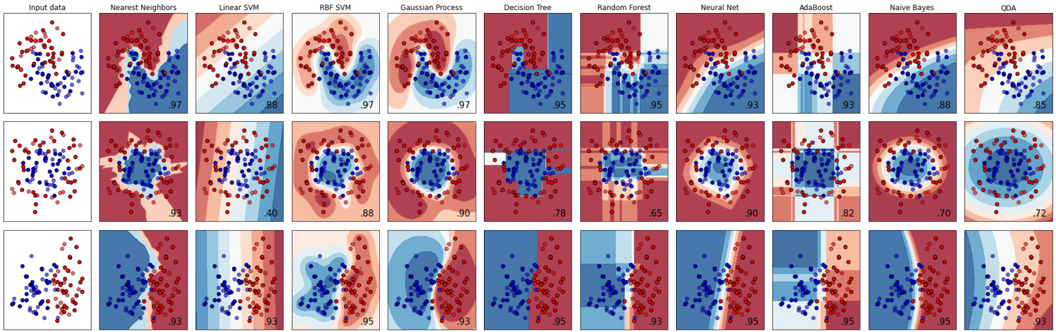
DESCR: str
The full description of the dataset.

(data, target) : *tuple if `return_X_y` is True*
New in version 0.18.

This is a copy of the test set of the UCI ML hand-written digits datasets
https://archive.ics.uci.edu/ml/datasets/Optical+Recognition+of+Handwritten+Digits

Classification

- Supervised:
  Label information is available and can be used for learning
- Unsupervised:
  No (initial) labels and learning needs to structure data on its own
- Many classification methods exist:



From scikit-learn documentation: Classifier comparison

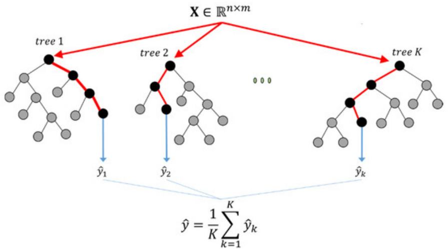
Regression

- Classification vs. Regression[1]:
  - Classify for categorical output
  - Regression: predicting continuous-valued attribute(s)
- Can be "by-products" of classification methods, e.g.: `RandomForestClassifier` and `RandomForestRegressor`, or `SVC` and `SVR`

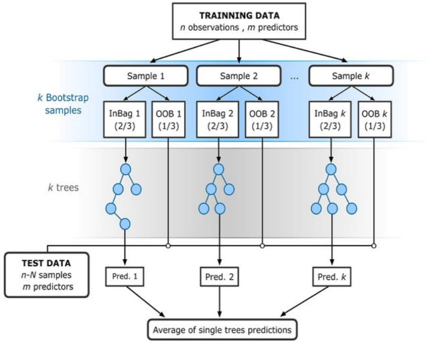
[1]As scikit-learn regards it.

Regression Example: Random Forest

- Ensemble of decision trees
- Perturb-and-combine technique applied to trees
- Considers diverse set of classifiers
- Randomization is achieved by selection of different classifiers
- Prediction is majority vote or average over all trees
- Easily extends to multi-output problems


$\mathbf{X} \in \mathbb{R}^{n\times m}$
tree 1
tree 2
tree K
$\hat{y}_1$
$\hat{y}_2$
$\hat{y}_K$
$\hat{y} = \frac{1}{K}\sum_{k=1}^{K}\hat{y}_k$

Process Variable Importance Analysis by Use of Random Forests in a Shapley Regression Framework, Aldrich

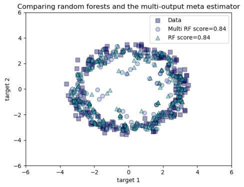


Modelling interannual variation in the spring and autumn land surface phenology of the European forest, Rodriguez-Galiano, et al.

Random Forest Example

```python
import numpy as np
import matplotlib.pyplot as plt
from sklearn.ensemble import RandomForestRegressor
from sklearn.model_selection import train_test_split
from sklearn.multioutput import MultiOutputRegressor


# Create a random dataset
rng = np.random.RandomState(1)
X = np.sort(200 * rng.rand(600, 1) - 100, axis=0)
y = np.array([np.pi * np.sin(X).ravel(),
              np.pi * np.cos(X).ravel()]).T
y += (0.5 - rng.rand(*y.shape))

X_train, X_test, y_train, y_test = train_test_split(
    X, y, train_size=400, test_size=200, random_state=4)

max_depth = 30
regr_multirf = MultiOutputRegressor(
                RandomForestRegressor(n_estimators=100,
                                      max_depth=max_depth,
                                      random_state=0))
regr_multirf.fit(X_train, y_train)

regr_rf = RandomForestRegressor(n_estimators=100,
                                max_depth=max_depth,
                                random_state=2)
regr_rf.fit(X_train, y_train)

# Predict on new data
y_multirf = regr_multirf.predict(X_test)
y_rf = regr_rf.predict(X_test)
```

```python
# Plot the results
plt.figure()
s = 50
a = 0.4
plt.scatter(y_test[:, 0], y_test[:, 1], edgecolor='k',
            c="navy", s=s, marker="s", alpha=a, label="Data")
plt.scatter(y_multirf[:, 0], y_multirf[:, 1], edgecolor='k',
            c="cornflowerblue", s=s, alpha=a,
            label="Multi␣RF␣score=%.2f" % regr_multirf.score(X_test,
                                                               y_test))
plt.scatter(y_rf[:, 0], y_rf[:, 1], edgecolor='k',
            c="c", s=s, marker="^", alpha=a,
            label="RF␣score=%.2f" % regr_rf.score(X_test, y_test))
plt.xlim([-6, 6])
plt.ylim([-6, 6])
plt.xlabel("target␣1")
plt.ylabel("target␣2")
plt.title("Comparing␣random␣forests␣and␣the␣multi-output␣" +
          "meta␣estimator")
plt.legend()
plt.show()
```

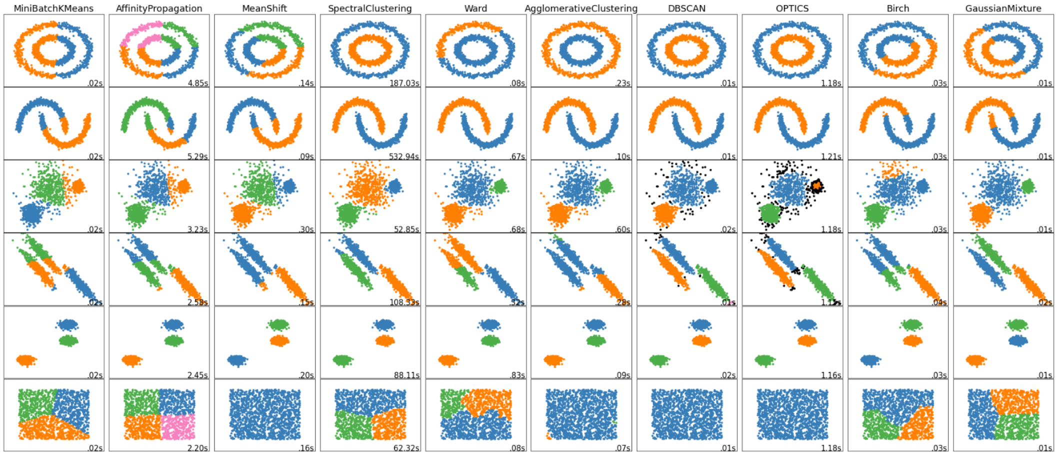


Python source code:

▸ Random Forest Regression

Clustering

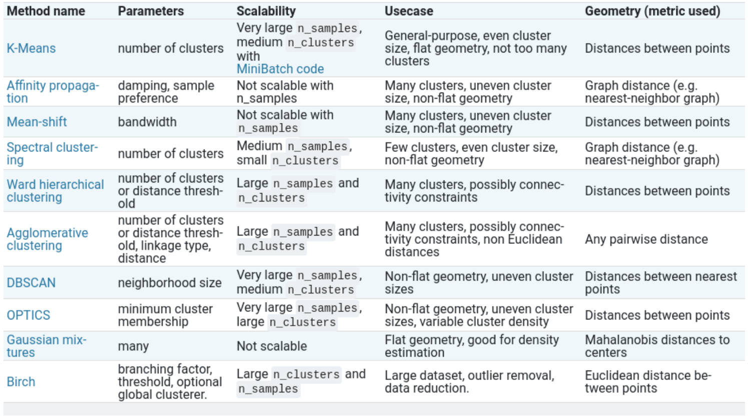
- Many clustering methods exist:



From scikit-learn documentation: ▸ Clustering comparison

Clustering

- Unsupervised: Find clusters (set of classes) automatically
- Clustering is applied in two steps:
  1. Train (i.e. identify) cluster with training data
  2. Retrieve the labels/metrics from the trained model



Method name	Parameters	Scalability	Usecase	Geometry (metric used)
K-Means	number of clusters	Very large `n_samples`, medium `n_clusters` with MiniBatch code	General-purpose, even cluster size, flat geometry, not too many clusters	Distances between points
Affinity propagation	damping, sample preference	Not scalable with n_samples	Many clusters, uneven cluster size, non-flat geometry	Graph distance (e.g. nearest-neighbor graph)
Mean-shift	bandwidth	Not scalable with `n_samples`	Many clusters, uneven cluster size, non-flat geometry	Distances between points
Spectral clustering	number of clusters	Medium `n_samples`, small `n_clusters`	Few clusters, even cluster size, non-flat geometry	Graph distance (e.g. nearest-neighbor graph)
Ward hierarchical clustering	number of clusters or distance threshold	Large `n_samples` and `n_clusters`	Many clusters, possibly connectivity constraints	Distances between points
Agglomerative clustering	number of clusters or distance threshold, linkage type, distance	Large `n_samples` and `n_clusters`	Many clusters, possibly connectivity constraints, non Euclidean distances	Any pairwise distance
DBSCAN	neighborhood size	Very large `n_samples`, medium `n_clusters`	Non-flat geometry, uneven cluster sizes	Distances between nearest points
OPTICS	minimum cluster membership	Very large `n_samples`, large `n_clusters`	Non-flat geometry, uneven cluster sizes, variable cluster density	Distances between points
Gaussian mixtures	many	Not scalable	Flat geometry, good for density estimation	Mahalanobis distances to centers
Birch	branching factor, threshold, optional global clusterer.	Large `n_clusters` and `n_samples`	Large dataset, outlier removal, data reduction.	Euclidean distance between points

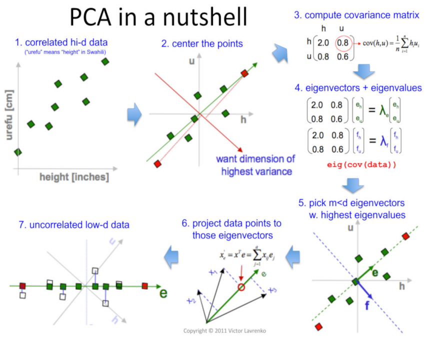
Table can be found in ▸ documentation

Dimensionality Reduction

- Richard Bellman: *The Curse of Dimensionality*
  The curse of dimensionality refers to various phenomena that arise when analyzing and organizing data in high-dimensional spaces that do not occur in low-dimensional settings such as the three-dimensional physical space of everyday experience.
- On the other hand, we want to work within dimensions as low as possible that still show the same/similar variance.

Dimensionality Reduction Example: PCA

- Principal Component Analysis (PCA):
  - Batched `PCA`
  - Mini-batch like `IncrementalPCA`
  - PCA with randomized Singular Value Decomposition (`svd_solver='randomized'`)
  - Kernel based PCA `KernelPCA` (e.g. RBF, polynomial, sigmoid)
- For some methods PCA might be a pre-requisite, e.g. SVM, K-Means
- Note that PCA looses information!

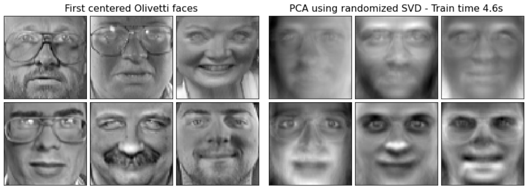

PCA in a nutshell
1. correlated hi-d data
("urefu" means "height" in Swahili)
urefu [cm]
height [inches]
2. center the points
u
h
want dimension of highest variance
3. compute covariance matrix
h u
h 2.0 0.8
u 0.8 0.6
$\mathrm{cov}(h,u) = \frac{1}{n}\sum_{i=1}^{n} h_i u_i$
4. eigenvectors + eigenvalues
$\begin{pmatrix} 2.0 & 0.8 \\ 0.8 & 0.6 \end{pmatrix}\begin{pmatrix} e_h \\ e_u \end{pmatrix} = \lambda_e \begin{pmatrix} e_h \\ e_u \end{pmatrix}$
$\begin{pmatrix} 2.0 & 0.8 \\ 0.8 & 0.6 \end{pmatrix}\begin{pmatrix} f_h \\ f_u \end{pmatrix} = \lambda_f \begin{pmatrix} f_h \\ f_u \end{pmatrix}$
eig(cov(data))
5. pick m<d eigenvectors w. highest eigenvalues
u
e
f
h
6. project data points to those eigenvectors
$x_e' = x^T e = \sum_{j=1}^{d} x_j e_j$
7. uncorrelated low-d data
e
Copyright © 2011 Victor Lavrenko

https://devopedia.org/principal-component-analysis

PCA Example: PCA with Randomized SVD

```python
import logging
from time import time
from numpy.random import RandomState
import matplotlib.pyplot as plt
from sklearn.datasets import fetch_olivetti_faces
from sklearn import decomposition

n_row, n_col = 2, 3
n_components = n_row * n_col
image_shape = (64, 64)
rng = RandomState(0)

# Load faces data
faces, _ = fetch_olivetti_faces(return_X_y=True,
                                shuffle=True,
                                random_state=rng)
n_samples, n_features = faces.shape

# global centering
faces_centered = faces - faces.mean(axis=0)
# local centering
faces_centered -= faces_centered.mean(axis=1)
                                .reshape(n_samples, -1)

def plot_gallery(title, images, n_col=n_col,
                 n_row=n_row, cmap=plt.cm.gray):
    ...
```

```python
plot_gallery("First centered Olivetti faces",
             faces_centered[:n_components])

estimator = decomposition.PCA(n_components=n_components,
                              svd_solver='randomized',
                              whiten=True)
t0 = time()
data = faces
data = faces_centered
estimator.fit(data)
train_time = (time() - t0)
print("done in %0.3fs" % train_time)
components_ = estimator.components_

plot_gallery('PCA using randomized SVD - Train time %.1fs'
             % (train_time), components_[:n_components])
plt.show()
```


First centered Olivetti faces
PCA using randomized SVD - Train time 4.6s

Python source code:

▸ Faces dataset decompositions

Model Selection

- For Estimators:
  - Cross-Validation (see hands-on exercise)
  - Tuning Hyper-Parameters
- Metrics and Scoring
- Validation Curves

Pre-Processing

- Standardization, or mean removal and variance scaling
- Non-linear transformation (e.g. mapping to distributions)
- Normalization
- Encoding categorical features
- Discretization
- Imputation of missing values
- Generating polynomial features
- Custom transformers

What Method is the Best for Me?

We cannot answer that instantly, but consider the following requirements:

- How much training data do you have?
- Is your problem continuous or discrete?
- What is the ratio $\#_{features}$ and $\#_{samples}$?
- Do you need a sparse model?
- Would reducing dimensionality be an option?
- Do you have a multi-task/-label problem?

Here's a great overview of (some) of the methods: ▸ Data Science Cheatsheet

VSB TECHNICAL UNIVERSITY OF OSTRAVA | IT4INNOVATIONS NATIONAL SUPERCOMPUTING CENTER

IT4Innovations National Supercomputing Center

VŠB – Technical University of Ostrava
Studentská 6231/1B
708 00 Ostrava-Poruba, Czech Republic
`www.it4i.cz`

EUROPEAN UNION
European Structural and Investment Funds
Operational Programme Research,
Development and Education

MINISTRY OF EDUCATION, YOUTH AND SPORTS