Scikit-learn: Machine Learning

Hendrik Speleers

Scikit-learn: Machine Learning

Università di Roma Tor Vergata

- Overview
  - Supervised vs. unsupervised learning
  - Data representation
  - Learning and predicting
    - Linear regression, support vector classification, clustering, ...
  - Dataset transformations
    - Preprocessing data, dimensionality reduction, ...
  - Model selection and evaluation
    - Performance measurement, dataset splitting, ...

Scikit-learn: Machine Learning

Università di Roma Tor Vergata

- Scikit-learn (also: Sklearn)
  - SciPy Toolkit for Learning
  - Python extension for machine learning
    - Simple and efficient tools for predictive data analysis
    - Regression, classification, clustering, dimensionality reduction, ...
    - Other Python libraries for learning: mlpy, PyTorch, TensorFlow, ...
  - Import convention

```
import sklearn
```

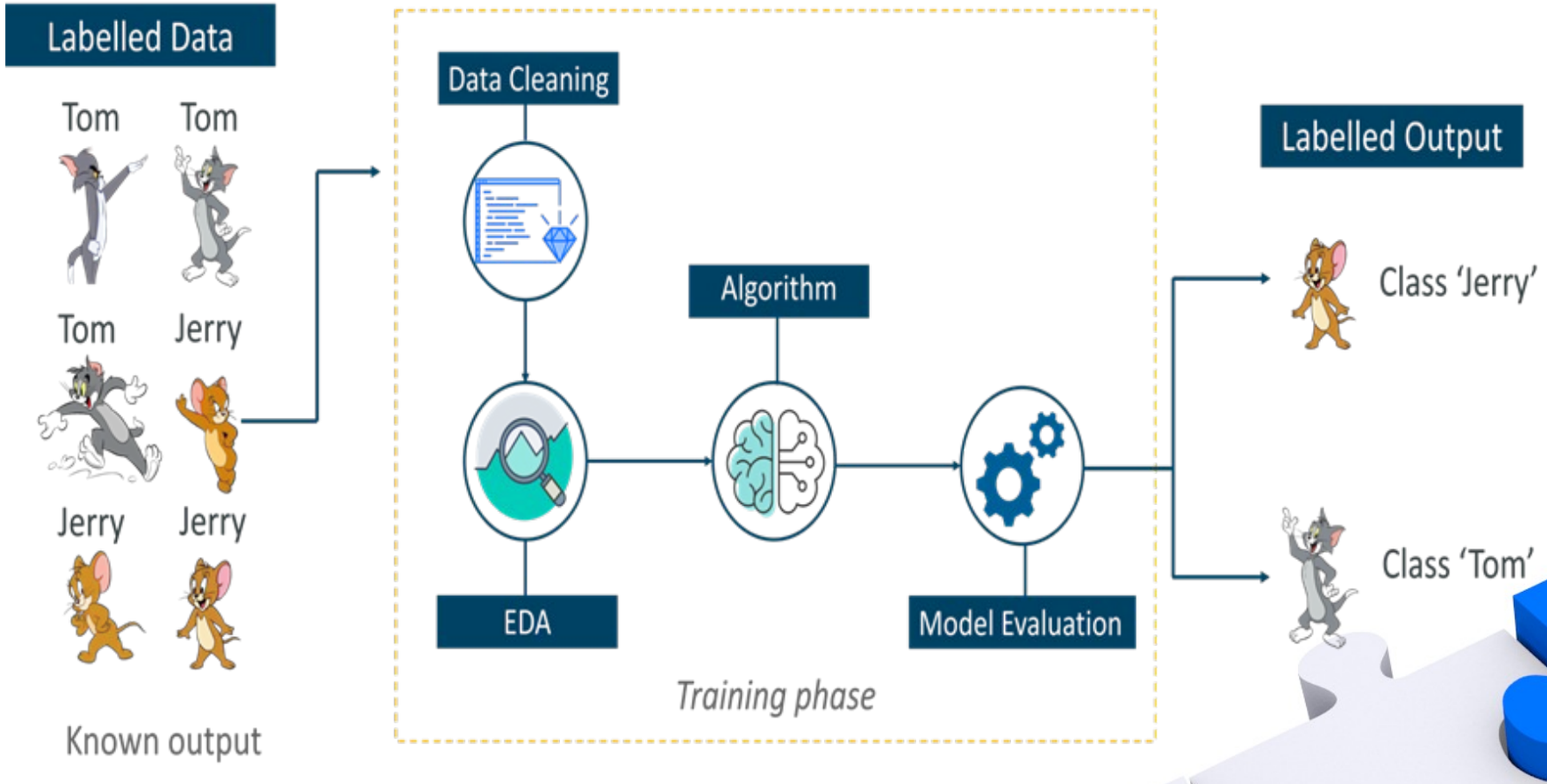
Scikit-learn: Machine Learning

Università di Roma Tor Vergata

- Machine learning
  - Supervised learning
    - Data must be in labelled format
    - Based on the training data, produces a function, which can be used for mapping new data
    - Two main types of problems
      - Regression (continuous output)
      - Classification (discrete output)
  - Unsupervised learning
    - Data is unlabelled
    - Draws a conclusion from a given dataset
    - Clustering is most popular method

Scikit-learn: Machine Learning

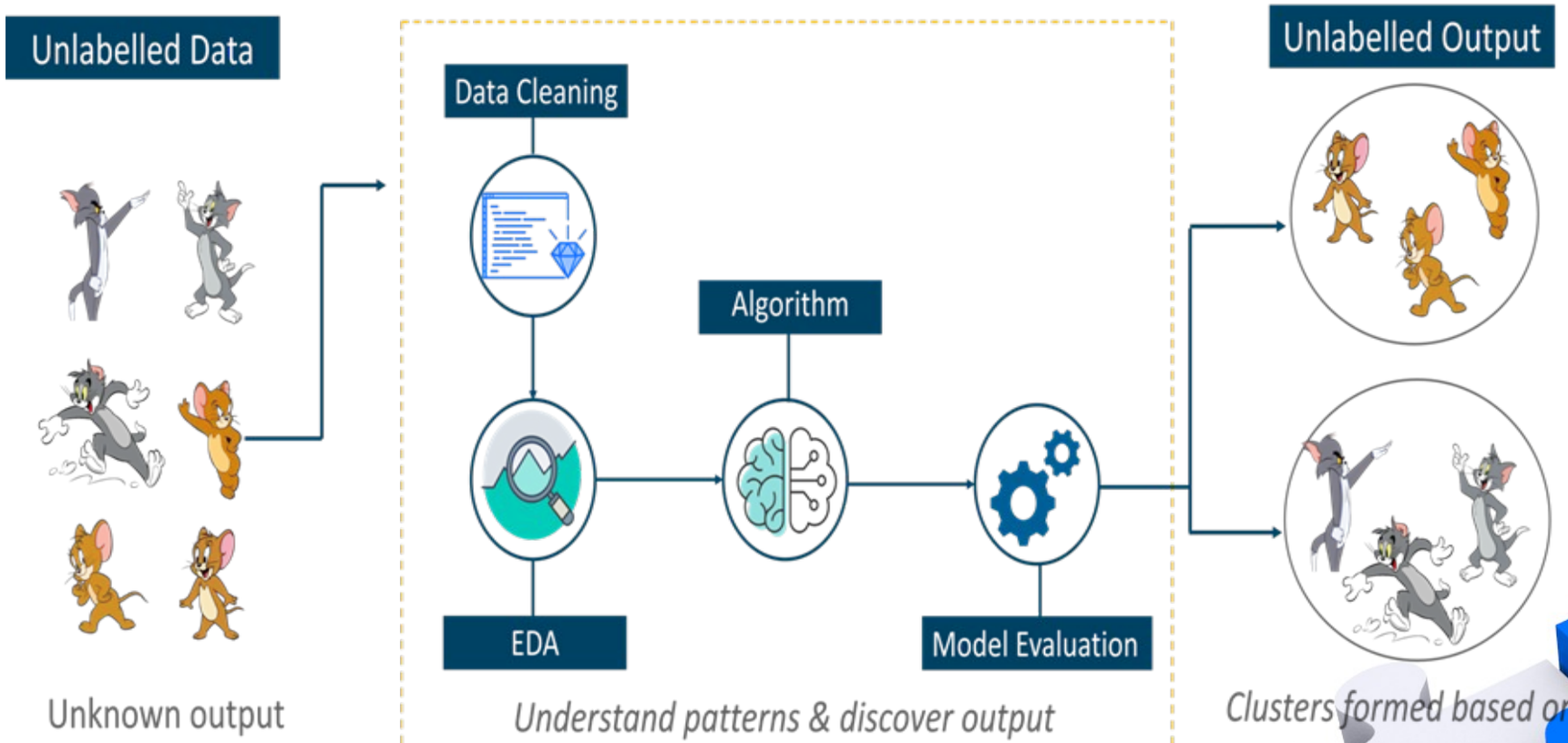
- Supervised learning



Scikit-learn: Machine Learning

Università di Roma Tor Vergata

- Unsupervised learning



Scikit-learn: Machine Learning

Università di Roma Tor Vergata

- Dataset
  - Data representation
    - Data matrix: two-dimensional array [n_samples, n_features]
      - Samples (i.e., rows) refer to the individual objects in the dataset
      - Features (i.e., columns) refer to the distinct observations describing a sample
    - Target vector: one-dimensional array [n_samples]
      - Targets or labels refer to the quantities of interest for the samples
  - Many example datasets

```
from sklearn import datasets
```

Scikit-learn: Machine Learning

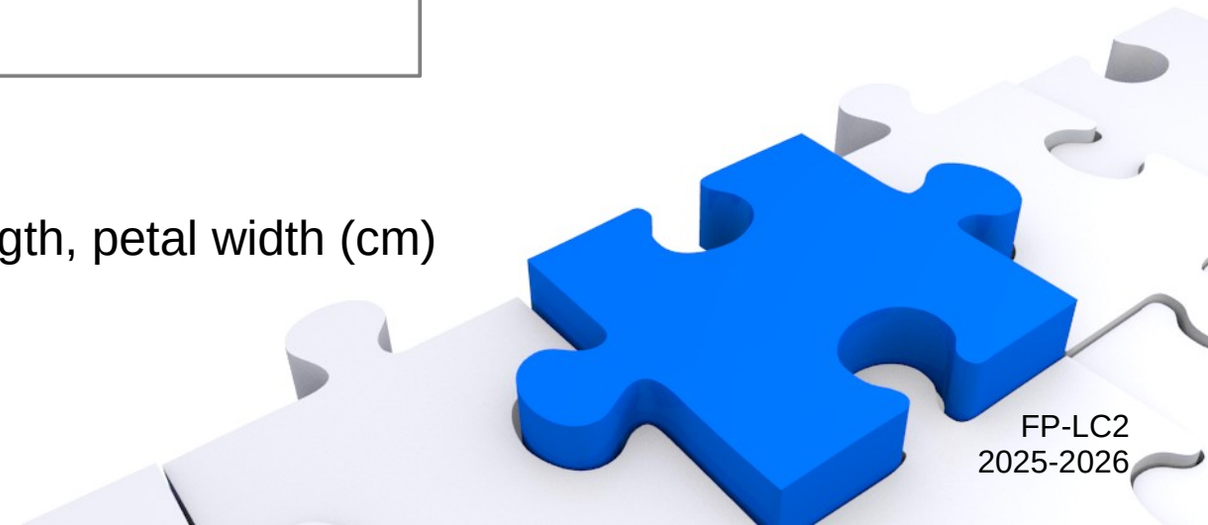
Università di Roma Tor Vergata

- Dataset
  - Example

```
In [1]: iris = datasets.load_iris()
In [2]: iris.data.shape
Out[2]: (150, 4)
In [3]: iris.target.shape
Out[3]: (150,)
```

Iris dataset:

- Features: sepal length, sepal width, petal length, petal width (cm)
- Target classes: Setosa, Versicolor, Virginica



Scikit-learn: Machine Learning

Università di Roma Tor Vergata

- Learning and predicting
  - Models
    - Uniform interfaces for all models (Estimator, Predictor, Transformer)
    - Supervised learning
      - Fit training data: `model.fit(X, y)`
      - Predict new data: `model.predict(X)`
      - Score method: `model.score(X, y)`
    - Unsupervised learning
      - Fit training data: `model.fit(X)`
      - Transform new data: `model.transform(X)`
      - Fit and transform: `model.fit_transform(X)`

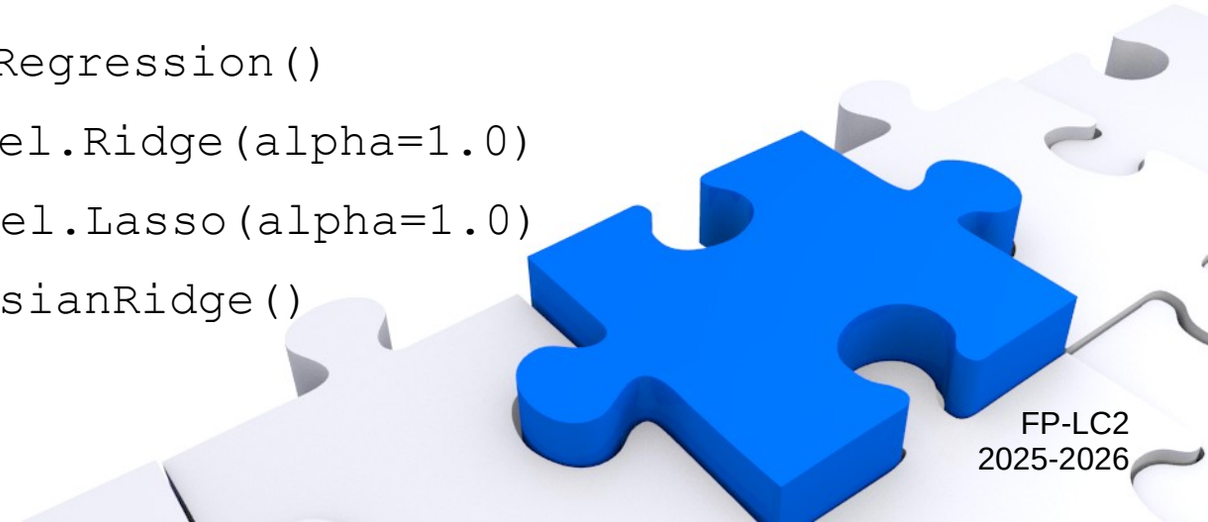
Scikit-learn: Machine Learning

Università di Roma Tor Vergata

- Learning and predicting
  - Linear regression models

```
from sklearn import linear_model
```

    - Fit a linear polynomial model through the data
      - $\hat{y}(x,c) = c_0 + c_1x_1 + \ldots + c_px_p$
    - Linear regression: `linear_model.LinearRegression()`
    - Ridge regression (L2 penalty): `linear_model.Ridge(alpha=1.0)`
    - Lasso regression (L1 penalty): `linear_model.Lasso(alpha=1.0)`
    - Bayesian regression: `linear_model.BayesianRidge()`

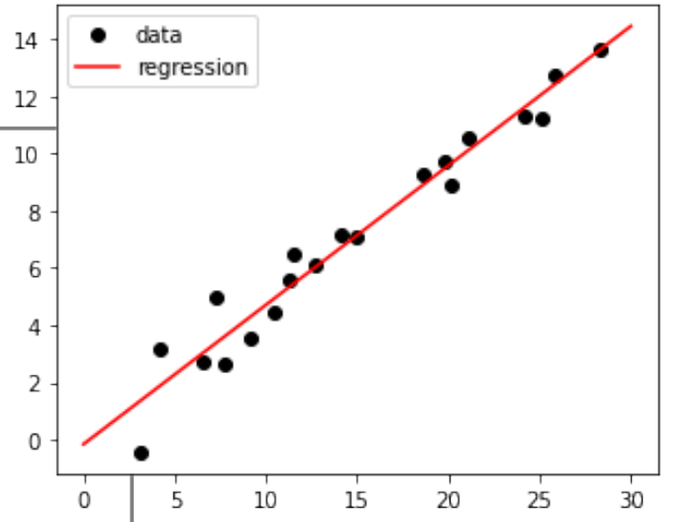


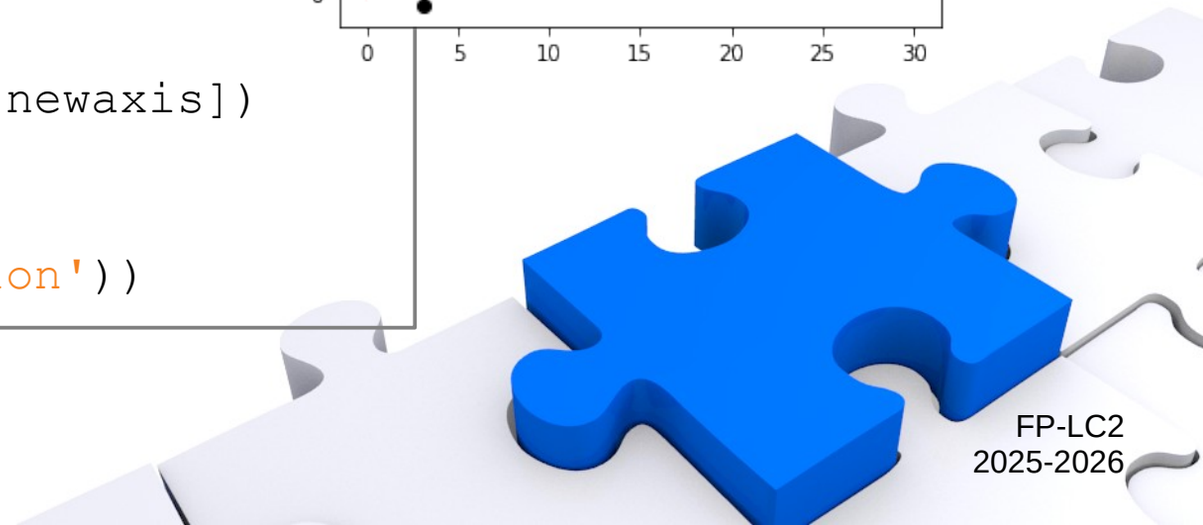
Scikit-learn: Machine Learning

Università di Roma Tor Vergata

- Learning and predicting
  - Example

```
In [1]: n = 20
   ...: x = 30 * np.random.rand(n, 1)
   ...: y = 0.5 * x + np.random.randn(n, 1)
In [2]: regr = linear_model.LinearRegression()
   ...: regr.fit(x, y)
In [3]: x_r = np.linspace(0, 30, 100)
   ...: y_r = regr.predict(x_r[:, np.newaxis])
In [4]: plt.plot(x, y, 'ko')
   ...: plt.plot(x_r, y_r, 'r--')
   ...: plt.legend(('data', 'regression'))
```





Scikit-learn: Machine Learning

Università di Roma Tor Vergata

- Learning and predicting
  - Support vector classification

```
from sklearn import svm
```

    - General kernel: `svm.SVC(C=1.0, kernel='rbf')`
    - Linear kernel: `svm.LinearSVC(C=1.0, penalty='l2', loss='squared_hinge')`
  - Naive Bayes classification

```
from sklearn import naive_bayes
```

    - Gaussian distribution: `naive_bayes.GaussianNB()`
    - Multinomial distribution: `naive_bayes.MultinomialNB()`

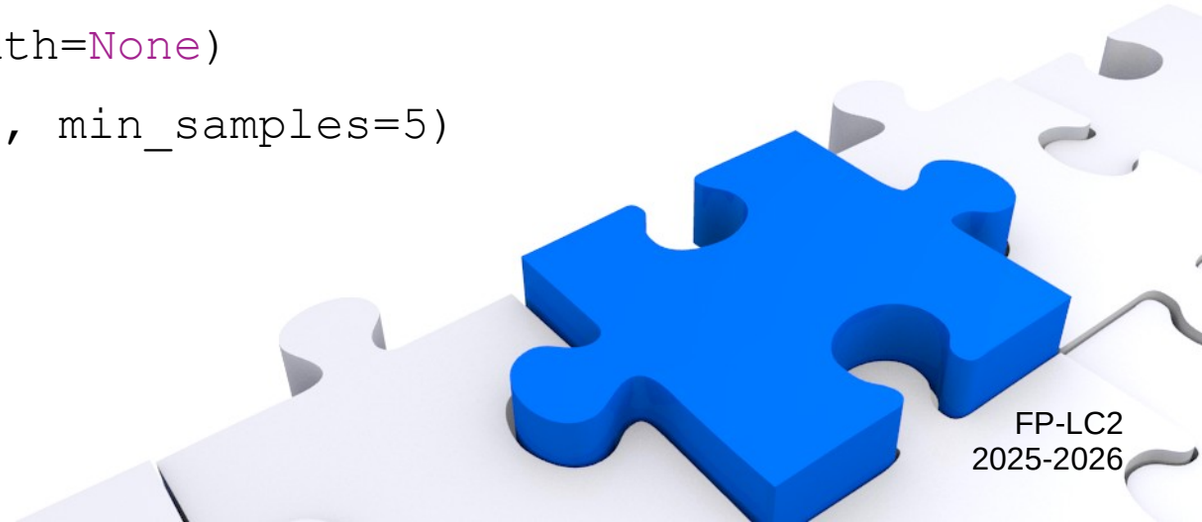
Scikit-learn: Machine Learning

Università di Roma Tor Vergata

- Learning and predicting
  - Clustering models

```
from sklearn import cluster
```

  - K-means: `cluster.KMeans(n_clusters=8)`
  - Spectral based: `cluster.SpectralClustering(n_clusters=8)`
  - Mean shift: `cluster.MeanShift(bandwidth=None)`
  - Density based: `cluster.DBSCAN(eps=0.5, min_samples=5)`

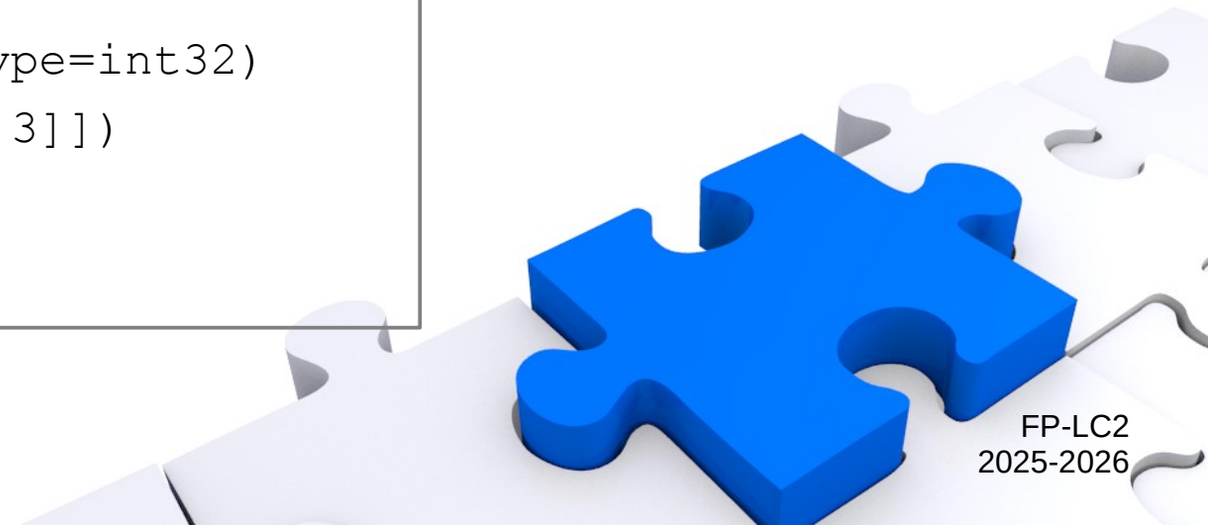


Scikit-learn: Machine Learning

Università di Roma Tor Vergata

- Learning and predicting
  - Example

```
In [1]: X = np.array([[1, 2], [1, 4], [1, 0],
   ...:               [10, 2], [10, 4], [10, 0]])
In [2]: kmeans = cluster.KMeans(n_clusters=2)
   ...: kmeans.fit(X)
In [3]: kmeans.labels_
Out[3]: array([1, 1, 1, 0, 0, 0], dtype=int32)
In [4]: kmeans.predict([[0, 0], [12, 3]])
Out[4]: array([1, 0], dtype=int32)
```



Scikit-learn: Machine Learning

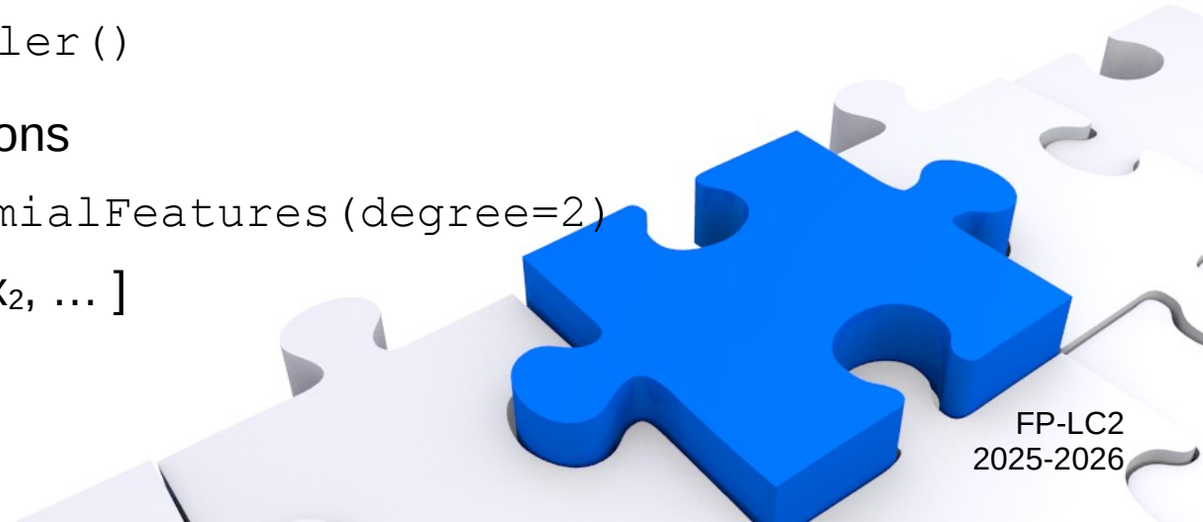
Università di Roma Tor Vergata

- Dataset transformations
  - Preprocessing data

```
from sklearn import preprocessing
```

    - Scale data to a range
      - Min-max: `preprocessing.MinMaxScaler(feature_range=(0, 1))`
      - Max abs: `preprocessing.MaxAbsScaler()`
    - Use linear models trained on nonlinear functions
      - Polynomials: `preprocessing.PolynomialFeatures(degree=2)`

$[x_1, x_2, \ldots] \rightarrow [1, x_1, x_2, \ldots, x_1x_1, x_1x_2, \ldots]$

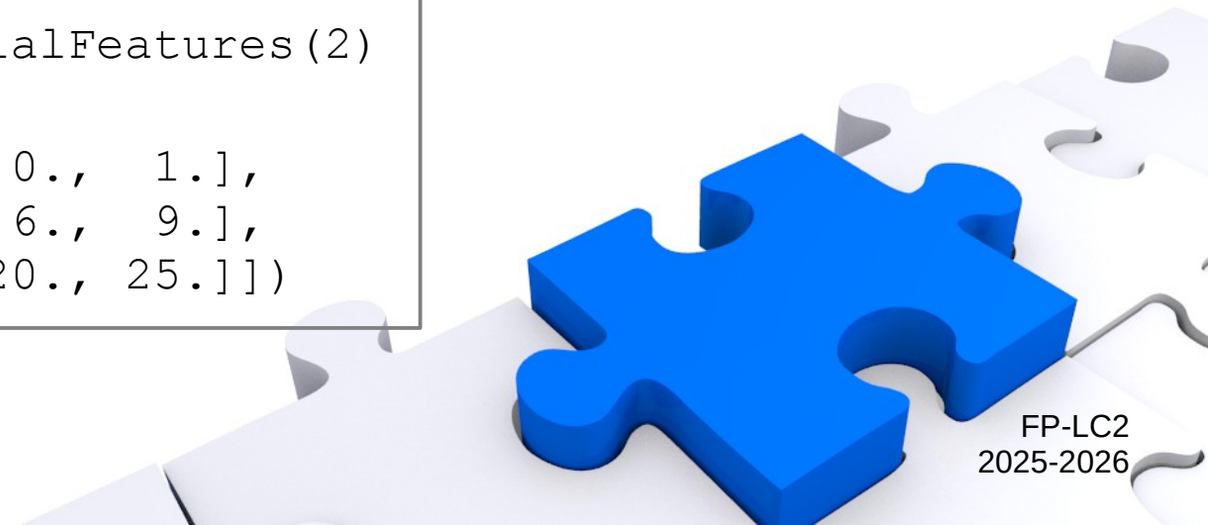


Scikit-learn: Machine Learning

Università di Roma Tor Vergata

- Dataset transformations
  - Example

```
In [1]: X = np.arange(6).reshape(3, 2)
   ...: X
Out[1]: array([[0, 1],
   ...:        [2, 3],
   ...:        [4, 5]])
In [2]: poly = preprocessing.PolynomialFeatures(2)
   ...: poly.fit_transform(X)
Out[2]: array([[ 1.,  0.,  1.,  0.,  0.,  1.],
   ...:        [ 1.,  2.,  3.,  4.,  6.,  9.],
   ...:        [ 1.,  4.,  5., 16., 20., 25.]])
```

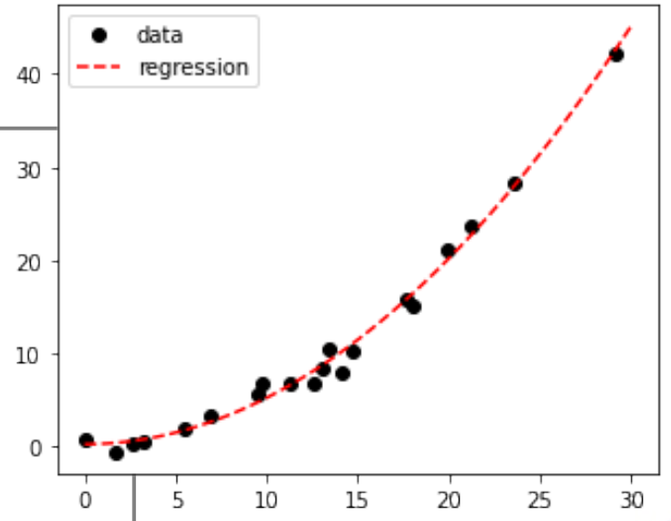


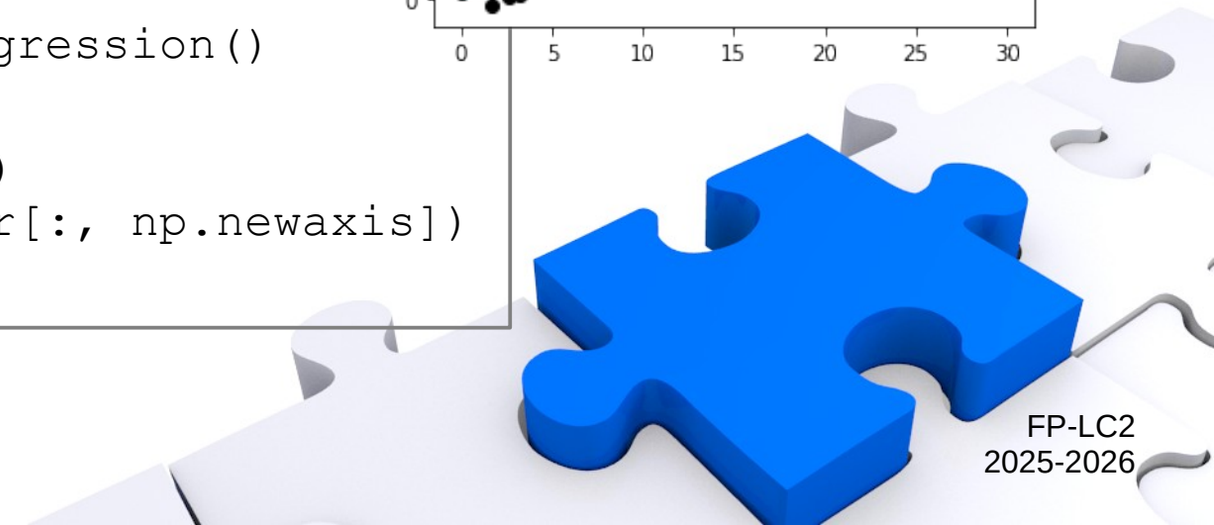
Scikit-learn: Machine Learning

Università di Roma Tor Vergata

- Dataset transformations
  - Example

```
In [1]: n = 20
   ...: x = 30 * np.random.rand(n, 1)
   ...: y = 0.05 * x**2 + np.random.randn(n, 1)
In [2]: poly = preprocessing.PolynomialFeatures(2)
   ...: x2 = poly.fit_transform(x)
In [3]: regr = linear_model.LinearRegression()
   ...: regr.fit(x2, y)
In [4]: x_r = np.linspace(0, 30, 100)
   ...: x2_r = poly.fit_transform(x_r[:, np.newaxis])
   ...: y_r = regr.predict(x2_r)
```





Scikit-learn: Machine Learning

Università di Roma Tor Vergata

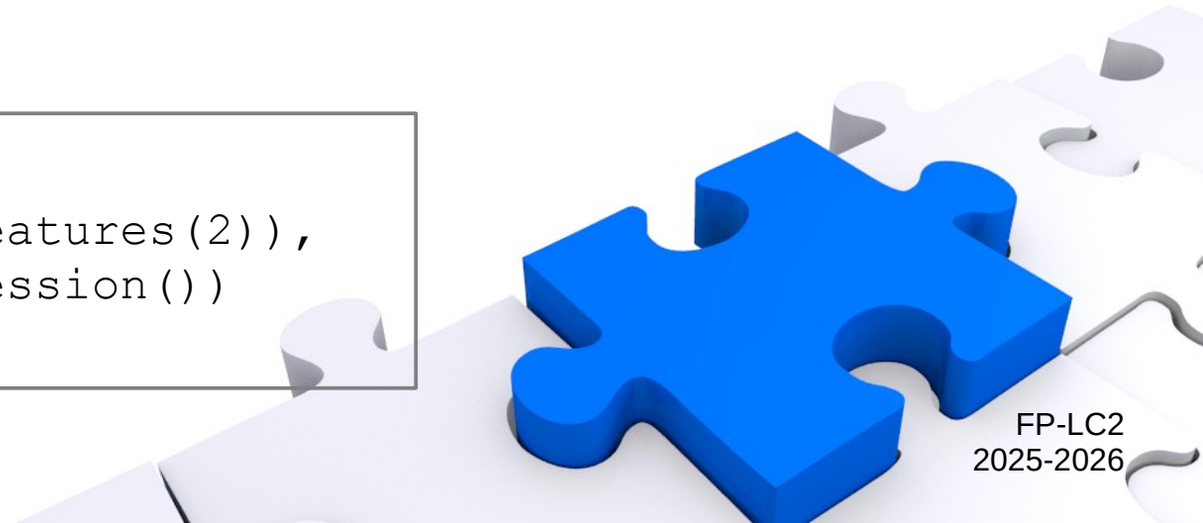
- Dataset transformations
  - Streamline the preprocessing

```
from sklearn import pipeline
```

Combine a list of transformers with an optional final predictor

- Basic syntax: `pipeline.Pipeline(steps)`
- Input: list of tuples (name, estimator)

```
pipe = pipeline.Pipeline([
   ('poly', preprocessing.PolynomialFeatures(2)),
   ('linear', linear_model.LinearRegression())
])
```



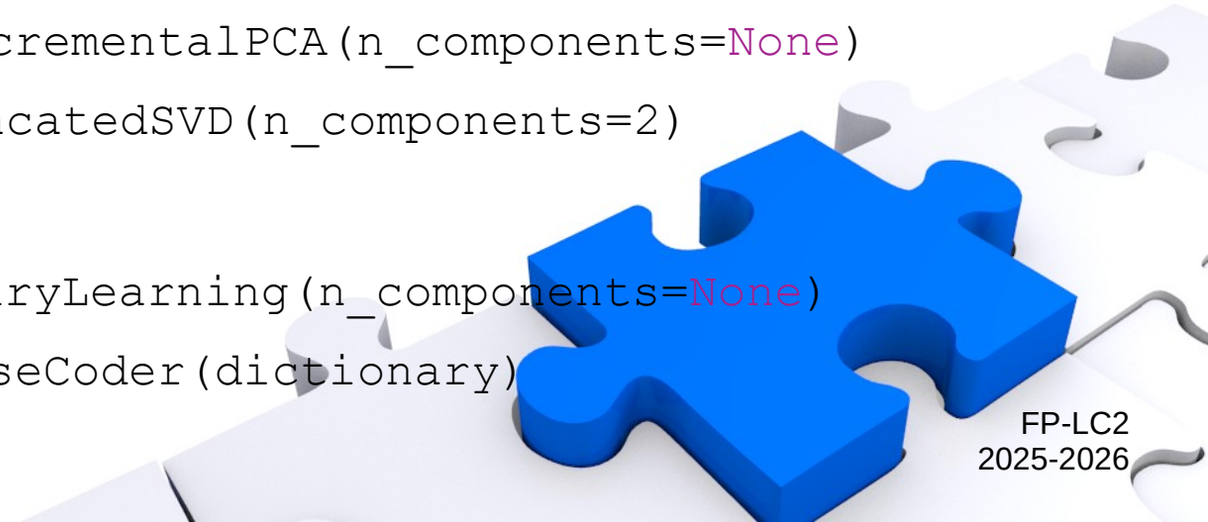
Scikit-learn: Machine Learning

Università di Roma Tor Vergata

- Dataset transformations
  - Dimensionality reduction

```
from sklearn import decomposition
```

    - Principal component analysis
      - Exact PCA: `decomposition.PCA(n_components=None)`
      - Incremental PCA: `decomposition.IncrementalPCA(n_components=None)`
      - Truncated SVD: `decomposition.TruncatedSVD(n_components=2)`
    - Dictionary learning
      - Dictionary: `decomposition.DictionaryLearning(n_components=None)`
      - Sparse coding: `decomposition.SparseCoder(dictionary)`

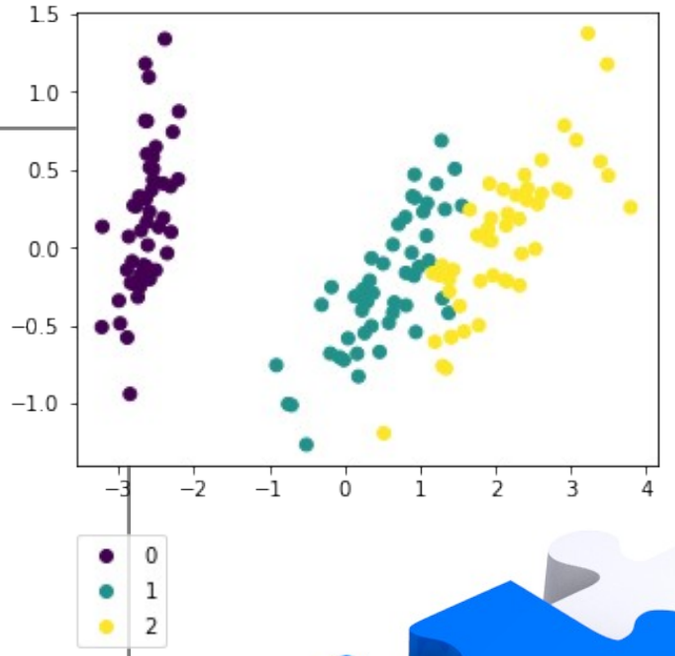


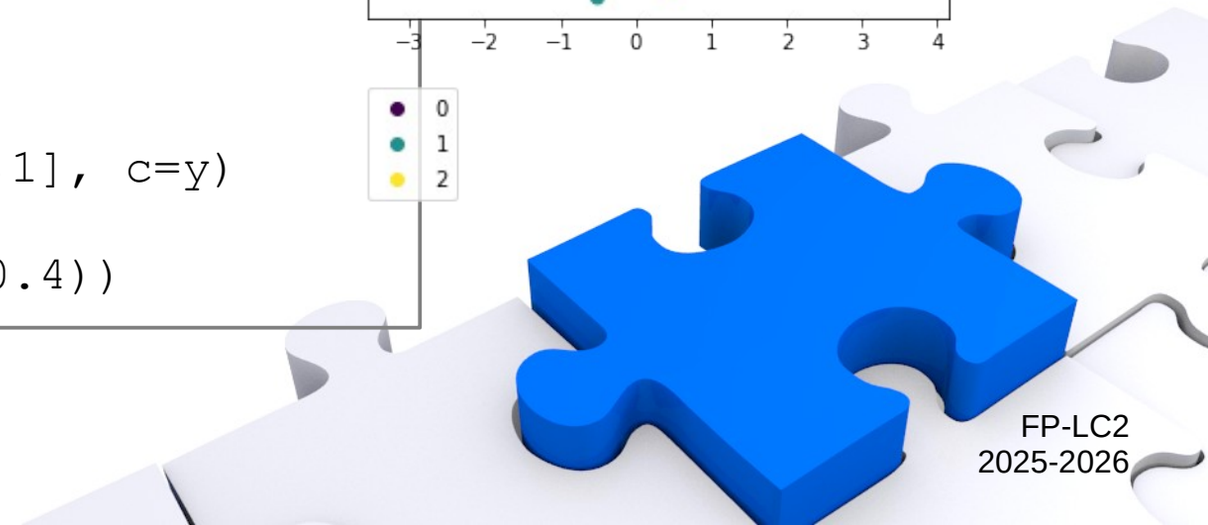
Scikit-learn: Machine Learning

Università di Roma Tor Vergata

- Dataset transformations
  - Example

```
In [1]: iris = datasets.load_iris()
   ...: X = iris.data
   ...: y = iris.target
In [2]: pca = decomposition.PCA(n_components=2)
   ...: P = pca.fit_transform(X)
   ...: P.shape
Out[2]: (150, 2)
In [3]: ps = plt.scatter(P[:,0], P[:,1], c=y)
   ...: ps_leg = ps.legend_elements()
   ...: plt.legend(*ps_leg, loc=(0,-0.4))
```





Scikit-learn: Machine Learning

Università di Roma Tor Vergata

- Model selection and evaluation
  - Performance measurement

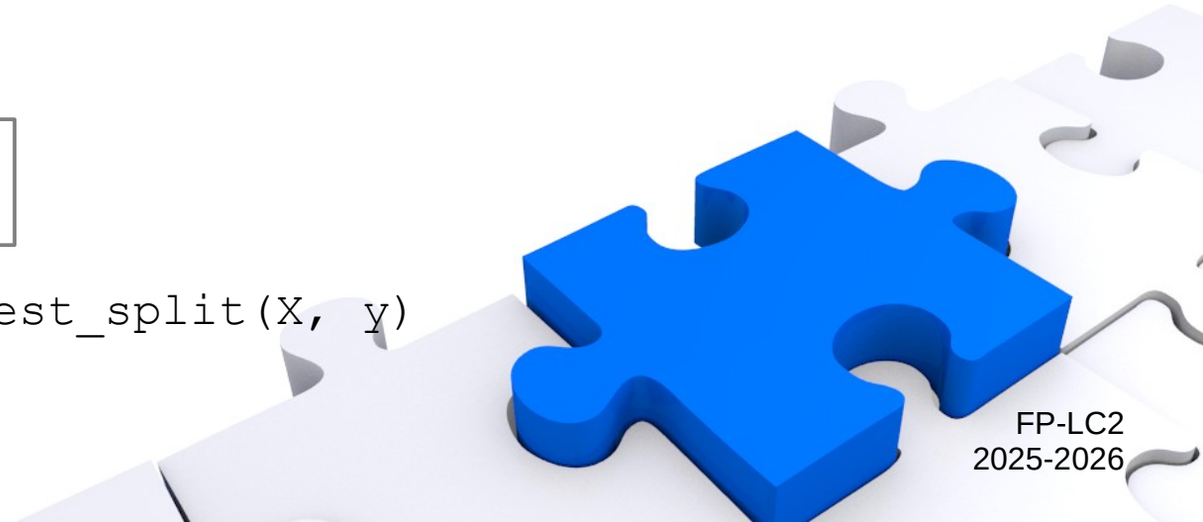
```
from sklearn import metrics
```

    - General kernel: `metrics.classification_report(y_true, y_pred)`
    - Linear kernel: `metrics.confusion_matrix(y_true, y_pred)`

  - Using a training and a testing set

```
from sklearn import model_selection
```

    - Split dataset: `model_selection.train_test_split(X, y)`



Scikit-learn: Machine Learning

Università di Roma Tor Vergata

- Model selection and evaluation
  - Choose a model (and its hyperparameters)
    - Step 1: Load a dataset
    - Step 2: Split the dataset
    - Step 3: Train the model
    - Step 4: Test the model
  - Select the best model
  - Cross-validation of the model