

# Scikit-Learn

## Machine Learning in Python

Prof. Massimo Fedeli

IIS Fermi Sacconi Cria

17 dicembre 2025

# Indice

- 1 Introduzione a Scikit-Learn
- 2 Struttura e Workflow
- 3 Preprocessing dei Dati
- 4 Supervised Learning: Classificazione
- 5 Supervised Learning: Regressione
- 6 Unsupervised Learning
- 7 Metriche e Valutazione
- 8 Model Selection e Tuning
- 9 Pipeline e Feature Engineering
- 10 Ensemble Methods

# Cos'è Scikit-Learn?

- **Libreria open-source** per Machine Learning in Python
- Costruita su NumPy, SciPy e Matplotlib
- Sviluppata inizialmente da David Cournapeau nel 2007
- Attualmente mantenuta da una grande comunità
- Licenza BSD (permissiva)

- La libreria SciPy è una raccolta di strumenti avanzati per il calcolo scientifico e ingegneristico in Python.
- Si appoggia a NumPy e ne estende le funzionalità, offrendo algoritmi pronti all'uso, efficienti e ben collaudati.
- Algebra lineare
- Integrazione ed equazioni differenziali
- Statistica e Probabilità

## Caratteristiche principali

- API semplice e consistente
- Ottima documentazione
- Algoritmi efficienti e testati
- Integrazione con l'ecosistema Python scientifico

# Installazione

## Metodi di installazione

```
# Usando pip
pip install scikit-learn

# Usando conda
conda install scikit-learn

# Installazione con dipendenze complete
pip install scikit-learn numpy scipy matplotlib pandas
```

## Verifica installazione

```
import sklearn
print(sklearn.__version__)
```

## Supervised Learning

- Classificazione
- Regressione
- Predizione

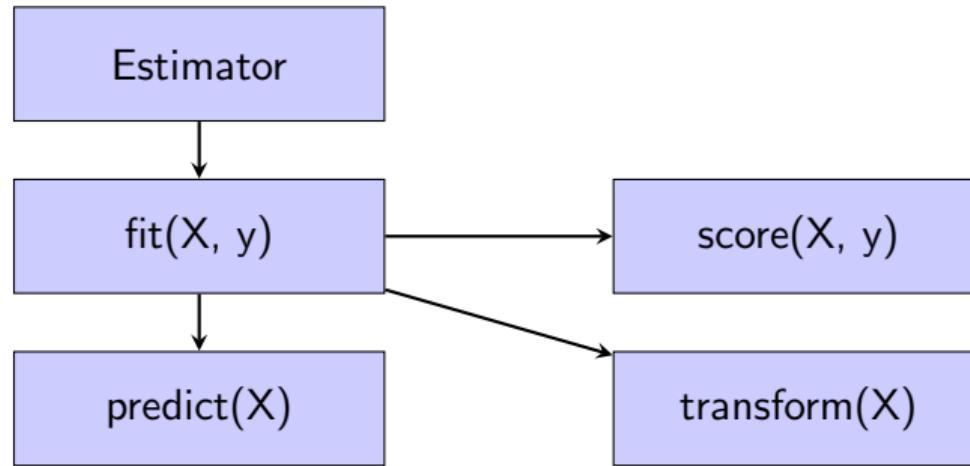
## Unsupervised Learning

- Clustering
- Riduzione dimensionalità
- Anomaly detection

## Altri ambiti

- Preprocessing dei dati
- Feature engineering
- Model selection
- Cross-validation
- Ensemble methods

# API Unificata di Scikit-Learn



## Metodi principali:

- `fit()`: addestra il modello
- `predict()`: effettua predizioni
- `transform()`: trasforma i dati
- `score()`: valuta le prestazioni

# Cos'è un Estimator?

**Definizione:** Un Estimator è il concetto fondamentale di Scikit-Learn

Un Estimator è **qualsiasi oggetto che può imparare dai dati** e implementa il metodo `fit()`.

## Tipologie di Estimator:

- **Modelli di ML:** regressione lineare, alberi decisionali, SVM, reti neurali
- **Transformer:** scalatori, encoder, PCA, selettori di feature
- **Preprocessori:** normalizzatori, gestori di valori mancanti
- **Pipeline:** combinazioni di più estimator

## Vantaggi dell'API unificata:

- **Uniformità:** stesso pattern per tutti gli algoritmi
- **Semplicità:** facile passare da un modello all'altro
- **Composabilità:** possibilità di combinare estimator

# Esempio: Utilizzo degli Estimator

Tutti gli estimator seguono lo stesso pattern:

```
from sklearn.linear_model import LinearRegression
from sklearn.tree import DecisionTreeClassifier
from sklearn.preprocessing import StandardScaler

# Pattern identico per tutti gli estimator:

# 1. Creazione
model = LinearRegression()

# 2. Addestramento
model.fit(X_train, y_train)

# 3. Utilizzo
predictions = model.predict(X_test)
accuratezza = model.score(X_test, y_test)
```

# Metodi Principali: Dettaglio

## fit(X, y) - Addestramento

- Impara dai dati di training
- X: features (matrice n\_samples × n\_features)
- y: target (vettore n\_samples)
- Restituisce l'oggetto stesso per method chaining

## predict(X) - Predizione

- Effettua predizioni su nuovi dati
- Richiede fit() precedente
- Restituisce array di predizioni

# Metodi Principali: Dettaglio (continua)

## transform(X) - Trasformazione

- Trasforma i dati secondo le regole apprese con `fit()`
- Usato principalmente nei preprocessors (scaler, encoder, etc.)
- Restituisce array trasformato
- Spesso combinato con `fit_transform()` per training

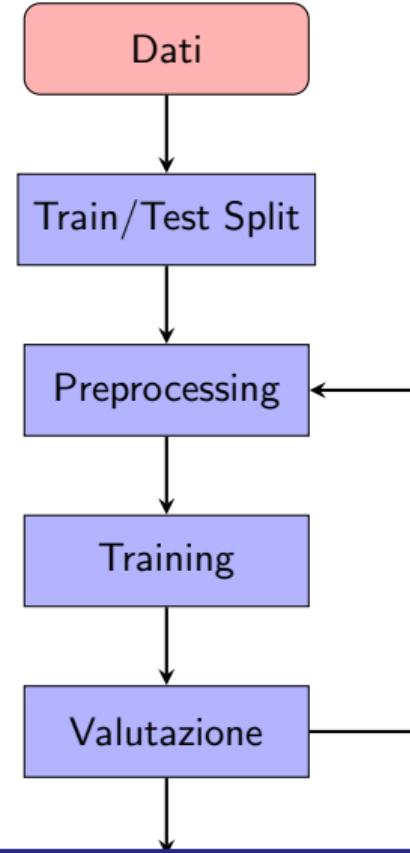
## score(X, y) - Valutazione

- Valuta le prestazioni del modello
- Per classificatori: restituisce l'accuracy
- Per regressori: restituisce  $R^2$  score
- Range tipico: [0, 1] (più alto = migliore)

# Esempio Pratico: Utilizzo dei Metodi

```
1  from sklearn.datasets import load_iris
2  from sklearn.tree import DecisionTreeClassifier
3  from sklearn.preprocessing import StandardScaler
4  from sklearn.model_selection import train_test_split
5
6  # Caricamento dati
7  X, y = load_iris(return_X_y=True)
8  X_train, X_test, y_train, y_test = train_test_split(
9      X, y, test_size=0.3, random_state=42
10 )
11
12 # Preprocessing: transform
13 scaler = StandardScaler()
14 X_train_scaled = scaler.fit_transform(X_train) # fit + transform
15 X_test_scaled = scaler.transform(X_test) # solo transform
16
17 # Classificazione: fit, predict, score
18 clf = DecisionTreeClassifier()
19 clf.fit(X_train_scaled, y_train) # addestramento
```

# Workflow Tipico del Machine Learning



# Primo Esempio: Classificazione Iris

```
1  from sklearn.datasets import load_iris
2  from sklearn.model_selection import train_test_split
3  from sklearn.tree import DecisionTreeClassifier
4  from sklearn.metrics import accuracy_score
5
6  # Caricamento dataset
7  iris = load_iris()
8  X, y = iris.data, iris.target
9
10 # Divisione train/test
11 X_train, X_test, y_train, y_test = train_test_split(
12     X, y, test_size=0.3, random_state=42
13 )
14
15 # Addestramento modello
16 clf = DecisionTreeClassifier()
17 clf.fit(X_train, y_train)
18
19 # Predizione e valutazione
```

## Perché è importante?

- Migliora le prestazioni dei modelli
- Gestisce dati mancanti e outlier
- Normalizza scale diverse
- Codifica variabili categoriche

## Moduli principali:

- `sklearn.preprocessing`: scaling, encoding
- `sklearn.impute`: gestione valori mancanti
- `sklearn.feature_selection`: selezione features

# Preprocessing: Concetti Fondamentali

## Cos'è il Preprocessing?

- Trasformazione dei dati grezzi
- Preparazione per il ML
- Fase cruciale del workflow
- Impatta direttamente i risultati

## Quando applicarlo?

- Prima del training
- Su train e test set
- Mai sul test prima del train!

## Problemi comuni risolti:

- Scale diverse tra features
- Valori mancanti (NaN)
- Variabili categoriche
- Outliers estremi
- Features ridondanti
- Distribuzioni non normali

## Regola d'oro

`fit()` solo sul training set, `transform()` su train e test!

# Preprocessing: Tecniche Principali

## 1. Scaling e Normalizzazione

**Problema:** Features con scale diverse (es: età 0-100, reddito 0-1M)

**Soluzioni:**

- StandardScaler: trasforma in media=0, std=1  $\rightarrow z = \frac{x-\mu}{\sigma}$
- MinMaxScaler: scala in range [0,1]  $\rightarrow x_{norm} = \frac{x-x_{min}}{x_{max}-x_{min}}$
- RobustScaler: resistente agli outliers (usa mediana e IQR)

## 2. Encoding Variabili Categoriche

**Problema:** ML lavora solo con numeri

**Soluzioni:**

- LabelEncoder: ordinali  $\rightarrow [0, 1, 2, \dots]$  (es: basso/medio/alto)
- OneHotEncoder: nominali  $\rightarrow$  vettori binari (es: rosso/verde/blu  $\rightarrow [1,0,0]$ )
- OrdinalEncoder: come LabelEncoder ma per multiple colonne

# Scaling e Normalizzazione

```
1 from sklearn.preprocessing import StandardScaler, MinMaxScaler  
2  
3 # StandardScaler: media 0, varianza 1  
4 scaler = StandardScaler()  
5 X_scaled = scaler.fit_transform(X_train)  
6  
7 # MinMaxScaler: range [0, 1]  
8 min_max_scaler = MinMaxScaler()  
9 X_normalized = min_max_scaler.fit_transform(X_train)
```

## Attenzione!

- Fai `fit()` solo sul training set
- Usa `transform()` sul test set
- Evita data leakage!

# Scaling e Normalizzazione: Perché sono necessari?

## Il Problema

Esempio: Dataset case

Casa	mq	Prezzo (€)
A	50	150.000
B	100	300.000
C	150	450.000

### Attenzione!

I mq variano [50-150], il prezzo [150k-450k]  
→ Il prezzo domina il calcolo delle distanze!

### Algoritmi Sensibili alla Scala

- K-Nearest Neighbors (KNN)
- Support Vector Machines (SVM)
- Regressione Lineare/Logistica
- Neural Networks
- K-Means Clustering
- Principal Component Analysis (PCA)

### Algoritmi NON Sensibili

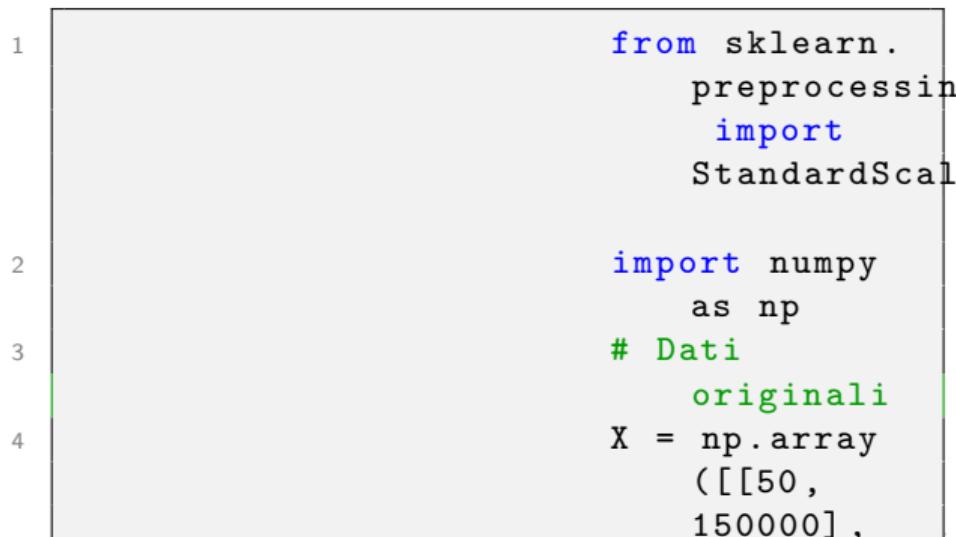
- Decision Trees
- Random Forest
- Gradient Boosting

# StandardScaler: Standardizzazione

## Formula

$$z = \frac{x - \mu}{\sigma}$$

Dove:  $\mu$  = media,  $\sigma$  = deviazione standard



## Caratteristiche:

# MinMaxScaler: Normalizzazione

## Formula

$$x_{norm} = \frac{x - x_{min}}{x_{max} - x_{min}}$$

Risultato: range [0, 1] (o personalizzabile)

## Caratteristiche:

- Range fisso [0, 1]
- Preserva la forma della distribuzione
- Molto sensibile agli outliers
- Valori compressi in [0, 1]

```
1      from sklearn.preprocessing
2          import MinMaxScaler
3
4      # Dati originali
5      X = np.array([[50, 150000],
6                  [100, 300000],
7                  [150, 450000]])
8
9      # Normalizzazione
10     scaler = MinMaxScaler()
11     X_norm = scaler.fit_transform(
12         X)
```

# RobustScaler e Confronto delle Tecniche

## RobustScaler: Resistente agli Outliers

$$x_{scaled} = \frac{x - Q_{50}}{Q_{75} - Q_{25}}$$

Usa mediana ( $Q_{50}$ ) e IQR (Interquartile Range) invece di media e std

```
from sklearn.preprocessing import
    RobustScaler

# Dati con outlier
X = np.array([[1], [2], [3], [4],
              [100]])

# Confronto
standard = StandardScaler().
    fit_transform(X)
minmax = MinMaxScaler().
    fit_transform(X)
```

Scaler	Pro	Contro
Standard	Funziona bene con dist. normale	Sensibile outliers
MinMax	Range fisso [0,1]	Molto sensibile outliers
Robust	Resistente outliers	Range variabile

# Encoding Variabili Categoricalhe

```
1  from sklearn.preprocessing import LabelEncoder, OneHotEncoder  
2  
3  # LabelEncoder per variabili ordinali  
4  le = LabelEncoder()  
5  labels = ['rosso', 'verde', 'blu', 'rosso']  
6  encoded = le.fit_transform(labels)  
7  # Output: [2, 1, 0, 2]  
8  
9  # OneHotEncoder per variabili nominali  
0  from sklearn.preprocessing import OneHotEncoder  
1  import numpy as np  
2  
3  ohe = OneHotEncoder(sparse_output=False)  
4  colors = np.array([['rosso'], ['verde'], ['blu']])  
5  encoded_ohe = ohe.fit_transform(colors)  
6  # Output: [[0, 0, 1],  
7  #           [0, 1, 0],  
8  #           [1, 0, 0]]
```

# Encoding Variabili Categoriche: Il Problema

## Perché serve l'Encoding?

- Gli algoritmi ML lavorano solo con numeri
- Le variabili categoriche sono testo
- Necessaria conversione numerica
- Attenzione: non tutte le conversioni sono uguali!

## Tipi di Variabili Categoriche:

- ① **Nominali:** nessun ordine  
Esempi: colore, città, marca
- ② **Ordinali:** ordine definito  
Esempi: taglia (S/M/L), voto (basso/medio/alto)

## Problema con encoding errato:

Colore	Codice
Rosso	0
Verde	1
Blu	2

### Attenzione!

Blu (2) > Verde (1) > Rosso (0)  
→ L'algoritmo pensa che Blu sia "maggiore"!  
→ Questo è SBAGLIATO per variabili nominali!

**Soluzione:** scegliere l'encoder giusto!

# LabelEncoder vs OneHotEncoder

## LabelEncoder

### Per variabili ORDINALI

Converte categorie in numeri sequenziali

```
from sklearn.  
preprocessing  
import 1  
LabelEncoder  
  
le =  
    LabelEncoder()  
  
# Esempio:  
taglie ( 3  
ordine! )
```

## OneHotEncoder

### Per variabili NOMINALI

Crea una colonna binaria per ogni categoria

```
from sklearn.  
preprocessing  
import  
OneHotEncoder  
  
import numpy  
as np  
  
ohe =  
    OneHotEncoder(
```

# Gestione Valori Mancanti

```
1 from sklearn.impute import SimpleImputer
2 import numpy as np
3
4 # Dati con valori mancanti
5 X = np.array([[1, 2], [np.nan, 3], [7, 6], [np.nan, np.nan]])
6
7 # Strategia: media
8 imputer = SimpleImputer(strategy='mean')
9 X_imputed = imputer.fit_transform(X)
10
11 # Altre strategie disponibili:
12 # - 'median': mediana
13 # - 'most_frequent': moda
14 # - 'constant': valore costante
```

# Algoritmi di Classificazione in Scikit-Learn

## Algoritmi Lineari

- Logistic Regression
- Linear SVM
- Perceptron
- SGD Classifier

## Algoritmi Non Lineari

- Decision Trees
- Random Forest
- K-Nearest Neighbors

## Metodi Avanzati

- Support Vector Machines
- Gradient Boosting
- Neural Networks (MLP)
- Naive Bayes

## Ensemble Methods

- Bagging
- AdaBoost
- Voting Classifier

# Logistic Regression

```
1  from sklearn.linear_model import LogisticRegression
2  from sklearn.datasets import make_classification
3  from sklearn.model_selection import train_test_split
4
5  # Generazione dataset
6  X, y = make_classification(n_samples=1000, n_features=4,
7  n_classes=2, random_state=42)
8
9  # Split dei dati
10 X_train, X_test, y_train, y_test = train_test_split(
11     X, y, test_size=0.2, random_state=42
12 )
13
14 # Addestramento
15 log_reg = LogisticRegression(max_iter=1000)
16 log_reg.fit(X_train, y_train)
17
18 # Predizione
19 y_pred = log_reg.predict(X_test)
```

# Decision Tree Classifier

```
1 from sklearn.tree import DecisionTreeClassifier, plot_tree
2 import matplotlib.pyplot as plt
3
4 # Creazione e addestramento
5 dt = DecisionTreeClassifier(max_depth=3, random_state=42)
6 dt.fit(X_train, y_train)
7
8 # Visualizzazione albero
9 plt.figure(figsize=(15, 10))
10 plot_tree(dt, filled=True, feature_names=['F1', 'F2', 'F3', 'F4'],
11            class_names=['Class 0', 'Class 1'])
12 plt.show()
13
14 # Importanza features
15 importances = dt.feature_importances_
16 for i, imp in enumerate(importances):
17     print(f"Feature {i}: {imp:.3f}")
```

# Random Forest Classifier

```
1  from sklearn.ensemble import RandomForestClassifier  
2  
3 # Creazione Random Forest  
4 rf = RandomForestClassifier(  
5     n_estimators=100,          # numero di alberi  
6     max_depth=5,            # profondità massima  
7     min_samples_split=5,    # min campioni per split  
8     random_state=42  
9 )  
10  
11 # Addestramento  
12 rf.fit(X_train, y_train)  
13  
14 # Valutazione  
15 train_score = rf.score(X_train, y_train)  
16 test_score = rf.score(X_test, y_test)  
17  
18 print(f"Train accuracy: {train_score:.3f}")  
19 print(f"Test accuracy: {test_score:.3f}")
```

# K-Nearest Neighbors (KNN)

```
1 from sklearn.neighbors import KNeighborsClassifier  
2  
3 # Creazione modello KNN  
4 knn = KNeighborsClassifier(  
5     n_neighbors=5,          # numero di vicini  
6     weights='distance',    # peso in base alla distanza  
7     metric='euclidean')    # metrica di distanza  
8  
9 # Addestramento  
10 knn.fit(X_train, y_train)  
11  
12 # Predizione con probabilita  
13 y_pred_proba = knn.predict_proba(X_test)  
14 print("Probabilita prime 5 predizioni:")  
15 print(y_pred_proba[:5])  
16  
17 # Accuracy  
18 accuracy = knn.score(X_test, y_test)
```

# Support Vector Machine (SVM)

```
1 from sklearn.svm import SVC
2
3 # SVM lineare
4 svm_linear = SVC(kernel='linear', C=1.0)
5 svm_linear.fit(X_train, y_train)
6
7 # SVM con kernel RBF
8 svm_rbf = SVC(kernel='rbf', C=1.0, gamma='scale')
9 svm_rbf.fit(X_train, y_train)
10
11 # SVM polinomiale
12 svm_poly = SVC(kernel='poly', degree=3, C=1.0)
13 svm_poly.fit(X_train, y_train)
14
15 # Confronto accuracy
16 print(f"Linear SVM: {svm_linear.score(X_test, y_test):.3f}")
17 print(f"RBF SVM: {svm_rbf.score(X_test, y_test):.3f}")
18 print(f"Poly SVM: {svm_poly.score(X_test, y_test):.3f}")
```

## Regressione Lineare

- Linear Regression
- Ridge Regression
- Lasso Regression
- ElasticNet

## Regressione Non Lineare

- Decision Tree Regressor
- Random Forest Regressor
- SVR

## Metodi Avanzati

- Gradient Boosting Regressor
- AdaBoost Regressor
- Multi-layer Perceptron

## Polynomial Features

- Polynomial Regression
- Feature Engineering

# Linear Regression

```
1  from sklearn.linear_model import LinearRegression
2  from sklearn.datasets import make_regression
3
4  # Generazione dataset
5  X, y = make_regression(n_samples=100, n_features=1,
6                         noise=10, random_state=42)
7
8  # Creazione e addestramento
9  lr = LinearRegression()
10 lr.fit(X, y)
11
12 # Parametri del modello
13 print(f"Coefficiente: {lr.coef_[0]:.3f}")
14 print(f"Intercetta: {lr.intercept_:.3f}")
15
16 # Predizione
17 y_pred = lr.predict(X)
18
19 # Metriche
```

# Ridge e Lasso Regression

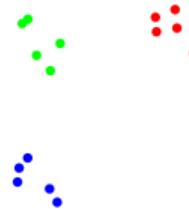
```
1 from sklearn.linear_model import Ridge, Lasso  
2  
3 # Ridge Regression (L2 regularization)  
4 ridge = Ridge(alpha=1.0)  
ridge.fit(X_train, y_train)  
ridge_score = ridge.score(X_test, y_test)  
5  
6 # Lasso Regression (L1 regularization)  
7 lasso = Lasso(alpha=0.1)  
lasso.fit(X_train, y_train)  
lasso_score = lasso.score(X_test, y_test)  
8  
9 # ElasticNet (combinazione L1 e L2)  
from sklearn.linear_model import ElasticNet  
elastic = ElasticNet(alpha=0.1, l1_ratio=0.5)  
elastic.fit(X_train, y_train)  
elastic_score = elastic.score(X_test, y_test)  
10  
11 print(f"Ridge R\textsuperscript{2}: {ridge_score:.3f}")  
12
```

# Polynomial Regression

```
1 from sklearn.preprocessing import PolynomialFeatures
2 from sklearn.pipeline import make_pipeline
3
4 # Creazione pipeline con features polinomiali
5 degree = 3
6 poly_model = make_pipeline(
7     PolynomialFeatures(degree),
8     LinearRegression()
9 )
10
11 # Addestramento
12 poly_model.fit(X_train, y_train)
13
14 # Valutazione
15 train_score = poly_model.score(X_train, y_train)
16 test_score = poly_model.score(X_test, y_test)
17
18 print(f"Training R\textsuperscript{2}: {train_score:.3f}")
19 print(f"Test R\textsuperscript{2}: {test_score:.3f}")
```

## Algoritmi Principali

- K-Means
- DBSCAN
- Hierarchical Clustering
- Mean Shift
- Gaussian Mixture



# K-Means Clustering

```
1  from sklearn.cluster import KMeans
2  from sklearn.datasets import make_blobs
3
4  # Generazione dataset
5  X, _ = make_blobs(n_samples=300, centers=4,
6                     cluster_std=0.6, random_state=42)
7
8  # K-Means con 4 cluster
9  kmeans = KMeans(n_clusters=4, random_state=42)
10 labels = kmeans.fit_predict(X)
11
12 # Centri dei cluster
13 centers = kmeans.cluster_centers_
14 print("Centri dei cluster:")
15 print(centers)
16
17 # Inerzia (somma distanze quadrate)
18 print(f"Inerzia: {kmeans.inertia_:.2f}")
```

# DBSCAN Clustering

```
1 from sklearn.cluster import DBSCAN
2
3 # DBSCAN clustering
4 dbSCAN = DBSCAN(eps=0.5, min_samples=5)
5 labels = dbSCAN.fit_predict(X)
6
7 # Numero di cluster trovati
8 n_clusters = len(set(labels)) - (1 if -1 in labels else 0)
9 n_noise = list(labels).count(-1)
10
11 print(f"Numero di cluster: {n_clusters}")
12 print(f"Punti noise: {n_noise}")
13
14 # Core samples
15 core_samples_mask = np.zeros_like(labels, dtype=bool)
16 core_samples_mask[dbSCAN.core_sample_indices_] = True
```

## Vantaggi DBSCAN:

## Metodi Lineari

- PCA (Principal Component Analysis)
- TruncatedSVD
- Factor Analysis

## Metodi Non Lineari

- t-SNE
- Isomap
- Locally Linear Embedding

## Applicazioni

- Visualizzazione dati
- Riduzione rumore
- Feature extraction
- Compressione dati
- Miglioramento performance

# Principal Component Analysis (PCA)

```
1  from sklearn.decomposition import PCA  
2  
3  # PCA con 2 componenti  
4  pca = PCA(n_components=2)  
5  X_pca = pca.fit_transform(X)  
6  
7  # Varianza spiegata  
8  print("Varianza spiegata per componente:")  
9  print(pca.explained_variance_ratio_)  
10 print(f"Varianza totale: {sum(pca.explained_variance_ratio_):.3f}")  
11  
12 # Componenti principali  
13 print("\nComponenti principali:")  
14 print(pca.components_)  
15  
16 # PCA con soglia di varianza  
17 pca_var = PCA(n_components=0.95) # 95% varianza  
18 X_reduced = pca_var.fit_transform(X)  
19 print(f"Dimensioni ridotte: {X_reduced.shape[1]}")
```

# Metriche per Classificazione

```
from sklearn.metrics import (accuracy_score, precision_score,
                             recall_score, f1_score,
                             confusion_matrix, classification_report)

# Calcolo metriche
accuracy = accuracy_score(y_test, y_pred)
precision = precision_score(y_test, y_pred, average='weighted')
recall = recall_score(y_test, y_pred, average='weighted')
f1 = f1_score(y_test, y_pred, average='weighted')

print(f"Accuracy: {accuracy:.3f}")
print(f"Precision: {precision:.3f}")
print(f"Recall: {recall:.3f}")
print(f"F1-Score: {f1:.3f}")

# Confusion Matrix
cm = confusion_matrix(y_test, y_pred)
print("\nConfusion Matrix:")
print(cm)
```

# Classification Report

```
1 from sklearn.metrics import classification_report  
2  
3 # Report completo  
4 report = classification_report(y_test, y_pred,  
5                                 target_names=['Class 0', 'Class 1'])  
6 print(report)
```

Output esempio:

	precision	recall	f1-score	support
--	-----------	--------	----------	---------

Class 0	0.85	0.90	0.87	100
Class 1	0.89	0.83	0.86	100

accuracy			0.87	200
macro avg	0.87	0.87	0.87	200
weighted avg	0.87	0.87	0.87	200

# ROC Curve e AUC

```
1 from sklearn.metrics import roc_curve, roc_auc_score
2 import matplotlib.pyplot as plt
3
4 # Calcolo ROC
5 y_proba = classifier.predict_proba(X_test)[:, 1]
6 fpr, tpr, thresholds = roc_curve(y_test, y_proba)
7 auc = roc_auc_score(y_test, y_proba)
8
9 # Visualizzazione
10 plt.figure(figsize=(8, 6))
11 plt.plot(fpr, tpr, label=f'ROC curve (AUC = {auc:.2f})')
12 plt.plot([0, 1], [0, 1], 'k--', label='Random')
13 plt.xlabel('False Positive Rate')
14 plt.ylabel('True Positive Rate')
15 plt.title('ROC Curve')
16 plt.legend()
17 plt.grid(True)
18 plt.show()
```

# Metriche per Regressione

```
1  from sklearn.metrics import (mean_squared_error,
2                                 mean_absolute_error,
3                                 r2_score,
4                                 mean_absolute_percentage_error)
5
6 # Calcolo metriche
7 mse = mean_squared_error(y_test, y_pred)
8 rmse = np.sqrt(mse)
9 mae = mean_absolute_error(y_test, y_pred)
10 r2 = r2_score(y_test, y_pred)
11 mape = mean_absolute_percentage_error(y_test, y_pred)
12
13 print(f"MSE: {mse:.3f}")
14 print(f"RMSE: {rmse:.3f}")
15 print(f"MAE: {mae:.3f}")
16 print(f"R\textsuperscript{2} Score: {r2:.3f}")
17 print(f"MAPE: {mape:.3f}")
```

# Train-Test Split

```
1  from sklearn.model_selection import train_test_split
2
3  # Split semplice
4  X_train, X_test, y_train, y_test = train_test_split(
5      X, y,
6      test_size=0.2,          # 20% test set
7      random_state=42,        # riproducibilita
8      stratify=y              # mantiene proporzioni classi
9 )
10
11 # Split triplo (train/validation/test)
12 X_temp, X_test, y_temp, y_test = train_test_split(
13     X, y, test_size=0.2, random_state=42
14 )
15 X_train, X_val, y_train, y_val = train_test_split(
16     X_temp, y_temp, test_size=0.25, random_state=42
17 )
18 # Risultato: 60% train, 20% val, 20% test
```

# Cross-Validation

```
1 from sklearn.model_selection import cross_val_score, KFold
2
3 # K-Fold Cross-Validation
4 kfold = KFold(n_splits=5, shuffle=True, random_state=42)
5 scores = cross_val_score(model, X, y, cv=kfold,
6                           scoring='accuracy')
7
8 print(f"Scores per fold: {scores}")
9 print(f"Media: {scores.mean():.3f}")
10 print(f"Deviazione standard: {scores.std():.3f}")
11
12 # Stratified K-Fold (per classificazione)
13 from sklearn.model_selection import StratifiedKFold
14 skfold = StratifiedKFold(n_splits=5, shuffle=True,
15                         random_state=42)
16 scores_strat = cross_val_score(model, X, y, cv=skfold)
```

# Grid Search CV

```
1  from sklearn.model_selection import GridSearchCV
2
3  # Definizione griglia parametri
4  param_grid = {
5      'n_estimators': [50, 100, 200],
6      'max_depth': [3, 5, 7, 10],
7      'min_samples_split': [2, 5, 10],
8      'min_samples_leaf': [1, 2, 4]
9  }
10
11 # Grid Search
12 grid_search = GridSearchCV(
13     RandomForestClassifier(random_state=42),
14     param_grid,
15     cv=5,
16     scoring='accuracy',
17     n_jobs=-1,
18     verbose=1
19 )
```

# Randomized Search CV

```
1 from sklearn.model_selection import RandomizedSearchCV
2 from scipy.stats import randint, uniform
3
4 # Distribuzioni parametri
5 param_distributions = {
6     'n_estimators': randint(50, 500),
7     'max_depth': randint(3, 20),
8     'min_samples_split': randint(2, 20),
9     'min_samples_leaf': randint(1, 10),
10    'max_features': uniform(0.1, 0.9)
11}
12
13 # Randomized Search
14 random_search = RandomizedSearchCV(
15     RandomForestClassifier(random_state=42),
16     param_distributions,
17     n_iter=100,          # numero iterazioni
18     cv=5,
19     random_state=42,
```

# Pipeline: Workflow Integrato

```
1  from sklearn.pipeline import Pipeline
2  from sklearn.preprocessing import StandardScaler
3  from sklearn.decomposition import PCA
4
5  # Creazione pipeline
6  pipeline = Pipeline([
7      ('scaler', StandardScaler()),
8      ('pca', PCA(n_components=2)),
9      ('classifier', RandomForestClassifier())
10 ])
11
12 # Addestramento pipeline
13 pipeline.fit(X_train, y_train)
14
15 # Predizione
16 y_pred = pipeline.predict(X_test)
17
18 # Accesso ai componenti
19 scaler = pipeline.named_steps['scaler']
```

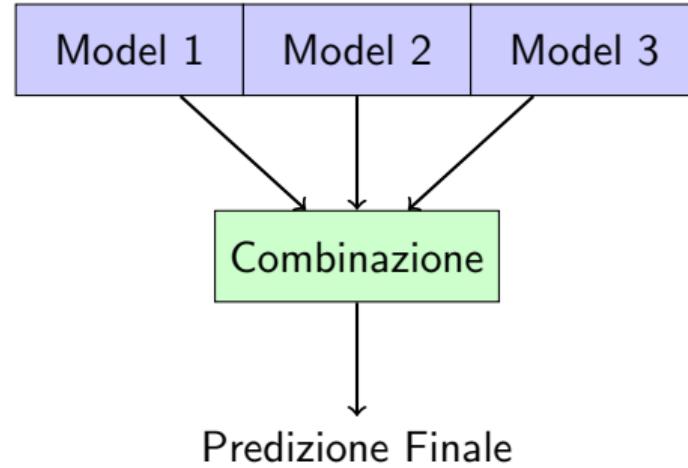
# ColumnTransformer

```
1  from sklearn.compose import ColumnTransformer
2  from sklearn.preprocessing import StandardScaler, OneHotEncoder
3
4  # Definizione trasformazioni per colonne diverse
5  preprocessor = ColumnTransformer(
6      transformers=[
7          ('num', StandardScaler(), ['age', 'salary']),
8          ('cat', OneHotEncoder(), ['city', 'gender'])
9      ]
10 )
11
12 # Pipeline completa
13 full_pipeline = Pipeline([
14     ('preprocessor', preprocessor),
15     ('classifier', LogisticRegression())
16 ])
17
18 # Addestramento
19 full_pipeline.fit(X_train, y_train)
```

# Feature Selection

```
1  from sklearn.feature_selection import (SelectKBest,
2                                         f_classif, RFE)
3
4  # SelectKBest: seleziona k migliori features
5  selector = SelectKBest(score_func=f_classif, k=5)
6  X_selected = selector.fit_transform(X_train, y_train)
7
8  # Recursive Feature Elimination
9  from sklearn.linear_model import LogisticRegression
0  rfe = RFE(estimator=LogisticRegression(), n_features_to_select=5)
1  X_rfe = rfe.fit_transform(X_train, y_train)
2
3  # Feature importance da Random Forest
4  rf = RandomForestClassifier()
5  rf.fit(X_train, y_train)
6  importances = rf.feature_importances_
7
8  # Selezione features importanti
9  threshold = 0.1
```

# Ensemble Methods



## Tipi principali:

- **Bagging:** Bootstrap Aggregating (Random Forest)
- **Boosting:** AdaBoost, Gradient Boosting
- **Voting:** Combinazione predizioni multiple
- **Stacking:** Meta-learning

# Voting Classifier

```
1  from sklearn.ensemble import VotingClassifier
2  from sklearn.tree import DecisionTreeClassifier
3  from sklearn.linear_model import LogisticRegression
4  from sklearn.svm import SVC
5
6  # Definizione modelli base
7  clf1 = LogisticRegression(max_iter=1000)
8  clf2 = DecisionTreeClassifier()
9  clf3 = SVC(probability=True)
10
11 # Voting Classifier
12 voting_clf = VotingClassifier(
13     estimators=[('lr', clf1), ('dt', clf2), ('svc', clf3)],
14     voting='soft' # 'hard' per majority voting
15 )
16
17 # Addestramento
18 voting_clf.fit(X_train, y_train)
```

# Gradient Boosting

```
1 from sklearn.ensemble import GradientBoostingClassifier  
2  
3 # Gradient Boosting  
4 gb = GradientBoostingClassifier(  
5     n_estimators=100,  
6     learning_rate=0.1,  
7     max_depth=3,  
8     random_state=42  
9 )  
10  
11 gb.fit(X_train, y_train)  
12  
13 # Feature importance  
14 importances = gb.feature_importances_  
15  
16 # Curve di apprendimento  
17 train_scores = []  
18 test_scores = []  
19 for i, y_pred in enumerate(gb.staged_predict(X_test)):  
20     pass
```

## Do

- Normalizza i dati
- Usa cross-validation
- Valuta con metriche appropriate
- Documenta il codice
- Versiona i modelli
- Monitora overfitting

## Don't

- Non fare fit su test set
- Non ignorare data leakage
- Non usare sempre accuracy
- Non trascurare l'EDA
- Non dimenticare feature engineering
- Non saltare la validazione

## Documentazione Ufficiale

- <https://scikit-learn.org/>
- [https://scikit-learn.org/stable/user\\_guide.html](https://scikit-learn.org/stable/user_guide.html)

## Tutorial e Corsi

- Scikit-Learn Tutorial ufficiale
- Kaggle Learn
- DataCamp, Coursera

## Community

- Stack Overflow
- GitHub Issues
- Reddit r/MachineLearning

## Punti Chiave

- Scikit-Learn è la libreria standard per ML in Python
- API semplice e consistente
- Vasta gamma di algoritmi
- Ottimi strumenti per preprocessing e valutazione
- Integrazione perfetta con NumPy, Pandas, Matplotlib

**Grazie per l'attenzione!**

Domande?