

Scikit-Learn

Machine Learning in Python

Prof. Massimo Fedeli

IIS Fermi Sacconi Cpia

2 dicembre 2025

- 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)

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

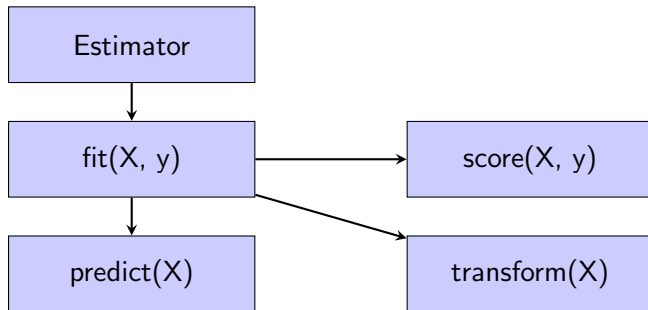
- Classificazione
- Regressione
- Predizione

Unsupervised Learning

- Clustering
- Riduzione dimensionalità
- Anomaly detection

Altri ambiti

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



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)
```


`fit(X, y)` - Addestramento

- Impara dai dati di training
- `X`: features (matrice $n_samples \times n_features$)
- `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

`transform(X)` - Trasformazione

- Trasforma i dati secondo le regole apprese con `fit()`
- Usato principalmente nei preprocessor (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

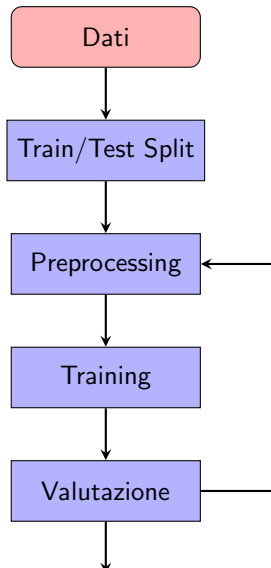
```
from sklearn.datasets import load_iris
from sklearn.tree import DecisionTreeClassifier
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import train_test_split

# Caricamento dati
X, y = load_iris(return_X_y=True)
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.3, random_state=42
)

# Preprocessing: transform
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train) # fit + transform
X_test_scaled = scaler.transform(X_test)      # solo transform

# Classificazione: fit, predict, score
clf = DecisionTreeClassifier()
clf.fit(X_train_scaled, y_train)               # addestramento
```

Workflow Tipico del Machine Learning



Primo Esempio: Classificazione Iris

```
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import accuracy_score

# Caricamento dataset
iris = load_iris()
X, y = iris.data, iris.target

# Divisione train/test
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.3, random_state=42
)

# Addestramento modello
clf = DecisionTreeClassifier()
clf.fit(X_train, y_train)

# 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!

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 Categoricalhe

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

```
from sklearn.preprocessing import StandardScaler, MinMaxScaler

# StandardScaler: media 0, varianza 1
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X_train)

# MinMaxScaler: range [0, 1]
min_max_scaler = MinMaxScaler()
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: μ = media, σ = deviazione standard

```
1 from sklearn.  
    preprocessing  
    import  
    StandardScaler  
  
2 import numpy  
    as np  
3 # Dati  
    originali  
4 X = np.array  
    ([[50,  
        150000],
```

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 Categorie

```
from sklearn.preprocessing import LabelEncoder, OneHotEncoder

# LabelEncoder per variabili ordinali
le = LabelEncoder()
labels = ['rosso', 'verde', 'blu', 'rosso']
encoded = le.fit_transform(labels)
# Output: [2, 1, 0, 2]

# OneHotEncoder per variabili nominali
from sklearn.preprocessing import OneHotEncoder
import numpy as np

ohe = OneHotEncoder(sparse_output=False)
colors = np.array([[ 'rosso'], [ 'verde'], [ 'blu']])
encoded_ohe = ohe.fit_transform(colors)
# Output: [[0, 0, 1],
#           [0, 1, 0],
#           [1, 0, 0]]
```

Encoding Variabili Categorie: 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 Categorie:

- 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  
    LabelEncoder  
  
le =  
    LabelEncoder  
    (  
  
3  
# Esempio:  
4     taglie (  
        ordine!)
```

OneHotEncoder

Per variabili NOMINALI

Crea una colonna binaria per ogni categoria

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

```



```
from sklearn.impute import SimpleImputer
import numpy as np

# Dati con valori mancanti
X = np.array([[1, 2], [np.nan, 3], [7, 6], [np.nan, np.nan]])

# Strategia: media
imputer = SimpleImputer(strategy='mean')
X_imputed = imputer.fit_transform(X)

# Altre strategie disponibili:
# - 'median': mediana
# - 'most_frequent': moda
# - 'constant': valore costante
```

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

```
from sklearn.linear_model import LogisticRegression
from sklearn.datasets import make_classification
from sklearn.model_selection import train_test_split

# Generazione dataset
X, y = make_classification(n_samples=1000, n_features=4,
                           n_classes=2, random_state=42)

# Split dei dati
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.2, random_state=42
)

# Addestramento
log_reg = LogisticRegression(max_iter=1000)
log_reg.fit(X_train, y_train)

# Predizione
y_pred = log_reg.predict(X_test)
```

Decision Tree Classifier

```
from sklearn.tree import DecisionTreeClassifier, plot_tree
import matplotlib.pyplot as plt

# Creazione e addestramento
dt = DecisionTreeClassifier(max_depth=3, random_state=42)
dt.fit(X_train, y_train)

# Visualizzazione albero
plt.figure(figsize=(15, 10))
plot_tree(dt, filled=True, feature_names=['F1', 'F2', 'F3', 'F4'],
          class_names=['Class 0', 'Class 1'])
plt.show()

# Importanza features
importances = dt.feature_importances_
for i, imp in enumerate(importances):
    print(f"Feature {i}: {imp:.3f}")
```

Random Forest Classifier

```
from sklearn.ensemble import RandomForestClassifier

# Creazione Random Forest
rf = RandomForestClassifier(
    n_estimators=100,      # numero di alberi
    max_depth=5,          # profondita massima
    min_samples_split=5,  # min campioni per split
    random_state=42
)

# Addestramento
rf.fit(X_train, y_train)

# Valutazione
train_score = rf.score(X_train, y_train)
test_score = rf.score(X_test, y_test)

print(f"Train accuracy: {train_score:.3f}")
print(f"Test accuracy: {test_score:.3f}")
```

K-Nearest Neighbors (KNN)

```
from sklearn.neighbors import KNeighborsClassifier

# Creazione modello KNN
knn = KNeighborsClassifier(
    n_neighbors=5,          # numero di vicini
    weights='distance',    # peso in base alla distanza
    metric='euclidean'     # metrica di distanza
)

# Addestramento
knn.fit(X_train, y_train)

# Predizione con probabilita
y_pred_proba = knn.predict_proba(X_test)
print("Probabilita prime 5 predizioni:")
print(y_pred_proba[:5])

# Accuracy
accuracy = knn.score(X_test, y_test)
```

Support Vector Machine (SVM)

```
from sklearn.svm import SVC

# SVM lineare
svm_linear = SVC(kernel='linear', C=1.0)
svm_linear.fit(X_train, y_train)

# SVM con kernel RBF
svm_rbf = SVC(kernel='rbf', C=1.0, gamma='scale')
svm_rbf.fit(X_train, y_train)

# SVM polinomiale
svm_poly = SVC(kernel='poly', degree=3, C=1.0)
svm_poly.fit(X_train, y_train)

# Confronto accuracy
print(f"Linear SVM: {svm_linear.score(X_test, y_test):.3f}")
print(f"RBF SVM: {svm_rbf.score(X_test, y_test):.3f}")
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

```
from sklearn.linear_model import LinearRegression
from sklearn.datasets import make_regression

# Generazione dataset
X, y = make_regression(n_samples=100, n_features=1,
                      noise=10, random_state=42)

# Creazione e addestramento
lr = LinearRegression()
lr.fit(X, y)

# Parametri del modello
print(f"Coefficiente: {lr.coef_[0]:.3f}")
print(f"Intercetta: {lr.intercept_:.3f}")

# Predizione
y_pred = lr.predict(X)

# Metriche
```

Ridge e Lasso Regression

```
from sklearn.linear_model import Ridge, Lasso

# Ridge Regression (L2 regularization)
ridge = Ridge(alpha=1.0)
ridge.fit(X_train, y_train)
ridge_score = ridge.score(X_test, y_test)

# Lasso Regression (L1 regularization)
lasso = Lasso(alpha=0.1)
lasso.fit(X_train, y_train)
lasso_score = lasso.score(X_test, y_test)

# 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)

print(f"Ridge R2: {ridge_score:.3f}")
```

Polynomial Regression

```
from sklearn.preprocessing import PolynomialFeatures
from sklearn.pipeline import make_pipeline

# Creazione pipeline con features polinomiali
degree = 3
poly_model = make_pipeline(
    PolynomialFeatures(degree),
    LinearRegression()
)

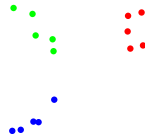
# Addestramento
poly_model.fit(X_train, y_train)

# Valutazione
train_score = poly_model.score(X_train, y_train)
test_score = poly_model.score(X_test, y_test)

print(f"Training R2: {train_score:.3f}")
print(f"Test R2: {test_score:.3f}")
```

Algoritmi Principali

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



K-Means Clustering

```
from sklearn.cluster import KMeans
from sklearn.datasets import make_blobs

# Generazione dataset
X, _ = make_blobs(n_samples=300, centers=4,
                  cluster_std=0.6, random_state=42)

# K-Means con 4 cluster
kmeans = KMeans(n_clusters=4, random_state=42)
labels = kmeans.fit_predict(X)

# Centri dei cluster
centers = kmeans.cluster_centers_
print("Centri dei cluster:")
print(centers)

# Inerzia (somma distanze quadrate)
print(f"Inerzia: {kmeans.inertia_:.2f}")
```

DBSCAN Clustering

```
from sklearn.cluster import DBSCAN

# DBSCAN clustering
dbscan = DBSCAN(eps=0.5, min_samples=5)
labels = dbscan.fit_predict(X)

# Numero di cluster trovati
n_clusters = len(set(labels)) - (1 if -1 in labels else 0)
n_noise = list(labels).count(-1)

print(f"Numero di cluster: {n_clusters}")
print(f"Punti noise: {n_noise}")

# Core samples
core_samples_mask = np.zeros_like(labels, dtype=bool)
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)

```
from sklearn.decomposition import PCA

# PCA con 2 componenti
pca = PCA(n_components=2)
X_pca = pca.fit_transform(X)

# Varianza spiegata
print("Varianza spiegata per componente:")
print(pca.explained_variance_ratio_)
print(f"Varianza totale: {sum(pca.explained_variance_ratio_):.3f}")

# Componenti principali
print("\nComponenti principali:")
print(pca.components_)

# PCA con soglia di varianza
pca_var = PCA(n_components=0.95) # 95% varianza
X_reduced = pca_var.fit_transform(X)
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

```
from sklearn.metrics import classification_report

# Report completo
report = classification_report(y_test, y_pred,
                              target_names=['Class 0', 'Class 1'])

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

```
from sklearn.metrics import roc_curve, roc_auc_score
import matplotlib.pyplot as plt

# Calcolo ROC
y_proba = classifier.predict_proba(X_test)[: , 1]
fpr, tpr, thresholds = roc_curve(y_test, y_proba)
auc = roc_auc_score(y_test, y_proba)

# Visualizzazione
plt.figure(figsize=(8, 6))
plt.plot(fpr, tpr, label=f'ROC curve (AUC = {auc:.2f})')
plt.plot([0, 1], [0, 1], 'k--', label='Random')
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('ROC Curve')
plt.legend()
plt.grid(True)
plt.show()
```

Metriche per Regressione

```
from sklearn.metrics import (mean_squared_error,
                              mean_absolute_error,
                              r2_score,
                              mean_absolute_percentage_error)

# Calcolo metriche
mse = mean_squared_error(y_test, y_pred)
rmse = np.sqrt(mse)
mae = mean_absolute_error(y_test, y_pred)
r2 = r2_score(y_test, y_pred)
mape = mean_absolute_percentage_error(y_test, y_pred)

print(f"MSE: {mse:.3f}")
print(f"RMSE: {rmse:.3f}")
print(f"MAE: {mae:.3f}")
print(f"R2 Score: {r2:.3f}")
print(f"MAPE: {mape:.3f}")
```

Train-Test Split

```
from sklearn.model_selection import train_test_split

# Split semplice
X_train, X_test, y_train, y_test = train_test_split(
    X, y,
    test_size=0.2,          # 20% test set
    random_state=42,        # riproducibilit 
    stratify=y              # mantiene proporzioni classi
)

# Split triplo (train/validation/test)
X_temp, X_test, y_temp, y_test = train_test_split(
    X, y, test_size=0.2, random_state=42
)
X_train, X_val, y_train, y_val = train_test_split(
    X_temp, y_temp, test_size=0.25, random_state=42
)

# Risultato: 60% train, 20% val, 20% test
```

Cross-Validation

```
from sklearn.model_selection import cross_val_score, KFold

# K-Fold Cross-Validation
kfold = KFold(n_splits=5, shuffle=True, random_state=42)
scores = cross_val_score(model, X, y, cv=kfold,
                          scoring='accuracy')

print(f"Scores per fold: {scores}")
print(f"Media: {scores.mean():.3f}")
print(f"Deviazione standard: {scores.std():.3f}")

# Stratified K-Fold (per classificazione)
from sklearn.model_selection import StratifiedKFold
skfold = StratifiedKFold(n_splits=5, shuffle=True,
                          random_state=42)
scores_strat = cross_val_score(model, X, y, cv=skfold)
```

Grid Search CV

```
from sklearn.model_selection import GridSearchCV

# Definizione griglia parametri
param_grid = {
    'n_estimators': [50, 100, 200],
    'max_depth': [3, 5, 7, 10],
    'min_samples_split': [2, 5, 10],
    'min_samples_leaf': [1, 2, 4]
}

# Grid Search
grid_search = GridSearchCV(
    RandomForestClassifier(random_state=42),
    param_grid,
    cv=5,
    scoring='accuracy',
    n_jobs=-1,
    verbose=1
)
```

Randomized Search CV

```
from sklearn.model_selection import RandomizedSearchCV
from scipy.stats import randint, uniform

# Distribuzioni parametri
param_distributions = {
    'n_estimators': randint(50, 500),
    'max_depth': randint(3, 20),
    'min_samples_split': randint(2, 20),
    'min_samples_leaf': randint(1, 10),
    'max_features': uniform(0.1, 0.9)
}

# Randomized Search
random_search = RandomizedSearchCV(
    RandomForestClassifier(random_state=42),
    param_distributions,
    n_iter=100,          # numero iterazioni
    cv=5,
    random_state=42,
```


Pipeline: Workflow Integrato

```
from sklearn.pipeline import Pipeline
from sklearn.preprocessing import StandardScaler
from sklearn.decomposition import PCA

# Creazione pipeline
pipeline = Pipeline([
    ('scaler', StandardScaler()),
    ('pca', PCA(n_components=2)),
    ('classifier', RandomForestClassifier())
])

# Addestramento pipeline
pipeline.fit(X_train, y_train)

# Predizione
y_pred = pipeline.predict(X_test)

# Accesso ai componenti
scaler = pipeline.named_steps['scaler']
```

ColumnTransformer

```
from sklearn.compose import ColumnTransformer
from sklearn.preprocessing import StandardScaler, OneHotEncoder

# Definizione trasformazioni per colonne diverse
preprocessor = ColumnTransformer(
    transformers=[
        ('num', StandardScaler(), ['age', 'salary']),
        ('cat', OneHotEncoder(), ['city', 'gender'])
    ]
)

# Pipeline completa
full_pipeline = Pipeline([
    ('preprocessor', preprocessor),
    ('classifier', LogisticRegression())
])

# Addestramento
full_pipeline.fit(X_train, y_train)
```

Feature Selection

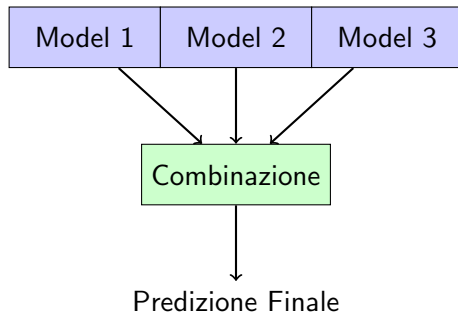
```
from sklearn.feature_selection import (SelectKBest,
                                       f_classif, RFE)

# SelectKBest: seleziona k migliori features
selector = SelectKBest(score_func=f_classif, k=5)
X_selected = selector.fit_transform(X_train, y_train)

# Recursive Feature Elimination
from sklearn.linear_model import LogisticRegression
rfe = RFE(estimator=LogisticRegression(), n_features_to_select=5)
X_rfe = rfe.fit_transform(X_train, y_train)

# Feature importance da Random Forest
rf = RandomForestClassifier()
rf.fit(X_train, y_train)
importances = rf.feature_importances_

# Selezione features importanti
threshold = 0.1
```



Tipi principali:

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

Voting Classifier

```
from sklearn.ensemble import VotingClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.svm import SVC

# Definizione modelli base
clf1 = LogisticRegression(max_iter=1000)
clf2 = DecisionTreeClassifier()
clf3 = SVC(probability=True)

# Voting Classifier
voting_clf = VotingClassifier(
    estimators=[('lr', clf1), ('dt', clf2), ('svc', clf3)],
    voting='soft' # 'hard' per majority voting
)

# Addestramento
voting_clf.fit(X_train, y_train)
```

Gradient Boosting

```
from sklearn.ensemble import GradientBoostingClassifier

# Gradient Boosting
gb = GradientBoostingClassifier(
    n_estimators=100,
    learning_rate=0.1,
    max_depth=3,
    random_state=42
)

gb.fit(X_train, y_train)

# Feature importance
importances = gb.feature_importances_

# Curve di apprendimento
train_scores = []
test_scores = []
for i, y_pred in enumerate(gb.staged_predict(X_test)):
```

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

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?