Preparare i dati

Garbage in, garbage out

Prima regola del Machine Learning: **i dati sono tutto**. Puoi avere l'algoritmo più figo del mondo, ma se i dati fanno schifo, il risultato farà schifo. "Garbage in, garbage out" dicono gli americani — spazzatura dentro, spazzatura fuori.

In questo capitolo impariamo a lavorare con i dati usando **NumPy** (per i calcoli), **Matplotlib** (per i grafici) e **scikit-learn** (per il ML vero e proprio).

NumPy: il motore dei dati

NumPy è la libreria Python per lavorare con array di numeri in modo veloce e compatto. Se hai fatto il capitolo sulle matrici, è ora di fare il salto di qualità!

```
import numpy as np

# Creare array
numeri = np.array([1, 2, 3, 4, 5])
print("Array:", numeri)
print("Tipo:", type(numeri))

# Operazioni su tutti gli elementi in un colpo!
print("Doppio:", numeri * 2)
print("Quadrato:", numeri ** 2)
print("Somma:", np.sum(numeri))
print("Media:", np.mean(numeri))
print("Dev. standard:", np.std(numeri))
```

Array 2D (tipo una tabella)

```
import numpy as np

# Dataset: 5 studenti, 3 features (ore_studio, assenze, voto_precedente)
dati = np.array([
    [5, 2, 7],
    [8, 0, 8],
    [2, 10, 5],
    [6, 3, 6],
    [7, 1, 7],
])

print("Shape:", dati.shape)  # (5, 3) → 5 righe, 3 colonne
print("Righe (studenti):", dati.shape[0])
print("Colonne (features):", dati.shape[1])

# Accesso: dati[riga, colonna]
print("
Studente 0, tutte le features:", dati[0])
print("Tutti gli studenti, colonna 0 (ore):", dati[:, 0])
print("Media ore studio:", np.mean(dati[:, 0]))
```

Generare dati casuali

Nel ML spesso creiamo **dati sintetici** per esercitarci. NumPy ha tutto quello che serve:

```
import numpy as np

# Seed: rende i numeri "casuali" riproducibili
np.random.seed(42)

# 100 numeri casuali tra 0 e 1
random_base = np.random.rand(100)

# 100 numeri con distribuzione normale (media=0, std=1)
normali = np.random.randn(100)

# Numeri interi casuali tra 1 e 10
interi = np.random.randint(1, 11, size=100)

print(f"Casuali [0,1]: media={random_base.mean():.2f}, min={random_base.
print(f"Normali:       media={normali.mean():.2f}, std={normali.std():.2
print(f"Interi [1,10]: media={interi.mean():.2f}, moda più comune: {np.b
```

Visualizzare i dati con Matplotlib

I grafici sono fondamentali nel ML: prima di far partire qualsiasi algoritmo, **guarda i dati!** Un grafico vale più di mille numeri.

Scatter plot (grafico a punti)

```
import numpy as np
import matplotlib.pyplot as plt

np.random.seed(42)

# Generiamo dati: ore di studio vs voto
ore_studio = np.random.uniform(1, 10, 50)
voti = ore_studio * 0.8 + 2 + np.random.randn(50) * 0.8  # Relazione lin

plt.figure(figsize=(8, 5))
plt.scatter(ore_studio, voti, color='dodgerblue', alpha=0.7)
plt.xlabel('Ore di studio')
plt.ylabel('Voto')
plt.title('Ore di studio vs Voto')
plt.grid(True, alpha=0.3)
plt.show()
```

Istogramma

```
import numpy as np
import matplotlib.pyplot as plt

np.random.seed(42)

# Voti di una classe (distribuzione "realistica")
voti = np.random.normal(6.5, 1.5, 100)
voti = np.clip(voti, 2, 10)  # Limita tra 2 e 10

plt.figure(figsize=(8, 5))
plt.hist(voti, bins=15, color='coral', edgecolor='black', alpha=0.7)
plt.xlabel('Voto')
plt.ylabel('Frequenza')
plt.title('Distribuzione dei voti della classe')
plt.axvline(np.mean(voti), color='red', linestyle='--', label=f'Media: {
plt.legend()
plt.grid(True, alpha=0.3)
plt.show()
```

Grafici con più serie

```
import numpy as np
import matplotlib.pyplot as plt

np.random.seed(42)

ore = np.linspace(1, 10, 50)

# Due gruppi: chi studia con metodo e chi studia a casaccio
con_metodo = ore * 0.9 + 1.5 + np.random.randn(50) * 0.5
senza_metodo = ore * 0.5 + 2 + np.random.randn(50) * 1.2

plt.figure(figsize=(8, 5))
plt.scatter(ore, con_metodo, color='green', alpha=0.7, label='Con metodo
plt.scatter(ore, senza_metodo, color='red', alpha=0.7, label='Senza meto
plt.xlabel('Ore di studio')
plt.ylabel('Voto')
plt.title('Studiare con metodo fa la differenza!')
plt.legend()
plt.grid(True, alpha=0.3)
plt.show()
```

Normalizzazione: mettere i dati in scala

Immagina di avere due features: **età** (15-19) e **reddito familiare** (20.000-80.000). Il reddito ha numeri MOLTO più grandi dell'età. Molti algoritmi ML impazziscono con scale diverse — pensano che il reddito sia più "importante" solo perché ha numeri più grandi!

La soluzione: **normalizzare** i dati, cioè portarli tutti sulla stessa scala.

Min-Max Scaling (porta tutto tra 0 e 1)

La formula è: `x_norm = (x - min) / (max - min)`

```
import numpy as np

# Dati con scale diverse
eta = np.array([15, 16, 17, 18, 19])
reddito = np.array([25000, 35000, 45000, 60000, 80000])

print("PRIMA della normalizzazione:")
print(f"  Età:     {eta}  (range: {eta.min()}-{eta.max()})")
print(f"  Reddito: {reddito}  (range: {reddito.min()}-{reddito.max()})")

# Normalizzazione a mano
eta_norm = (eta - eta.min()) / (eta.max() - eta.min())
reddito_norm = (reddito - reddito.min()) / (reddito.max() - reddito.min(

print("
DOPO la normalizzazione (0-1):")
print(f"  Età:     {eta_norm}")
print(f"  Reddito: {np.round(reddito_norm, 2)}")
print("
Ora entrambe sono tra 0 e 1!")
```

Con scikit-learn (più comodo!)

```
import numpy as np
from sklearn.preprocessing import MinMaxScaler

# Dataset con 2 features su scale diverse
dati = np.array([
    [15, 25000],
    [16, 35000],
    [17, 45000],
    [18, 60000],
    [19, 80000],
])

scaler = MinMaxScaler()
dati_norm = scaler.fit_transform(dati)

print("Prima:")
print(dati)
print("
Dopo normalizzazione:")
print(np.round(dati_norm, 2))
```

!!! warning "Quando normalizzare?"

```
- **Sempre** con KNN, reti neurali, e regressione se le features hanno s
- **Non necessario** con alberi di decisione (non sono sensibili alla sc
- Regola pratica: nel dubbio, normalizza!
```

Dividere i dati con scikit-learn

Abbiamo visto come dividere a mano. Ma scikit-learn ha una funzione che lo fa meglio, **mescolando** anche i dati in modo casuale (importantissimo!):

```
import numpy as np
from sklearn.model_selection import train_test_split

# Creiamo un dataset di esempio
np.random.seed(42)
X = np.random.rand(20, 3)  # 20 campioni, 3 features
y = np.random.randint(0, 2, 20)  # 20 etichette (0 o 1)

# Split: 80% training, 20% test
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.2, random_state=42
)

print(f"Dataset completo: {X.shape[0]} campioni")
print(f"Training set:     {X_train.shape[0]} campioni")
print(f"Test set:         {X_test.shape[0]} campioni")
print(f"
Etichette training: {y_train}")
print(f"Etichette test:     {y_test}")
```

!!! tip "Perché mescolare i dati?"

```
Immagina un dataset ordinato: prima tutti gli studenti promossi, poi tut
```

Creare un dataset sintetico completo

Mettiamo tutto insieme! Creiamo un dataset "vero" da usare nei prossimi capitoli:

```
import numpy as np
import matplotlib.pyplot as plt

np.random.seed(42)
n = 100  # 100 studenti

# Generiamo le features
ore_studio = np.random.uniform(0, 10, n)
assenze = np.random.randint(0, 15, n)
voto_precedente = np.random.uniform(3, 10, n)

# Il voto finale dipende dalle features (con un po' di rumore)
voto_finale = (
    ore_studio * 0.5
    + voto_precedente * 0.4
    - assenze * 0.2
    + np.random.randn(n) * 0.5
)
voto_finale = np.clip(voto_finale, 2, 10)  # Limita tra 2 e 10

# Mettiamo tutto in un array
X = np.column_stack([ore_studio, assenze, voto_precedente])
y = voto_finale

print(f"Dataset creato!")
print(f"Features (X): shape {X.shape}")
print(f"Labels (y): shape {y.shape}")
print(f"
print(f"
Primi 5 studenti:")
print(f"{'Ore':>6} {'Ass':>5} {'Prec':>6} {'Voto':>6}")
print("-" * 26)
for i in range(5):
    print(f"{X[i,0]:>6.1f} {X[i,1]:>5.0f} {X[i,2]:>6.1f} {y[i]:>6.1f}")

# Visualizziamo
fig, axes = plt.subplots(1, 3, figsize=(14, 4))

axes[0].scatter(X[:, 0], y, alpha=0.5, color='blue')
axes[0].set_xlabel('Ore studio')
```

```
axes[0].set_ylabel('Voto finale')
axes[0].set_title('Ore studio vs Voto')

axes[1].scatter(X[:, 1], y, alpha=0.5, color='red')
axes[1].set_xlabel('Assenze')
axes[1].set_title('Assenze vs Voto')

axes[2].scatter(X[:, 2], y, alpha=0.5, color='green')
axes[2].set_xlabel('Voto precedente')
axes[2].set_title('Voto prec. vs Voto')

plt.tight_layout()
plt.show()
```

Esercizi

Esercizio 1: Esplora un dataset

Crea un dataset con 50 studenti (ore_studio, media_precedente) e calcola: media, minimo, massimo e deviazione standard di ogni feature.

```
import numpy as np

np.random.seed(123)

# Crea il dataset (50 studenti, 2 features)

# Calcola e stampa le statistiche per ogni feature
```

??? success "Soluzione"

```pyodide
import numpy as np
np.random.seed(123)
ore_studio = np.random.uniform(0, 10, 50)
media_prec = np.random.uniform(4, 10, 50)
dataset = np.column_stack([ore_studio, media_prec])
nomi = ["Ore di studio", "Media precedente"]
for i, nome in enumerate(nomi):
    col = dataset[:, i]
    print(f"{nome}:")
    print(f"  Media: {col.mean():.2f}")
    print(f"  Min: {col.min():.2f}")
    print(f"  Max: {col.max():.2f}")
    print(f"  Std: {col.std():.2f}
```

") ```

Esercizio 2: Visualizza e normalizza

Genera un dataset con 2 features su scale molto diverse (es. altezza in cm e peso in kg), visualizzalo con uno scatter plot, poi normalizzalo e visualizzalo di nuovo.

```
import numpy as np
import matplotlib.pyplot as plt

np.random.seed(42)

# Crea i dati (altezza: 150-190 cm, peso: 45-95 kg)

# Visualizza PRIMA della normalizzazione

# Normalizza

# Visualizza DOPO la normalizzazione
```

??? success "Soluzione"

```pyodide install="matplotlib,numpy"
import numpy as np
import matplotlib.pyplot as plt
np.random.seed(42)
altezza = np.random.uniform(150, 190, 50)
peso = np.random.uniform(45, 95, 50)
fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(10, 4))
ax1.scatter(altezza, peso)
ax1.set_xlabel("Altezza (cm)")
ax1.set_ylabel("Peso (kg)")
ax1.set_title("Prima della normalizzazione")
alt_norm = (altezza - altezza.min()) / (altezza.max() - altezza.min())
peso_norm = (peso - peso.min()) / (peso.max() - peso.min())
ax2.scatter(alt_norm, peso_norm)
ax2.set_xlabel("Altezza (normalizzata)")
ax2.set_ylabel("Peso (normalizzato)")
ax2.set_title("Dopo la normalizzazione")
plt.tight_layout()
plt.show()
```

Esercizio 3: Train/test split personalizzato

Senza usare `train_test_split` di sklearn, scrivi una funzione che divide un dataset in training e test set con una percentuale a scelta. La funzione deve anche mescolare i dati!

```
import numpy as np

def split_dataset(X, y, test_size=0.2, seed=42):
    # Mescola i dati
    # Dividi in train e test
    # Restituisci X_train, X_test, y_train, y_test
    pass

# Testa con un dataset di esempio
np.random.seed(42)
X = np.random.rand(20, 2)
y = np.random.randint(0, 2, 20)

# X_train, X_test, y_train, y_test = split_dataset(X, y, test_size=0.2)
```

??? success "Soluzione"

```
```pyodide
import numpy as np

def split_dataset(X, y, test_size=0.2, seed=42):
    np.random.seed(seed)
    n = len(X)
    indici = np.arange(n)
    np.random.shuffle(indici)
    n_test = int(n * test_size)
    test_idx = indici[:n_test]
    train_idx = indici[n_test:]
    return X[train_idx], X[test_idx], y[train_idx], y[test_idx]

np.random.seed(42)
X = np.random.rand(20, 2)
y = np.random.randint(0, 2, 20)
X_train, X_test, y_train, y_test = split_dataset(X, y, test_size=0.2)
print(f"Dataset totale: {len(X)}")
print(f"Training set: {len(X_train)}")
print(f"Test set: {len(X_test)}")
```
```