Regressione lineare

Trovare la retta migliore

La regressione lineare è il primo vero algoritmo di Machine Learning che imparerai. L'idea è semplicissima: hai un po' di punti su un grafico e devi trovare la **retta che li descrive meglio**. Tipo: "più studi, più il voto sale" — ma *quanto* sale esattamente?

Se ricordi la matematica delle medie, la retta è: **y = mx + b**

- **m** = pendenza (quanto sale/scende la retta)
- **b** = intercetta (dove la retta "parte" sull'asse y)
- **x** = la feature (input)
- **y** = la previsione (output)

Il lavoro del ML è trovare i valori di **m** e **b** che fanno passare la retta il più vicino possibile a tutti i punti!

Vediamola in azione

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

np.random.seed(42)

# Dati: ore di studio vs voto
ore = np.array([1, 2, 3, 4, 5, 6, 7, 8, 9, 10], dtype=float)
voti = ore * 0.7 + 2.5 + np.random.randn(10) * 0.5

plt.figure(figsize=(8, 5))
plt.scatter(ore, voti, color='dodgerblue', s=80, zorder=5)
plt.xlabel('Ore di studio')
plt.ylabel('Voto')
plt.title('Ore di studio vs Voto')
plt.grid(True, alpha=0.3)
plt.show()

print("Vedi una tendenza? I punti salgono da sinistra a destra!")
print("Ora troviamo la retta che li descrive meglio...")
```

Gradient Descent: trovare m e b

Come fa il computer a trovare la retta migliore? Usa un algoritmo chiamato **Gradient Descent** (discesa del gradiente). L'idea è tipo camminare in montagna con gli occhi bendati: fai un passo, senti se vai in discesa, e continui a scendere finché non arrivi al punto più basso.

In pratica:

1. Parti con valori casuali di **m** e **b**
2. Calcola quanto "sbagli" con quei valori (l'**errore**)
3. Aggiusta m e b un pochino nella direzione che riduce l'errore
4. Ripeti finché l'errore non si stabilizza

```
import numpy as np

np.random.seed(42)

# Dati
X = np.array([1, 2, 3, 4, 5, 6, 7, 8, 9, 10], dtype=float)
y = X * 0.7 + 2.5 + np.random.randn(10) * 0.5

# Gradient Descent
m = 0.0  # Partiamo da zero
b = 0.0
lr = 0.01  # Learning rate: quanto "grandi" sono i passi
n = len(X)

print("Addestramento in corso...")
print(f"{'Epoca':>6} {'m':>8} {'b':>8} {'Errore':>10}")
print("-" * 36)

for epoca in range(1000):
    # Previsioni attuali
    y_pred = m * X + b

    # Errore medio quadratico (MSE)
    errore = np.mean((y - y_pred) ** 2)

    # Gradienti (la direzione per migliorare)
    dm = -2/n * np.sum(X * (y - y_pred))
    db = -2/n * np.sum(y - y_pred)

    # Aggiorna m e b
    m -= lr * dm
    b -= lr * db

    # Stampa ogni 200 epoche
    if epoca % 200 == 0:
        print(f"{epoca:>6} {m:>8.4f} {b:>8.4f} {errore:>10.4f}")

print(f"
```

```
Risultato: y = {m:.2f}x + {b:.2f}")
print(f"(La formula vera era: y = 0.70x + 2.50)")
```

!!! tip "Il Learning Rate"

```
Il `lr` (learning rate) è tipo la lunghezza dei passi in montagna. Se è
```

Visualizzare la retta trovata

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

np.random.seed(42)

X = np.array([1, 2, 3, 4, 5, 6, 7, 8, 9, 10], dtype=float)
y = X * 0.7 + 2.5 + np.random.randn(10) * 0.5

# Gradient Descent (veloce)
m, b, lr, n = 0.0, 0.0, 0.01, len(X)
for _ in range(1000):
    y_pred = m * X + b
    m -= (-2/n) * lr * np.sum(X * (y - y_pred))
    b -= (-2/n) * lr * np.sum(y - y_pred)

# Grafico
plt.figure(figsize=(8, 5))
plt.scatter(X, y, color='dodgerblue', s=80, zorder=5, label='Dati reali'
plt.plot(X, m * X + b, color='red', linewidth=2, label=f'y = {m:.2f}x +
plt.xlabel('Ore di studio')
plt.ylabel('Voto')
plt.title('Regressione lineare: la retta trovata')
plt.legend()
plt.grid(True, alpha=0.3)
plt.show()
```

Con scikit-learn (3 righe di codice)

Tutto quel lavoro di gradient descent... scikit-learn lo fa in **3 righe**. Ecco il potere delle librerie!

```
import numpy as np
from sklearn.linear_model import LinearRegression

np.random.seed(42)

X = np.array([1, 2, 3, 4, 5, 6, 7, 8, 9, 10], dtype=float).reshape(-1, 1
y = X.flatten() * 0.7 + 2.5 + np.random.randn(10) * 0.5

# 3 righe magiche!
modello = LinearRegression()
modello.fit(X, y)         # Addestra
previsioni = modello.predict(X)  # Prevedi

print(f"Coefficiente (m): {modello.coef_[0]:.4f}")
print(f"Intercetta (b):   {modello.intercept_:.4f}")
print(f"Formula: y = {modello.coef_[0]:.2f}x + {modello.intercept_:.2f}"
```

!!! warning "Perché reshape(-1, 1)?"

```
Scikit-learn vuole le features come una matrice (array 2D), non come un
```

Valutare il modello

Ok, la retta sembra buona. Ma *quanto* buona? Servono delle **metriche**:

MSE (Mean Squared Error)

La media dei quadrati degli errori. Più è bassa, meglio è.

R² (R-squared)

Indica quanto bene la retta "spiega" i dati. Va da 0 a 1:

- **$R^2 = 1$**: la retta passa perfettamente per tutti i punti (impossibile nella realtà!)
- **$R^2 = 0.8$**: la retta spiega l'80% della variazione nei dati (molto buono!)
- **$R^2 = 0$**: la retta non spiega nulla (meglio tirare a caso)

```
import numpy as np
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error, r2_score
from sklearn.model_selection import train_test_split

np.random.seed(42)

# Dataset più grande
X = np.random.uniform(1, 10, 50).reshape(-1, 1)
y = X.flatten() * 0.7 + 2.5 + np.random.randn(50) * 0.8

# Split train/test
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,

# Addestra
modello = LinearRegression()
modello.fit(X_train, y_train)

# Prevedi sul TEST set (dati mai visti!)
y_pred = modello.predict(X_test)

# Metriche
mse = mean_squared_error(y_test, y_pred)
r2 = r2_score(y_test, y_pred)

print(f"Formula: y = {modello.coef_[0]:.2f}x + {modello.intercept_:.2f}"
print(f"MSE:  {mse:.4f} (piu' basso = meglio)")
print(f"R2:   {r2:.4f} (piu' vicino a 1 = meglio)")

if r2 > 0.7:
    print("
Il modello e' buono!")
elif r2 > 0.4:
    print("
Il modello e' accettabile, ma si puo' migliorare.")
else:
    print("
Il modello non funziona bene.")
```

Esempio pratico: prevedere i voti

Mettiamo tutto insieme con un esempio completo. Vogliamo prevedere il voto di uno studente in base alle ore di studio:

```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.linear_model import LinearRegression
from sklearn.model_selection import train_test_split
from sklearn.metrics import r2_score

np.random.seed(42)

# Dataset: 100 studenti
ore = np.random.uniform(0, 10, 100)
voti = ore * 0.65 + 2.8 + np.random.randn(100) * 0.7
voti = np.clip(voti, 2, 10)

X = ore.reshape(-1, 1)
y = voti

# Split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,

# Addestra
modello = LinearRegression()
modello.fit(X_train, y_train)

# Valuta
r2 = r2_score(y_test, modello.predict(X_test))

# Grafico
plt.figure(figsize=(8, 5))
plt.scatter(X_train, y_train, alpha=0.5, label='Training')
plt.scatter(X_test, y_test, alpha=0.5, color='orange', label='Test')
x_line = np.linspace(0, 10, 100).reshape(-1, 1)
plt.plot(x_line, modello.predict(x_line), color='red', linewidth=2,
         label=f'y = {modello.coef_[0]:.2f}x + {modello.intercept_:.2f}'
plt.xlabel('Ore di studio')
plt.ylabel('Voto')
plt.title(f'Previsione voti (R2 = {r2:.2f})')
plt.legend()
plt.grid(True, alpha=0.3)
plt.show()
```

```
# Previsioni su studenti nuovi
print("
Previsioni per nuovi studenti:")
nuove_ore = [2, 5, 8]
for ore_s in nuove_ore:
    voto_previsto = modello.predict([[ore_s]])[0]
    print(f"  Studio {ore_s} ore -> voto previsto: {voto_previsto:.1f}")
```

Esercizi

Esercizio 1: Gradient Descent personalizzato

Modifica il gradient descent visto sopra per trovare la retta che passa meglio per questi dati. Prova con diversi learning rate (0.001, 0.01, 0.1) e confronta i risultati.

```
import numpy as np

X = np.array([2, 4, 6, 8, 10, 12], dtype=float)
y = np.array([5, 10, 14, 20, 24, 30], dtype=float)

# Implementa il gradient descent con diversi learning rate
```

Esercizio 2: Regressione con sklearn

Crea un dataset dove il voto dipende da ORE DI STUDIO e VOTO PRECEDENTE (2 features). Addestra un modello di regressione lineare e fai delle previsioni.

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

np.random.seed(42)

# Crea il dataset con 2 features

# Addestra il modello

# Fai previsioni e valuta
```

Esercizio 3: Confronto visuale

Genera 3 dataset diversi (uno molto lineare, uno con tanto rumore, uno non lineare) e mostra come la regressione lineare funziona bene solo sul primo.

```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.linear_model import LinearRegression

np.random.seed(42)

# Dataset 1: molto lineare
# Dataset 2: lineare con molto rumore
# Dataset 3: non lineare (es. quadratico)

# Per ognuno: addestra, prevedi, calcola R2, plotta
```