SVILUPPO APPLICAZIONI E ANALISI DATI CON PYTHON

Machine Learning

Fondamenti e Applicazioni Pratiche

Scikit-learn, Pandas, NumPy - Python 3.x

Docente: Marino Speciale

Indice

- Cos'è Machine Learning? Supervised/Unsupervised
- Workflow ML: Data → Train → Test → Evaluate
- Train-Test Split e Preprocessing (Scaling)
- Supervised Learning: Regressione e Classificazione
- Unsupervised Learning: Clustering e Dimensionality Reduction
- Metriche di Valutazione e Cross-Validation
- Hyperparameter Tuning e Best Practices

Cos'è Machine Learning?

- ML = Apprendimento da dati, non da istruzioni esplicite
- Il modello impara pattern dai dati di addestramento
- Poi predice/generalizza su dati nuovi (test set)
- Applicazioni: spam detection, raccomandazioni, Computer Vision, NLP

Analizziamoli in dettaglio 

Apprendimento da dati vs Istruzioni esplicite

Nel software tradizionale (programmazione esplicita), il programmatore scrive regole del tipo SE (if) succede questo, ALLORA (then) fai quello.

Nel ML, questo paradigma si ribalta.

- Programmazione Classica: Input + Regole → Output.
- ML: Input + Output desiderati → Regole (il Modello).

Analogia: Invece di dare a un robot un manuale di 1000 pagine su come riconoscere un gatto (istruzioni esplicite), gli mostri 10.000 foto di gatti e gli dici: "Trova cosa hanno in comune" (apprendimento dai dati).

Esempio di apprendimento da dati (Induzione): Nella programmazione classica, tu scrivi l'algoritmo f e fornisci i dati x per ottenere y ($f(x) = y$). Nel ML, fornisci x e y all'elaboratore affinché esso "apprenda" la funzione f più probabile. Tecnicamente, parliamo di approssimazione di funzioni.

Imparare pattern dai dati di addestramento

Il "pattern" è una regolarità statistica. Se addestriamo un modello a riconoscere email di spam, l'algoritmo non "legge" come un umano, ma cerca correlazioni matematiche.

- **Esempio tecnico**: Il modello potrebbe notare che la presenza della parola "Gratis" combinata con un link sospetto e un mittente mai visto ha una probabilità del 98% di essere spam. Questa combinazione di fattori è il pattern.
- **Approfondimento**: Matematicamente, questo significa minimizzare una Funzione di Costo (Loss Function). Il modello prova una regola, vede quanto sbaglia rispetto ai dati reali, e aggiusta i suoi parametri interni tramite un processo chiamato Gradient Descent (Discesa del Gradiente).

Il modello cerca regolarità statistiche nei dati.
Se parliamo di una rete neurale, questi pattern sono memorizzati sotto forma di pesi (ω) e bias (b) all'interno dei neuroni artificiali.

Predire e Generalizzare (Test Set)

Il vero obiettivo del ML non è ricordare i dati che ha già visto, ma capire quelli futuri.

Questo si chiama Generalizzazione.

- Training Set: È il libro di testo su cui il modello studia.
- Test Set: È l'esame finale con domande (dati) che il modello non ha mai visto prima.
- Il rischio (Overfitting): Se il modello "impara a memoria" i dati di addestramento ma fallisce sul test set, si dice che è in *overfitting*. È come uno studente che impara le risposte a memoria ma non capisce la materia: se cambi una virgola nella domanda, non sa rispondere.

Generalizzazione vs Overfitting: L'obiettivo non è che il modello "impari a memoria" i dati di addestramento (questo sarebbe overfitting), ma che estragga regole applicabili a dati mai visti. Il Test Set serve proprio a misurare questa capacità di generalizzazione su dati "unseen".

Le Applicazioni

Spam Detection (Filtro email): È un classificatore. Analizza le frequenze delle parole e i metadati per assegnare un'etichetta (Spam o Non Spam).

- Tecnica comune: Naive Bayes o Logistic Regression.

Raccomandazioni (Netflix, Amazon): Algoritmi che creano un "profilo matematico" dei tuoi gusti e lo confrontano con quello di milioni di altri utenti.

- Tecnica comune: Collaborative Filtering. Se l'utente A e l'utente B sono simili e B ha guardato un film che A non ha visto, il sistema lo consiglia ad A.

CV - Computer Vision (Vista artificiale): Permette alle macchine di interpretare i pixel. Un'immagine per il computer è solo una griglia di numeri (intensità di colore). Il ML trasforma quei numeri in concetti (volto, auto, strada).

- Tecnica core: CNN (Convolutional Neural Networks), che filtrano l'immagine per strati, dai bordi alle forme complesse.

NLP - Natural Language Processing (Linguaggio naturale): Il computer traduce le parole in vettori numerici (Word Embeddings). In questo modo, può calcolare la "distanza" semantica tra le parole (es. "Re" e "Regina" sono matematicamente vicini).

- Tecnica core: Transformers (la tecnologia dietro ChatGPT), che usano il meccanismo di Attention per capire il contesto di una parola all'interno di una frase.

Supervised vs Unsupervised Learning

🏷️ SUPERVISED:

- Hai: X (features) + y (labels)
- Tipi: Regressione, Classificazione

🔍 UNSUPERVISED:

- Hai: X (features) SOLO
- Tipi: Clustering, Dimensionality Reduction

Supervised Learning (Apprendimento Supervisionato)

In questo scenario, il modello ha un "insegnante".

Ogni dato di input (*X*) è accompagnato dalla risposta corretta (*y*), definita **Label** o etichetta.

I due sottogruppi principali:

1. Regressione: L'output *y* è un valore continuo (numerico).

- Esempio: Predire il prezzo di una casa (*y*) basandosi sulla superficie e la posizione (*X*).
- Approfondimento: Si basa sulla minimizzazione dell'errore quadratico medio (*MSE*):

$$MSE = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2$$

2. Classificazione: L'output *y* è un valore discreto (una categoria).

- Esempio: Spam detection (Spam / Non Spam) o riconoscimento di immagini (Gatto / Cane).
- Tecnica: Si usa spesso la funzione Softmax per ottenere una distribuzione di probabilità sulle varie classi.

Unsupervised Learning (Apprendimento non supervisionato)

Qui il modello lavora "al buio". Non ci sono label (*y*), solo dati grezzi (*X*). L'obiettivo è scoprire la struttura intrinseca dei dati.

I due sottogruppi principali:

1. **Clustering**: Raggruppare i dati in base alla loro somiglianza matematica (distanza euclidea o similarità del coseno).
   - Algoritmo principe: K-Means. Viene usato per la segmentazione della clientela nel marketing.
2. **Dimensionality Reduction** (Riduzione della Dimensionalità): Ridurre il numero di variabili (features) mantenendo però le informazioni più importanti.
   - Tecnica chiave: PCA (Principal Component Analysis). Serve a semplificare dataset enormi o a visualizzare dati complessi in 2D o 3D, eliminando il "rumore" statistico.

Workflow Machine Learning

1. Load – Caricamento dei dataset (CSV, DB, API)
2. Explore – Analisi, visualizzazione, statistiche
3. Preprocess – Pulizia, missing values, scaling
4. Split – Train (70-80%) e Test (20-30%)
5. Model Selection – Scegli algoritmo
6. Train – Fit modello su train set
7. Evaluate – Misura su test set (accuracy, MSE)
8. Tune – Ottimizza hyperparameters
9. Deploy – Usa in produzione

Preprocessing

Il **Pre-processing** è la fase del ciclo di vita del Machine Learning che trasforma i dati grezzi (*raw data*) in un formato adatto all'addestramento dei modelli.

Poiché gli algoritmi sono essenzialmente motori matematici, la qualità dei risultati dipende direttamente dalla pulizia e dalla coerenza dei dati in ingresso (concetto noto come *Garbage In, Garbage Out*).

Si basa su tre pilastri:

1. Pulisci (Data Cleaning)
2. Missing Values (Dati Mancanti)
3. Scaling (Normalizzazione e Standardizzazione)

Preprocessing - Pulizia (Data Cleaning)

La pulizia dei dati mira a rimuovere il "rumore" e le incongruenze che potrebbero fuorviare l'apprendimento del modello.

- Rimozione dei duplicati: Record identici possono dare un peso eccessivo a determinati pattern, portando all'overfitting.
- Gestione degli outlier (Valori Anomali): Identificazione di punti dati che deviano significativamente dalla distribuzione statistica. Tecnicamente si usano metodi come il Z-score o l'Interquartile Range (IQR) per decidere se eliminare o correggere questi valori.
- Correzione di errori strutturali: Uniformare le stringhe (es. "Milano", "milano", "MI") o correggere errori di battitura macroscopici.

Preprocessing - Missing Values (Dati Mancanti)

In molti dataset reali, alcune informazioni sono assenti (indicate come `NaN` o `NULL`). I modelli di ML, tranne rare eccezioni (come XGBoost), non possono processare dati mancanti.

Le strategie di gestione includono:

- Deletion (Cancellazione): Eliminare l'intera riga o colonna. Si usa solo se la perdita di dati è trascurabile (<5%).
- Imputazione univariata: Sostituire il valore mancante con la media, la mediana (più robusta agli outlier) o la moda.
- Imputazione multivariata (K-NN Imputation): Si utilizza un algoritmo (come il K-Nearest Neighbors) per stimare il valore mancante basandosi sui valori degli altri record simili.

Preprocessing - Scaling (Normalizzazione e Standardizzazione)

Lo Scaling è il processo di trasformazione delle feature numeriche affinché abbiano tutte lo stesso ordine di grandezza. Senza scaling, una variabile con valori grandi (es. lo stipendio, 50.000€) dominerebbe impropriamente su una piccola (es. l'età, 30 anni) durante il calcolo delle distanze matematiche.Le due tecniche principali sono:

A. Normalizzazione (Min-Max Scaling)

Porta tutti i valori nell'intervallo [0, 1]. È utile quando la distribuzione dei dati non è necessariamente gaussiana.

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

B. Standardizzazione (Z-score Normalization)

Trasforma i dati affinché abbiano media $\mu = 0$ e deviazione standard $\sigma = 1$. È preferibile quando l'algoritmo assume una distribuzione normale (es. Linear Regression o SVM).

$$x_{new} = \frac{x - \mu}{\sigma}$$

Preprocessing: Scaling

❌ SENZA SCALING: Se le feature (colonne del dataset) hanno scale diverse — ad esempio, una è tra 0 e 1, un'altra tra 0 e 10.000 — i modelli possono dare troppa importanza a quelle con valori grandi. → Questo succede soprattutto con modelli che usano distanze (es. KNN, SVM) o gradiente (es. regressione, reti neurali).

✅ CON SCALING: Lo StandardScaler trasforma ogni feature per avere:

- media = 0
- deviazione standard = 1

```
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)  # su training set
X_test_scaled = scaler.transform(X_test)        # su test set
```

✅ Questo rende tutte le feature comparabili e migliora la performance del modello.

- `fit_transform` calcola media e `std` → va fatto solo sul training set
- `transform` applica la stessa trasformazione → va fatto sul test set
- Così eviti data leakage (il modello non "vede" il test set durante l'addestramento)

Train-Test Split (70-30)

```
# Importa la funzione che divide il dataset
from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.3, random_state=42)

model.fit(X_train, y_train)              # Allena: Il modello impara dai dati di training
accuracy = model.score(X_test, y_test) # Valuta: Misura l'accuratezza del modello sui dati di test
```

- X = input (feature)
- y = target (label)
- test_size=0.3 → 30% test, 70% train
- random_state=42 → rende la divisione ripetibile

È una tecnica fondamentale nel ML per valutare le prestazioni di un modello.
Obiettivo: dividere il dataset in due parti:
- 70% → per allenare il modello (X_train, y_train)
- 30% → per valutarlo su dati mai visti (X_test, y_test)

Perché è importante:
- Evita overfitting: il modello non deve "memorizzare" tutto
- Permette di stimare quanto bene generalizza
- È la base per tecniche più avanzate come cross-validation

Regressione Lineare

Usando la libreria scikit-learn vediamo un modello semplice ma potente per **prevedere valori numerici** in base a una variabile.
Obiettivo: Prevedere il **prezzo di una casa** in base alla **superficie in m²**.
Trova una relazione lineare del tipo: Prezzo = a * Area +b

```
# Importa il modello di regressione lineare
from sklearn.linear_model import LinearRegression

X = [[100], [150], [200]]              # Area in m² - x = superficie della casa
y = [200000, 300000, 400000]           # Prezzo in € - y = prezzo corrispondente

#Crea il modello e lo allena sui dati: trova la retta che meglio collega superficie e prezzo
model = LinearRegression()
model.fit(X, y)

# Prevede il prezzo di una casa di 175 m². Il risultato è € 350.000
y_pred = model.predict([[175]])  # 350000
```

Classificazione: Logistic Regression

Classificazione binaria usando la Logistic Regression di scikit-learn. È un modello molto usato per problemi come spam detection, diagnosi mediche, previsioni di eventi, ecc.
In questo esempio si prevede se un'email è spam (1) o non spam (0) in base a delle feature (es. numero di link, parole sospette, ecc.).

```
from sklearn.linear_model import LogisticRegression

model = LogisticRegression()                          # crea il modello
model.fit(X_train, y_train)  # 0=non spam, 1=spam     # lo allena sui dati di training

y_prob = model.predict_proba(X_train) # restituisce le probabilità che ogni es. sia spam o no

y_pred = model.predict(X_train)       # restituisce la classe predetta: 0 o 1

accuracy = accuracy_score(y_train, y_pred)  # Calcola l'accuratezza: % di predizioni corrette
```

Decision Tree

Un albero decisionale è un modello che:

- Divide lo spazio dei dati in rami basati su condizioni (es. "se X > 5")
- Arriva alle foglie che rappresentano le classi (es. "spam" o "non spam")

È molto intuitivo e interpretabile, ma può diventare troppo preciso sui dati di training.

```
from sklearn.tree import DecisionTreeClassifier

model = DecisionTreeClassifier(
    max_depth=5,           # limita la profondità dell'albero (quanti livelli di decisione)
    min_samples_split=10)  # ogni nodo deve avere almeno 10 esempi per essere diviso
model.fit(X_train, y_train)# allena il modello sui dati
```

⚠️ ATTENZIONE: Tendente all'overfitting! Un albero troppo profondo può:

- Imparare rumore invece di regole generali
- Fare predizioni perfette sul training set
- Fallire completamente sul test set

Soluzione: Random Forest (ensemble)

Random Forest (Ensemble)

Una Random Forest è un ensemble di tanti alberi:

- Ogni albero vede solo una parte dei dati
- Le predizioni vengono mediate
- Il risultato è più robusto, meno soggetto a overfitting

```
from sklearn.ensemble import RandomForestClassifier

model = RandomForestClassifier(
    n_estimators=100,           # Numero di alberi
    max_depth=10,
    random_state=42)
model.fit(X_train, y_train)
```

Unsupervised Learning: K-Means

K-Means è un algoritmo che:

- raggruppa i dati in k cluster (gruppi distinti dei dati)
- ogni punto viene assegnato al centroide più vicino (le X rosse nel grafico)
- i centroidi vengono aggiornati finché la soluzione si stabilizza

È utile quando non abbiamo etichette (y) e vogliamo scoprire strutture nascoste nei dati.

Quando si usa K-Means?

- Segmentazione clienti
- Raggruppamento di immagini
- Analisi esplorativa
- Riduzione della complessità

Esempio: Clustering K-Means (Unsupervised)

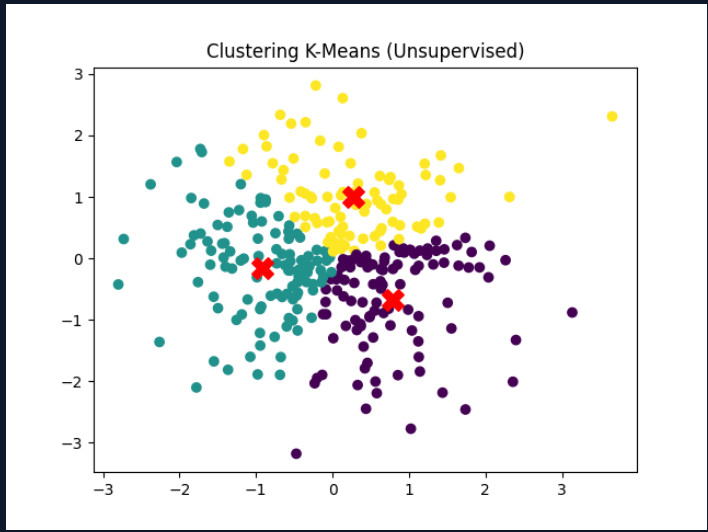
```
import matplotlib.pyplot as plt
import numpy as np
from sklearn.cluster import KMeans
from sklearn.preprocessing import StandardScaler # Per lo Scaling

# 1. GENERAZIONE DATI (Creazione della variabile X)
# Creiamo 300 punti casuali in uno spazio 2D
X = np.random.randn(300, 2)

# 2. PREPROCESS - Scaling
# Lo scaling è fondamentale per il K-Means perché si basa sulla distanza euclidea
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)  # Applichiamo il scaler ai dati

# 3. MODELLO
model = KMeans(n_clusters=3,        # Crea un modello con 3 cluster
               random_state=42)     # 42 serve per rendere il risultato ripetibile
clusters = model.fit_predict(X_scaled)  # Applichiamo il modello ai dati scalati
                                        # Niente etichette y! è unsupervised learning
# 4. VISUALIZZAZIONE
# Recuperiamo i centri dei cluster (centroidi)
centers = model.cluster_centers_

plt.scatter(X_scaled[:, 0], X_scaled[:, 1], c=clusters, cmap='viridis') # Visualizza i punti colorati in base al cluster assegnato
plt.scatter(centers[:, 0], centers[:, 1], c='red', marker='X', s=200, label='Centroidi')    # Mostra i centroidi dei cluster con una X rossa
plt.title("Clustering K-Means (Unsupervised)")
plt.show()
```



DBSCAN

Mentre l'esempio precedente usava KMeans, il DBSCAN (Density-Based Spatial Clustering of Applications with Noise) è un algoritmo di Unsupervised Learning molto più avanzato:

- **Non serve definire K**: A differenza di KMeans, non devi dire quanti cluster vuoi. Li trova lui in base alla densità.
- **Gestione del Rumore (Noise)**: È uno dei pochi algoritmi che identifica automaticamente gli outlier (punti isolati), etichettandoli con -1. Questo si ricollega alla fase di "Pulizia" citata nelle slide.
- **Parametri Chiave**:
  - `eps=30`: La distanza massima per considerare due punti "vicini". Nota: se scali i dati (StandardScaler), un valore di 30 è troppo alto; solitamente si usa tra 0.1 e 1.0.
  - `min_samples=2`: Il numero minimo di punti necessari per formare un cluster denso.

DBSCAN è molto sensibile al valore di eps. Potrai approfondire l'utilizzo del Metodo del K-Gomito (K-Elbow) applicato alle distanze dei vicini (K-Nearest Neighbors) per calcolare matematicamente il valore ottimale di eps invece di sceglierlo a caso.

Esempio: DBSCAN

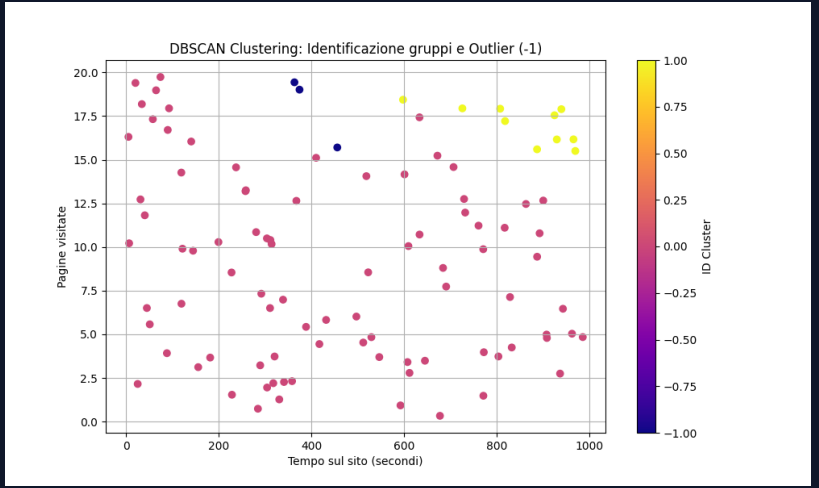
```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.cluster import DBSCAN
from sklearn.preprocessing import StandardScaler

# 1. Creazione dati X (Feature Space)
# Generiamo dati casuali per simulare il comportamento degli utenti
np.random.seed(42)
X = np.random.rand(100, 2) * [1000, 20] # Tempo fino a 1000s, Pagine fino a 20

# 2. PREPROCESS - Scaling
# Senza questo, DBSCAN fallirebbe a causa delle diverse unità di misura
X_scaled = StandardScaler().fit_transform(X)

# 3. MODELLO DBSCAN (Unsupervised Learning)
# Usiamo eps=0.5 perché i dati ora sono scalati (media 0, varianza 1)
dbscan = DBSCAN(eps=0.5, min_samples=5)
db_clusters = dbscan.fit_predict(X_scaled)

# 4. VISUALIZZAZIONE
plt.figure(figsize=(10, 6))
scatter = plt.scatter(X[:, 0], X[:, 1], c=db_clusters, cmap='plasma')
plt.title("DBSCAN Clustering: Identificazione gruppi e Outlier (-1)")
plt.xlabel("Tempo sul sito (secondi)")
plt.ylabel("Pagine visitate")
plt.colorbar(scatter, label='ID Cluster')
plt.grid(True)
plt.show()
```



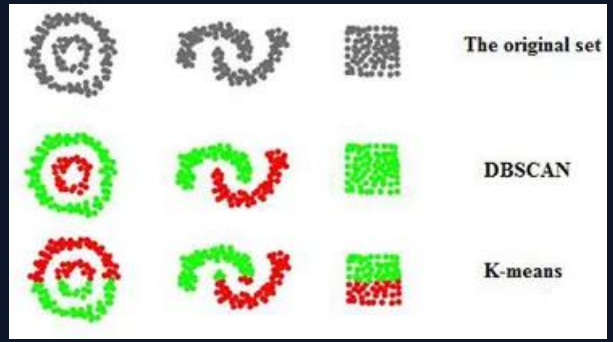
Differenza tra K-Means e DBSCAN

K-Means

- Tipo di algoritmo: Si basa sui centroidi.
- Configurazione: Richiede che l'utente specifichi in anticipo il numero di cluster (*k*).
- Gestione del rumore: Non gestisce efficacemente i punti di rumore (outlier).
- Forma dei cluster: È ottimizzato per trovare cluster di forma circolare o sferica.

DBSCAN

- Tipo di algoritmo: Si basa sulla densità.
- Configurazione: Non richiede di specificare il numero esatto di cluster in anticipo.
- Gestione del rumore: È in grado di gestire e identificare i punti considerati rumore.
- Forma dei cluster: Può rilevare cluster di forma arbitraria e complessa.



PCA: Dimensionality Reduction

Principal Component Analysis è una tecnica di riduzione dimensionale molto usata nel machine learning e nell'analisi dei dati.
La PCA serve per:

- ridurre il numero di feature (es. da 1000 a 2)
- mantenere più informazione possibile
- semplificare il dataset per visualizzazione, modellazione o preprocessing

Lo fa trovando nuove variabili (componenti principali) che:

- sono combinazioni lineari delle originali
- massimizzano la varianza (cioè l'informazione)

Quando si usa PCA?

- Per visualizzare dati complessi (es. in 2D)
- Per accelerare modelli (meno feature → meno calcoli)
- Per eliminare rumore o ridondanza
- Come preprocessing prima di clustering o classificazione

PCA: Dimensionality Reduction

```
from sklearn.decomposition import PCA


# Dataset: 1000 features → Riduci a 2
pca = PCA(n_components=2)
X_reduced = pca.fit_transform(X)


print(f'Varianza spiegata: {pca.explained_variance_ratio_}')
# [0.45, 0.23, ...] → Prime 2: 68%
```

Approfondimento tecnico sulla PCA

Perché ridurre le dimensioni?
In molti dataset (come quelli con 1000 features), molte variabili sono correlate tra loro o rappresentano semplice "rumore". La PCA trasforma le variabili originali in un nuovo set di variabili chiamate Componenti Principali (PC) che sono:

1. **Ortogonali** (indipendenti l'una dall'altra).
2. **Ordinate per importanza** (la PC1 cattura la massima varianza possibile).

Il concetto di Varianza (risultato: [0.45, 0.23] significa che:

- La prima componente (PC1) da sola riesce a riassumere il 45% dell'informazione totale contenuta nelle 1000 variabili originali.
- La seconda componente (PC2) aggiunge un ulteriore 23%.
- Insieme, riducendo i dati da 1000 a sole 2 dimensioni, hai mantenuto il 68% dell'informazione originale, eliminando il 32% di rumore o ridondanza.

Connessione con il Preprocessing
Matematicamente, la PCA effettua la Decomposizione agli Autovalori della matrice di covarianza. Se non pulisci i dati e non applichi lo scaling, le variabili con unità di misura più grandi domineranno la matrice di covarianza, rendendo la riduzione di dimensionalità tecnicamente errata.
Visualizzazione (Il vantaggio della PCA): Uno dei motivi principali per cui si riduce a n_components=2 è la possibilità di visualizzare dati complessi su un grafico 2D:

Principali Metriche di Valutazione per Modelli di classificazione

Sono fondamentali per capire quanto bene il modello sta funzionando, soprattutto quando le classi sono sbilanciate.

```
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score
```

- Accuracy = (TP + TN) / Total - Percentuale di predizioni corrette (sia positivi che negativi)
- Precision = TP / (TP + FP) - Quanto sono affidabili i positivi predetti
- Recall = TP / (TP + FN) - Quanto riesce a catturare tutti i veri positivi
- F1 = Media armonica tra Precision e Recall - Utile quando vuoi bilanciare precisione e copertura
- Confusion Matrix = [[TN, FP], [FN, TP]]
  - TN = True Negative → non spam predetto correttamente
  - FP = False Positive → non spam scambiato per spam
  - FN = False Negative → spam non rilevato
  - TP = True Positive → spam rilevato correttamente

K-Fold Cross-Validation

È un metodo per testare il modello su più suddivisioni del dataset, invece di una sola. L'obiettivo è ottenere una valutazione più affidabile delle prestazioni.

Come funziona

1. Il dataset viene diviso in K parti (fold) uguali.
2. Per ogni fold:
   - Si usa 1 parte come test
   - Le altre K−1 parti come training
3. Si ripete il processo K volte, cambiando ogni volta il fold di test.
4. Si calcolano le metriche (es. accuracy) per ogni fold.
5. Si fa la **media** e si calcola la **deviazione standard**.

K-Fold Cross-Validation

```
from sklearn.model_selection import cross_val_score

scores = cross_val_score(model, X, y, cv=5) # usa 5 fold (5 suddivisioni)
# scores = [0.85, 0.88, 0.86, 0.87, 0.84]   # contiene 5 valori di accuratezza, uno per ogni fold

print(f'Mean: {scores.mean():.3f}')    # accuratezza media del modello
print(f'Std: {scores.std():.3f}')      # contiene 5 valori di accuratezza, uno per ogni fold

→ K-Fold: più robusto di single split
```

Perché è meglio di un singolo train-test split?

Metodo	Vantaggi	Svantaggi
Train-Test Split	Veloce, semplice	Dipende dalla divisione casuale
K-Fold CV	Più robusto, meno variabile	Più lento, ma più affidabile

Hyperparameter Tuning: GridSearchCV

Ogni modello ha dei parametri esterni (detti iperparametri) che influenzano il suo comportamento:

- Non vengono appresi dai dati
- Vanno scelti manualmente o cercati automaticamente

Esempi:

- `n_estimators` → numero di alberi in una Random Forest
- `max_depth` → profondità massima di ogni albero

Perché è utile?

- Evita di scegliere iperparametri “a caso”
- Migliora la generalizzazione del modello
- Usa la validazione incrociata per evitare overfitting

Hyperparameter Tuning:

```
from sklearn.model_selection import GridSearchCV

params = {'n_estimators': [50, 100, 200], 'max_depth': [5, 10, 15]}  # Totale: 9 combinazioni

grid = GridSearchCV(model, params, cv=5)# Crea l'oggetto GridSearchCV
                                        # cv=5 → usa 5-fold cross-validation per ogni combinazione
                                        # Valuta ogni modello su 5 suddivisioni diverse
grid.fit(X_train, y_train)              # Esegue la ricerca su tutti i modelli
                                        # Trova la combinazione con le prestazioni migliori

best_model = grid.best_estimator_       # Restituisce il modello ottimale
                                        # Puoi usarlo per fare predizioni
```

Overfitting vs Underfitting

Cruciale per capire **quanto bene un modello generalizza** sui dati nuovi

I tre scenari di fitting

🔴 Underfitting — Modello troppo semplice

- Non riesce a catturare la struttura dei dati
- Fa errori sia sul training che sul test
- Es: una linea retta per dati curvi

Sintomi:

- Accuracy bassa su training
- Accuracy bassa su test

🟢 Just Right — Modello bilanciato

- Impara abbastanza da generalizzare
- Training ≈ Test → buone prestazioni su entrambi

Sintomi:

- Accuracy alta su training
- Accuracy simile su test

🔵 Overfitting — Modello troppo complesso

- Impara anche il rumore dei dati di training
- Fa predizioni perfette sul training, ma fallisce sul test

Sintomi:

- Accuracy altissima su training
- Accuracy bassa su test

7 best practices fondamentali per costruire modelli di ML

✅ random_state=42 per reproducibilità
- Impostare nei modelli, nei , nei , ecc.
- Garantisce che i risultati siano sempre uguali ogni volta che esegui il codice
- è uno standard ironico, ma puoi usare qualsiasi intero

✅ Sempre dividi in train/test (NO leakage!)
- Serve per valutare il modello su dati mai visti
- Evita il data leakage, cioè che il modello “impari” da dati che dovrebbe ignorare
- Usa

✅ Scala le features (StandardScaler/MinMaxScaler)
- Molti modelli (es. SVM, KNN, PCA) sono sensibili alla scala
- `StandardScaler` → media 0, deviazione standard 1
- `MinMaxScaler` → valori tra 0 e 1

7 best practices fondamentali per costruire modelli di ML

- ✅ Cross-validation per valutazione robusta
  - Valuta il modello su più suddivisioni del dataset
  - Riduce la varianza della stima
  - Usa `cross_val_score(model, X, y, cv=5)`
- ✅ Monitora train vs test accuracy
  - Serve per diagnosticare underfitting o overfitting
  - Se train ≫ test → overfitting
  - Se train ≈ test → modello bilanciato
- ✅ Hyperparameter tuning con GridSearchCV
  - Cerca automaticamente la combinazione migliore di iperparametri
  - Usa
  - Migliora la generalizzazione
- ✅ Feature engineering: crea features significative
  - Le feature contano più del modello!
  - Trasforma, combina, o crea nuove variabili che catturano meglio il fenomeno
  - Es: da “data” puoi estrarre “mese”, “giorno della settimana”, “stagione”

Workflow completo

1. LOAD + EXPLORE - carica i dati (può essere da file, API, o generati) - esplora con X.shape, X.head(), y.value_counts() - verifica distribuzioni, outlier, tipi di variabili

```
X, y = load_data()
```

2. PREPROCESS - standardizza le feature (media 0, std 1) - essenziale per modelli sensibili alla scala: SVM, KNN, PCA - puoi usare anche MinMaxScaler o RobustScaler

```
X_scaled = StandardScaler().fit_transform(X)
```

3. SPLIT - divide i dati in training e test - garantisce riproducibilità - evita data leakage: non usare test per preprocessing!

```
X_tr, X_te, y_tr, y_te = train_test_split()
```

4-8. MODEL, TRAIN, EVALUATE, TUNE, DEPLOY - scegli algoritmo (es. RandomForestClassifier) - allena sul training set - valuta su test set (accuracy, precision, recall…) - usa GridSearchCV per ottimizzare iperparametri - salva il modello (joblib.dump) e documenta il flusso

```
model.fit(X_tr, y_tr)
```

```
score = model.score(X_te, y_te)
```

Strumenti di Sviluppo

📓 Jupyter Notebook
→ Interattivo, perfetto per EDA + Demo

📝 VS Code + Pylance
→ Leggero, debugging, snippet

🎨 PyCharm Professional
→ Full IDE, refactoring, analysis

Librerie Complementari

- XGBoost, LightGBM: Gradient boosting avanzato
- TensorFlow/Keras: Deep Learning e reti neurali
- Statsmodels: Analisi statistiche e serie temporali
- SHAP: Explainability (perché il modello predice)
- MLflow: Experiment tracking e deployment

Quando usare quale algoritmo?

LINEAR REGRESSION → Relazione lineare, continuo

LOGISTIC REGRESSION → Classificazione binaria

DECISION TREE → Interpretabile

RANDOM FOREST → Ensemble, alta accuratezza

SVM → Classificazione non-lineare

K-MEANS → Clustering unsupervised

PCA → Riduzione dimensionalità

NEURAL NETWORKS → Problemi complessi