Relazione Elaborato

Introduzione

L'obiettivo di questo elaborato è sviluppare un esperimento di classificazione in Python a partire dal dataset Wine Quality – White, che contiene misure chimiche di vini bianchi e un punteggio di qualità. Ho scelto questo contenitore per misurarmi su qualcosa di leggermente più complesso di Iris ma anche perchè sono affascinato dal mondo del vino.

Lo scopo è costruire un modello in grado di distinguere tra vini "buoni" e "non buoni" a partire dalle caratteristiche chimiche, applicando le principali tecniche imparate al corso:

1) tecniche di pre-processing (target binario, split train/validation/test, standardizzazione)
2) analisi tramite PCA
3) algoritmi di classificazione (SVM, Logistic Regression, Random Forest, Naive Bayes)
4) valutazione: accuracy, precision, recall, F1-score, ROC-AUC
5) visualizzazione della matrice di confusione e della curva ROC

Descrizione del dataset

Il dataset utilizzato è winequality-white.csv, che contiene vini bianchi con le seguenti feature numeriche:

- fixed acidity
- volatile acidity
- citric acid
- residual sugar
- chlorides
- free sulfur dioxide
- total sulfur dioxide
- density
- pH
- sulphates
- alcohol

e una variabile di output:

- quality: punteggio di qualità (valori interi tipicamente tra 3 e 9).

Tutte le feature sono di tipo numerico continuo e rappresentano caratteristiche chimico-fisiche del vino.

3. Definizione del target e analisi di bilanciamento

Poiché la variabile quality è numerica, è stata trasformata in un target binario per formulare un problema di classificazione:

target = 1 se quality >= 6 → vino buono
target = 0 se quality < 6 → vino non buono

In Python:

data["target"] = (data["quality"] >= 6).astype(int)

Successivamente è stato calcolato il conteggio e la percentuale delle classi:
classe 1 (buono) ≈ 66%
classe 0 (non buono) ≈ 34%

Il dataset risulta quindi sbilanciato, con una prevalenza di vini classificati come buoni.

Suddivisione train / validation / test (70% / 20% / 10%)

Per allenare e valutare correttamente i modelli, ho scelto la suddivisione più completa del dataset (70% training, 20% validation e 10% test), ottenuta con train_test_split di Scikit-Learn.

Per mantenere la proporzione delle classi ed evitare sbilanciamento ho usato la funzione stratify:

```
X = data.drop(columns=["quality", "target"])
y = data["target"]

# 70% train, 30% temporaneo
X_train, X_temp, y_train, y_temp = train_test_split(
    X, y,
    test_size=0.30,
    stratify=y,
    random_state=42
)

# dal 30% rimanente: 20% val, 10% test
X_val, X_test, y_val, y_test = train_test_split(
    X_temp, y_temp,
    test_size=1/3,
    stratify=y_temp,
    random_state=42
)
```

Sono state poi verificate sia le dimensioni dei tre insiemi sia le proporzioni del target, che risultano coerenti con quelle originali.

Standardizzazione delle feature

Poiché molti algoritmi utilizzati (in particolare SVM, Logistic Regression e PCA) sono sensibili alla scala delle feature, è stata applicata la standardizzazione:

media di ogni feature → ~0
deviazione standard → ~1

La funzione utilizzata è StandardScaler di Scikit-Learn:

```
scaler = StandardScaler()

X_train_s = scaler.fit_transform(X_train)
X_val_s   = scaler.transform(X_val)
X_test_s  = scaler.transform(X_test)
```

Fit_transform calcola medie e deviazioni standard ed applica la standardizzazione. È usata solo sull'insieme Training per evitare che il modello apprenda informazioni da insiemi di dati che dovranno essere utilizzati per verificare la bontà del modello stesso (Val e Test). Lo Scaler quindi

memorizza medie e deviazioni standard del primo insieme e le riutilizza per standardizzare anche gli altri due.
Dalle feature standardizzate è stato poi ricostruito un DataFrame, per controllare media e deviazione standard sull'insieme Train:

```
scaled_df = pd.DataFrame(X_train_s, columns=X.columns)
print(scaled_df.mean())
print(scaled_df.std())
```

I risultati mostrano medie tendenti a 0 (ordini di grandezza molto piccoli) e deviazioni standard tendenti a 1, confermando che la standardizzazione è stata applicata correttamente.

PCA (Principal Component Analysis)

Adesso applichiamo la PCA alle feature standardizzate con l'obiettivo di ridurre la dimensionalità. Nello specifico si trovano due componenti principali PC1 e PC2, comprimendo i dati delle variabili originali.

```
pca = PCA(n_components=2)
X_train_pca = pca.fit_transform(X_train_s)
X_val_pca   = pca.transform(X_val_s)
X_test_pca  = pca.transform(X_test_s)

print("Varianza spiegata da PC1 e PC2:")
print(pca.explained_variance_ratio_.round(4))
```

È stato poi creato un grafico scatter in 2D.

```
pca_df = pd.DataFrame({
    "PC1": X_train_pca[:, 0],
    "PC2": X_train_pca[:, 1],
    "target": y_train.values
})
sns.scatterplot(data=pca_df, x="PC1", y="PC2", hue="target", palette="viridis", alpha=0.7)
```

Interpretazione della PCA

Le prime due componenti principali spiegano una quota limitata della varianza totale (circa il 43% complessivo), quindi il dataset è complesso.
Nel grafico PCA le due classi (target = 0 e target = 1) risultano ampiamente sovrapposte.
Questo suggerisce che la qualità del vino non è facilmente separabile in uno spazio bidimensionale a partire dalle sole combinazioni lineari delle feature.

La PCA quindi è utile per esplorare la struttura dei dati e la varianza, ma non è sufficiente per distinguere in modo netto vini buoni e non buoni. Da qui la necessità di ricorrere a modelli di classificazione più complessi.

Modelli di classificazione

Sono stati addestrati e valutati tre modelli di classificazione:

1. SVM (Support Vector Machine) con kernel RBF
3. Random Forest
4. Naive Bayes (GaussianNB)

Tutti i modelli sono stati addestrati sui dati standardizzati del training set (X_train_s, y_train) e valutati sul validation set (X_val_s, y_val).

Per ciascun modello sono state calcolate le metriche:

- Accuracy
- Precision
- Recall
- F1-score
- ROC-AUC

e sono state rappresentate:

- la matrice di confusione
- la curva ROC

SVM

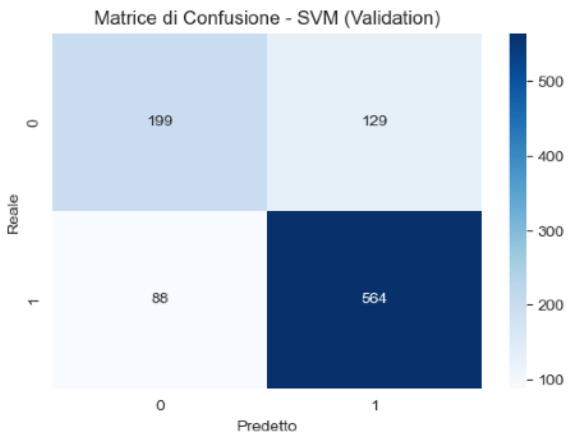
```
svm = SVC(
    C=1,
    kernel='rbf',
    gamma='scale',
    probability=True
)

svm.fit(X_train_s, y_train)

y_val_pred  = svm.predict(X_val_s)
y_val_proba = svm.predict_proba(X_val_s)[:,1]
```

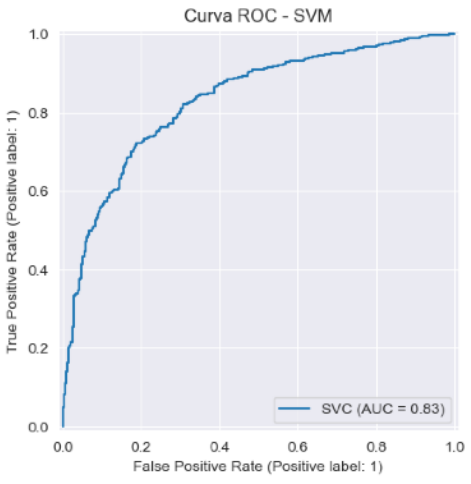
Le metriche ottenute sul Validation set sono:

Accuracy ≈ 0.78
Precision ≈ 0.81
Recall ≈ 0.87
F1-score ≈ 0.84
ROC-AUC ≈ 0.83

L'SVM mostra un buon equilibrio tra precision e recall, ed un'ottima capacità discriminativa (AUC > 0.8).

Successivamente mostriamo la matrice di confusione e la Curva ROC attraverso matplotlib e seaborn, importate precedentemente.





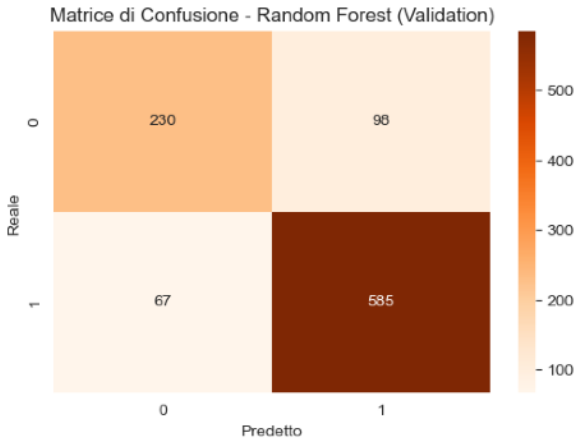
Random Forest

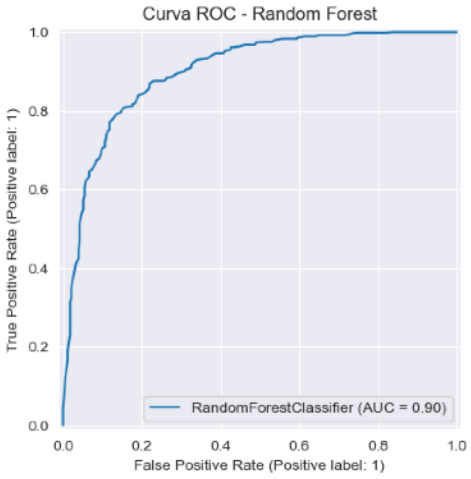
```
rf = RandomForestClassifier(
    n_estimators=200,
    max_depth=None,
    random_state=42
)

rf.fit(X_train_s, y_train)

y_val_pred_rf  = rf.predict(X_val_s)
y_val_proba_rf = rf.predict_proba(X_val_s)[:, 1]
```

Metriche sul validation set:

Accuracy ≈ 0.83
Precision ≈ 0.86
Recall ≈ 0.90
F1-score ≈ 0.88
ROC-AUC ≈ 0.90

La Random Forest tende a ottenere metriche molto competitive e si dimostra un modello robusto, adatto alla complessità delle variabili chimiche del vino.
Inoltre stampiamo anche la scala di importanza delle feature determinata dall'algoritmo e scopriamo che "alcohol" e "volatile acidity" sono state reputate le due feature con maggiore peso nel processo selettivo.
Il vantaggio di questo algoritmo ed essere altamente spiegabile.





Naive Bayes (GaussianNB)

Utilizziamo la variante Gaussian perchè abbiamo feature continue:

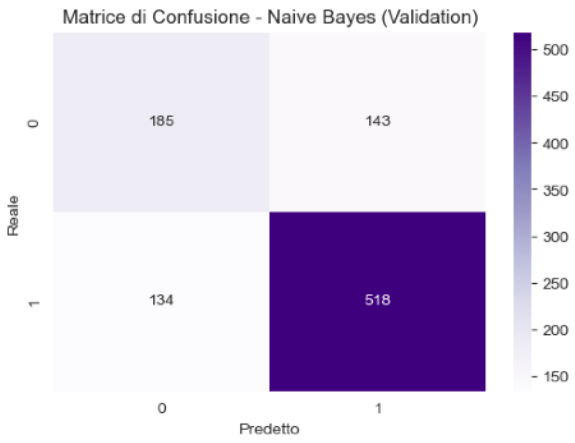
```
nb = GaussianNB()
nb.fit(X_train_s, y_train)

y_val_pred_nb  = nb.predict(X_val_s)
y_val_proba_nb = nb.predict_proba(X_val_s)[:,1]
```

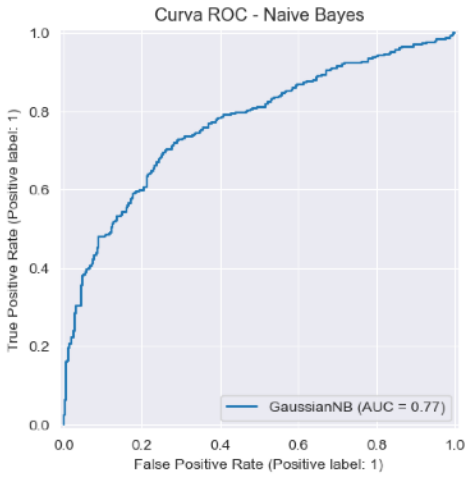
Metriche sul validation set:

Accuracy ≈ 0.72
Precision ≈ 0.78
Recall ≈ 0.79
F1-score ≈ 0.79
ROC-AUC ≈ 0.77

Come atteso, le prestazioni di Naive Bayes risultano inferiori rispetto agli altri modelli.

Questo è coerente con la teoria: Naive Bayes assume l'indipendenza tra le feature, ipotesi non realistica nel dataset Wine Quality, dove le variabili chimiche sono spesso correlate (es. zucchero residuo, densità, alcol).





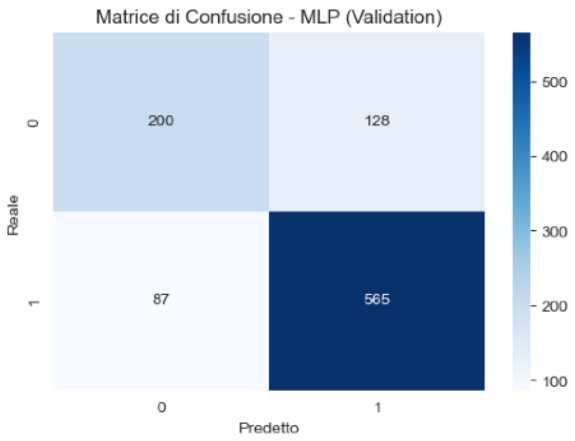
Tensorflow (Deep Learning)

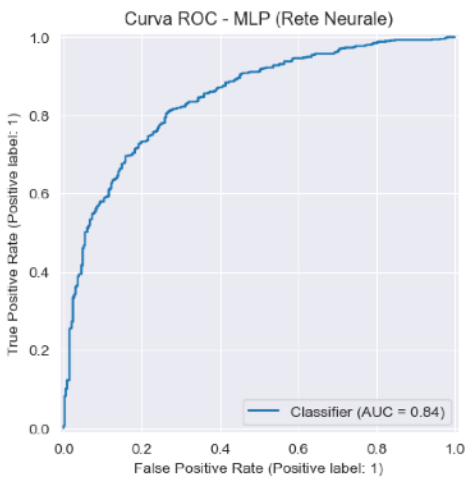
Lo scopo di questo ultimo esperimento è creare, addestrare e valutare una rete neurale feedforward che predice se un vino è buono (1) o non buono (0).

Si crea il modello con input e struttura della rete neurale: 32 neuroni nel primo layer nascosto (Dense) e 16 nel secondo. Con Dropout si vanno a spegnere alcuni neuroni a caso per limitare l'overfitting.

Si addestra il modello facendo 100 ripetizioni così da generare dei pesi affidabili e si passa poi alle predizioni.
La learning curve dimostra che il modello si comporta positivamente diminuendo via via sia il training loss che il validation loss.

Concludo poi stampando matrice di confusione e Curva Roc.

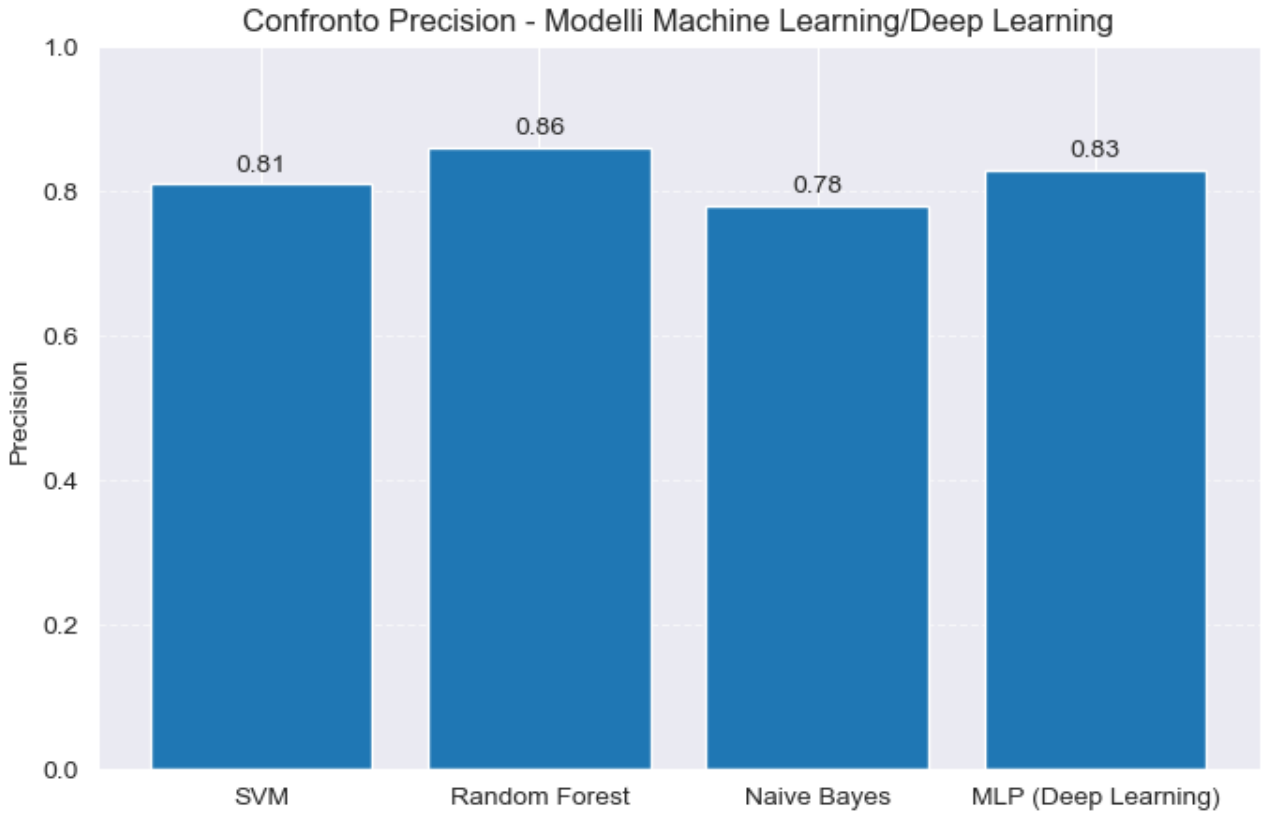




Queste le metriche sul validation set:

Accuracy ≈ 0.78
Precision ≈ 0.83
Recall ≈ 0.85
F1-score ≈ 0.84
ROC-AUC ≈ 0.83

Confronto tra i modelli e commento finale



L'analisi comparativa dei modelli di Machine Learning e Deep Learning ha evidenziato prestazioni complessivamente buone per tutti gli algoritmi considerati. In particolare, osservando la Precision si nota come il modello Random Forest ottenga il valore più elevato (0,86), risultando quello più affidabile nel riconoscere correttamente i vini buoni tra quelli classificati come tali.

La rete neurale MLP raggiunge una Precision pari a 0,83, molto vicina a quella della SVM (0,81) e superiore al Naive Bayes (0,78). Questo risultato conferma che il Deep Learning, pur con una rete relativamente semplice e un numero di dati non particolarmente elevato, è in grado di apprendere efficacemente le relazioni non lineari tra le variabili chimiche e la qualità del vino.

Da un punto di vista applicativo, una Precision elevata risulta importante per ridurre il numero di falsi positivi, ossia quei casi in cui un vino viene erroneamente considerato buono. Tuttavia, nella selezione vinicola può essere altrettanto rilevante massimizzare il Recall, in modo da evitare lo scarto di vini potenzialmente validi. Per questo, nella valutazione complessiva del modello occorre

considerare anche altre metriche, come F1-score e ROC-AUC, che confermano ulteriormente la buona capacità predittiva dei modelli proposti.

In conclusione, il modello Random Forest risulta essere il più efficace nel contesto analizzato, mentre il modello MLP rappresenta un'interessante alternativa moderna con prestazioni competitive, mostrando come il Deep Learning possa essere applicato con successo anche a dataset tabellari di dimensioni moderate.

Andrea Carli
Matricola 0322500035

Codice Python

```python
# importazione delle librerie principali

import numpy as np #gestione di array e matrici

import pandas as pd #manipolazione dei dati

import matplotlib.pyplot as plt #creazioni di grafici

import seaborn as sns #creazioni di grafici più sofisticati


# lettura del database su vini bianchi

data=pd.read_csv("data/winequality-white.csv", sep=';')

#print(data.columns)

#data.head()


# definizione del target per classificazione (binaria)


data["target"] = (data["quality"] >= 6).astype(int)

print("Conteggio target (totale osservazioni):")

print(data [['target']].value_counts())


print("\nPercentuali:")
```

```
print((data["target"].value_counts(normalize=True) *
100).round(2))



#%%

#il dataset è sbilanciato, utilizzo stratify per dividere i
dati mantenendo le stesse percentuali delle classi per
testarlo in modo equo


from sklearn.model_selection import train_test_split


# Variabili

X = data.drop(columns=["quality", "target"])

y = data["target"]


# Split stratificato (prima 70/30 per isolare il training,
poi 20/10 per validation e test)


X_train, X_temp, y_train, y_temp = train_test_split(

    X, y,

    test_size=0.30,    # 30% va in temp

    stratify=y,

    random_state=42

)


X_val, X_test, y_val, y_test = train_test_split(

    X_temp, y_temp,
```

```
    test_size=1/3,   # cioè 10% del totale
    stratify=y_temp,
    random_state=42
)

# Verifica proporzioni
print("Proporzioni originali:")
print(data["target"].value_counts(normalize=True).round(3))

print("\nProporzioni training set:")
print(y_train.value_counts(normalize=True).round(3))

print("\nProporzioni validation set:")
print(y_val.value_counts(normalize=True).round(3))

print("\nProporzioni test set:")
print(y_test.value_counts(normalize=True).round(3))



# Controllo delle dimensioni
print("Training set:", X_train.shape)
print("Validation set:", X_val.shape)
print("Test set:", X_test.shape)
#%%
# adesso standardizziamo le feature con la funzione
standarscaler
```

```
from sklearn.preprocessing import StandardScaler

scaler = StandardScaler()


X_train_s = scaler.fit_transform(X_train)   # fit + transform SOLO sul training per evitare che il modello apprenda informazioni future. Fit salva dentro scaler le medie e deviazioni. Transform le riutilizza per standardizzare val e test

X_val_s = scaler.transform(X_val)

X_test_s = scaler.transform(X_test)

# controllo

print("Train scalato:", X_train_s.shape)

print("Val scalato:", X_val_s.shape)

print("Test scalato:", X_test_s.shape)


#adesso abbiamo due tre nuove matrici standardizzate che lasciano invariato il database di partenza. le medie  sono tendenti a 0 mentre la deviazione standard a 1


scaled_df = pd.DataFrame(X_train_s, columns=X.columns)

print(scaled_df.mean())

print(scaled_df.std())


#%%

#adesso applichiamo la PCA: prendiamo le 11 feature del vino e le combiniamo in due nuove variabili pc1 e pc2 che catturano la massima varianza possibile e sono tra loro indipendenti.
```

```
from sklearn.decomposition import PCA

pca = PCA(n_components=2)

X_train_pca = pca.fit_transform(X_train_s)   # fit +
transform SOLO su train come sopra

X_val_pca = pca.transform(X_val_s)

X_test_pca = pca.transform(X_test_s)


print("Varianza spiegata da PC1 e PC2:")

print(pca.explained_variance_ratio_.round(4))


# DataFrame PCA per grafico

pca_df = pd.DataFrame({

    "PC1": X_train_pca[:, 0],

    "PC2": X_train_pca[:, 1],

    "target": y_train.values

})


# Grafico PCA 2D

plt.figure(figsize=(9,6))

sns.scatterplot(

    data=pca_df,

    x="PC1", y="PC2",

    hue="target",

    palette="viridis",

    alpha=0.7

)

plt.title("PCA - Prime 2 Componenti (Training Set)")
```

```
plt.xlabel(f"PC1 ({pca.explained_variance_ratio_[0]*100:.2f}% varianza)")
plt.ylabel(f"PC2 ({pca.explained_variance_ratio_[1]*100:.2f}% varianza)")
plt.legend(title="Target (0 = non buono, 1 = buono)")
plt.show()


#%%
# partiamo con algoritmo SVC
from sklearn.svm import SVC
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score, roc_auc_score, confusion_matrix, RocCurveDisplay


svm = SVC(
    C=1,                # penalità standard
    kernel='rbf',       # kernel gaussiano
    gamma='scale',      # impostazione moderna e stabile
    probability=True    # necessario per ROC curve
)

# Addestramento sul training set
svm.fit(X_train_s, y_train)

# Predizione sul validation set
y_val_pred = svm.predict(X_val_s)
y_val_proba = svm.predict_proba(X_val_s)[:,1]
```

```
# Metriche di valutazione
print("Accuracy:",  accuracy_score(y_val, y_val_pred))
print("Precision:", precision_score(y_val, y_val_pred))
print("Recall:",    recall_score(y_val, y_val_pred))
print("F1-score:",  f1_score(y_val, y_val_pred))
print("ROC-AUC:",   roc_auc_score(y_val, y_val_proba))

# Matrice di confusione
cm = confusion_matrix(y_val, y_val_pred)
plt.figure(figsize=(6,4))
sns.heatmap(cm, annot=True, fmt="d", cmap="Blues")
plt.title("Matrice di Confusione - SVM (Validation)")
plt.xlabel("Predetto")
plt.ylabel("Reale")
plt.show()

# ROC Curve
RocCurveDisplay.from_estimator(svm, X_val_s, y_val)
plt.title("Curva ROC - SVM")
plt.show()



#%%
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score, precision_score,
recall_score, f1_score, roc_auc_score, confusion_matrix,
RocCurveDisplay
```

```python
# Modello Random Forest base
rf = RandomForestClassifier(
    n_estimators=200,      # numero alberi
    max_depth=None,        # profondità libera
    random_state=42
)

# Addestramento sul training
rf.fit(X_train_s, y_train)

# Predizione sulla validation
y_val_pred_rf = rf.predict(X_val_s)
y_val_proba_rf = rf.predict_proba(X_val_s)[:, 1]

# Metriche di valutazione
print("Accuracy:",  accuracy_score(y_val, y_val_pred_rf))
print("Precision:", precision_score(y_val, y_val_pred_rf))
print("Recall:",    recall_score(y_val, y_val_pred_rf))
print("F1-score:",  f1_score(y_val, y_val_pred_rf))
print("ROC-AUC:",   roc_auc_score(y_val, y_val_proba_rf))

# Matrice di confusione
cm_rf = confusion_matrix(y_val, y_val_pred_rf)

plt.figure(figsize=(6,4))
sns.heatmap(cm_rf, annot=True, fmt="d", cmap="Oranges")
```

```
plt.title("Matrice di Confusione - Random Forest (Validation)")
plt.xlabel("Predetto")
plt.ylabel("Reale")
plt.show()

# ROC Curve
RocCurveDisplay.from_estimator(rf, X_val_s, y_val)
plt.title("Curva ROC - Random Forest")
plt.show()

# Importanza delle feature Random Forest
feature_imp = pd.Series(rf.feature_importances_, index=X_train.columns).sort_values(ascending=False)

print("\nImportanza delle feature:")
print(feature_imp)

#%%
from sklearn.naive_bayes import GaussianNB
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score, roc_auc_score, confusion_matrix, RocCurveDisplay

# Modello Naive Bayes utilizzando Gaussian perchè abbiamo feature continue
nb = GaussianNB()
```

```python
# Addestramento sul training set
nb.fit(X_train_s, y_train)

# Predizione sulla validation
y_val_pred_nb = nb.predict(X_val_s)
y_val_proba_nb = nb.predict_proba(X_val_s)[:,1]

# Metriche di valutazione
print("Accuracy:",  accuracy_score(y_val, y_val_pred_nb))
print("Precision:", precision_score(y_val, y_val_pred_nb))
print("Recall:",    recall_score(y_val, y_val_pred_nb))
print("F1-score:",  f1_score(y_val, y_val_pred_nb))
print("ROC-AUC:",   roc_auc_score(y_val, y_val_proba_nb))

# Matrice di confusione
cm_nb = confusion_matrix(y_val, y_val_pred_nb)

plt.figure(figsize=(6,4))
sns.heatmap(cm_nb, annot=True, fmt="d", cmap="Purples")
plt.title("Matrice di Confusione - Naive Bayes (Validation)")
plt.xlabel("Predetto")
plt.ylabel("Reale")
plt.show()

# ROC Curve
RocCurveDisplay.from_estimator(nb, X_val_s, y_val)
plt.title("Curva ROC - Naive Bayes")
```

```
plt.show()

#%%


#%%

from tensorflow.keras import Input

from tensorflow.keras.models import Sequential #costrusce la rete strato dopo strato

from tensorflow.keras.layers import Dense, Dropout #definisce i neuroni e una tecnica che li spegne a caso per ridurre overfitting

from tensorflow.keras.optimizers import Adam #ottimizzatore dei pesi durante l'addestramento


model = Sequential([

    Input(shape=(X_train_s.shape[1],)),  #strato che riceve i dati chimici del vino

    Dense(32, activation='relu'), #primo strato nascosto con 32 neuroni che ignora i valori negativi

    Dropout(0.2), #spegne il 20% dei neuroni attivi per evitare overfitting

    Dense(16, activation='relu'), #secondo strato nascosto con 16 neuroni che migliorano il ragionamento

    Dropout(0.2),

    Dense(1, activation='sigmoid') #strato di uscita con 1 neurone che indica 0 o 1 in caso di vino cattivo o buono

])


model.compile(

    optimizer=Adam(learning_rate=0.001), # definizione di quanto velocemente impara la rete
```

```
    loss='binary_crossentropy', #funzione che indica quanto la rete sbaglia ad ogni previsione
    metrics=['accuracy'] #misura quante volte indovina sulle classi finali
)

# Addestramento
history = model.fit(
    X_train_s, y_train,
    validation_data=(X_val_s, y_val),
    epochs=100, #ripetizione dell'addestramento
    batch_size=16, #ogni quanti esempi aggiorna i pesi
    verbose=0
)

# Predizioni
y_val_proba = model.predict(X_val_s).ravel()  # probabilità classe 1 = vino buono
y_val_pred = (y_val_proba >= 0.5).astype(int) #applichiamo una soglia >=0,5 - vino buono

# Metriche
print("Accuracy:",  accuracy_score(y_val, y_val_pred))
print("Precision:", precision_score(y_val, y_val_pred))
print("Recall:",    recall_score(y_val, y_val_pred))
print("F1-score:",  f1_score(y_val, y_val_pred))
print("ROC-AUC:",   roc_auc_score(y_val, y_val_proba))
```

```
# Learning curves
plt.figure(figsize=(7,5))
plt.plot(history.history['loss'], label='Training Loss')
plt.plot(history.history['val_loss'], label='Validation
Loss')
plt.title("Learning Curve - MLP")
plt.xlabel("Epoch")
plt.ylabel("Loss")
plt.legend()
plt.grid(True)
plt.show()

cm = confusion_matrix(y_val, y_val_pred)

plt.figure(figsize=(6,4))
sns.heatmap(cm, annot=True, fmt="d", cmap="Blues")
plt.title("Matrice di Confusione - MLP (Validation)")
plt.xlabel("Predetto")
plt.ylabel("Reale")
plt.show()

plt.figure(figsize=(7,5))
RocCurveDisplay.from_predictions(y_val, y_val_proba)
plt.title("Curva ROC - MLP (Rete Neurale)")
plt.grid(True)
plt.show()
#%%
```

```
# Precision ottenute dai tuoi modelli
precision_svm = 0.81
precision_rf  = 0.86
precision_dt  = 0.78
precision_mlp = 0.83

model_names = ["SVM", "Random Forest", "Naive Bayes", "MLP (Deep Learning)"]
precision_scores = [precision_svm, precision_rf, precision_dt, precision_mlp]

# Grafico a colonne
plt.figure(figsize=(8,5))
bars = plt.bar(model_names, precision_scores)

# Etichette sopra le barre
for bar in bars:
    height = bar.get_height()
    plt.text(bar.get_x() + bar.get_width()/2, height + 0.01, f"{height:.2f}",
             ha='center', va='bottom')

plt.title("Confronto Precision - Modelli Machine Learning/ Deep Learning")
plt.ylabel("Precision")
plt.ylim(0,1) # scala da 0 a 1
plt.grid(axis='y', linestyle='--', alpha=0.6)
plt.show()
```