Relazione Finale — Progetto di Apprendimento Automatico

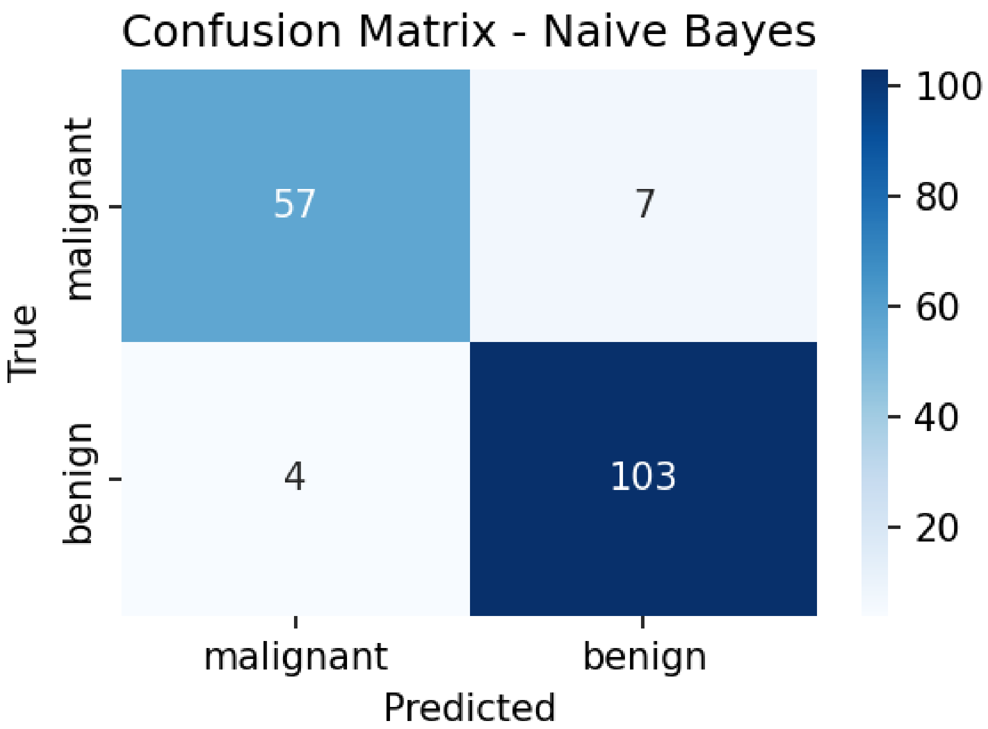
Dataset: **Breast Cancer Wisconsin (Diagnostic)**. Obiettivo: classificare tumori in benigni o maligni.

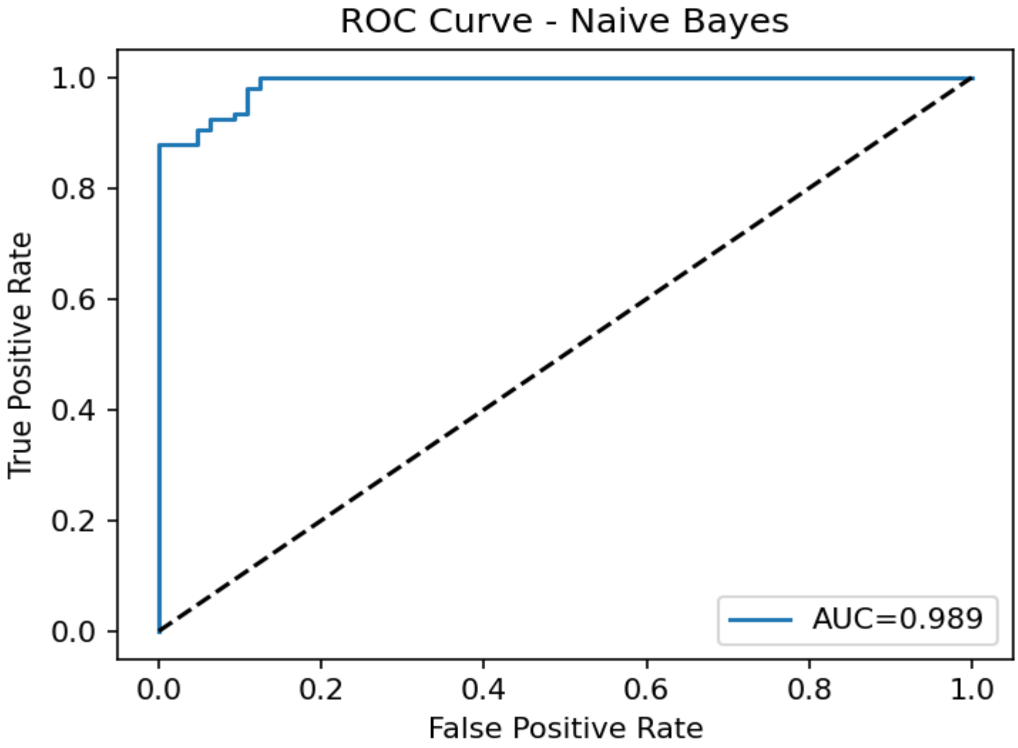
Scelte progettuali (estesa)
Motivazioni: dataset noto e adatto a task di classificazione binaria; 30 feature numeriche; dimensione gestibile. Preprocessing: standardizzazione (StandardScaler). Split stratificato 70/30. Modelli scelti: Naive Bayes (baseline probabilistica), Decision Tree (interpretabilità), Random Forest (robustezza/ensemble).

Procedura: standardizzazione -> split stratificato -> training -> test -> metriche e visualizzazioni (confusion matrix, ROC).

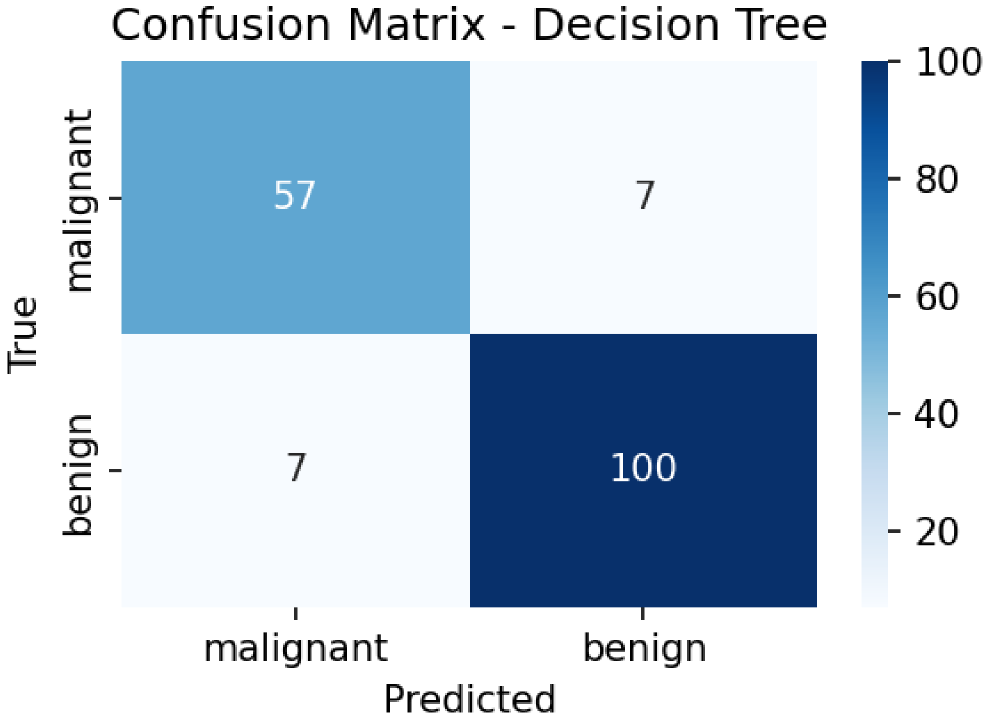
Model	Accuracy	Precision	Recall	F1	ROC AUC
Naive Bayes	0.9357	0.9364	0.9626	0.9493	0.9892
Decision Tree	0.9181	0.9346	0.9346	0.9346	0.9126
Random Forest	0.9357	0.9444	0.9533	0.9488	0.9913

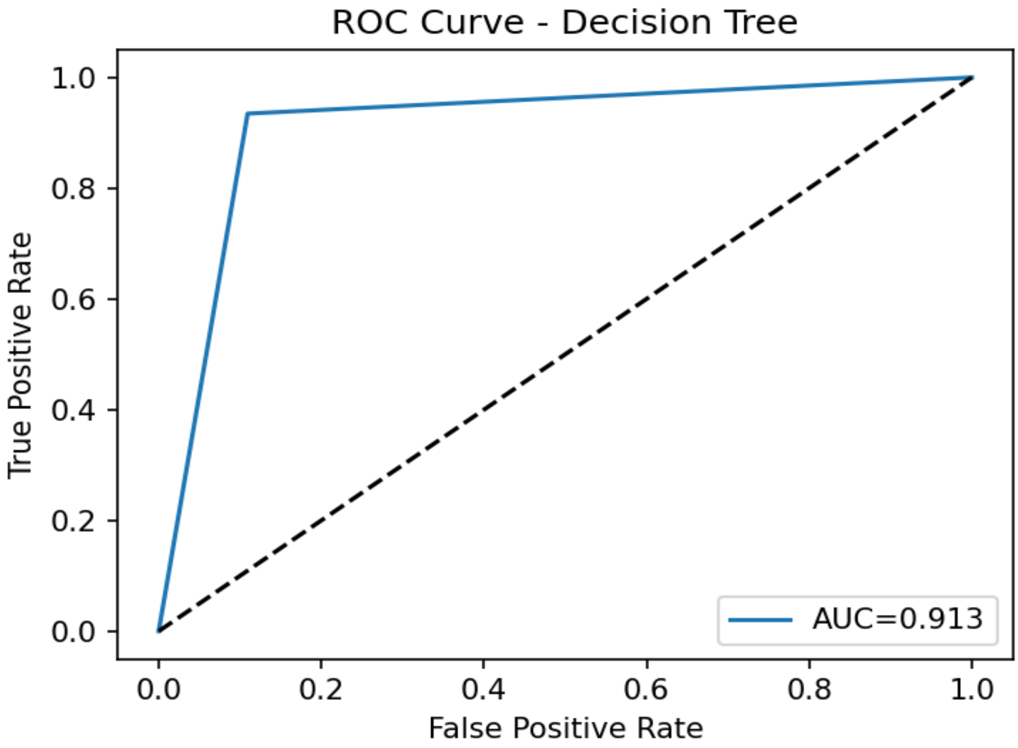
Naive Bayes



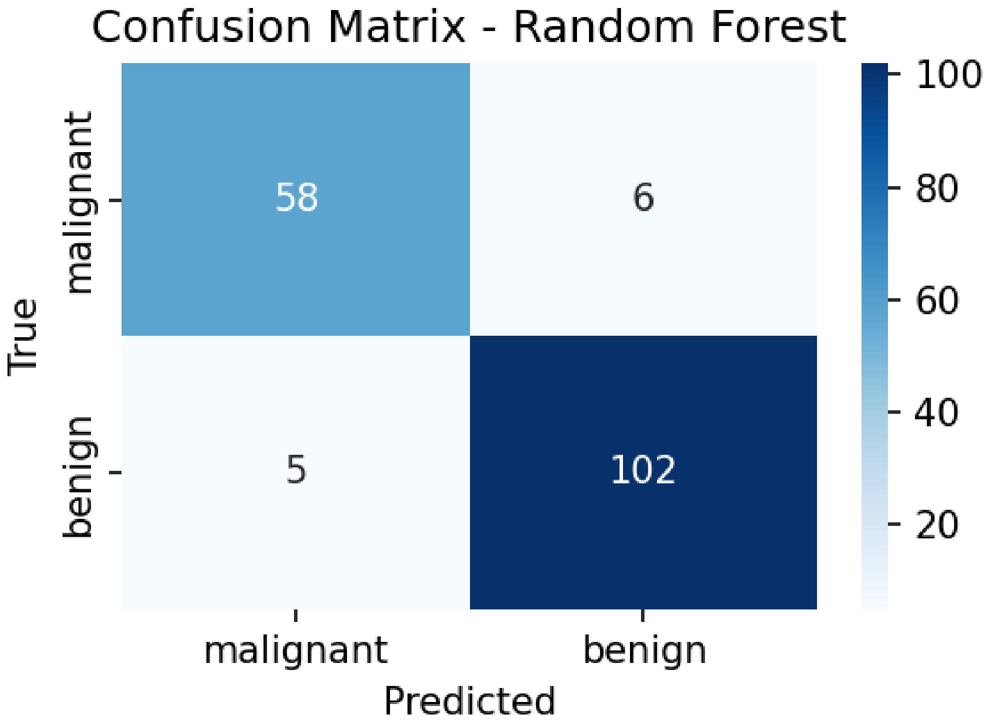


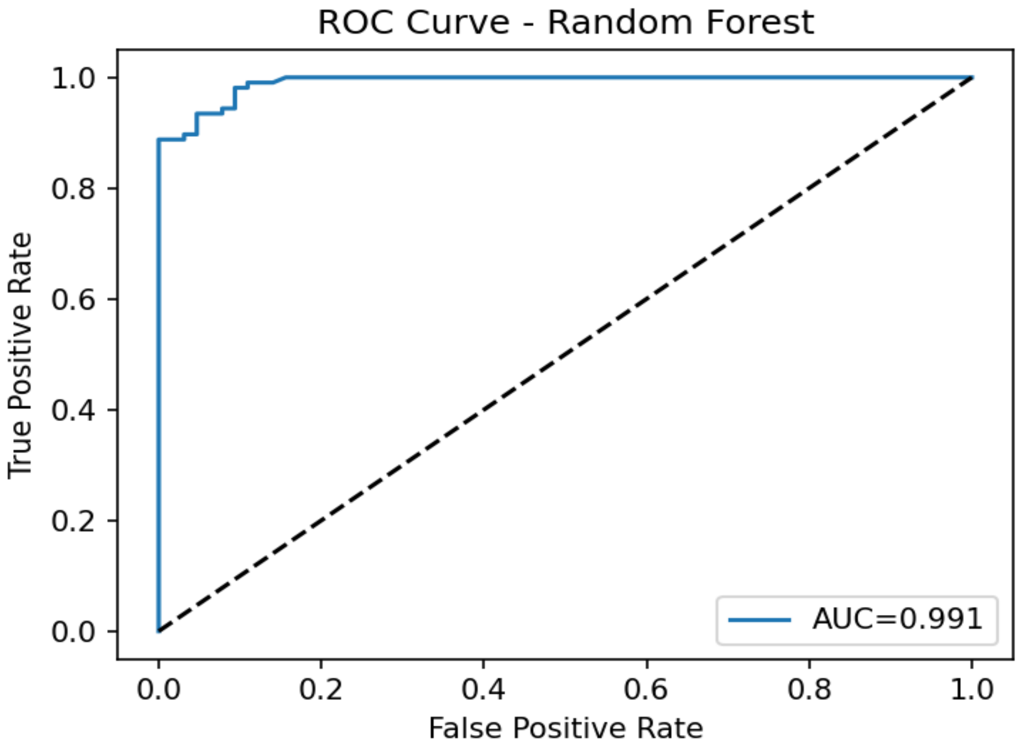
Decision Tree





Random Forest





Appendice: codice principale

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.datasets import load_breast_cancer
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.naive_bayes import GaussianNB
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score, confusion_matrix, roc_curve, au


data = load_breast_cancer()
X = data.data
y = data.target
feature_names = data.feature_names
class_names = data.target_names
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)
X_train, X_test, y_train, y_test = train_test_split(X_scaled, y, test_size=0.3, random_state=42, stratify=y)
print('Train size:', X_train.shape[0], 'Test size:', X_test.shape[0])

classifiers = {
    'Naive Bayes': GaussianNB(),
    'Decision Tree': DecisionTreeClassifier(random_state=42, ccp_alpha=0.0),
    'Random Forest': RandomForestClassifier(n_estimators=100, random_state=42, min_samples_leaf=1, max_features='sq
}
results = {}
for name, clf in classifiers.items():
    clf.fit(X_train, y_train)
    y_pred = clf.predict(X_test)
    if hasattr(clf, 'predict_proba'):
        y_score = clf.predict_proba(X_test)[:,1]
    else:
        y_score = clf.decision_function(X_test)
    acc = accuracy_score(y_test, y_pred)
    prec = precision_score(y_test, y_pred)
    rec = recall_score(y_test, y_pred)
    f1 = f1_score(y_test, y_pred)
    fpr, tpr, _ = roc_curve(y_test, y_score)
    roc_auc = auc(fpr, tpr)
    results[name] = {'accuracy': acc, 'precision': prec, 'recall': rec, 'f1': f1, 'roc_auc': roc_auc}
    print(name, results[name])

for name, clf in classifiers.items():
    y_pred = clf.predict(X_test)
    if hasattr(clf, 'predict_proba'):
        y_score = clf.predict_proba(X_test)[:,1]
    else:
        y_score = clf.decision_function(X_test)
    cm = confusion_matrix(y_test, y_pred)
    plt.figure(figsize=(5,4))
    sns.heatmap(cm, annot=True, fmt='d', cmap='Blues', xticklabels=class_names, yticklabels=class_names)
    plt.title(f'Confusion Matrix - {name}')
    plt.show()
    fpr, tpr, _ = roc_curve(y_test, y_score)
    plt.figure(figsize=(5,4))
    plt.plot(fpr, tpr, label=f'AUC={auc(fpr,tpr):.3f}')
    plt.plot([0,1],[0,1],'k--')
    plt.title(f'ROC Curve - {name}')
    plt.xlabel('FPR')
    plt.ylabel('TPR')
    plt.legend()
    plt.show()
```