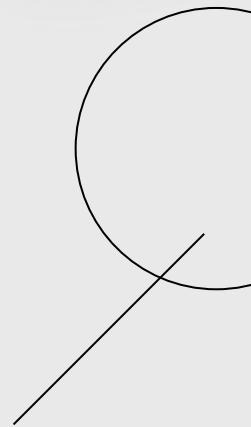


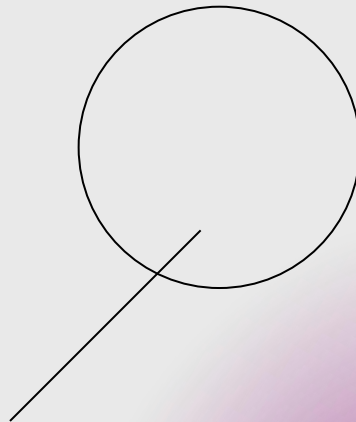
Algoritmi di classificazione

Implementazione e Confronto su dataset
Mushroom e Rice



Introduzione

In questo progetto andremo ad analizzare i dataset Mushroom e Rice. Addestreremo dei modelli di classificazione in grado di categorizzare i diversi dati in base alle loro caratteristiche. Mostrando le relative metriche di ogni algoritmo andremo a valutare l'efficacia dei vari modelli implementati.



Basi Teoriche

Classificazione

è una tipologia di apprendimento supervisionato in cui il target (variabile dipendente y) è una variabile **discreta** o **categorica**.

Obiettivo: Costruire un modello capace di assegnare una determinata istanza (fungo o chicco di riso) a una delle classi predefinite (es. Edibile/Poisoned) basandosi sulle caratteristiche (feature) fornite.

Classificazione Binaria: Nel nostro progetto affrontiamo casi binari, dove l'output appartiene a un insieme di due sole etichette $\{0,1\}$.

Metriche e metodi di valutazione

Accuracy : misura il numero dei casi classificati correttamente diviso il numero dei casi totali.

Precision : è il numero di esempi correttamente classificati come positivi diviso il numero di esempi che sono stati classificati come positivi.

Recall : è il numero di esempi correttamente classificati come positivi diviso il numero di esempi che sono effettivamente positivi.

Matrice di confusione = matrice binaria dove vengono mostrati il numero di classificazioni corrette/incorrette positive/negative.

n-fold-cross-validation: il dataset viene partizionato in n sottoinsiemi disgiunti con la stessa cardinalità, viene usato ogni sottoinsieme come test e i restanti $n-1$ usati come addestramento.

Impurità di Gini: misura statistica che indica quanto sono "misti" i dati in un nodo.

Preprocessing dei dati

Label Encoding: Trasforma le classi (es. "edible", "poisonous") in numeri (0, 1).

One-Hot Encoding: Trasforma categorie in colonne binarie separate.

Standard Scaling: Normalizza i dati affinché abbiano media 0 e varianza 1. Questo è cruciale per algoritmi come il k-NN che calcolano "distanze" fisiche.

Basi Teoriche

Algoritmi di classificazione

Alberi di decisione

Nodo Radice: È l'attributo che meglio divide i dati iniziali.

Decision Nodes: Ogni nodo pone una domanda su una caratteristica.

Split (Divisione): I dati vengono divisi in rami (Sì/No o valori numerici) in base alla risposta.

Foglie: Sono i nodi finali che contengono la classificazione.

K-Nearest Neighbors (k-NN)

È un algoritmo "geometrico". Non costruisce un modello complesso, ma memorizza i dati.

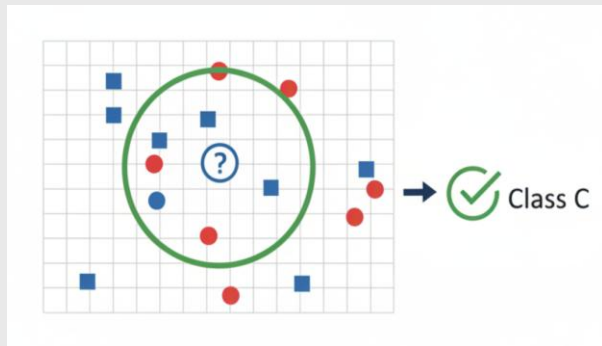
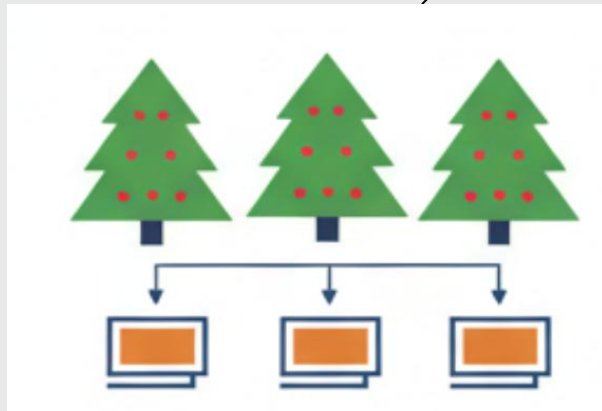
Per classificare un nuovo dato, guarda i **K vicini** più prossimi nello spazio cartesiano.

Random Forest

È un metodo di Ensemble, invece di creare un solo albero, per un maggiore confronto crea tanti alberi decisionali.

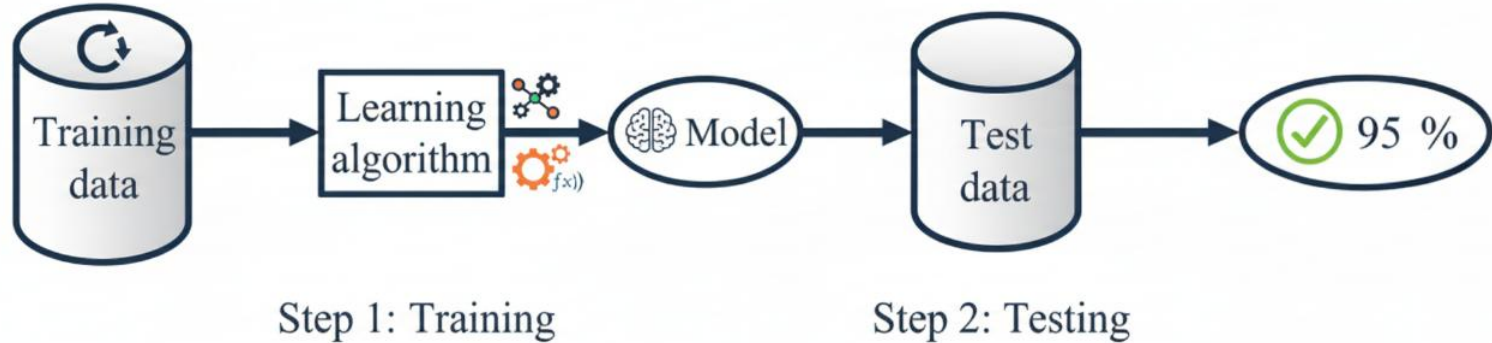
Ogni albero viene addestrato su un sottoinsieme diverso dei dati.

Alla fine, la foresta prende una decisione a maggioranza. È solitamente più preciso e robusto di un albero singolo.



Basi Teoriche

FLUSSO DI LAVORO



Linguaggi e Strumenti Utilizzati

- **Python**

- **Librerie:**

scikit-learn: Per la creazione, addestramento e valutazione dei modelli di classificazione (Decision Tree, k-NN, Random Forest).

matplotlib e *seaborn*: Per la visualizzazione grafica dei risultati, come alberi decisionali, matrici di confusione e grafici di importanza delle feature.

pandas: Per la gestione e manipolazione dei dataset.

numpy: Per operazioni numeriche e array.

scipy: Per il caricamento di dataset in formato ARFF e altre funzioni scientifiche.

- **Visual Studio Code**: Ambiente di sviluppo utilizzato per scrivere, modificare e testare il codice.

- **AI (Gemini)**: utilizzato per la creazione dei grafici su python.

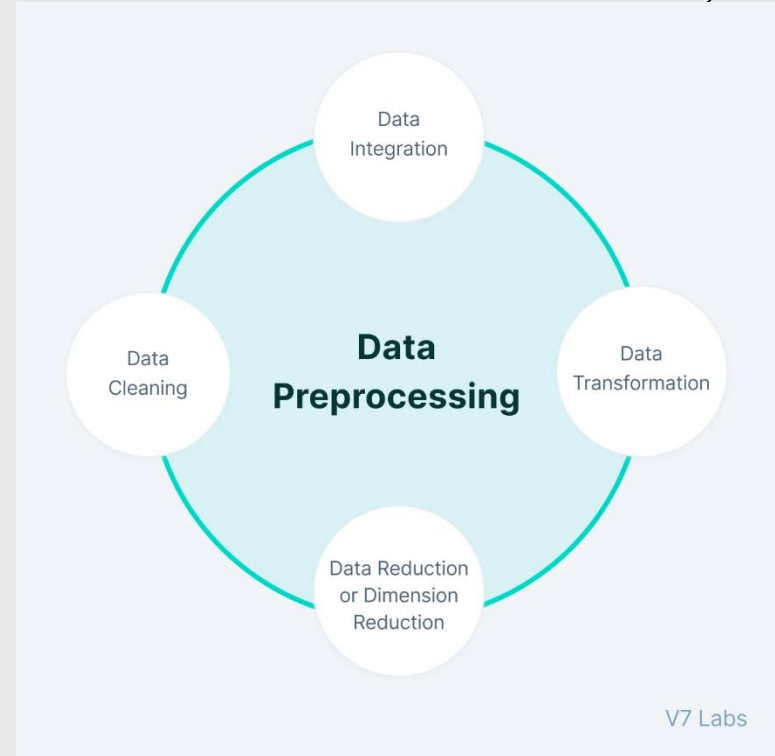


AI & Machine Learning

Powered by Scikit-learn

01

Dataset e Preprocessing



Descrizione Dataset Mushroom

Numero di Record:

8.124 istanze totali (4.208 commestibili, 3.916 velenosi).

Colonne Rimosse:

veil-type: Rimossa perché ha un unico valore costante per tutti i record (non porta informazione).

Nota: In una versione alternativa ([decision_tree.py](#)) è stata rimossa anche stalk-root per i troppi dati mancanti, ma nel modello finale ([models.py](#)) i mancanti sono gestiti.

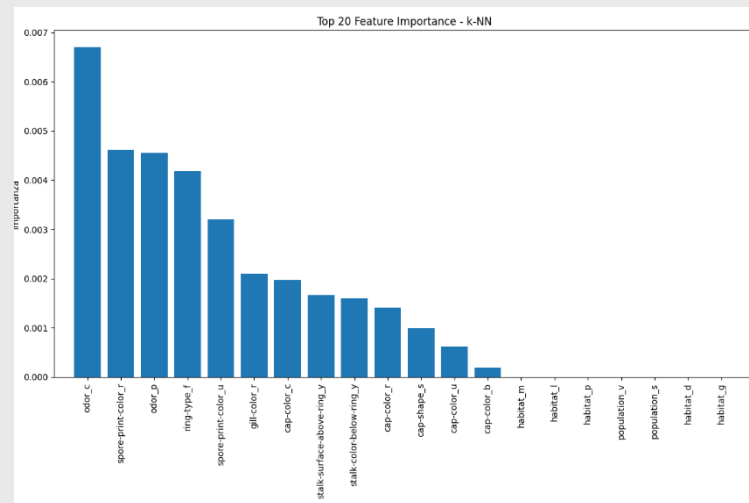
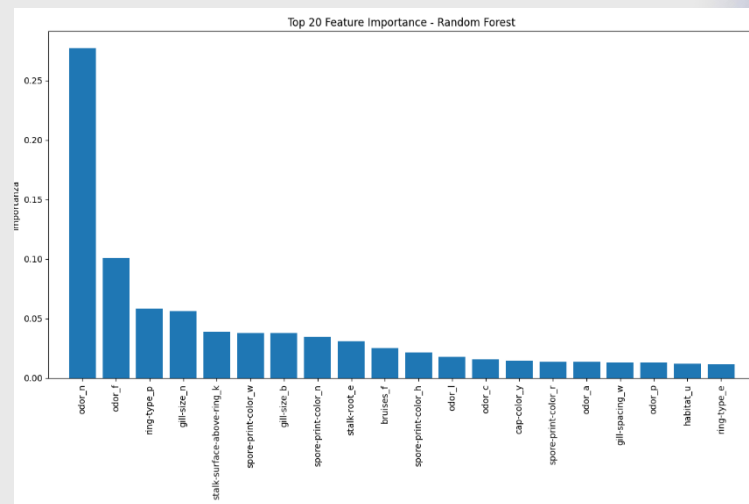
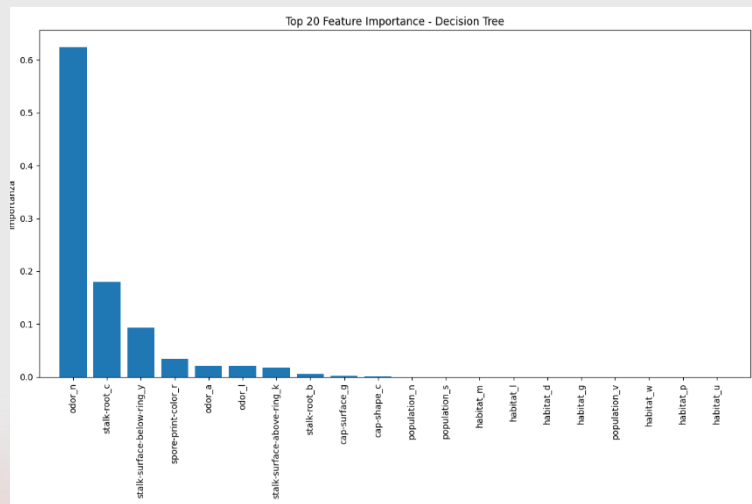
Grafici di Importanza:

Analisi effettuata tramite *Feature Importance* (per alberi) e *Permutation Importance* (per k-NN) per identificare le caratteristiche più discriminanti (es. odor, spore-print-color).



Importanza delle caratteristiche del dataset Mushroom per ogni modello

Eliminazione della colonna veil-type



Descrizione Dataset Rice

Numero di Record:

3.810 istanze totali (1.630 Cammeo, 2.180 Osmancik).

Preprocessing e Pulizia:

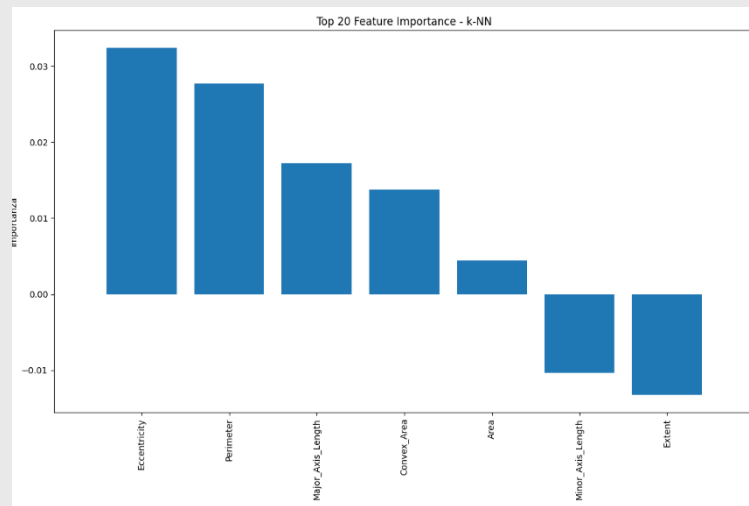
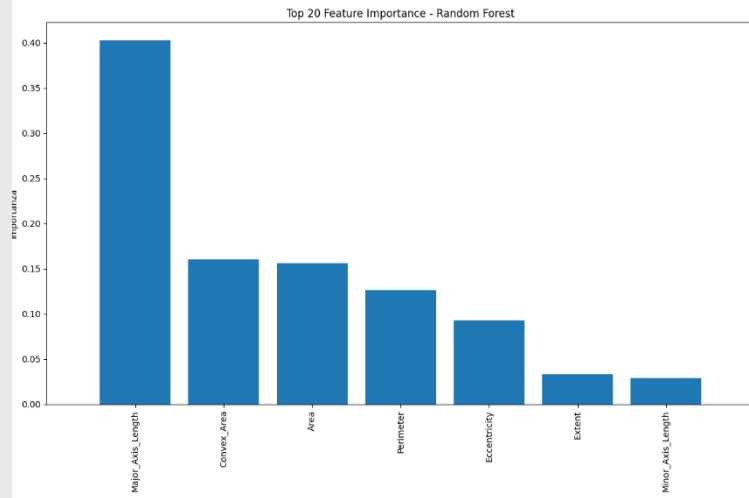
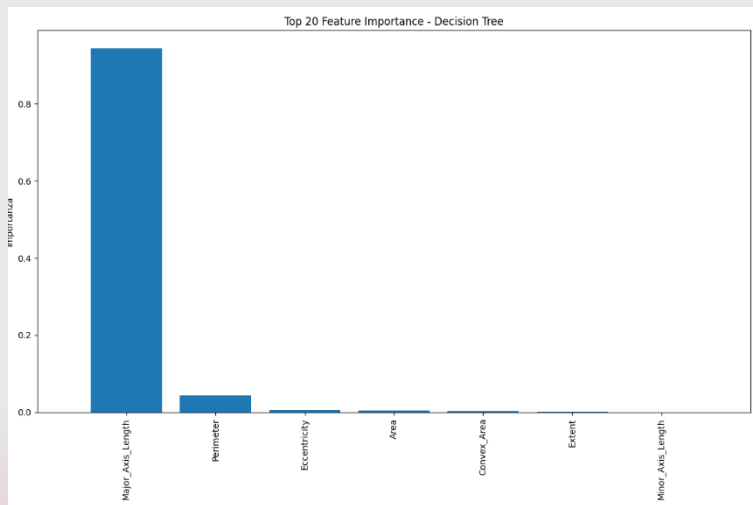
Decodifica delle stringhe UTF-8 per corretta lettura dal formato .arff.
Nessuna colonna rimossa: tutte le 7 feature geometriche sono state ritenute valide.

Standardizzazione:

Fondamentale l'uso di StandardScaler per normalizzare le misurazioni geometriche (es. Area vs Perimetro) e permettere il corretto funzionamento di algoritmi basati sulla distanza come k-NN.



Importanza delle caratteristiche del dataset Rice per ogni modello



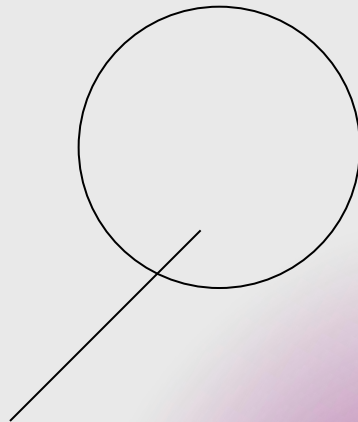
Preprocessing dei Dati

Queste funzioni si occupano di revisionare i dati del dataset Mushroom e del dataset Rice per prepararli all'addestramento e al testing dei modelli.

```
FUNCTION preprocess_data_mushrooms()  
    transform the class column with LabelEncoder //valori 0 e 1 per commestibile e velenoso  
    remove veil-type and stalk-root columns due to poor impact  
    convert the attributes with one-hot-encoding and scale the information  
  
    RETURN attributes and classes  
  
FUNCTION preprocess_data_rice()  
    convert the input data into utf-8 strings  
    transform the class column with LabelEncoder  
    scale the features  
  
    RETURN attributes and classes
```

Divisione Training/Test e Scaling

Dopo il pre-processing, abbiamo diviso i dati in *training* (80%) e *test* (20%) per valutare i modelli in modo equo.



02

Algoritmi e Implementazioni

Modelli Scikit-Learn: Decision Tree, k-NN, Random Forest

```
FUNCTION train_tree_model(x_train, y_train):  
    trains the Decision Tree on the training data  
    with sklearn library with max_depth = 5 to avoid overfitting  
  
    RETURN the Decision Tree  
  
FUNCTION train_knn_model(x_train, y_train):  
    trains the K-NN on the training data with sklearn library  
  
    RETURN the K-NN classifier  
  
FUNCTION train_rf_model(x_train, y_train)  
    trains the Random Forest with 10 trees on the training data  
    with sklearn library  
  
    RETURN the Random Forest
```

Abbiamo addestrato 3 modelli della libreria Scikit-Learn: *Decision Tree* con profondità limitata a 5 per evitare l'overfitting, *k-NN* con 5 vicini per sfruttare la distanza tra punti, e *Random Forest*, un ensemble di 10 alberi che migliora la stabilità e l'accuratezza grazie al voto collettivo.

Custom Decision Tree: struttura e logica

```
FUNCTION BuildTree(dataset, labels)
  IF all labels are equal
    RETURN leaf with that label

  FOR each feature
    FOR each possible split value
      compute Gini impurity

  SELECT split with minimum impurity

  CREATE left and right subsets

  left_subtree ← BuildTree(left subset)
  right_subtree ← BuildTree(right subset)

  RETURN decision node

FUNCTION gini_impurity(y)
  FOR each class
    calculates impurity

  RETURN impurity //grado di disordine dei dati

FUNCTION decision_tree_predict(tree, sample)
  IF node is a leaf node
    return node value

  IF sample's feature value is <= than the node's threshold
    return decision_tree_predict(left tree, sample)
  ELSE
    return decision_tree_predict(right tree, sample)
```

Abbiamo inoltre provato a sviluppare un Decision Tree custom.

L'algoritmo costruisce un decision tree partendo dai dati di training. Ad ogni passo sceglie la feature e la soglia che separano meglio i dati, riducendo al minimo la Gini Impurity. La costruzione dell'albero avviene in modo ricorsivo fino ad ottenere nodi foglia che rappresentano una classe. In fase di predizione, il campione viene fatto scorrere nell'albero seguendo le condizioni fino alla foglia finale

```
CLASS DecisionNode:
  feature      //indice feature di split
  threshold    //valore della feature su cui effettuare split
  left         //figlio sinistro
  right        //figlio destro
  value        //etichetta di classe per i nodi foglia, altrimenti null
```




03

Valutazione e analisi dei Risultati

Metriche di performance e cross-validation

Abbiamo valutato i modelli usando metriche come **accuracy**, **precision** e **recall**, fondamentali per capire la qualità delle classificazioni.

La validazione fissata a 10 fold ha garantito che i risultati fossero affidabili e non dovuti al caso, mostrando performance realistiche su dati mai visti prima.



Mushroom vs Rice

Valutazione dei modelli sul Test set...

Decision Tree Accuracy: 100.00%

k-NN Accuracy: 100.00%

Random Forest Accuracy: 100.00%

Calcolo di Precision e Recall per ogni modello...

--- Metriche dettagliate per Decision Tree ---

Precision: 100.00%

Recall: 100.00%

--- Metriche dettagliate per k-NN ---

Precision: 100.00%

Recall: 100.00%

--- Metriche dettagliate per Random Forest ---

Precision: 100.00%

Recall: 100.00%

Cross-Validation dei modelli...

Decision Tree CV Accuracy (Media): 96.73%

Decision Tree CV Accuracy (Deviazione Standard): 9.41%

k-NN CV Accuracy (Media): 96.56%

k-NN CV Accuracy (Deviazione Standard): 8.25%

Random Forest CV Accuracy (Media): 96.85%

Random Forest CV Accuracy (Deviazione Standard): 9.45%

Valutazione dei modelli sul Test set...

Decision Tree Accuracy: 91.73%

k-NN Accuracy: 90.55%

Random Forest Accuracy: 91.73%

Calcolo di Precision e Recall per ogni modello...

--- Metriche dettagliate per Decision Tree ---

Precision: 91.65%

Recall: 93.20%

--- Metriche dettagliate per k-NN ---

Precision: 90.87%

Recall: 91.75%

--- Metriche dettagliate per Random Forest ---

Precision: 92.87%

Recall: 91.75%

Cross-Validation dei modelli...

Decision Tree CV Accuracy (Media): 91.89%

Decision Tree CV Accuracy (Deviazione Standard): 1.97%

k-NN CV Accuracy (Media): 91.76%

k-NN CV Accuracy (Deviazione Standard): 1.91%

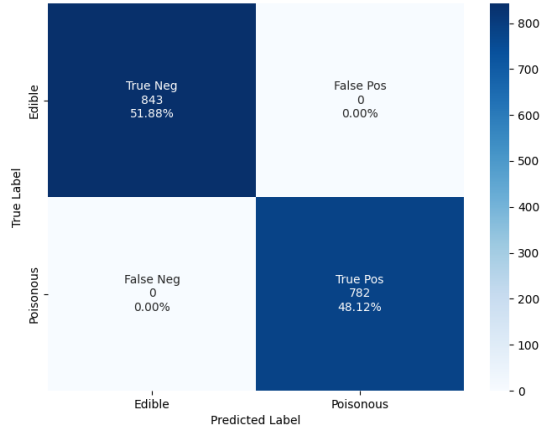
Random Forest CV Accuracy (Media): 91.76%

Random Forest CV Accuracy (Deviazione Standard): 2.20%

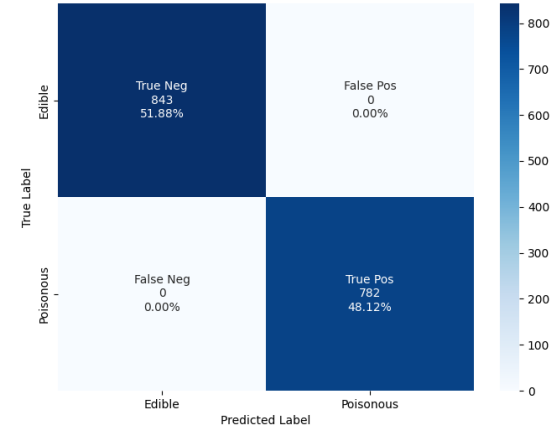
Visualizzazioni delle matrici di confusione, confini decisionali

Matrice di confusione Dataset Mushroom

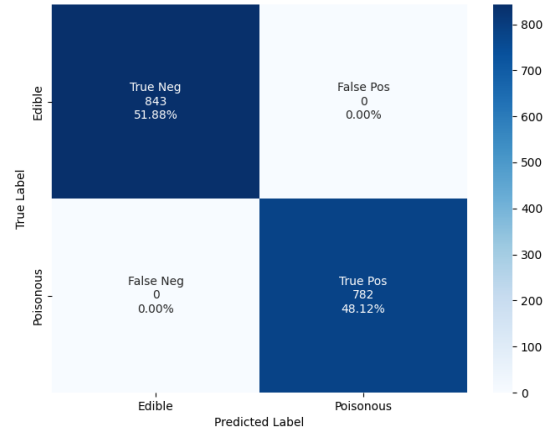
Matrice di Confusione - Decision Tree



Matrice di Confusione - Random Forest

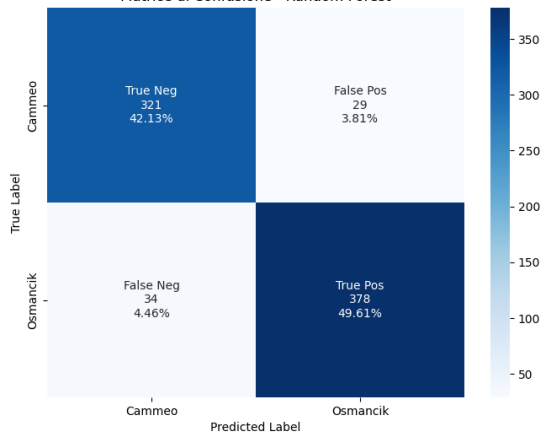


Matrice di Confusione - k-NN

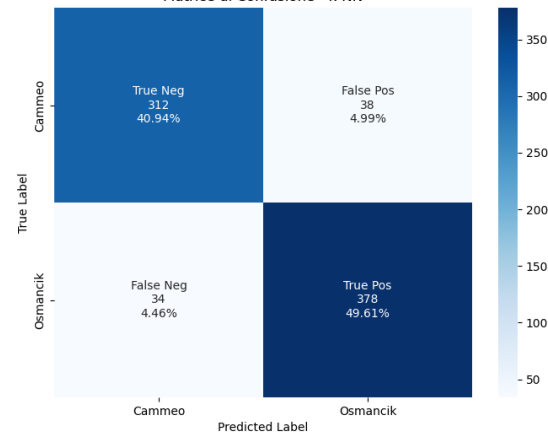


Matrice di confusione Dataset Rice

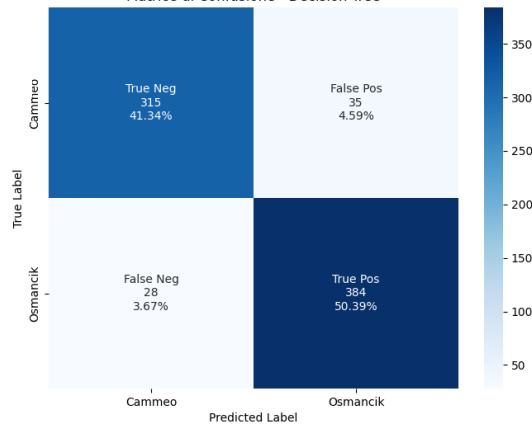
Matrice di Confusione - Random Forest



Matrice di Confusione - k-NN

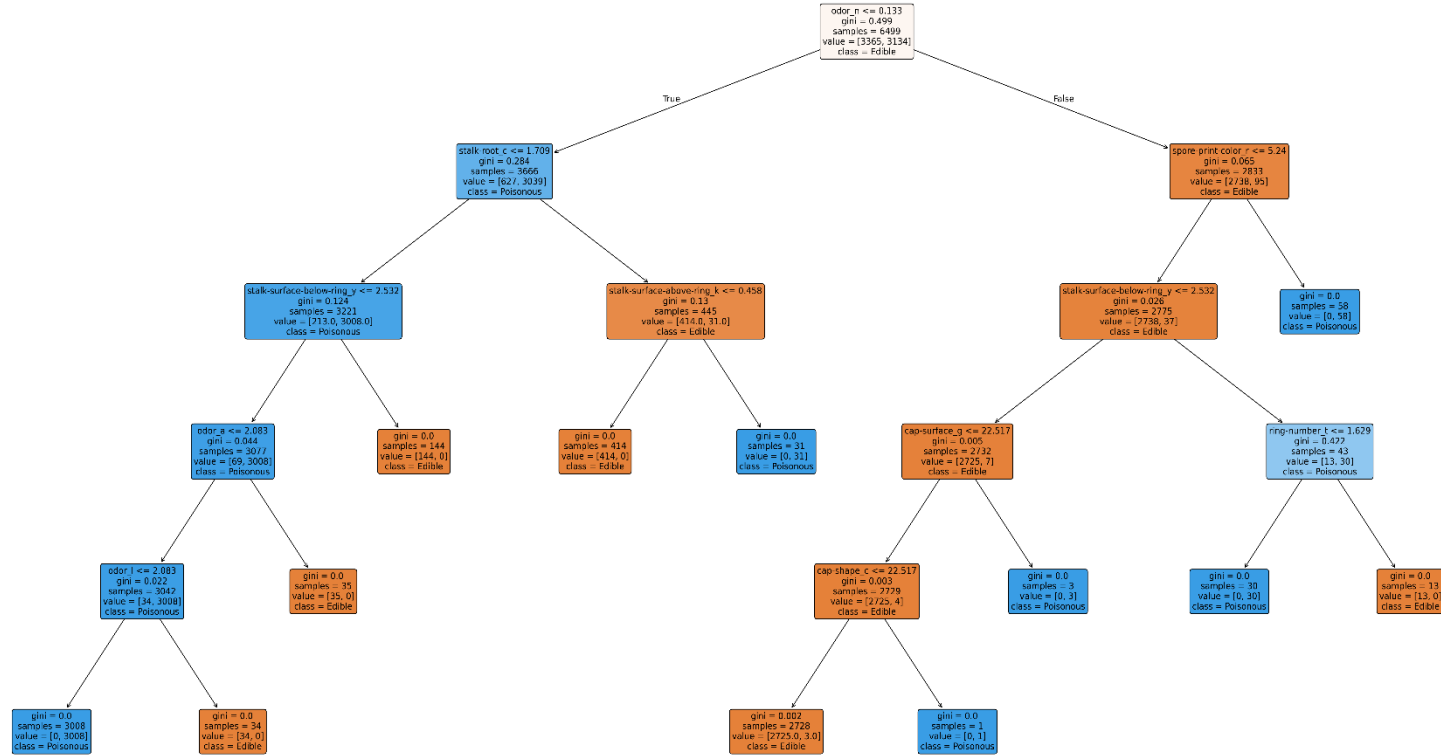


Matrice di Confusione - Decision Tree



Decision-tree Dataset Mushroom

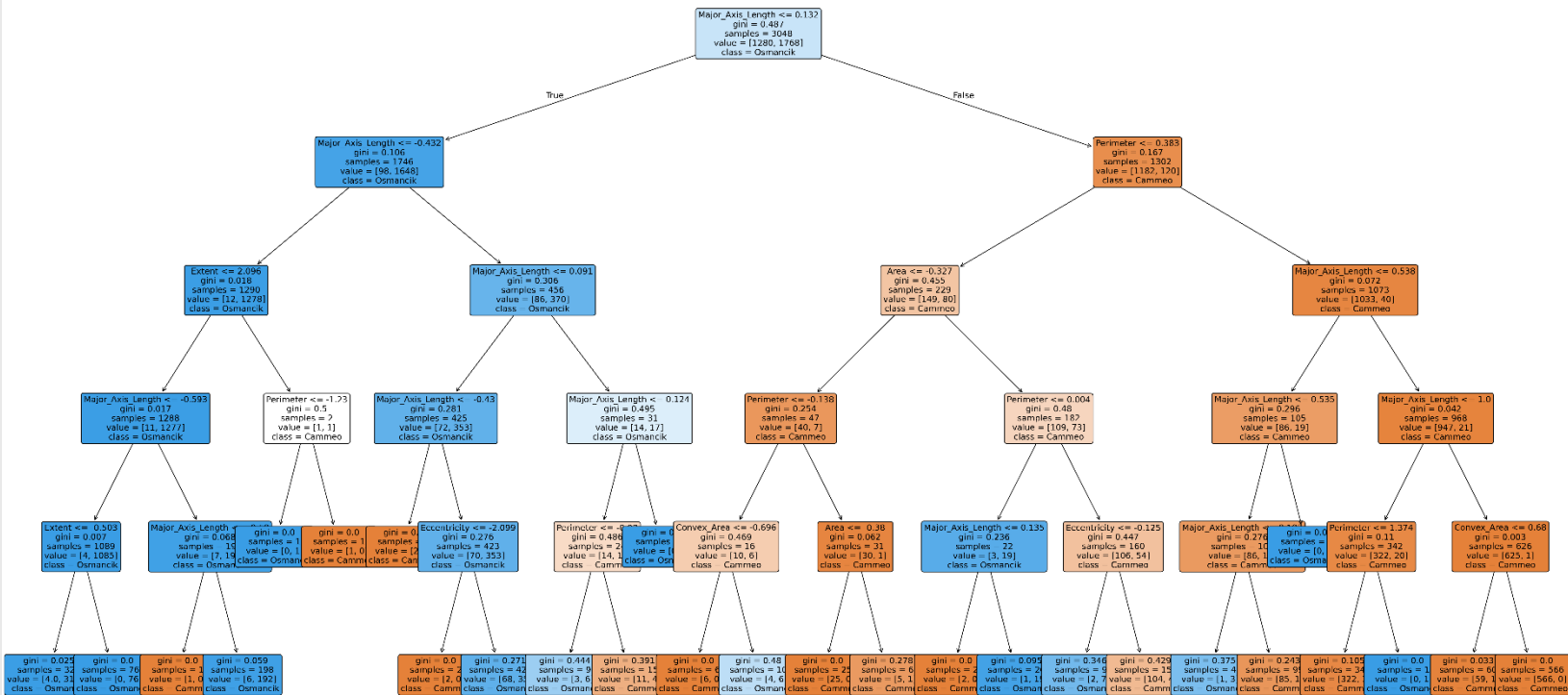
Visualizzazione Albero di Decisione



Decision tree visualization for the 'Major Axis Length' feature. The tree structure is as follows:

- Root Node: Major Axis Length <= 0.132 (gini = 0.487, samples = 3048, value = [1280, 1768], class = Osmanick)
 - True Branch: Major Axis Length <= -0.432 (gini = 0.106, samples = 1746, value = [98, 1648], class = Osmanick)
 - Extent <= 2.096 (gini = 0.018, samples = 1290, value = [12, 1279], class = Osmanick)
 - Major Axis Length <= -0.593 (gini = 0.017, samples = 1288, value = [11, 1277], class = Osmanick)
 - Perimeter <= -1.23 (gini = 0.5, samples = 2, value = [1, 1], class = Cammeo)
 - Major Axis Length <= 0.091 (gini = 0.308, samples = 456, value = [86, 370], class = Osmanick)
 - Major Axis Length <= -0.43 (gini = 0.281, samples = 425, value = [72, 353], class = Osmanick)
 - Major Axis Length <= 0.124 (gini = 0.495, samples = 31, value = [14, 17], class = Osmanick)
 - False Branch: Perimeter <= 0.383 (gini = 0.167, samples = 1302, value = [1182, 120], class = Cammeo)
 - Area <= -0.327 (gini = 0.453, samples = 229, value = [149, 80], class = Cammeo)
 - Perimeter <= -0.138 (gini = 0.254, samples = 47, value = [40, 7], class = Cammeo)
 - Perimeter <= 0.004 (gini = 0.48, samples = 187, value = [109, 73], class = Cammeo)
 - Major Axis Length <= 0.558 (gini = 0.072, samples = 1073, value = [1033, 40], class = Cammeo)
 - Major Axis Length <= 0.535 (gini = 0.296, samples = 105, value = [86, 19], class = Cammeo)
 - Major Axis Length <= 1.0 (gini = 0.042, samples = 968, value = [947, 21], class = Cammeo)

The tree continues to split based on various features like 'Perimeter', 'Convex Area', 'Eccentricity', and 'Major Axis Length' until it reaches leaf nodes with final gini index, sample counts, and class distributions.

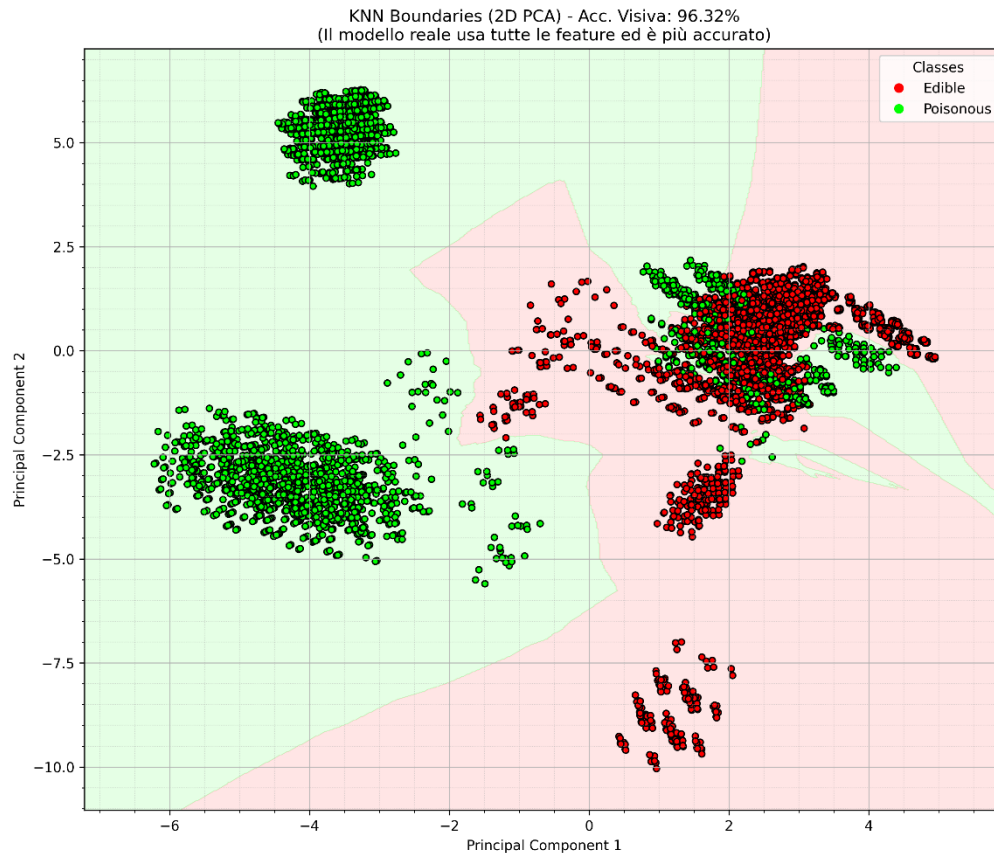


KNN Dataset Mushroom

Rappresentazione grafica di come l' algoritmo taglia lo spazio delle feature per separare le classi.

Riduzione della Dimensionalità (PCA):

Poiché il dataset ha oltre 20 feature, abbiamo utilizzato la **Principal Component Analysis** per proiettare i dati in un piano 2D, permettendoci di vedere le aree di competenza di ogni classe.

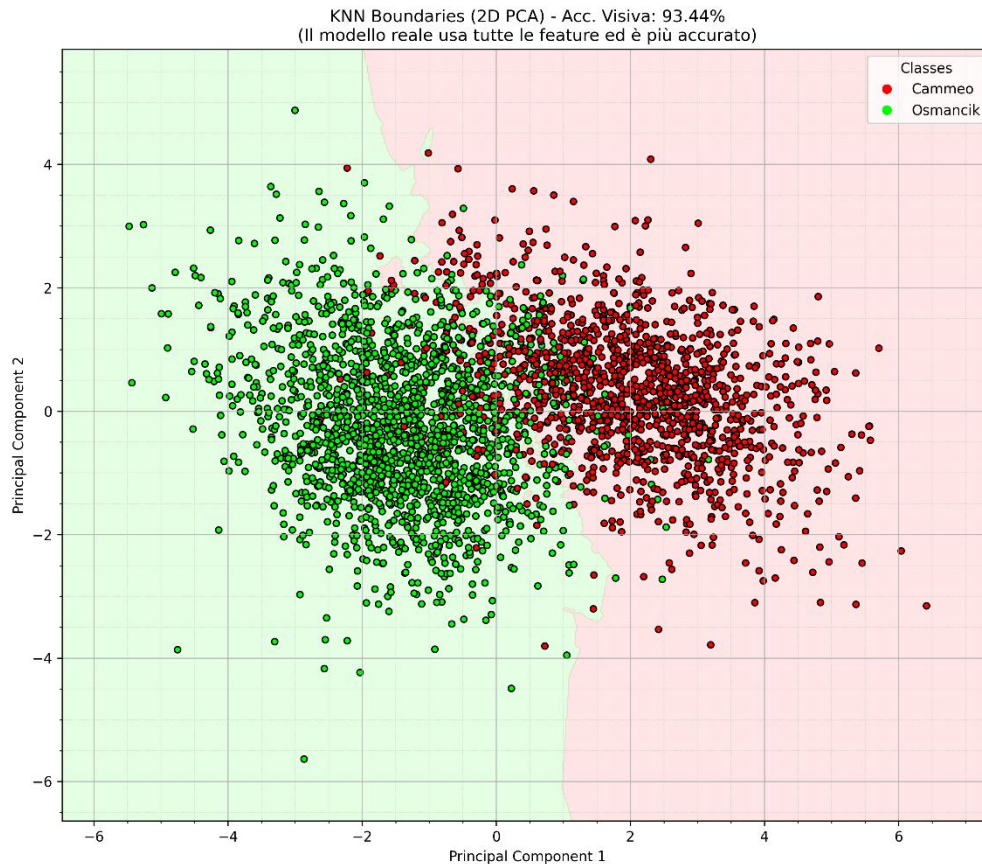


KNN Dataset Rice

Rappresentazione grafica di come l' algoritmo taglia lo spazio delle feature per separare le classi.

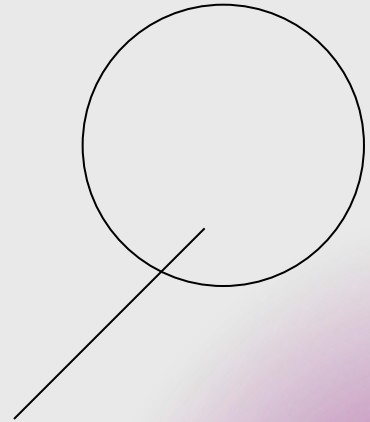
Riduzione della Dimensionalità (PCA):

Poiché il dataset ha più di 6 feature, abbiamo utilizzato la **Principal Component Analysis** per proiettare i dati in un piano 2D, permettendoci di vedere le aree di competenza di ogni classe.



Analisi comparativa e conclusioni

Confrontare i tempi di esecuzione degli algoritmi, confronto tra costo e accuratezza, differenza di accuratezza tra i due dataset, spiegazione di quale dei due dataset sia il migliore.



Tempo di esecuzione degli algoritmi su dataset Mushroom vs Rice

```
Addestramento del Decision Tree...  
Decision Tree training time: 0.0436s  
Addestramento del k-NN...  
k-NN training time: 0.0026s  
Addestramento del Random Forest...  
Random Forest training time: 0.0425s  
Addestramento completato.
```

```
Addestramento dei modelli...  
Addestramento del Decision Tree...  
Decision Tree training time: 0.0455s  
Addestramento del k-NN...  
k-NN training time: 0.0126s  
Addestramento del Random Forest...  
Random Forest training time: 0.0641s  
Addestramento completato.
```

KNN è il miglior algoritmo per quanto riguarda la velocità di addestramento

Tempo di esecuzione dell'albero di decisione scritto da noi

```
-----  
Addestramento dell'albero di decisione...
```

```
Tempo di addestramento: 0.4927 secondi  
-----
```

```
Valutazione del modello sui dati di test...
```

```
Accuracy: 100.00%  
Precision: 100.00%  
Recall: 100.00%
```

```
Modello Decision Tree addestrato e valutato.  
-----
```

```
Cross-Validation con 10 fold...
```

```
Accuratezza con Cross-Validation: 100.00% (+/- 0.00%)  
-----
```

```
-----  
Addestramento dell'albero di decisione...
```

```
Tempo di addestramento: 20.2282 secondi  
-----
```

```
Valutazione del modello sui dati di test...
```

```
Accuracy: 88.45%  
Precision: 88.03%  
Recall: 91.02%
```

```
Modello Decision Tree addestrato e valutato.  
-----
```

```
Cross-Validation con 10 fold...
```

```
Accuratezza con Cross-Validation: 88.06% (+/- 0.94%)  
-----
```

Considerazioni finali

Nel dataset **Mushroom**, la presenza di feature altamente discriminanti (come *l'odore*) ha reso il problema linearmente separabile, permettendo anche a modelli semplici di raggiungere il 100% di accuratezza.

Nel dataset **Rice**, le caratteristiche geometriche sono valori continui e spesso sovrapposti tra le classi. Per questo è stata necessaria una normalizzazione tramite Standard Scaler. Questa complessità mette in difficoltà i modelli con confini decisionali lineari.

L'importanza del Preprocessing

Senza il **One-Hot Encoding**, l'albero avrebbe potuto interpretare i colori come una scala ordinata, introducendo un *bias* inesistente.

Senza lo **Scaling**, il k-NN darebbe troppa importanza ai valori più alti, ignorando quelli più piccoli e sbagliando le previsioni

Il **Decision-Tree** vince per trasparenza: possiamo seguire il percorso logico (white-box) ed è fondamentale per spiegare il "perché" di una decisione.

Il **Random Forest** vince per stabilità: la tecnica di Bagging ha eliminato la varianza, confermandosi la scelta migliore per generalizzare su nuovi dati.



Considerazioni finali

Abbiamo confrontato i due dataset chiedendoci: "Cosa accadrebbe se il modello sbagliasse?"

La risposta cambia completamente tra i due casi.

Per quanto riguarda il dataset Mushroom, un errore di classificazione di un fungo velenoso come commestibile, può comportare gravi conseguenze. Per questo è fondamentale che il modello sia quanto più accurato possibile, anche a fronte di un più alto costo computazionale.

Per il dataset Rice, stiamo solo distinguendo due varietà di chicchi (Cammeo e Osmancik).

Questo può dar la possibilità di trovare dei compromessi tra accuratezza e costo computazionale, siccome in questo ambito un errore nella classificazione non comporterebbe una conseguenza grave come nel caso del dataset Mushroom.



Considerazioni finali

Per quanto riguarda il dataset Mushroom, valutiamo che il modello più adeguato per eseguire operazioni di classificazione sia Random Forest, questo perché, nonostante sia computazionalmente più costoso, a fronte della Cross-Fold-Validation, restituisce una Accuracy media maggiore.

```
Addestramento del Decision Tree...  
Decision Tree training time: 0.0436s  
Addestramento del k-NN...  
k-NN training time: 0.0026s  
Addestramento del Random Forest...  
Random Forest training time: 0.0425s
```

```
Decision Tree CV Accuracy (Media): 96.73%  
Decision Tree CV Accuracy (Deviazione Standard): 9.41%  
  
k-NN CV Accuracy (Media): 96.56%  
k-NN CV Accuracy (Deviazione Standard): 8.25%  
  
Random Forest CV Accuracy (Media): 96.85%  
Random Forest CV Accuracy (Deviazione Standard): 9.45%
```

Per quanto riguarda invece il dataset Rice, il modello più adeguato è K-NN perché garantisce un'accuracy quasi a pari degli altri modelli, ma con un costo computazionale significativamente minore.

```
Addestramento del Decision Tree...  
Decision Tree training time: 0.0455s  
Addestramento del k-NN...  
k-NN training time: 0.0126s  
Addestramento del Random Forest...  
Random Forest training time: 0.0641s
```

```
Decision Tree CV Accuracy (Media): 91.89%  
Decision Tree CV Accuracy (Deviazione Standard): 1.97%  
  
k-NN CV Accuracy (Media): 91.76%  
k-NN CV Accuracy (Deviazione Standard): 1.91%  
  
Random Forest CV Accuracy (Media): 91.76%  
Random Forest CV Accuracy (Deviazione Standard): 2.20%
```


Fine

Orsini Fabio
Migliaccio Matteo

