Machine Learning

Università Roma Tre
Dipartimento di Ingegneria

Anno Accademico 2021 - 2022

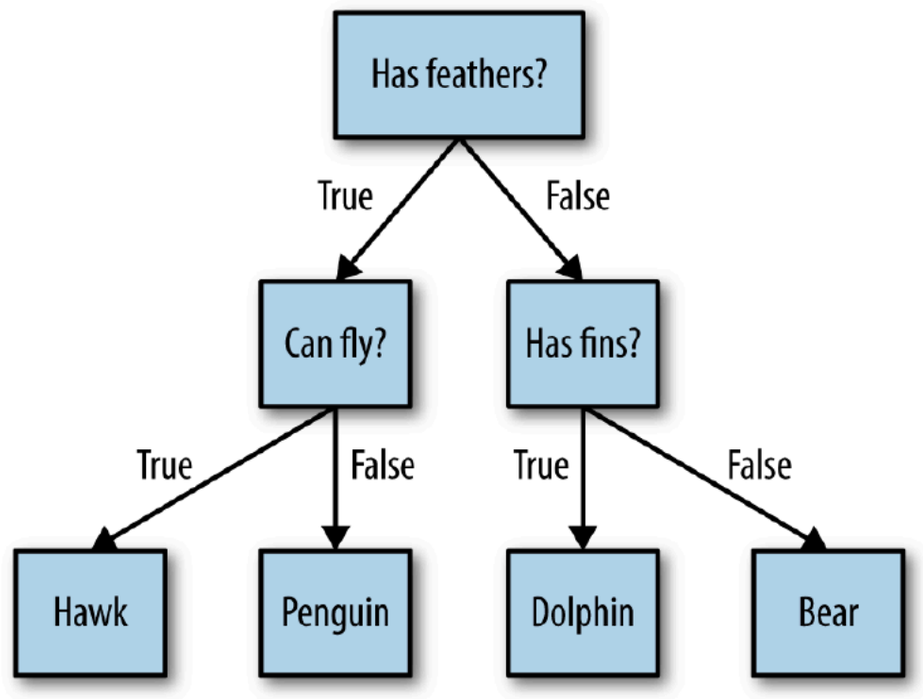
Esercitazione: Decision Trees (Ex05)

Sommario

- Richiami
- scikit-learn e decision trees
- Visualizzazione
- Feature importance
- Decision trees e regressione
- Pruning

Richiami: Decision Trees

- Impiegati spesso per la classificazione e regressione.
- In sintesi creano una albero di nodi if/else che porta ad una certa decisione.
- Si può rappresentare come un albero dove le foglie contengono la risposta.



Richiami: Decision Trees

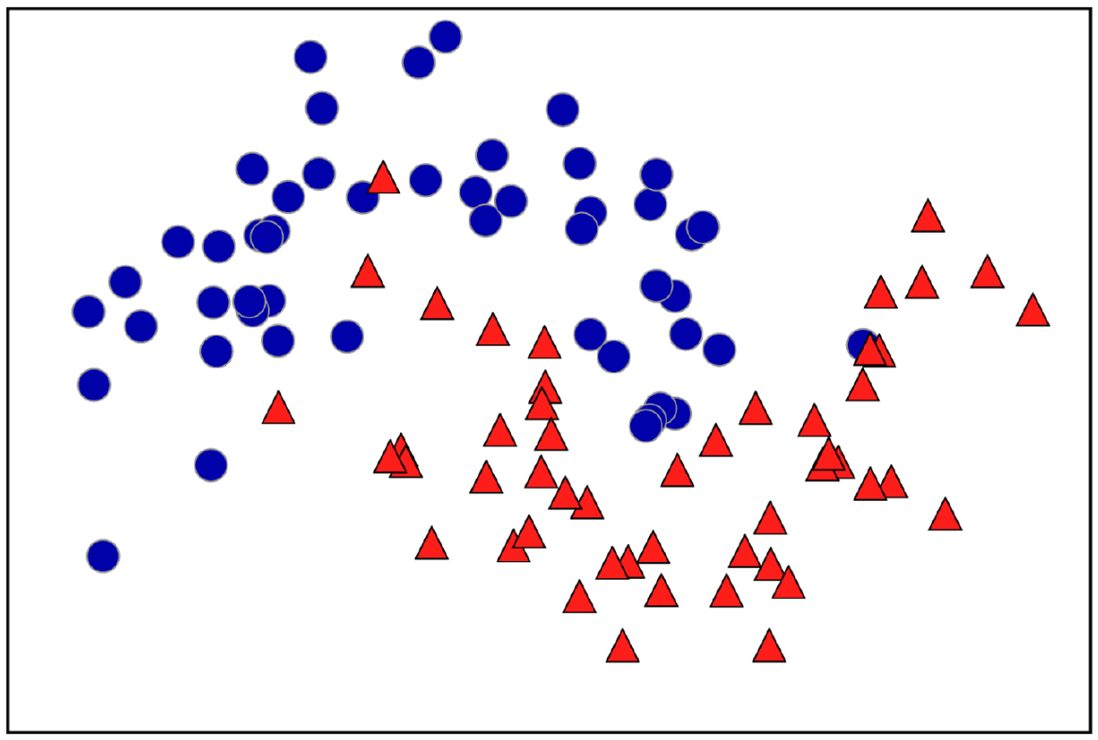
- In ML, ogni "domanda" in un nodo è chiamata comunemente *test*, ed è spesso codificata con feature su domini continui, ad esempio:
  - la feature *i* è maggiore del valore *a*?
- L'algoritmo si focalizza nello scegliere le sequenze if/else che portano ad una riposta più velocemente, ovvero sono più *informative* per la variabile target.

Dataset two_moons

- Toy dataset generato da scikit-learn

sklearn.datasets.make_moons(*n_samples=100*, *, *shuffle=True*, *noise=None*, *random_state=None*)

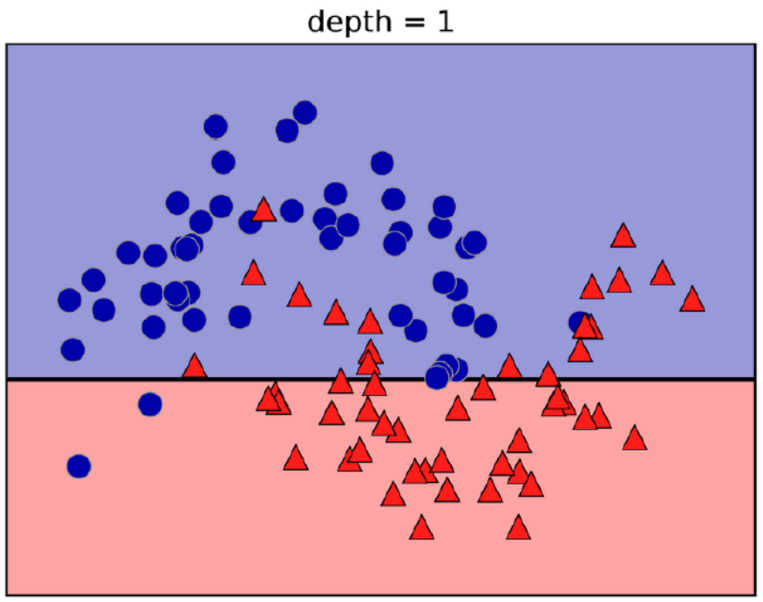
- Ogni istanza ha 2 valori.
- Ad esempio, per 75 istanze otteniamo:

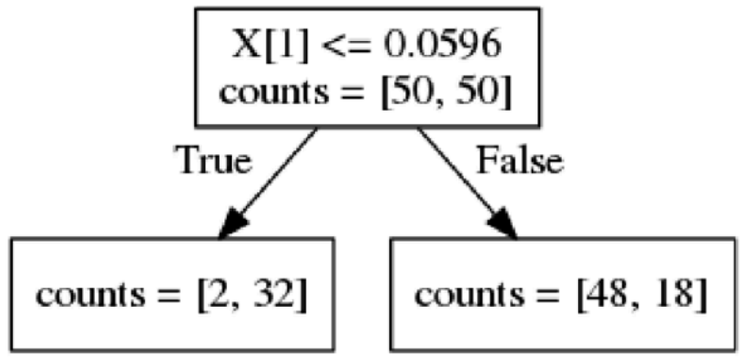


- Per la profondità 0 dell'albero, quale test immagineresti?

Decision Trees su two_moons dataset

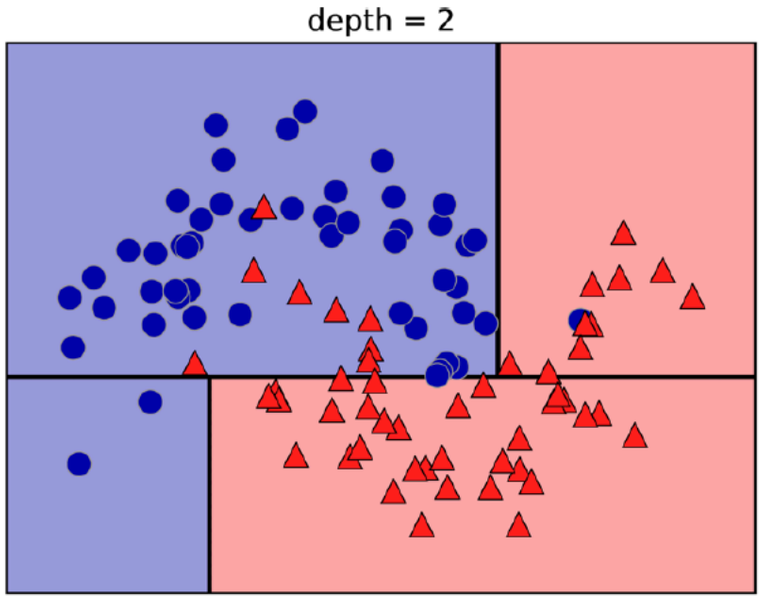
- Depth = 1

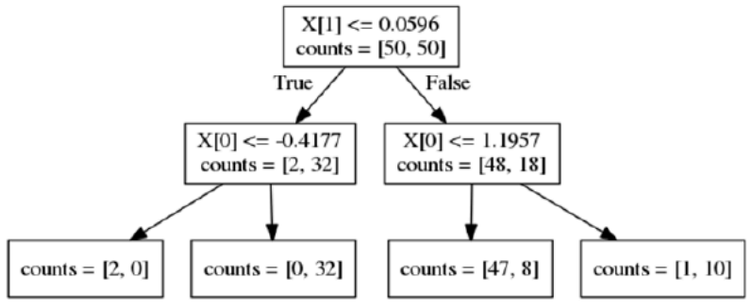




dove [2,32] indica che 2 istanze appartengono alla classe 1 e 32 alla classe 2

- Depth = 2





- …

Se in una foglia ci sono istanze appartenenti ad una sola classe allora la foglia si chiama *pure*.

Richiami: Decision Trees

- Ogni test considera una singola feature, perciò la relativa decisione è rappresentata come una asse parallelo ad uno degli assi.
- Nella predizione, una volta arrivati ad una foglia, si assegna la classe target che appare più spesso nella regione associata. In modo simile per la regressione si opera una media dei valori delle istanze nella regione.
- Per dataset grandi, creare foglie pure è molto dispendioso in termini di risorse computazione e può creare fenomeni di overfitting.
  - Si possono implementare tecniche di early stopping limitando la profondità dell'albero (*pre-pruning*), oppure rimuovere o fondere foglie che contengono poca informazione (*post-pruning* o *pruning*)

scikit-learn e decision trees

- La classe **DecisionTreeClassifier** del modulo **DecisionTreeRegressor** implementa l'algoritmo.
- Esercizio: prova ad impiegarlo nel dataset Breast Cancer e valuta l'accuracy sul training e test set

```
from sklearn.tree import DecisionTreeClassifier
cancer = load_breast_cancer()
X_train, X_test, y_train, y_test = train_test_split(
cancer.data, cancer.target, stratify=cancer.target, random_state=42)
...
```

scikit-learn e decision trees

- La classe **DecisionTreeClassifier** del modulo **DecisionTreeRegressor** implementa l'algoritmo.
- Esercizio: prova ad impiegarlo nel dataset Breast Cancer e valuta l'accuracy sul training e test set

```
from sklearn.tree import DecisionTreeClassifier
cancer = load_breast_cancer()
X_train, X_test, y_train, y_test = train_test_split(
cancer.data, cancer.target, stratify=cancer.target, random_state=42)
tree = DecisionTreeClassifier(random_state=0)
tree.fit(X_train, y_train)
print("Accuracy on training set: {:.3f}".format(tree.score(X_train, y_train)))
print("Accuracy on test set: {:.3f}".format(tree.score(X_test, y_test)))
```

```
> Accuracy on training set: 1.000
> Accuracy on test set: 0.937
```

- Ti aspettavi una accuracy del 100%?

scikit-learn e decision trees

```
> Accuracy on training set: 1.000
> Accuracy on test set: 0.937
```

- Ti aspettavi una accuracy del 100%? Sì, per come funziona l'algoritmo l'albero cresce fino a creare foglie pure che rappresentazione perfettamente l'appartenenza delle istanze alle relative label.
- L'accuracy sul test set è leggermente inferiore ai modelli lineari (95% ca).
- Esercizio: Prova a impostare una profondità col parametro *max_depth* durante la costruzione dell'oggetto DecisionTreeClassifier. Cosa ti aspetti sulle due accuracy?

scikit-learn e decision trees

```
tree = DecisionTreeClassifier(max_depth=4, random_state=0)
tree.fit(X_train, y_train)
print("Accuracy on training set: {:.3f}".format(tree.score(X_train, y_train)))
print("Accuracy on test set: {:.3f}".format(tree.score(X_test, y_test)))
```

```
> Accuracy on training set: 0.988
> Accuracy on test set: 0.951
```

- Più bassa sul training, ma migliora (meno overfitting) sul test.

scikit-learn: visualizzare i decision trees

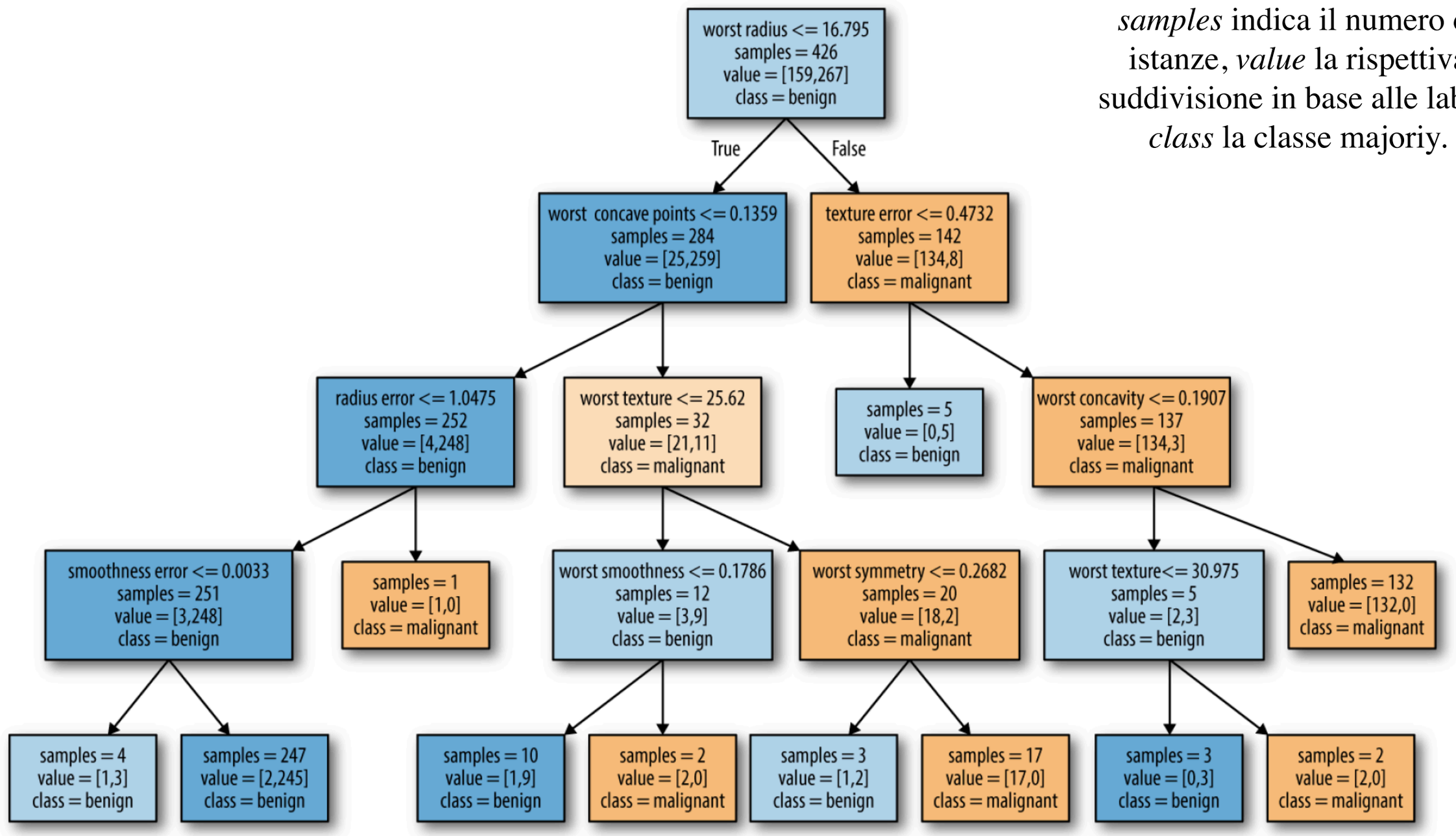
- La funzione **export_graphviz** del modulo **tree** permette di visualizzare l'albero. Salva un file .dot che può essere importato per la visualizzazione.

```
from sklearn.tree import export_graphviz

export_graphviz(tree, out_file="tree.dot", class_names=["malignant",
"benign"], feature_names=cancer.feature_names, impurity=False, filled=True)

import graphviz
with open("tree.dot") as f:
dot_graph = f.read()
graphviz.Source(dot_graph)
```

- Visualizzare il "comportamento" di un algoritmo di ML è molto utile per spiegarne l'output (*explaination*), in questo caso anche ai non-esperti.

scikit-learn: visualizzare i decision trees

- L'albero generato:



samples indica il numero di istanze, *value* la rispettiva suddivisione in base alle label, *class* la classe majoriy.

scikit-learn: visualizzare i decision trees

- Un altro modo per esplorare i decision trees è assegnare una misura di *importanza* alle feature in base al funzionamento dell'algoritmo.
- La variabile feature_importances_ del modello è un array con valori in [0,1] dove 1 indica "predice perfettamente in valore target".

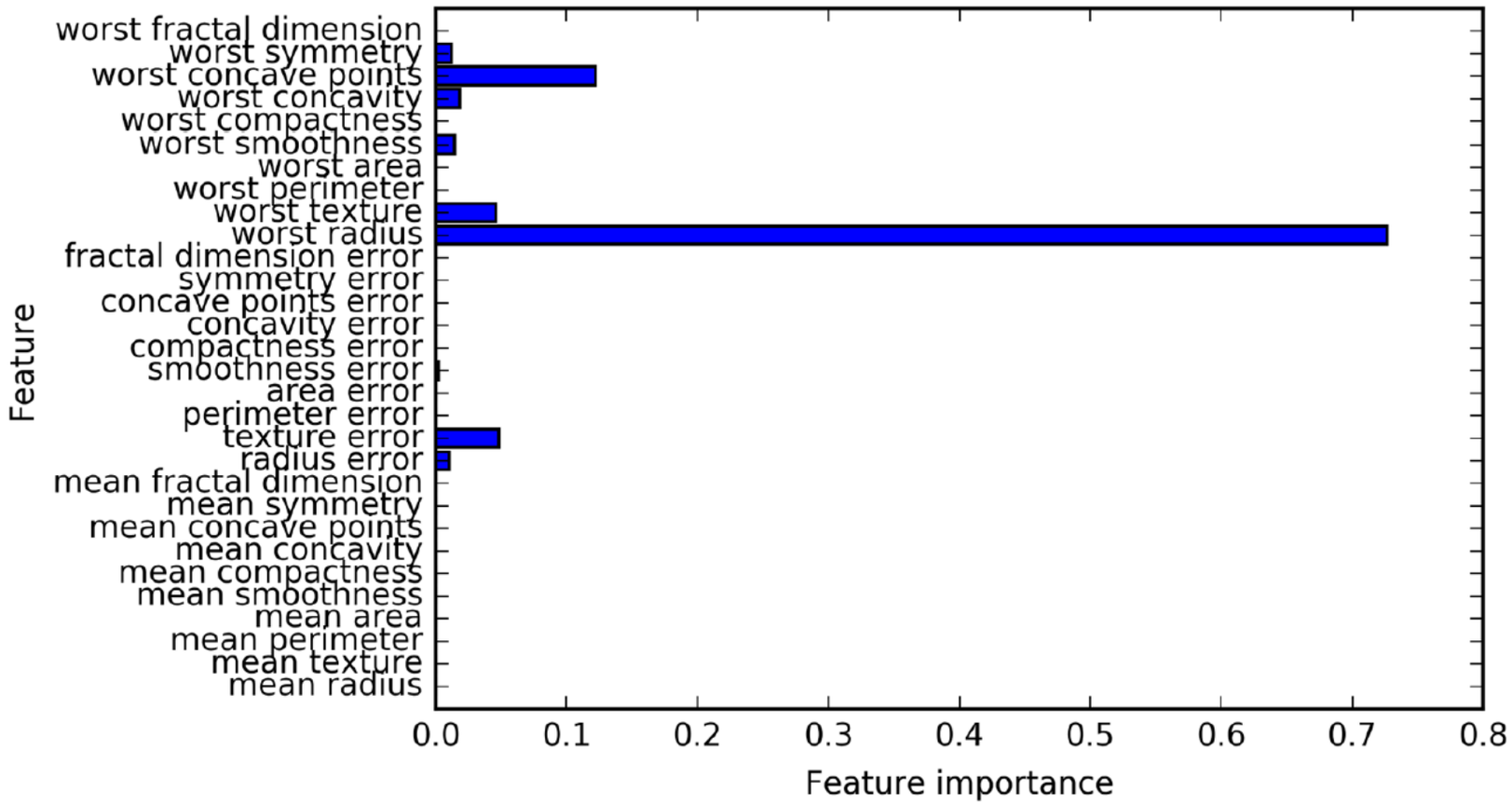
```
print("Feature importances:\n{}".format(tree.feature_importances_))
Out[62]:

[ 0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.01
0.048 0.  0.  0.002 0.  0.  0.  0.  0.  0.727 0.046
0.  0.  0.014 0.  0.018 0.122 0.012 0.  ]

def plot_feature_importances_cancer(model):
  n_features = cancer.data.shape[1]
  plt.barh(range(n_features), model.feature_importances_, align='center')
  plt.yticks(np.arange(n_features), cancer.feature_names)
  plt.xlabel("Feature importance")
  plt.ylabel("Feature")
plot_feature_importances_cancer(tree)
```

Decision trees e feature importance

- Un altro modo per esplorare i decision trees è assegnare una misura di *importanza* alle feature in base al funzionamento dell'algoritmo.
- La variabile **feature_importances_** del modello è un array con valori in [0,1] dove 1 indica "predice perfettamente in valore target". È valutata mediante la metrica GINI.

```
print("Feature importances:\n{}".format(tree.feature_importances_))
Out[62]:

[ 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.01
0.048 0. 0. 0.002 0. 0. 0. 0. 0. 0.727 0.046
0. 0. 0.014 0. 0.018 0.122 0.012 0. ]

def plot_feature_importances_cancer(model):
  n_features = cancer.data.shape[1]
  plt.barh(range(n_features), model.feature_importances_, align='center')
  plt.yticks(np.arange(n_features), cancer.feature_names)
  plt.xlabel("Feature importance")
  plt.ylabel("Feature")
plot_feature_importances_cancer(tree)
```

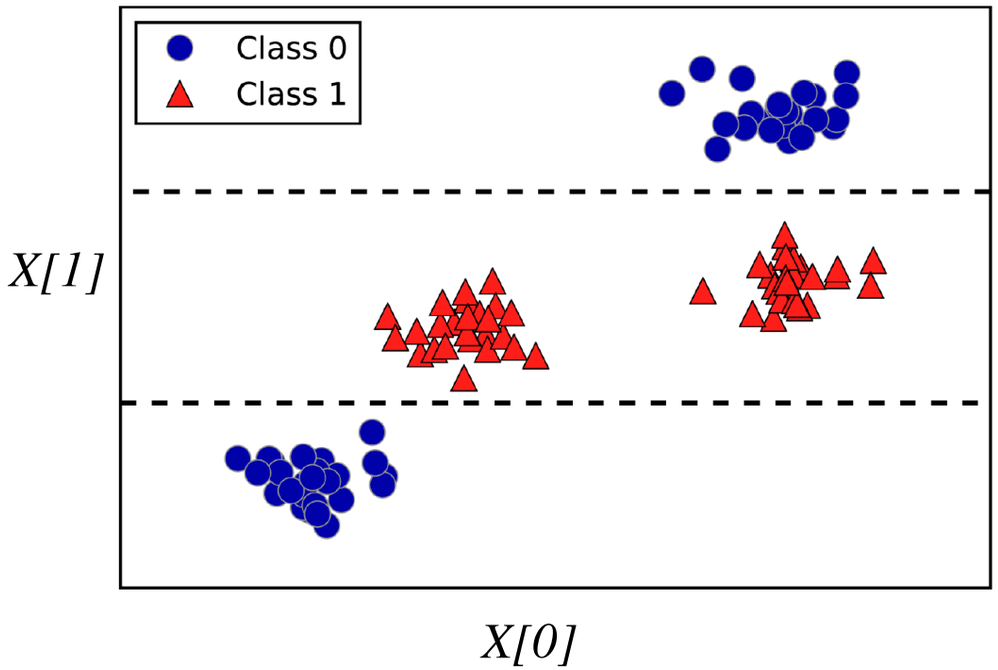
Decision trees e feature importance



- Si può notare come worst radius è la feature più discriminante. Questo indica anche che l'albero è ben costruito avendo questa feature in cima.
- Puoi dire che una *feature* con bassa importance è poco discriminante?

Esempio: feature importance

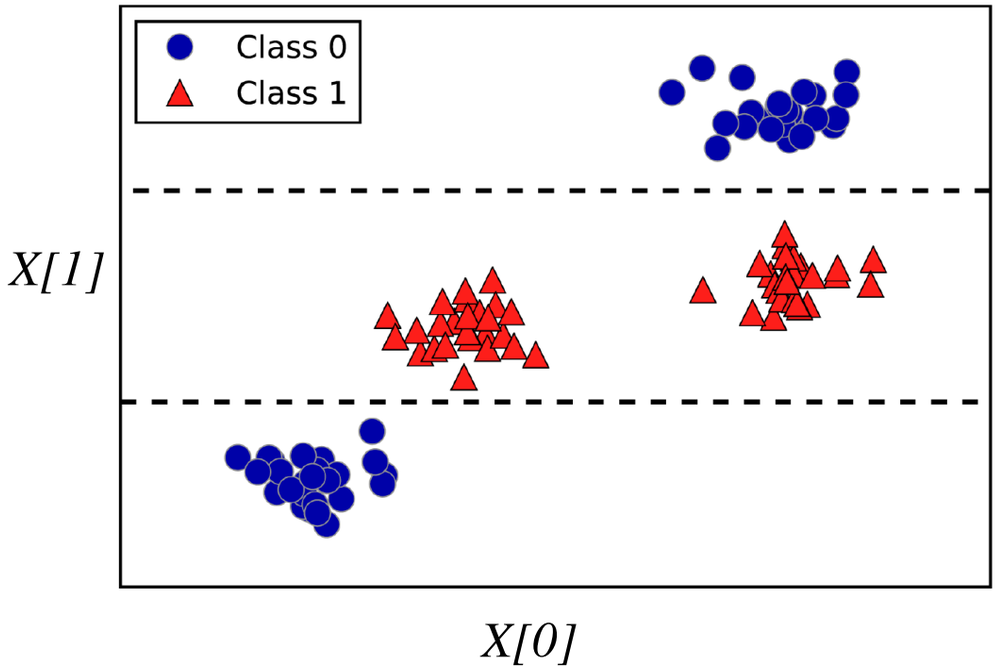
```
tree = mglearn.plots.plot_tree_not_monotone()
display(tree)
```

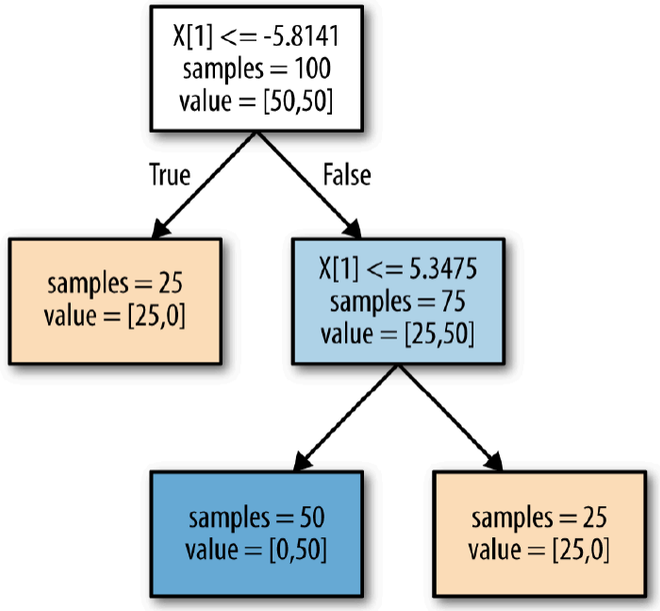


- **Esercizio:** In questo esempio come costruiresti l'albero di decisione?

Esempio: feature importance

```
tree = mglearn.plots.plot_tree_not_monotone()
display(tree)
```



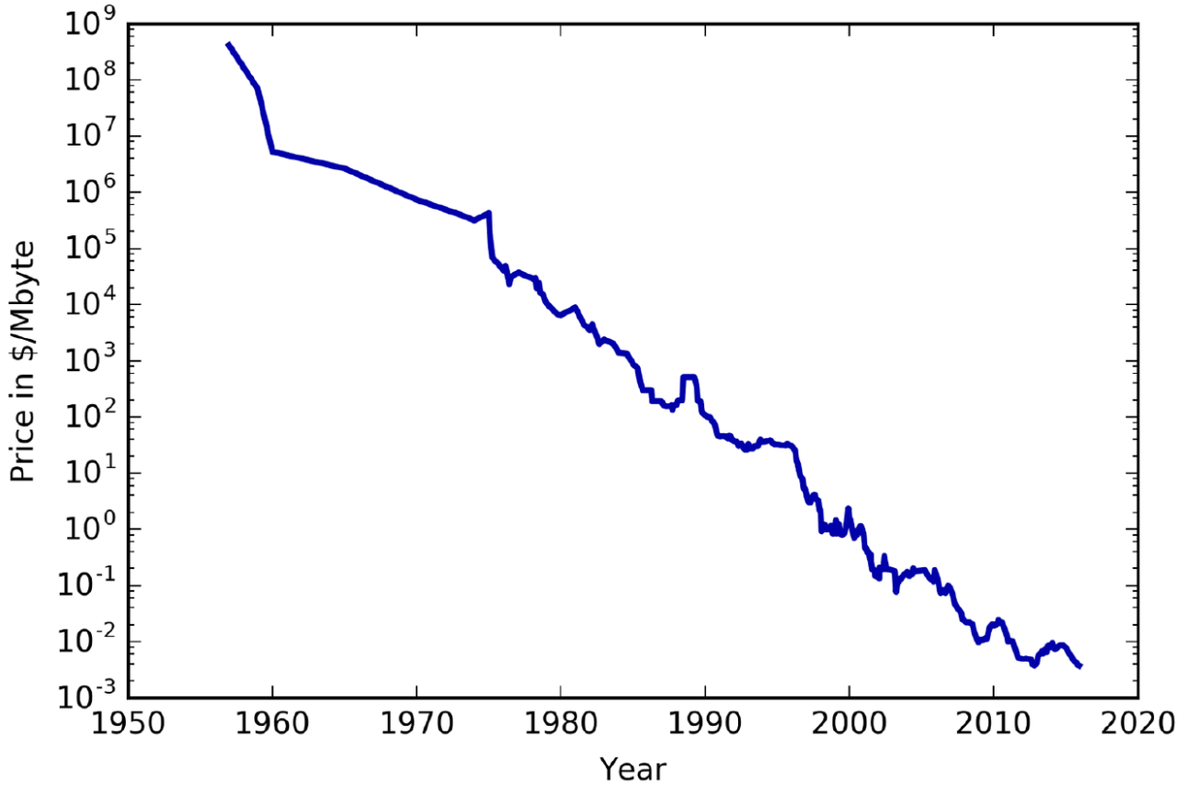


- In questo esempio, l'informazione rilevante è contenuta in X[1]. Infatti non possiamo dire che un valore alto per la feature X[0] identifica la classe 0, e uno basso la classe 1. L'albero effettivamente impiega la feature corretta.

scikit-learn: decision tree per la regressione

- La classe DecisionTreeRegressor impiega lo stesso algoritmo in ambito di regressione

```
import pandas as pd
ram_prices = pd.read_csv("data/ram_price.csv")
plt.semilogy(ram_prices.date, ram_prices.price)
plt.xlabel("Year")
plt.ylabel("Price in $/Mbyte")
```



Su scala logaritmica per le y si può ipotizzare una relazione lineare

https://github.com/amueller/introduction_to_ml_with_python/blob/master/data/ram_price.csv

scikit-learn: decision tree per la regressione

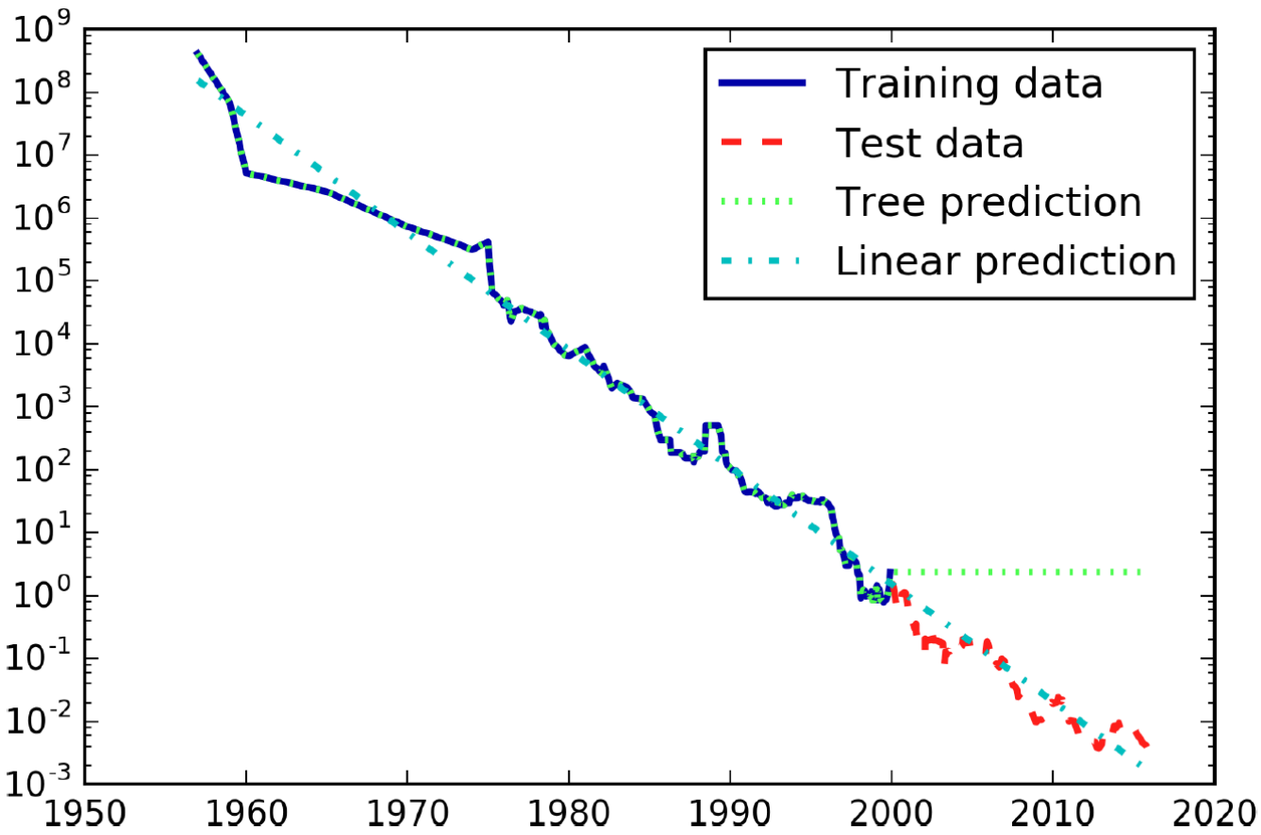
- Confrontiamo i decision trees con un modelli lineare, con l'accortezza di convertire i dati in valori logaritmo altrimenti il modello lineare non può funzionare.

```
from sklearn.tree import DecisionTreeRegressor
# use historical data to forecast prices after the year 2000
data_train = ram_prices[ram_prices.date < 2000]
data_test = ram_prices[ram_prices.date >= 2000]

# predict prices based on date
X_train = data_train.date[:, np.newaxis]
# we use a log-transform to get a simpler relationship of data to target
y_train = np.log(data_train.price)
tree = DecisionTreeRegressor().fit(X_train, y_train)
linear_reg = LinearRegression().fit(X_train, y_train)

# predict on all data
X_all = ram_prices.date[:, np.newaxis]
pred_tree = tree.predict(X_all)
pred_lr = linear_reg.predict(X_all)

# undo log-transform
price_tree = np.exp(pred_tree)
price_lr = np.exp(pred_lr)
```

- Cosa ti aspetti?

scikit-learn: decision tree per la regressione

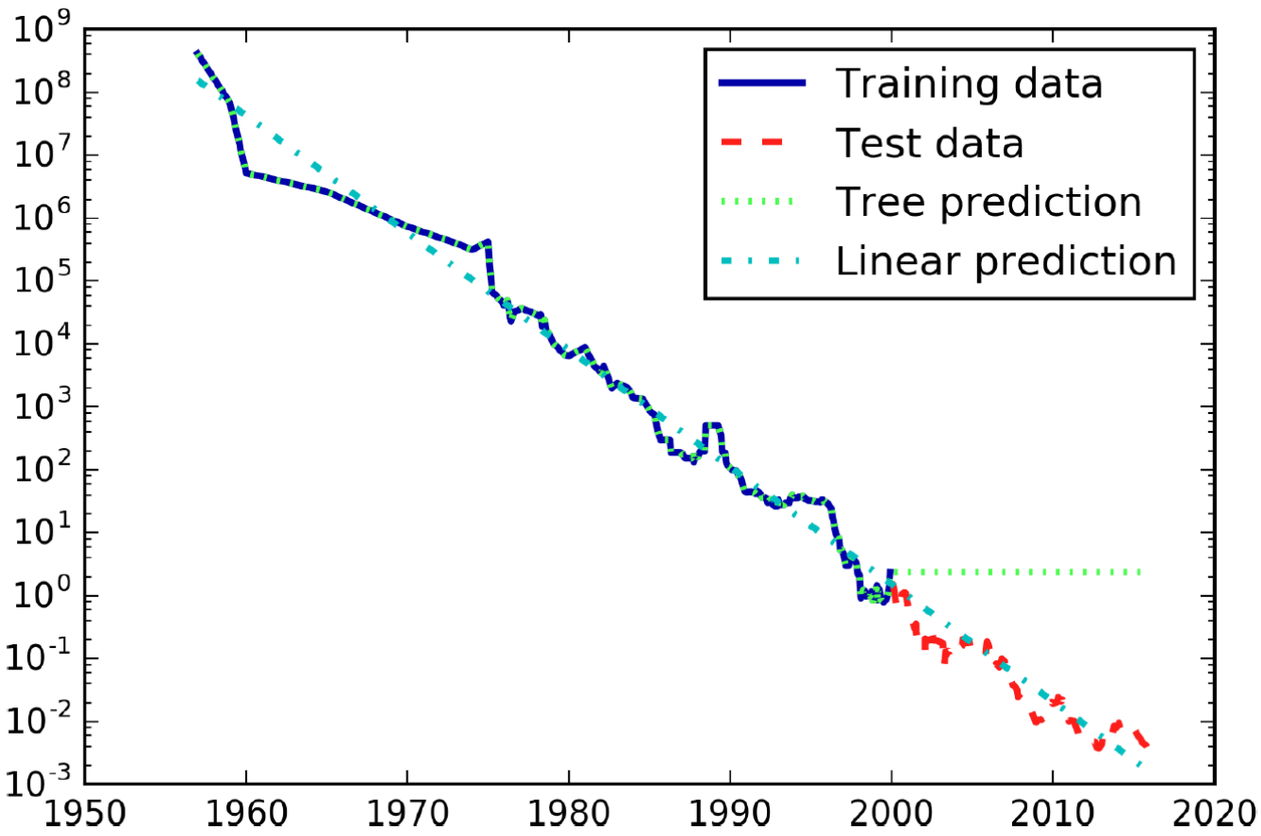
```
plt.semilogy(data_train.date, data_train.price, label="Training data")
plt.semilogy(data_test.date, data_test.price, label="Test data")
plt.semilogy(ram_prices.date, price_tree, label="Tree prediction")
plt.semilogy(ram_prices.date, price_lr, label="Linear prediction")
plt.legend()
```



- Il modello lineare approssima con una retta. Il decision tree è molto più accurato nella predizione.
- Ma che succede dopo l'ultima data presente nel dataset?

scikit-learn: decision tree per la regressione

```
plt.semilogy(data_train.date, data_train.price, label="Training data")
plt.semilogy(data_test.date, data_test.price, label="Test data")
plt.semilogy(ram_prices.date, price_tree, label="Tree prediction")
plt.semilogy(ram_prices.date, price_lr, label="Linear prediction")
plt.legend()
```



- Il modello lineare approssima con una retta. Il decision tree è molto più accurato nella predizione.
- **Attenzione:** L'algoritmo decision tree non è in grado di fare predizioni su nuovi dati con la data oltre a quella contenuta nel dataset.

scikit-learn: decision tree e pruning

- Per limitare l'overfitting e la complessità, solitamente è sufficiente impiegare una tecnica di pre-pruning con uno dei seguenti parametri:

  `max_depth`, `max_leaf_nodes`, o. `min_samples_leaf`

- Nota: min_samples_leaf indica il minimo numero di istanze per foglia.

- **Esercizio**: prova ad addestrare nuovamente il decision trees sul breast cancer dataset impostano a turno uno di questi valori e valutare le variazioni di accuracy.

Testi di Riferimento

- Andreas C. Müller, Sarah Guido. *Introduction to Machine Learning with Python: A Guide for Data Scientists*. O'Reilly Media 2016
- Aurélien Géron. *Hands-On Machine Learning with Scikit-Learn and TensorFlow: Concepts, Tools, and Techniques to Build Intelligent Systems*. O'Reilly Media 2017