RANDOM FOREST PER RILEVARE L'OCCUPAZIONE DI UNA STANZA

Gabriele Gadaleta, Francesco Marchetto, Andrea Morandini

Obiettivo e applicazioni

Obiettivo

L'obiettivo del progetto è quello di usare un algoritmo di **machine learning** per **prevedere la presenza di persone in una stanza**, utilizzando i dati provenienti da dei **sensori ambientali**.

Applicazioni

Alcune possibili applicazioni di questo lavoro possono essere:

- Ottimizzazione del **consumo energetico** negli edifici
- Sistemi di **sicurezza**
- **Organizzazione** e gestione degli spazi

Il Progetto

1. Visualizzazione

Abbiamo visualizzato il dataset, contenente **informazioni da sensori ambientali**. Dalle visualizzazioni abbiamo cercato di capire **quali features potevano essere importanti** per la classificazione e se esistevano dei “**pattern**” che si ripetevano.

2. Classificazione

Una volta capito come era strutturato il dataset abbiamo proseguito con:

1. **Subsampling** per ottenere training test e testing set
2. Selezione degli **iperparametri**
3. **Fitting** del modello
4. **Valutazione**

Il dataset

Il dataset contiene delle rilevazioni **minuto per minuto**, provenienti da dei **sensori** ambientali relativamente a:

- **Temperatura** (`Temperature` - °C)
- **Umidità** (`Humidity` - %)
- **Luminosità** (`Light` - Lux)
- **Anidride Carbonica** (`CO2` - ppm)
- **Rapporto di umidità** (`HumidityRatio` - kgwater-vapor/kg-air)

Viene indicata anche una **Data** (`date`)

Ad ogni rilevazione è associato un valore numerico intero uguale ad:

- **1** ⇒ se la stanza è occupata
- **0** ⇒ se la stanza non è occupata

Il dataset

Ai fini della visualizzazione e della classificazione, abbiamo scelto di aggiungere due features al dataset (derivandole dalla data):

- **Giorno della settimana** (`weekday`)
- **Ora** (`hour`, arrotondata all'ora più vicina)

Abbiamo inoltre scelto di **unire due dei tre dataset** che venivano forniti dagli autori, tenendo il terzo da parte per una **valutazione finale complessiva**.

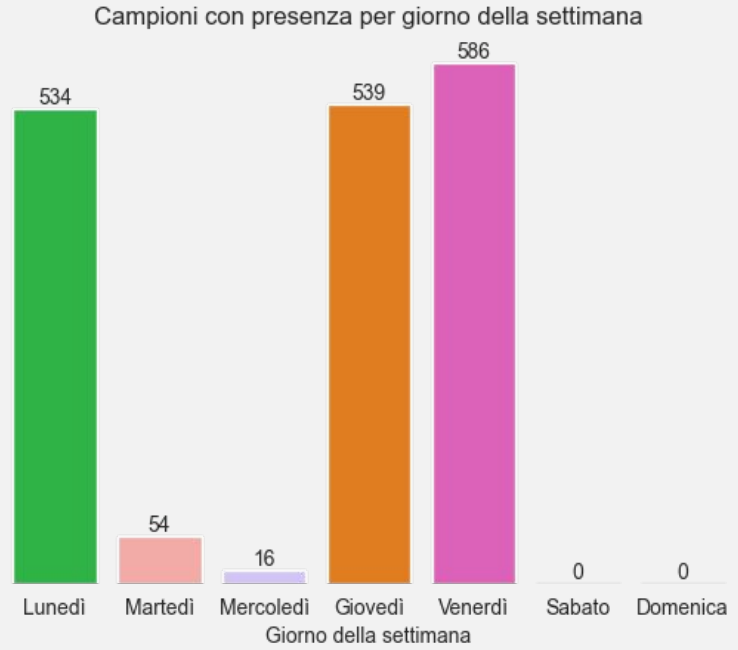
Visualizzazione

Bar chart per giorno

Una prima visualizzazione a cui possiamo pensare è un **bar chart** che mostri il **numero di campioni con presenza per giorno della settimana**.

Dal grafico è evidente che nel weekend non ci sia mai nessuno nella stanza ⇒ Questo potrebbe essere utilizzato nella classificazione.

Essendo che le misurazioni sono iniziate di mercoledì e sono terminate di martedì abbiamo un gap di mezza giornata nel dataset in questi giorni.



Heatmap orari

Proviamo anche a visualizzare come i campioni con presenza **si distribuiscono sulle ore della giornata** utilizzando una **heat map**.

Vediamo che gli orari più caldi sono la mattina e il pomeriggio (la notte non è mai stata rilevata neanche una presenza) ⇒ Anche questo sembra essere utile ai fini della classificazione.

Andamento temporale

Proviamo quindi a visualizzare l'**andamento temporale delle feature**, mettendo in **evidenza quando la stanza è occupata** (in rosso).

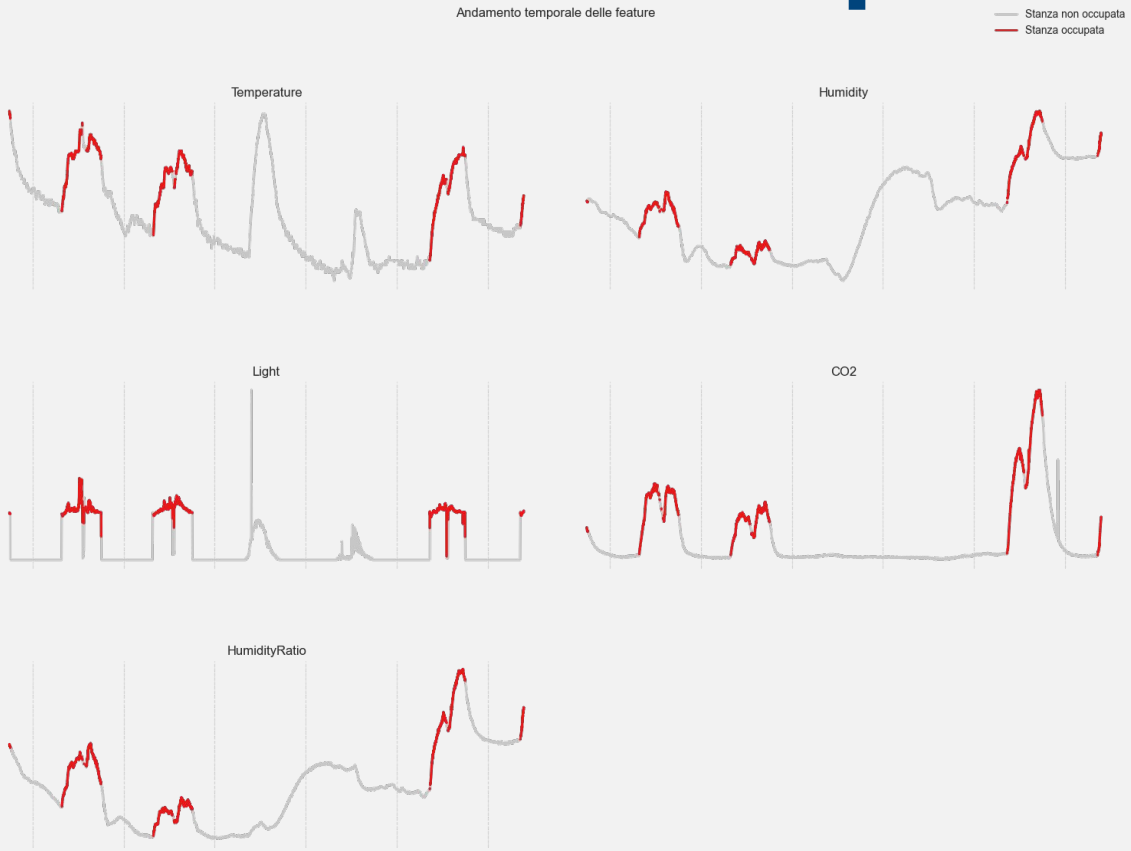
Notiamo che:

- **Luce**
- **CO2**

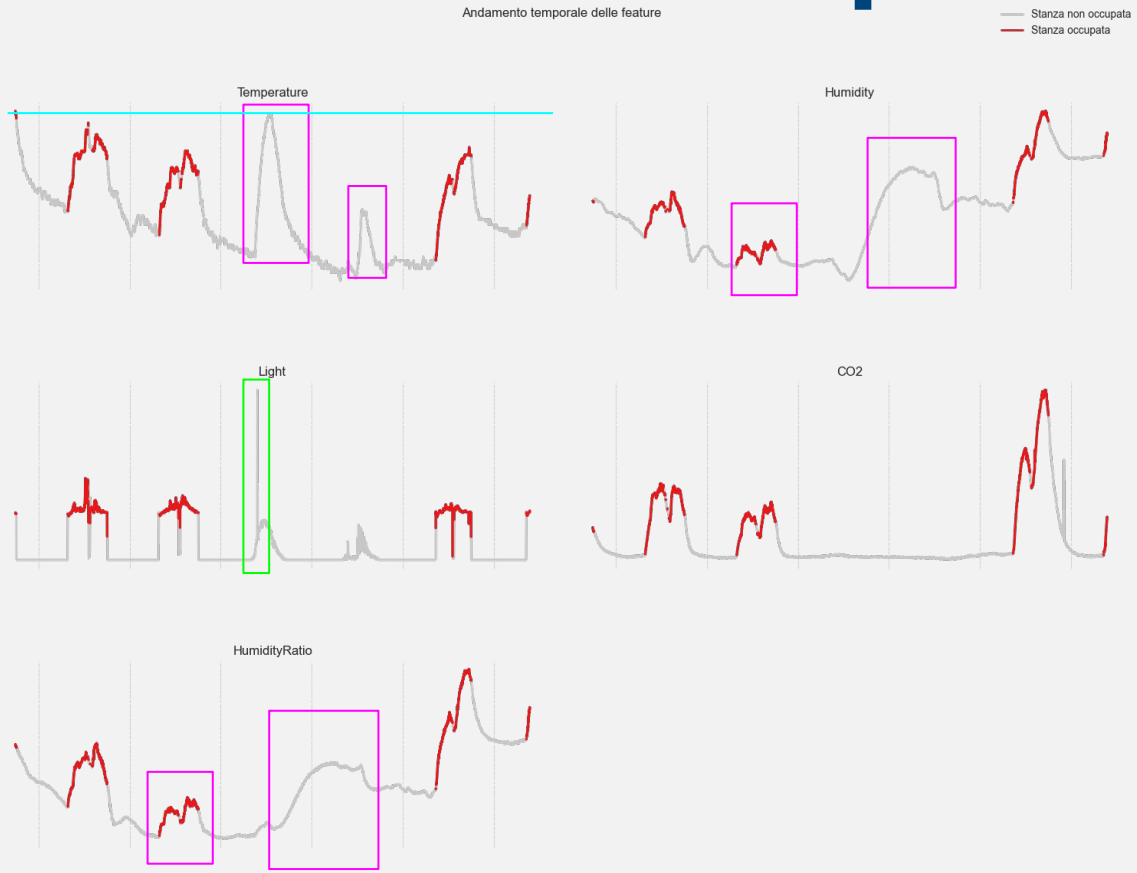
Sembrano indicare una separazione netta fra quando la stanza è occupata e quando non.

Le altre metriche, invece, crescono quando la stanza è occupata, ma anche in altri momenti ⇒ non sono conclusive e avranno probabilmente **meno importanza ai fini della classificazione**

Andamento temporale



Andamento temporale



Box plot features

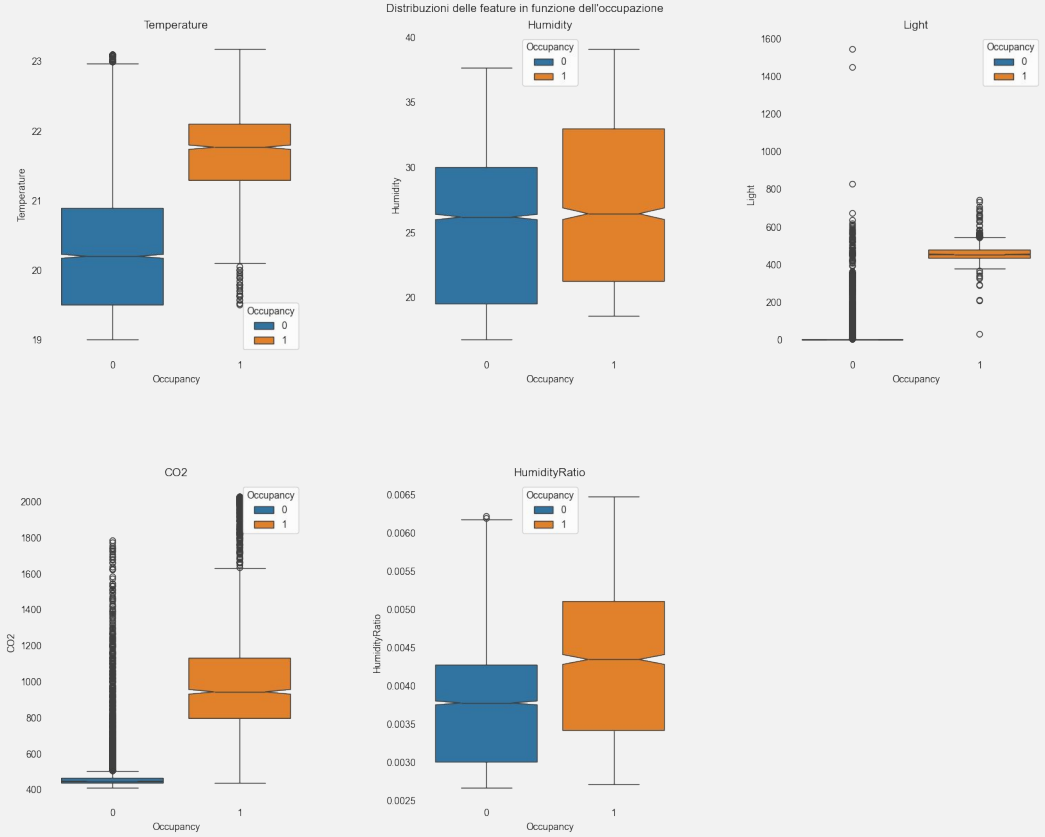
Visualizziamo quindi le distribuzioni delle features rispetto all'occupazione con dei boxplot, per visualizzare l'importanza delle singole ai fini della classificazioni.

Boxplot **Humidity** ⇒ notch overlapping
Occupancy *true* e *false* potrebbero avere la stessa distribuzione. Immaginiamo quindi che tale feature sarà poco rilevante ai fini della classificazione.

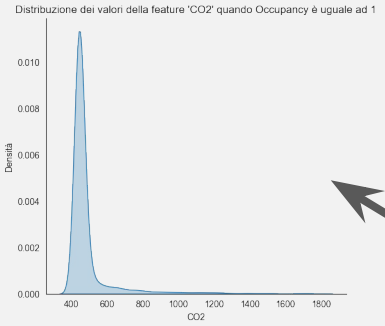
Luce, **CO2** e **temperatura** presentano invece notch molto ristretti e separati in base alla *Occupancy* per cui possiamo prevedere che le distribuzioni siano diverse, e che tali feature saranno più rilevanti nella classificazione.

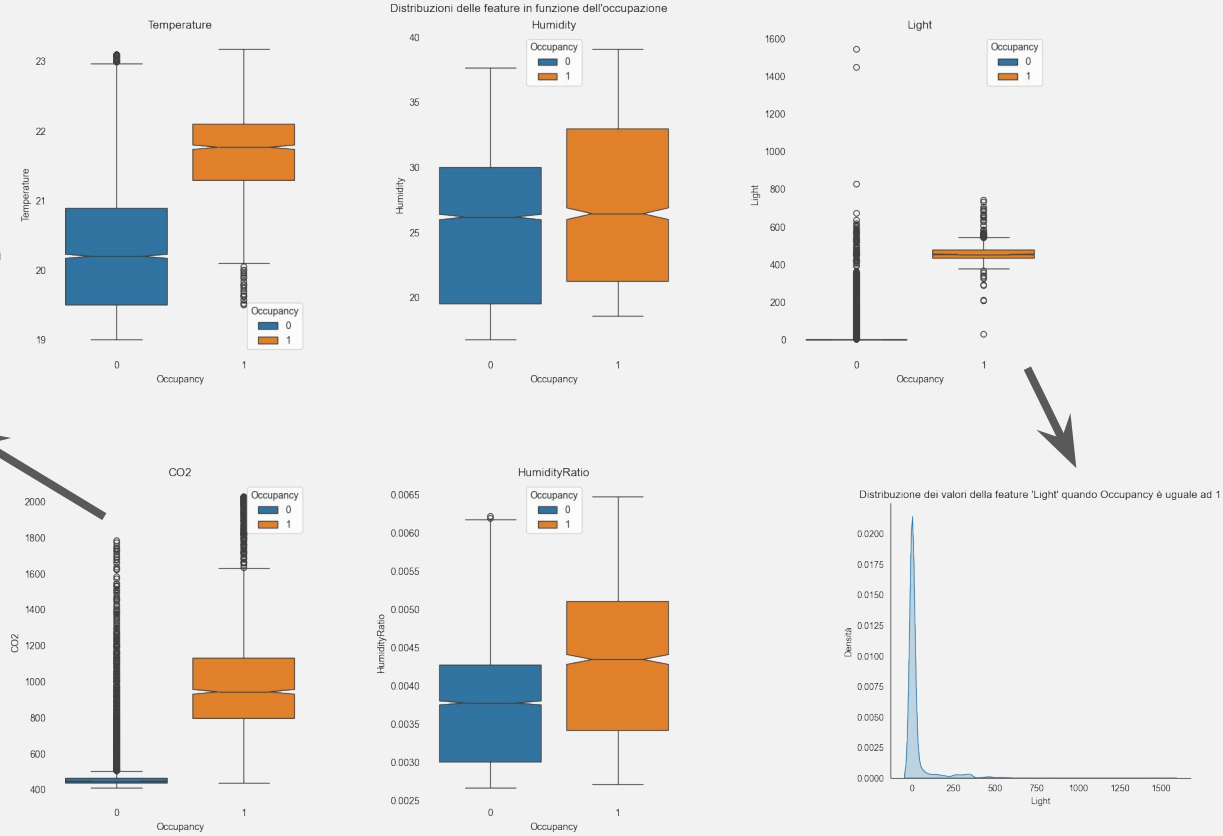
HumidityRatio presenta anch'essa notch separati, ma distribuzioni più "vicine" ⇒ sarà meno importante ai fini della classificazione.

Box plot features



Box plot features





Classificazione

Subsampling

Per prima cosa abbiamo effettuato il **subsampling** del dataset secondo una regola 75/25 (training/testing), utilizzando un campionamento **stratificato**, in modo da mantenere il bilanciamento presente nel dataset.

Questo ci permette di usare molti valori per fare il fitting del modello, mantenendo comunque una buona dimensione del testing set, necessario per validare le performance del modello.

Iperparametri

Iperparametri ⇒ i parametri che non vengono appresi direttamente dal modello (la random forest in questo caso)

Nel nostro caso, utilizziamo come iperparametri:

- **`n_estimators`** ⇒ Il numero di alberi nella foresta
- **`min_samples_split`** ⇒ Il numero minimo di campioni perchè un nodo venga ulteriormente diviso *(se il valore presente è minore di questo iperparametro il nodo sarà una foglia)*
- **`min_samples_leaf`** ⇒ Il numero minimo di campioni che deve avere una foglia *(se lo split produce foglie di dimensione inferiore a questo parametro lo split non avviene)*
- **`max_samples`** ⇒ La percentuale di campioni “estratti” per addestrare ogni albero
- **`max_depth`** ⇒ Profondità massima di ogni albero della foresta

Iperparametri

Grid Search Cross-Validation ⇒ per la selezione degli iperparametri andiamo a fare una ricerca esaustiva (provando tutte le combinazioni possibili *“brute force”*) su una griglia di parametri per trovare quali performano meglio, sfruttando la 5-fold cross-validation.

Come insieme di iperparametri su cui fare la Grid Search scegliamo alcuni dei valori più comunemente utilizzati per le random forest.

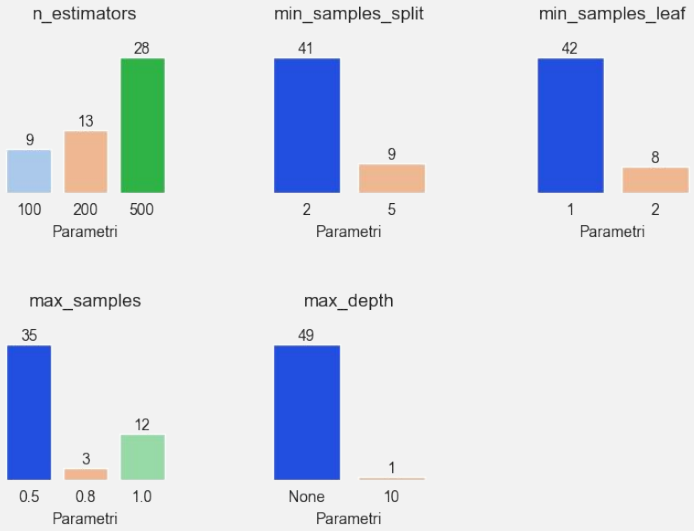
```
param_grid = {
        'n_estimators': [100, 200, 500],
        'min_samples_split': [2, 5, 10],
        'min_samples_leaf': [1, 2, 5],
        'max_samples': [0.5, 0.8, 1.0],
        'max_depth': [None, 10, 30]
    }
```

Iperparametri

`random_state` ⇒ il seed di generazione dei numeri casuali utilizzato

Dato che con `random_state` diversi, otteniamo **iper-parametri leggermente diversi** con la grid search, andiamo a cercare gli iper-parametri "ottimali" su **50** `random_state` diversi (calcolo oneroso - 2h ca).

Quindi, calcoliamo la **moda**, in modo da ottenere gli iper-parametri più frequenti e quindi, probabilmente, i migliori in generale per la nostra Random Forest.



Fitting

Una volta individuati il training set e gli iperparametri possiamo procedere a fare il fitting del modello:

```
classifier_with_tuned_paramaters = RandomForestClassifier(
    n_jobs=-1,
    n_estimators=500,
    max_samples=0.5,
)
```

```
classifier_with_tuned_paramaters.fit(X_train, y_train)
```

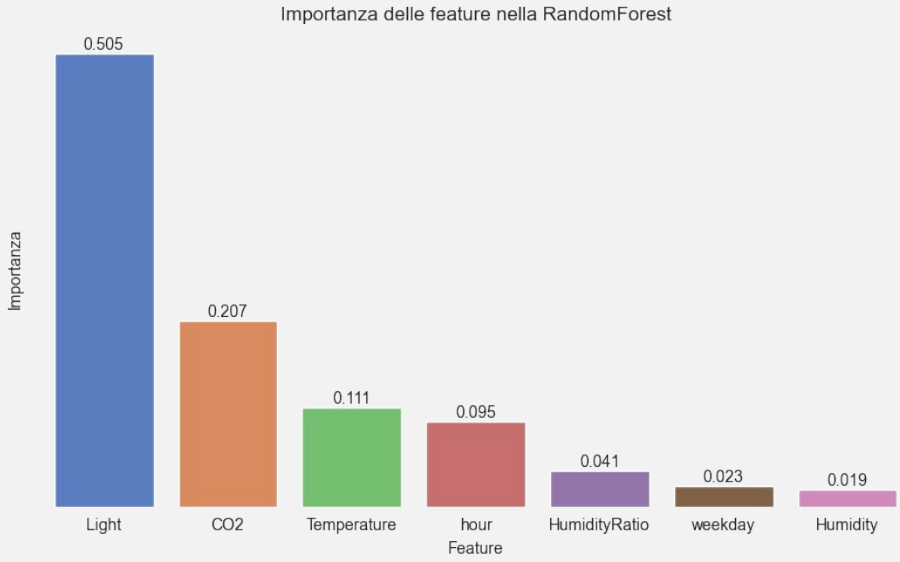
Andiamo a vedere più nel dettaglio la random forest che abbiamo ottenuto...

Importanza features

Andiamo quindi a visualizzare l'importanza di ognuna delle feature, utilizzate per le regole di split nella random forest.

Feature importanti ⇒ Minimizzano l'impurità dei nodi figli

Notiamo che le feature che ci sembravano "migliori" quando abbiamo visualizzato il dataset, ora risultano più importanti.



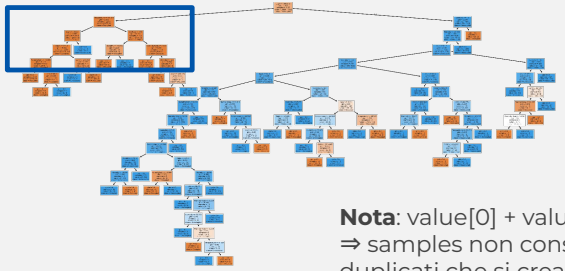

```
classifier_with_tuned_paramaters.feature_importances_
```

Un albero della foresta

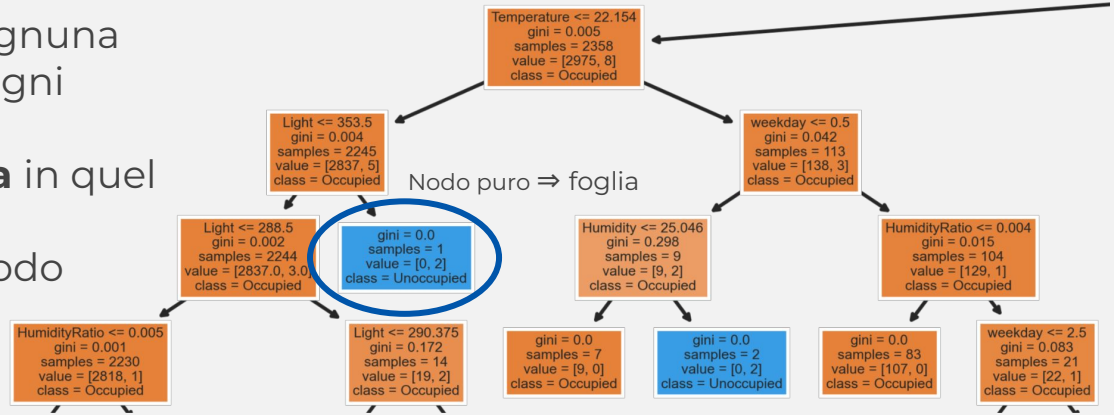
Andiamo a vedere nel dettaglio un albero di decisione *(il 423°, sui 500 della foresta)*.

Nella visualizzazione troviamo indicati:

- La **split rule**
- Il **Gini inequality index**
- Il numero di **campioni** per ognuna delle **due classi** presenti in ogni nodo
- La classe che verrebbe **scelta** in quel nodo
- Il numero di **campioni** nel nodo



Nota: value[0] + value[1] ≠ samples ⇒ samples non considera i duplicati che si creano durante il bootstrapping



Metriche di valutazione

Andiamo quindi a valutare il modello ottenuto, utilizzando come metriche:

- **Precision** ⇒ Ci indica, fra tutti i positivi trovati dal modello, quali lo sono davvero. È importante quando i falsi positivi sono costosi.

  $$P = \frac{T_p}{T_p + F_p}$$

- **Recall** ⇒ Ci indica, fra tutti i veri positivi, quanti ne ha trovati il modello. È importante quando i falsi negativi sono costosi.

  $$R = \frac{T_p}{T_p + F_n} = TPR$$

- **F1 Score** ⇒ Una “media armonizzata” di precision e recall. È utile quando si hanno classi sbilanciate.

  $$\mathrm{F1} = \frac{2 * T_p}{2 * T_p + F_p + F_n}$$

Metriche di valutazione

- **Confusion Matrix** ⇒ Un modo di visualizzare i risultati dell'algoritmo di classificazione
- **ROC** ⇒ Curva che traccia il valore di TPR (Recall!) e FPR al variare della soglia di decisione

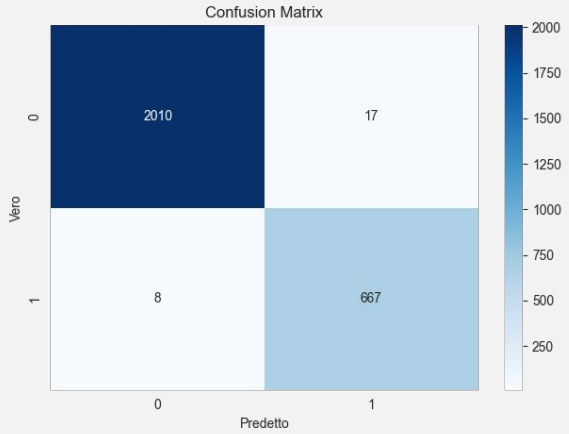
$$FPR = \frac{F_p}{T_p + F_n} \qquad R = \frac{T_p}{T_p + F_n} = TPR$$

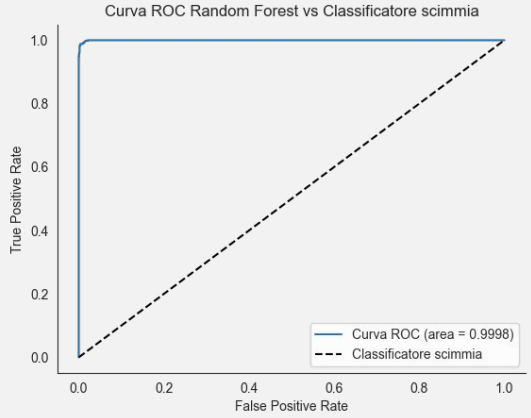
- **ROC-AUC** ⇒ Area sotto alla curva ROC. Valuta il modello nel complesso
- **Accuracy** ⇒ La proporzione di classificazioni corrette

$$\text{Accuracy} = \frac{\text{correct classifications}}{\text{total classifications}} = \frac{T_p + T_n}{T_p + T_n + F_p + F_n}$$

Valutazione

Abbiamo fatto una prima valutazione del modello utilizzando un testing set ottenuto dal subsampling iniziale, ecco i risultati:



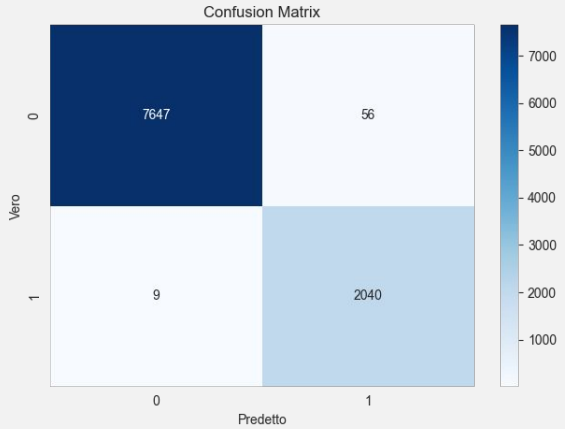


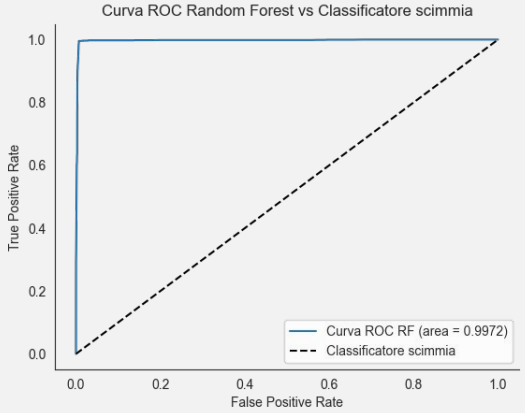
Positiva	Precision	Recall	F1-Score	Supporto
0	1.00	0.99	0.99	2027
1	0.98	0.99	0.98	675
Macro AVG	0.99	0.99	0.99	2702
Weighted AVG	0.99	0.99	0.99	2702

Accuracy = 0.99

Valutazione

Abbiamo fatto una seconda valutazione del modello utilizzando un set di testing messo a disposizione degli autori del dataset, per simulare il comportamento del modello costruito in uno scenario reale:





Positiva	Precision	Recall	F1-Score	Supporto
0	1.00	0.99	1.00	7703
1	0.97	1.00	0.98	2049
Macro AVG	0.99	0.99	0.99	9752
Weighted AVG	0.99	0.99	0.99	9752

Accuracy = 0.99


FINE