Progetto di Data Mining

Aldo Iannarelli, 602009 Vitalba Macaluso, 603160 Daniele Tribuzio, 602020

2019/2020

Indice

1 Comprensione dei dati

I dati forniti riguardano il rilevamento dell'occupazione di una stanza d'ufficio, suddivisi in 3 dataset (in ordine *datatraining*, *datatest* e *datatest2*) e ognuno dei quali costituiti da 7 attributi:

- **Date** *anno-mese-giorno ore:minuti:secondi*, indica il momento del rilevamento in un periodo compreso tra il 4 Febbraio 2015 e il 10 Febbraio dello stesso anno;
- **Temperature** *gradi Celsius*, indica la temperatura della stanza al momento del rilevamento;
- **Humidity** *relativa %*, indica il grado di umidità, ovvero il rapporto tra la quantità di vapore acqueo contenuta in una massa d'aria e la quantità massima che la stessa può contenere nelle medesime condizioni di temperatura e pressione;
- **Light** *Lux*, indica l'intensità di luce rilevata nella stanza;
- **CO2** *ppm:ppm*, indica la concentrazione di anidride carbonica nella stanza;
- **HumidityRatio** *kg(vapore acqueo)/kg(aria)*, quantità derivata dalla temperatura e dall'umidità relativa;
- **Occupancy** *binaria (0,1)*, indica se la stanza risulta occupata (1) o non occupata (0) al momento del rilevamento.

I dataset non presentano valori mancanti e presentano rispettivamente 8143, 2665 e 9752 records.
Il progetto verrà suddiviso in 3 Tasks, ognuna riguardante argomenti specifici.

1. Basic Classifiers and Evaluation
2. Advanced Classifiers and Evaluation
3. Time Series Analysis and Forecasting/Classification

L'analisi dei dati verrà svolta considerando il *datatest* come dataset di testing dei modelli, mentre il *datatest2* sarà preso in considerazione come dataset per la Task relativa al forecasting.

1.1 Distribuzione delle variabili e statistiche

La variabile target Occupancy risulta sbilanciata, anche se non fortemente, con 6414 records che assumono valore di Occupancy pari a 0 e 1729 records in cui tale variabile assume valore 1. La Figura 1.1 rappresenta la distribuzione della variabile.

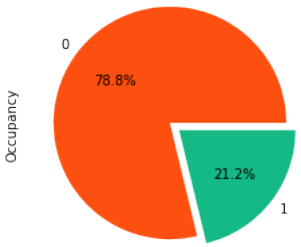

Figura 1.1: Distribuzione della variabile target

La temperatura minima della stanza, quando questa è occupata, risulta leggermente superiore alla stessa quando la stanza è invece vuota. La Figura 1.2 mostra che il 50% delle osservazioni con valore di Occupancy pari a 0 riporta una temperatura compresa tra i 19 e i 20.2 gradi, mentre il 50% delle osservazioni con valore di Occupancy pari a 1 riporta una temperatura compresa tra i 19.5 e i 21.8 gradi circa. Le temperature medie sono evidenziate dal marker '*'.

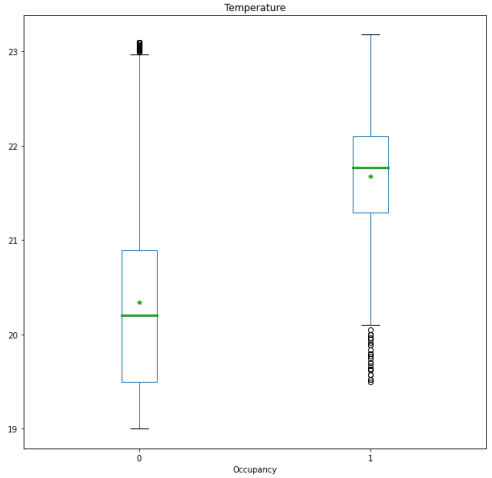

Figura 1.2: Distribuzione della variabile Temperature per Occupancy

La Figura 1.3 mostra che la concentrazione di CO2 raggiunge livelli maggiori quando la stanza d'ufficio è occupata, con una media di 1038 ppm:ppm. Quando la stanza è vuota, la concentrazione media è pari invece a circa 490 ppm:ppm, come indicato dalle linee sul grafico.

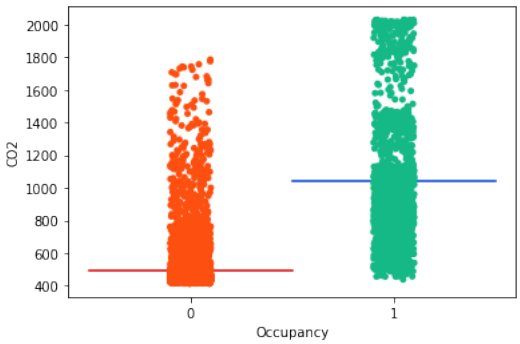

Figura 1.3: Distribuzione della variabile CO2 per Occupancy

Essendo la variabile HumidityRatio una quantità derivata dalla variabile Humidity, ci si aspetta che il loro andamento sia pressochè simile (seppure in un diverso ordine di grandezza) e che la loro correlazione sia elevata, come si vedrà in seguito. La Figura 1.4 dimostra questa ipotesi.

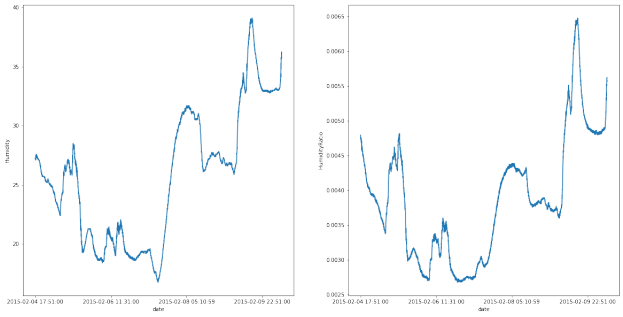

Figura 1.4: Andamento delle variabili Humidity (a sx) e HumidityRatio (a dx)

La variabile Light quando la stanza risulta occupata assume valori tra 31 e 744 Lux, con una media di circa 460 Lux. Quando la stanza è vuota, invece, il 75% delle osservazioni riporta un valore pari a 0 Lux, con una media di 27 Lux. La Figura 1.5 mostra che la maggior parte dei valori assunti dalla variabile Light quando la stanza è occupata risulta superiore agli stessi quando invece la stanza è vuota. Questo porta a ipotizzare che tale variabile sarebbe da considerarsi una buona variabile di classificazione per la variabile target. Dalla rappresentazione grafica, inoltre, si possono osservare due valori al di sopra 1400 Lux per Occupancy = 0; questi sono da ritenersi possibili outliers.

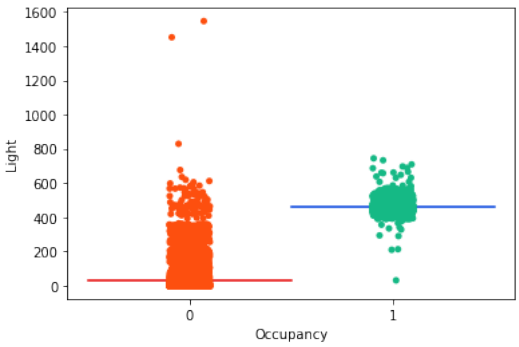

Figura 1.5: Distribuzione della variabile Light per Occupancy

1.2 Correlazione tra le variabili

La variabile Temperature risulta totalmente incorrelata alla variabile Humidity e mediamente correlata con le variabili Light e CO2. La variabile Humidity, come si era supposto, è quasi totalmente correlata alla variabile HumidityRatio. Si ritiene quindi che HumidityRatio sia ridondante e, nella fase di preparazione del dataset verrà eliminata.
La scelta di eliminare la variabile HumidityRatio è dettata dal fatto che quest'ultima assume valori di un ordine di grandezza decisamente inferiore rispetto alle altre.
Infine, si può notare un'alta correlazione tra la variabile Light e la variabile target Occupancy. Come si era già ipotizzato in precedenza, la variabile Light potrebbe essere una buona variabile da tenere in considerazione per la classificazione. Nella Figura 1.6 è mostrata la Heat map delle correlazioni, in cui a colori più chiari sono associate correlazioni più elevate.

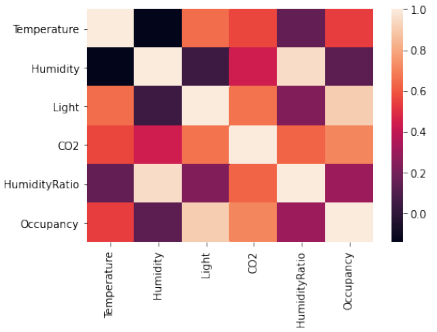

Figura 1.6: Heat map della correlazione tra le variabili

2 Preparazione dei dati

2.1 Variabile *Date*

Le Tasks 1 e 2 hanno a che vedere con la classificazione. In questa fase, quindi, si sceglie di non tenere in considerazione la dimensione temporale dei dati escludendo la variabile Date. Al fine di non perdere del tutto le informazioni derivate da tale variabile, si è scelto di creare due nuove variabili: Day

e Hour. Ciò che ci si apetta è che ci siano delle sostanziali differenze nei valori assunti dalle altre variabili contenute del dataset, tenendo conto del fatto che in alcuni giorni (ad esempio, il fine settimana) e in alcune fasce orarie (ad esempio, la pausa pranzo o le ore notturne) la stanza d'ufficio ci si aspetta risulti non occupata. A titolo esemplificativo si rappresentano le distribuzioni delle variabili CO2 e Light distinte per le nuove variabili create Day e Hour,con ulteriore distinzione per Occupancy.
La Figura 2.1 mostra (a sx) che nei giorni non lavorativi, ovvero sabato e domenica, la stanza d'ufficio non è occupata e il livello di concentrazione di CO2 è molto più basso rispetto agli altri giorni. Allo stesso modo, la Figura mostra (a dx) che negli orari di chiusura dell'ufficio il livelli di concentrazione di CO2 sono più bassi.

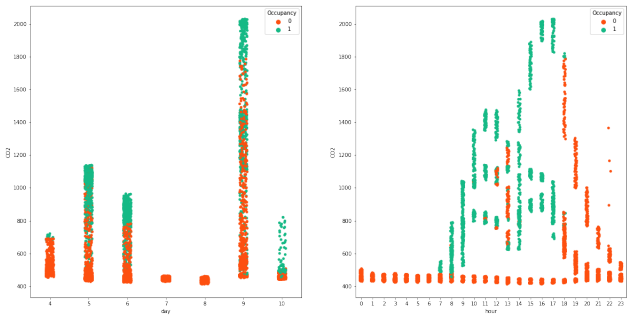

Figura 2.1: Distribuzione della variabile CO2 per Day/Hour e Occupancy

La Figura 2.2 evidenzia come la variabile Light assuma valori maggiori quando la stanza è occupata e quindi non nei giorni 7 e 8 Febbraio (rispettivamente, sabato e domenica). Lo stesso vale per gli orari di aperutra dell'ufficio.

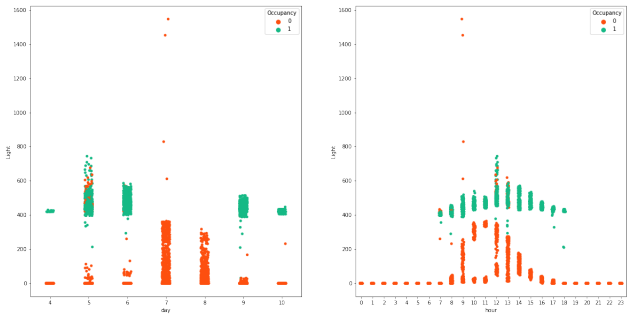

Figura 2.2: Distribuzione della variabile Light per Day/Hour e Occupancy

2.2 Variabile *HumidityRatio*

La variabile HumidityRatio risulta ridondante e pertanto si è scelto di escluderla dall'insieme delle variabili da utilizzare per svolgere le Tasks. Al posto di questa variabile, si utilizzerà una nuova variabile che chiameremo *HumidexIndex*. L'indice Humidex è una misura utilizzata in meteorologia per indicare il grado di benessere/malessere di una persona media in relazione alla temperatura e all'umidità. La formula mediante la quale di ottiene il valore di Humidex è la seguente:

$$Humidex = T + \frac{5}{9} * (0.0611 * RH * 10^{0.03T} - 10) \qquad (1)$$

dove T rappresenta la temperatura e RH l'umidità relativa.
In base ai valori ottenuti per l'indice Humidex, il grado di benessere/malessere sarà indicato come nella Tabella 2.1.

Humidex	**Categoria**
H < 27	Benessere
27 <= H < 30	Leggero disagio
30 <= H < 40	Disagio
40 <= H < 55	Grande disagio
H >= 55	Pericolo

Tabella 2.1: Conversione dell'indice Humidex in categorie

La Figura 2.3 mette in evidenza come il livello minimo riscontrato della variabile HumidexIndex sia 'Disagio' e che quest'ultimo si ha soltanto quando la stanza d'ufficio non è occupata. Quando la stanza d'ufficio è occupata, la sensazione di possibile malessere aumenta fino a raggiungere livelli di 'Pericolo'. Il fatto che la sensazione di 'Disagio' si riscontri soltanto quando la stanza d'ufficio è libera, potrebbe essere d'aiuto durante la classificazione.

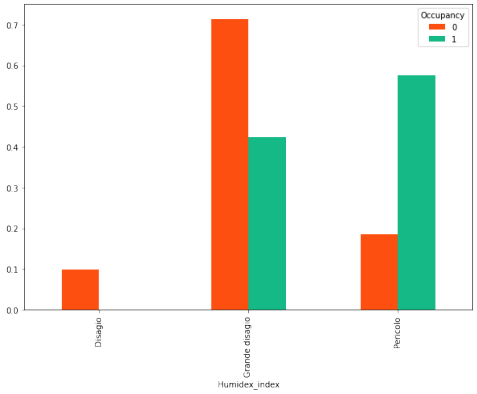

Figura 2.3: Distribuzione della variabile HumidexIndex per Occupancy

Infine, per facilitare successivamente la classificazione, la variabile HumidexIndex verrà binarizzata, formando così 3 variabili binarie chiamate 'Disagio', 'Grande disagio' e 'Pericolo'.

3 Task 1 - Basic Classifiers and Evaluation

3.1 K - Nearest Neighbors

Per la stima dei parametri del classificatore K-NN è stato utilizzato l'approccio esaustivo GridSearchCV, considerando quindi tutte le combinazioni degli iperparametri da valutare. La distanza tra le coppie di punti sono state calcolate usando la distanza Euclidea. Le performance degli iperparametri e del modello selezionati sono state valutate tenendo conto dell'accuracy e misurate su un set di validazione di ampiezza 10, secondo il metodo 10-fold Cross Validation. I parametri investigati sono stati:

- il numero k di vicini, per valori di k da 1 a $\sqrt{N}$ (convenzionalmente k viene fissato pari a quest'ultimo valore);
- il peso *weights* attribuito ai punti di ogni vicinato, scelto tra uniforme e inversamente proporzionale alla distanza tra ogni coppia di punti.

La Figura 3.1 rappresenta il grado di accuracy per diversi valori di k dei migliori modelli selezionati per *weights* = *'uniform'* e *weights* = *'distance'*. Per quanto riguarda il modello con peso uguale attribuito ai punti, si ottiene accuracy pari a 0.977 ponendo k = 32. Il modello in cui il peso attribuito ai punti è inversamente proporzionale alla distanza permette, invece, di avere un'accuracy massima pari a 0.975 ponendo k = 38.

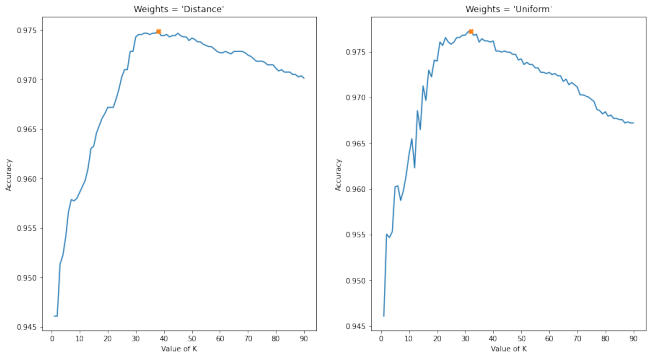

Figura 3.1: Livello di accuracy per diversi valori di k

Considerato che l'accuracy relativa ai due modelli è pressocchè simile, si opta per il modello più semplice, ovvero quello per cui il valore di k è minore.

La Figura 3.2 mostra il color scatterplot con la variabile Light sull'asse x e la variabile Temperature sull'asse y. Il colore verde indica l'area del grafico in cui la stanza viene classificata come 'Occupata', mentre il colore arancione quella in cui la stanza viene classificata come 'Non occupata'. Dalla rappresentazione grafica emerge come la variabile Light sia una buona variabile di classificazione.

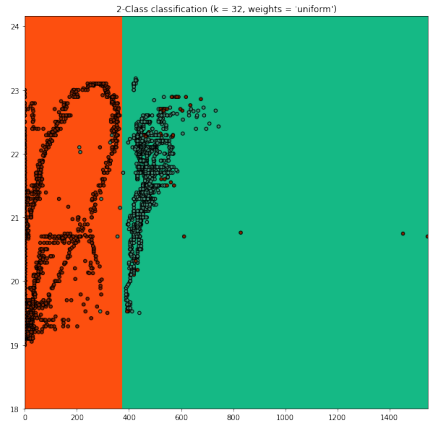

Figura 3.2: Color plot per K-NN con weights = 'uniform'

La Tabella 3.1 sintetizza le performance del modello scelto sul test set. Ogni record classificato come Occupancy = 0 appartiene a questa classe (precisione = 100%) e il 97% dei record che effettivamente appartiene alla classe Occupancy = 0 è stato classificato come appartenente ad essa (recall = 97%). Il modello riesce a classificare correttamente il 98% dei record presenti nel test set.

	precision	recall	f1-score
0	1.00	0.97	0.98
1	0.95	1.00	0.97
accuracy			0.98

Tabella 3.1: Performance del modello K-NN sul test set

L'AUC in Figura 3.3 è pari al 98%. Ciò significa che il modello è in grado di predire abbastanza bene quando la stanza d'ufficio è occupata o no.

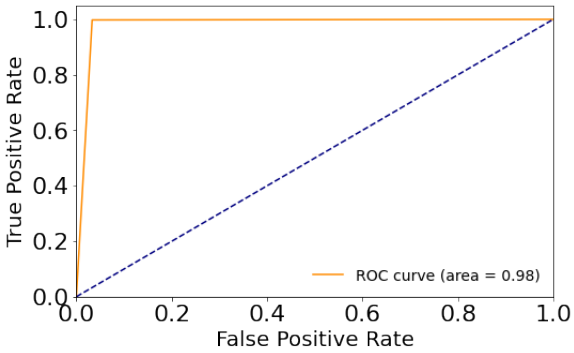

Figura 3.3: ROC curve - K-NN

Il test set contiene il 36.5% di record con Occupancy = 1. La Figura 3.4 dimostra che il modello ottenuto è ottimo in quanto, osservando la curva per la classe 1, si nota che oltre il 36% circa dell'ampiezza del campione il tasso di veri positivi rimane pari a 1 fino alla fine della classificazione. Lo stesso discorso vale per il 63.5% di record con Occupancy = 1.

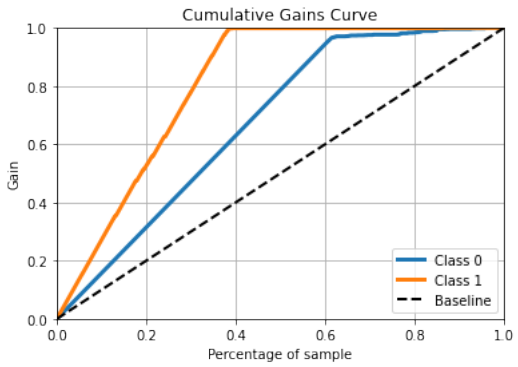

Figura 3.4: Lift chart - K-NN

3.2 Naive Bayes

Gli attributi dei training e test set sono stati suddivisi in:

- **numerici** - Temperature, Humidity, Light e CO2;
- **categoriali** - Day, Hour, Disagio, Grande disagio e Pericolo.

Sul training set costituito dalle variabili numeriche è stato implementato l'algoritmo Naive Bayes gaussiano per la classificazione, assumendo pertanto che la distribuzione della probabilità condizionata sia di tipo normale. Sul training set costituito dalle variabili categoriali è stato invece implementato l'agoritmo Naive Bayes categoriale per la classificazione, assumendo un valore di $\alpha = 1$ per il parametro di smoothing Laplaciano. Per entrambi gli algoritmi non è stata definita la probabilità a priori delle classi, ma per il CategoricalNB è stato chiesto che questa venisse imparata durante il training. Nella Tabella 3.2 sono presentati gli stimatori media e varianza per il modello GaussianNB, distinti per Occupancy = 0 e Occupancy = 1.

Attributo	**Mean_0**	**Sigma_0**	**Mean_1**	**Sigma_1**
Temperature	20.33	0.83	21.67	0.39
Humidity	25.35	28.03	27.15	37.54
Light	27.78	8026.67	459.85	1787.14
CO2	490.32	23380.76	1037.70	142501.77

Tabella 3.2: Stimatori media e varianza del modello guassiano

I modelli sul training set con gli attributi numerici e categoriali presentano accuracy rispettivamente pari a 0.978 e 0.941.

Le Tabelle 3.3 e 3.4 sintetizzano le performance dei modelli sul test set.

	precision	recall	f1-score
0	1.00	0.97	0.98
1	0.95	0.99	0.97
accuracy			0.98

Tabella 3.3: Performance del modello Gaussian Naive Bayes sul test set

Il modello basato sul training set con i soli dati categoriali presenta delle performance inferiori sul test set. L'accuracy è infatti pari a 0.89 e l'84% dei record che effettivamente appartiene alla classe Occupancy = 1 è stato classificato come appartenente ad essa.

	precision	recall	f1-score
0	0.91	0.92	0.92
1	0.86	0.84	0.85
accuracy			0.89

Tabella 3.4: Performance del modello Categorical Naive Bayes sul test set

L'AUC in Figura 3.5 è pari al 98%: il modello sul test set con attributi numerica è in grado di distinguere quasi perfettamente quando la stanza d'ufficio è occupata o no.

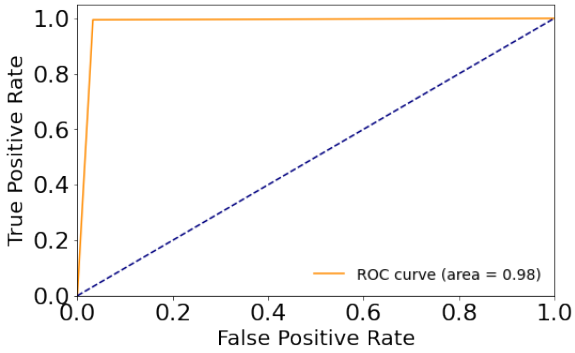

Figura 3.5: ROC curve - Gaussian Naive Bayes

L'AUC in Figura 3.6 indica, invece, che il modello è in grado di distinguere abbastanza bene le due classi, anche se in maniera leggermente peggiore rispetto al Gaussian Naive Bayes.

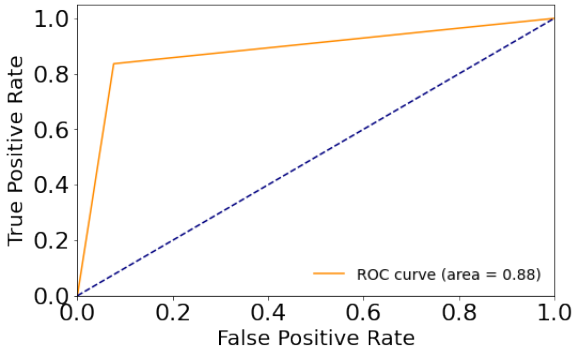

Figura 3.6: ROC curve - Categorical Naive Bayes

Il test set contiene il 63.5% di record con Occupancy = 0. La Figura 3.7 dimostra che il modello ottenuto è ottimo in quanto, osservando la curva per la classe 1, si nota che oltre il 63% circa dell'ampiezza del campione il tasso di veri positivi rimane pari a 1 fino alla fine della classificazione. Lo stesso vale per i record per cui Occupancy = 0.

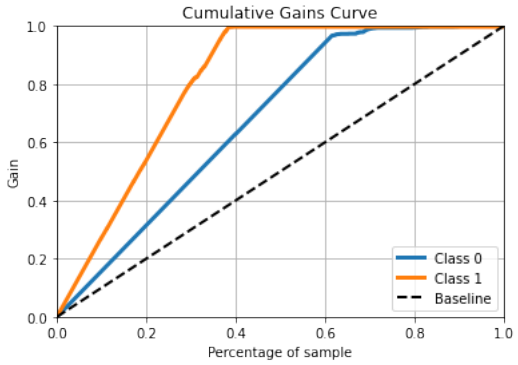

Figura 3.7: Lift chart - Gaussian Naive Bayes

La curva Cumulative Gain in Figura 3.8 conferma che il modello che tiene conto degli attributi categoriali non è ottimale in quanto alcuni record con valore di Occupancy pari a 0 vengono classificati erroneamente nella classe 1 e alcuni record con valori di Occupancy pari a 1 vengono classificati erroneamente nella classe 0.

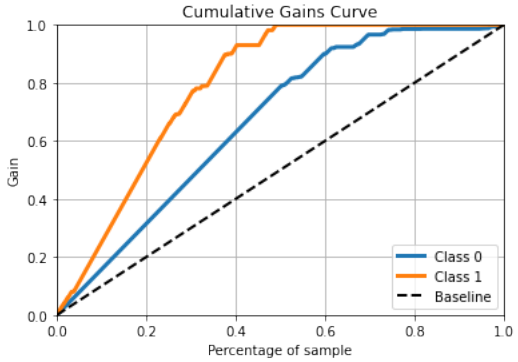

Figura 3.8: Lift chart - Categorical Naive Bayes

3.3 Logistic Regression

Per la stima dei parametri della regressione logistica è stato utilizzato l'approccio CV mediante l'algoritmo LogisticRegressionCV, che implementa la regressione logistica con il supporto integrato della Cross Validation. Le performance degli iperparametri e del modello selezionati sono state valutate tenendo conto dell'accuracy e misurate su un set di validazione di ampiezza 10, secondo il metodo 10-fold Cross Validation. Al fine di semplificare tale operazione, sono state selezionate soltanto le variabili numeriche e standardizzati i loro valori per avere media nulla e deviazione standard pari a 1. Inoltre, è stata lasciata la regolarizzazione di default l_2 che lavora molto bene per evitare situazioni di overfitting e il solver di default *lbfgs* che permette di trovare più velocemente la convergenza.

Il modello di regressione logistica ottenuto sul training set è il seguente:

$$expit(x) = \frac{1}{1 + e^{-(\beta_0 + \beta_j * X_j)}} \qquad (2)$$

Nella Tabella 3.5 sono inseriti i coefficienti di regressione e l'intercetta del modello.

Attributo	**Coefficiente**
Temperature	-0.62392622
Humidity	-0.04364801
Light	3.04521523
CO2	1.31450001
Intercetta	-3.42948643

Tabella 3.5: Coefficienti di regressione logistica

Il modello logistico sul training set ha accuracy pari a 0.988. Nel grafico in Figura 3.9 è rappresentato la funzione logistica sigmoidale considerando la variabile target in funzione della variabile Light.

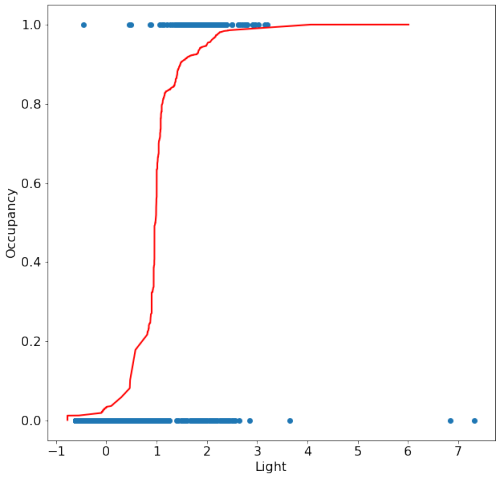

Figura 3.9: Expit Light - Occupancy

L'87% dei record classificati come Occupancy = 0 appartiene a questa classe, ma il 99% dei record che effettivamente appartiene alla classe Occupancy = 0 è stato classificato come appartenente ad essa. Il modello ha una precisione maggiore per la classe 1, ma una misura di recall del 0.73. Ciò significa che il 73% dei record che effettivamente appartiene alla classe 1 è stato correttamente classificato. L'accuracy del modello sul test set è inferiore all'accuracy del modello sul set di training.(Tabella 3.6)

	precision	recall	f1-score
0	0.87	0.99	0.92
1	0.98	0.73	0.84
accuracy			0.90

Tabella 3.6: Performance del modello di regressione logistica sul test set

L'AUC relativa alla regressione logistica (Figura 3.10) indica che il modello distingue bene tra le due classi, anche se al momento questo modello risulta il peggiore.

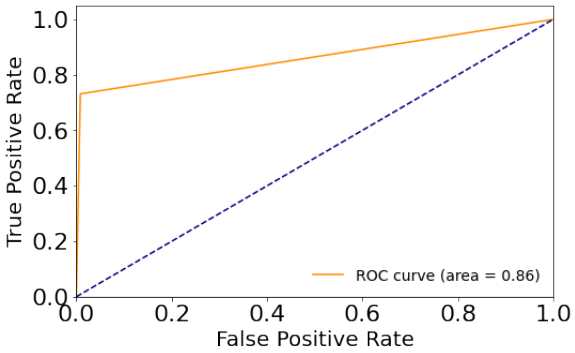

Figura 3.10: ROC curve - Logistic Regression

In termini di Cumulative Gain (Figura 3.11), il modello di regressione logistica sembra ottimo, ma, date le precedenti considerazioni, si ritiene che nn sia il migliore modello per i dati in analisi.

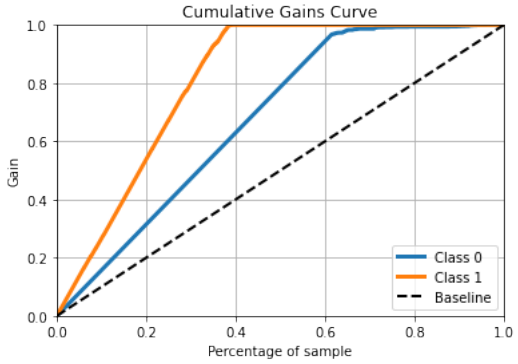

Figura 3.11: Lift chart - Logistic Regression

3.4 Decision Tree

Anche per la stima dei parametri del classificatore Decision Tree è stato utilizzato l'approccio esaustivo GridSearchCV. Le performance degli iperparametri e del modello selezionati sono state valutate tenendo conto dell'accuracy e misurate su un set di validazione di ampiezza 5, secondo il metodo 5-fold Cross Validation. Inoltre, il peso attribuito alle classi è stato bilanciato considerando le occorrenze delle classi della variabile target Occupancy. I parametri investigati sono stati:

- le misure di impurità *criterion*, selezionando sia l'indice Gini che l'entropia;
- la massima ampiezza dell'albero *max_ depth*, investigata in valori tra 1 e 9;
- il minimo numero di record richiesti per dividere un nodo interno *min_ sample_ split*, selezionati tra 2, 5, 10, 20, 50, 100, 150 e 200;
- il numero minimo di record richiesti in un nodo foglia *min_ sample_ leaf*, selezionati tra 10, 50, 100, 150 e 200.

Il miglior modello sui dati di training ha accuracy pari a 0.993 ed è risultato avere i seguenti parametri: *criterion* = 'entropy', *max_ depth* = 4, *min_ sample_ split* = 2, *min_ sample_ leaf* = 50.
L'attributo maggiormente importante nella costruzione del Decision Tree è Light (importance = 0.965), seguito da Humidity (importance = 0.013) e Hour (importance = 0.008). In Figura 3.14 è raffigurato l'albero decisionale ottenuto.

La Tabella 3.7 sintetizza le performance del modello scelto sul test set. Il 93% dei record classificati come Occupancy = 0 appartiene a questa classe e il 97% dei record che effettivamente appartiene alla classe Occupancy = 0 è stato classificato come appartenente ad essa (recall = 97%). La precisione del modello per la classe 1 è lievemente maggiore (precision = 95%), ma la misura di recall è pari a 0.87. In generale, il modello sul test set presenta accuracy pari a 0.93.

	precision	recall	f1-score
0	0.93	0.97	0.95
1	0.95	0.87	0.90
accuracy			0.93

Tabella 3.7: Performance del modello Decision Tree sul test set

L'AUC in Figura 3.12 è pari al 92%. Ciò significa che il modello è in grado di predire quando la stanza d'ufficio è occupata o no meglio del modello di regressione logistica.

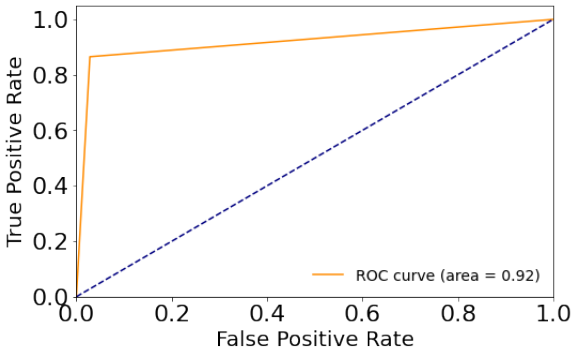

Figura 3.12: ROC curve - Decision Tree

La curva Cumulative Gain in Figura 3.13 indica che il modello è ottimale, considerando che il 36.5% e il 63.5% dei test record appartengono rispettivamente alle classi Occupancy = 1 e Occupancy = 1. Nonostante ciò, le altre misure di performance non confermano che il Decision Tree sia il miglior modello tra quelli presi in esame.

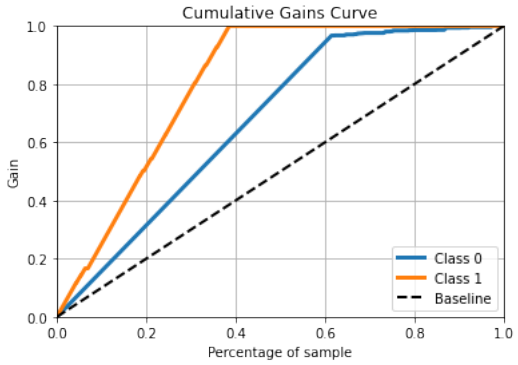

Figura 3.13: Lift chart - Decision Tree

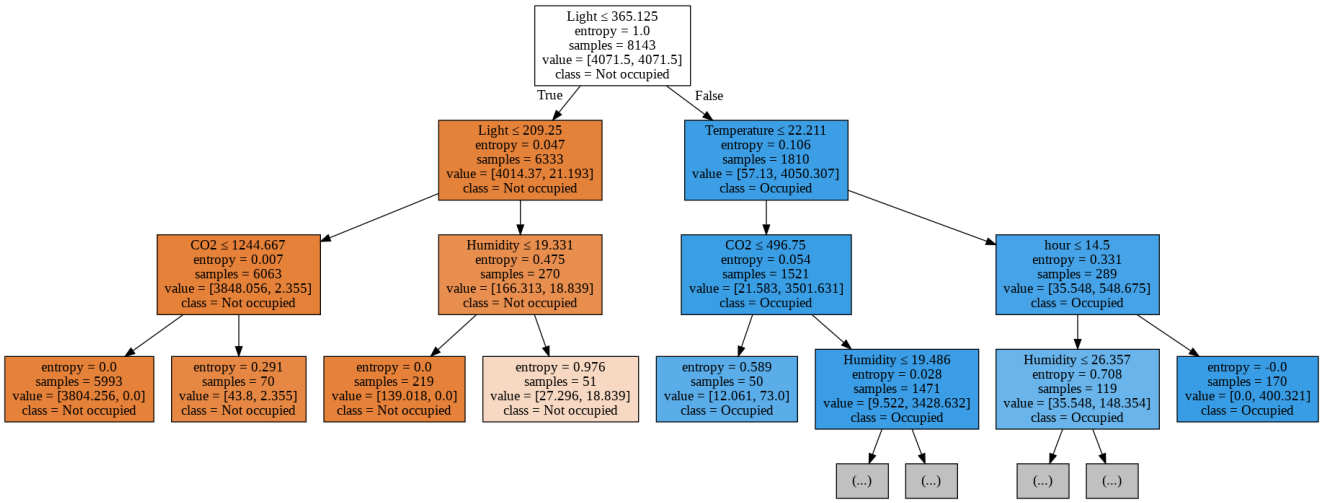

Figura 3.14: Decision Tree

3.5 Riduzione della dimensionalità

In questa fase si cercherà di ridurre la dimensionalità del dataset considerando tutti i classificatori analizzati finora, ponendo particolare attenzione su quelli che presentano performance migliori a seguito della riduzione. I metodi utilizzati per ridurre la dimensionalità saranno i seguenti:

- **Feature Selection:** Variance Threshold, Univariate Feature Selection usando l'algoritmo SelectKBest e ANOVA test usando f_classif;
- **Feature Projection:** PCA.

3.5.1 Variance Threshold

La soglia della varianza è stata posta pari a 0.2. Per gli algoritmi K-NN e Decision Tree, che sono stati implementati inizialmente sul dataset completo, la dimensione del training set si è ridotta da 9 a 7 attributi senza alterare però in alcun modo le performance dei modelli sul test set. Per l'algoritmo Logistic Regression, implementato inizialmente sul dataset contenente i soli attributi numerici standardizzati, qualunque soglia della varianza da 0 a 0.9 non riduce ulteriormente la dimensionalità del dataset e non influisce, di conseguenza, sulle performance del modello sul test set. Analogamente a ciò che accade per la Logistic Regression, non vi è alcuna riduzione della dimensionalità del training set utilizzato per implemetare l'algoritmo GaussianNB e le performance del modello sul test set rimangono invariate. Diversa è, invece, la situazione riscontrata nell'implementazione del CategoricalNB. La soglia della varianza di 0.2 riduce la dimensionalità del dataset di training da 5 a 3 e ciò comporta un miglioramento delle performance del modello sul test set ridotto: la Tabella 3.8 mostra le performance di quest'ultimo modello sul dataset con dimensionalità ridotta.

	precision	recall	f1-score
0	0.89	0.97	0.92
1	0.93	0.78	0.85
accuracy			0.90

Tabella 3.8: Performance del modello CategoricalNB sul test set ridotto

La misura di precisione aumenta da 0.86 a 0.93 per la classe Occupancy = 1, a discapito della precisione per la

classe Occupancy = 0 che diminuisce di 0.02. Al contrario, la misura di recall aumenta per la classe 0 e diminuisce per la classe 1, lasciando però invariato l'F1-score. L'accuracy del modello aumenta lievemente da 0.89 a 0.90.
I risultati ottenuti su tutti i classificatori fanno pensare che, sia nel K-NN che nel Decision Tree, i due attributi non considerati a seguito della riduzione della dimensionalità siano proprio i due attributi categoriali esclusi nel CategoricalNB. Inoltre, considerata la differenza delle performance del modello CategoricalNB prima e dopo la dimensionality reduction, si ritiene che analoghi cambiamenti non siano stati riscontrati rispettivamente in K-NN e Decision Tree perchè maggiormente influenzati dagli attributi numerici.

3.5.2 Univariate Feature Selection

Il numero k di attributi da selezionare è stato sperimentato dal numero massimo al numero minimo per ogni classificatore analizzato. Di conseguenza:

- Per K-NN e DecisionTree, k è stato impostato ricorsivamente da 9 a 1;
- Per GaussianNB e LogisticRegression, k è stato impostato ricorsivamente da 4 a 1;
- Per CategoricalNB, k da 5 a 1.

L'algoritmo K-NN non varia le proprie performance sul test set per i diversi valori di k considerati. Inoltre, per k = 1, l'attributo rimasto è Light. Ciò significherebbe che tale variabile è la sola veramente importante nel modello e che le performance di quest'ultimo dipendano proprio da essa. Analogamente, ciò avviene anche considerando l'algoritmo GaussianNB.
Utilizzando l'algoritmo SelectKBest, il classificatore CategoricalNB presenta le performance migliori se la dimensione del training set non viene ridotta e quindi per k = 5.
L'algoritmo LogisticRegression presenta invece delle performance lievemente migliori sul test set per la misura di recall relativa alla classe 1 e l'F1-score relativo alla classe 0 (Figura 3.9) se k viene posto pari a 2: in questo caso, il SelectKBest considera gli attributi Light e CO2.

	precision	recall	f1-score
0	0.87	0.99	0.93
1	0.98	0.74	0.84
accuracy			0.90

Tabella 3.9: Performance del modello LogisticRegression sul test set ridotto

Il Decision Tree presenta delle performance nettamente migliori a partire da k = 8. Tali performance rimangono invariate considerando tutti i valori di k compresi tra 8 e 1. Anche in questo caso, l'attributo che influisce maggiormente sul modello è Light e la Tabella 3.10 sintetizza ciò che è stato riscontrato, che vale per k = [8,7,6,5,4,3,2,1].

	precision	recall	f1-score
0	1.00	0.97	0.98
1	0.95	1.00	0.97
accuracy			0.98

Tabella 3.10: Performance del modello DecisionTree sul test set ridotto

3.5.3 ANOVA test

L'ANOVA test è stato effettuato sul training set con i soli attributi numerici. Mediante l'utilizzo dell'algoritmo f_classif e per k = 4 (numero di attributi numerici nel dataset), è stato ottenuto il grafico in Figura 3.15.

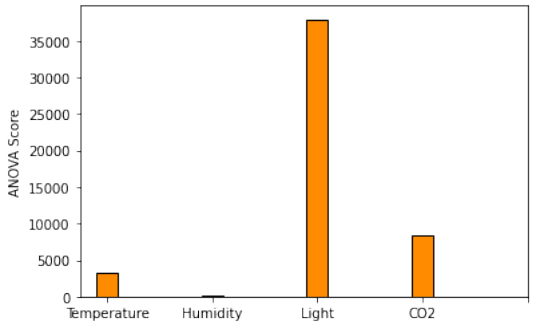

Figura 3.15: Score dell'ANOVA test per gli attributi numerici

L'attributo Light risulta essere il più significativo e il più importante all'interno del dataset, ovvero quello che spiega la maggior parte della variabilità dei dati, seguito in ordine da CO2 e Temperature. Questo risultato è in linea con i risultati ottenuti in precedenza. Infatti, in tutti i metodi di riduzione della dimensionalità del dataset, l'unico attributo che persisteva era proprio Light e, quando k era scelto pari a 2, il secondo attributo era risultato proprio CO2.

3.5.4 Principal Component Analysis

Per l'analisi delle componenti principali è stato utilizzato il costructor *make_pipeline* dalla libreria sklearn in cui è stato costruito il modello sia sui dati originari che sui dati standardizzati, utilizzando anche in questo caso una cross-validation. A prescindere dal classificatore utilizzato, la rappresentazione grafica del training set dopo la PCA è la stessa e varia soltanto se i dati sono o no standardizzati. Guardando l'accuracy delle previsioni per il test set e gli eigenvalues (Tabella 3.11), la PCA con 2 componenti sembra funzionare meglio sui dati non standardizzati. Le due componenti principali sui dati originari, infatti, riescono a catturare quasi il 100% della variabilità dei dati (0.9998); mentre le 2 componenti principali sui dati standardizzati riescono a catturarne circa l'87% (0.8695).

	Dati originari	Dati standardizzati
Firt PC	0.87122206	0.57040612
Second PC	0.12861901	0.29909627
Accuracy		
K-NN	97.82%	89.61%
GaussianNB	94.82%	93.06%
LogisticRegression	-	86.57%
DecisionTree	97.67%	92.46%

Tabella 3.11: Accuracy delle previsioni per il test set con PCA

La Figura 3.16 mostra il grafico del training set per i dati originari e standardizzati dopo la PCA considerando la prima e la seconda componente principale.
I modelli migliori presentano adesso i seguenti parametri:

- **K-NN** sui dati originari: *n_neighbors*= 31, *weights*= 'uniform'; sui dati standardizzati: *n_neighbors*= 83, *weights*= 'uniform';
- **GaussianNB**: sui dati originari: *sigma_0*= [21506.19, 9905.82], *theta_0*= [-145.00, -30.03], *sigma_1*= [113980.94, 30333.69], *theta_1*= [537.89, 111.38];
- **Logistic Regression** sui dati standardizzati: *coef*= [0.955, 0.087], *intercept*= -1.773;
- **Decision Tree**: sui dati originari: *criterion*= 'gini', *max_depth*= 3, *min_samples_leaf*= 50, *min_samples_split*= 2; sui dati standardizzati: *criterion*= 'gini', *max_depth*= 1, *min_samples_leaf*= 10, *min_samples_split*= 2.

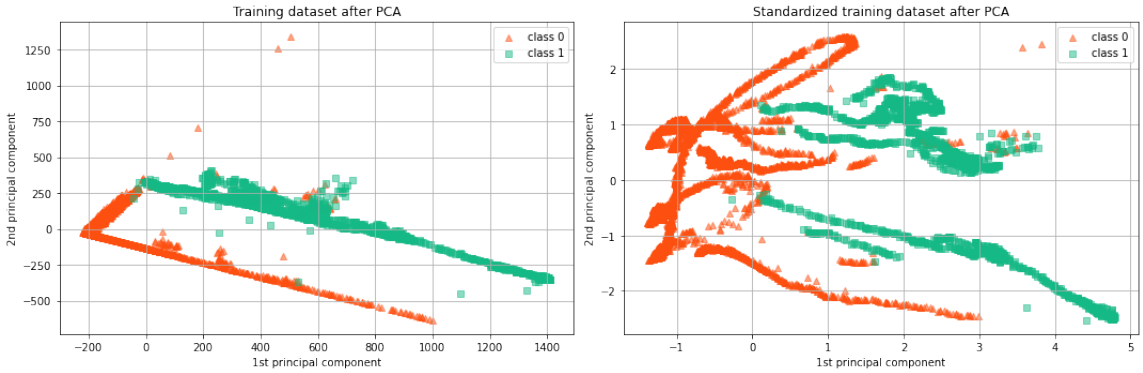

Figura 3.16: Plot del training set dopo la PCA

3.6 Dati fortemente sbilanciati

Il training set originario è composto da 6414 record con valore di Occupancy = 0 e 1729 record con valore di Occupancy = 1, rispettivamente il 78.8% e il 21.2% circa. Al fine di ottenere un dataset fortemente sbilanciato con il 98% di record appartenenti alla classe 0 e il 2% di record appartenente alla classe 1, mediante proporzione è risultato che il numero di record appartenenti alla classe Occupancy = 1 dovesse essere pari a 162 (1567 in meno). Il dataset così composto è stato ottenuto selezionando in maniera del tutto casuale i record da eliminare. Per far sì che la selezione casuale rimanesse sempre la stessa è stato posto *np.random.seed(18)*. Successivamente, sono stati costruiti i modelli di classificazione utilizzando il training set con le classi fortemente sbilanciate.
Inoltre, è stato posto, ove necessario, *random_state = 42*, in modo tale da poter lavorare sempre sullo stesso subset casuale. Di seguito, si riportano i valori di accuracy e AUC dei modelli sul test set:

- **K-NN:** accuracy = 0.87 e AUC = 0.910;
- **GaussianNB:** accuracy = 0.98 e AUC = 0.989;
- **CategoricalNB:** accuracy = 0.88 e AUC = 0.909;
- **LogisticRegression:** accuracy = 0.64 e AUC = 0.959;
- **DecisionTree:** accuracy = 0.89 e AUC = 0.855.

3.6.1 Adjust the Decision Threshold

La soglia, come da convenzione, è stata posta pari a 0.50. Utilizzando questa tecnica per l'algoritmo K-NN, il risultato è che l'accuracy del modello è diminuita da 0.87 a 0.81 mentre l'AUC è rimasta invariata. Per i modelli GaussianNB e Logistic Regression, i risultati sulla ROC curve e dell'accuracy sono rimasti invariati, eccetto l'accuracy del GaussianNB che è scesa a 0.97. Il Decision Tree risulta invariato i termini di accuracy, ma si nota un lieve aumento in termini di AUC, che adesso è pari a 0.865. Le performance dei modelli sono sintetizzate nella Tabella 3.12.

	Accuracy	AUC
K-NN	0.81	0.910
GaussianNB	0.97	0.989
Logistic Regression	0.64	0.959
Decision Tree	0.89	0.865

Tabella 3.12: Performance dei modelli usando Adjust Decision Threshold

In sintesi, la tecnica di bilanciamento qui considerata non mostra effetti significativi sull'AUC. Diversamente, questa incide maggiormente e negativamente sull'accuracy del classificatore K-NN.

3.6.2 Random UnderSampling

La classe con frequenza maggiore è stata ridotta ad una numerosità di 162 (la stessa numerosità della classe minore) estraendo record in maniera casuale senza ripetizioni.
Utilizzando questa tecnica per l'algoritmo K-NN, il risultato è che l'accuracy del modello aumenta notevolmente da 0.87 a 0.96, così come l'AUC, che raggiunge un valore pari a 0.979. I risultati sulla ROC curve e dell'accuracy del modello GaussianNB sul test set diminuiscono lievemente rispettivamente a 0.988 e 0.96. La Logistic Regression mostra, invece, un'accuracy lievemente maggiore (0.65) e anche l'AUC migliora, di circa il 3%. Per quanto riguarda il Decision Tree, la Random Undersampling ha migliorato le performance del modello: l'accuracy è pari a 0.92, mentre l'AUC adesso è intorno al 90%, come mostrato nella Tabella 3.13.

	Accuracy	**AUC**
K-NN	0.96	0.979
GaussianNB	0.96	0.988
Logistic Regression	0.65	0.993
Decision Tree	0.92	0.897

Tabella 3.13: Performance dei modelli usando Random UnderSampling

La tecnica Random Undersampling migliora notevolmente le performance dei modelli K-NN, in termini di accuracy, e Decision Tree, per la misura AUC. Miglioramenti lievi si hanno anche per il modello Logistic Regression, mentre non se ne riscontrano per il GaussianNB.

3.6.3 Condensed Nearest Neighbor

Mediante la tecnica CondensedNearestNeighbor sono stati individuati 162 record appartenenti alla classe Occupancy = 1 e rispettivamente per Occupancy = 0:

- 42 record sui dati originari;
- 43 record sul dataset con i soli attributi numerici, utilizzato per l'implementazione del GaussianNB;
- 26 record sui dati standardizzati, utilizzati per l'implemetazione dell'algoritmo Logistic Regression.

La Tabella 3.14 mostra le performance dei modelli dopo l'utilizzo della tecnica presa in esame. L'accuracy del modello K-NN sul test set diminuisce del 3% ma si riscontra un miglioramento del 2.4% in termini di AUC. Il modello GaussianNB presenta, invece, un'accuracy molto più bassa (di circa il 9%), anche se questo notevole peggioramento non si riscontra per l'AUC dal momento che la sua diminuzione è al di sotto dell'1%. Il modello Logistic Regression risultava inizialmente il più penalizzato, in termini di accuracy, dal forte sbilanciamento dei dati ma dopo l'utilizzo del Condensed Nearest Neighbor l'AUC aumenta da 0.959 a 0.975, mentre l'accuracy aumenta da 0.64 a 0.90. Per il modello Decisoion Tree si riscontra, infine, una diminuzione dell' accuracy dal 90% all'86%.

	Accuracy	**AUC**
K-NN	0.84	0.934
GaussianNB	0.89	0.983
Logistic Regression	0.90	0.975
Decision Tree	0.86	0.855

Tabella 3.14: Performance dei modelli usando Condensed Nearest Neighbor

La tecnica di bilanciamento considerata non permette di ottenere notevoli incrementi nelle performance dei modelli per quanto riguarda l'AUC. Tutti i modelli, eccetto la Logistic Regression, risultano piuttosto penalizzati, in modo particolare il GaussianNB (che vede una diminuzione del 9% dell'accuracy).

3.6.4 Random OverSampling

La classe con frequenza minore è stata sovra-campionata ad una numerosità di 6414 (la stessa numerosità della classe maggiore) estraendo record in maniera casuale con ripetizioni.
Utilizzando questa tecnica di bilanciamento per l'algoritmo K-NN, l'accuracy del modello aumenta da 0.87 a 0.90, mentre l'AUC mostrata rimane invariata, come mostrato nella Tabella 3.15. Per i modelli GaussianNB e Logistic Regression, le accuracy dei modelli sul test set non variano; mentre per il modello Logistic Regression si riscontra, piuttosto, un miglioramento del 3% sull'AUC. Il Decision Tree è l'unico algoritmo penalizzato da questa tecnica di bilanciamento: sia l'accuracy che l'AUC infatti diminuiscono del 2% circa.

	Accuracy	**AUC**
K-NN	0.90	0.910
GaussianNB	0.98	0.989
Logistic Regression	0.64	0.990
Decision Tree	0.88	0.835

Tabella 3.15: Performance dei modelli usando Random OverSampling

In definitiva, si ritiene che la tecnica Random Oversampling sia una buona tecnica di bilanciamento per il modello K-NN. Le perfomance dei modelli GaussianNB e Logistic Regression risultano pressochè invariate, mentre le stesse per il Decision Tree sono risultate peggiorate.

3.6.5 SMOTE

La classe con frequenza minore è stata sovra-campionata ad una numerosità di 6414 introducendo dei synthetic record mediante interpolazione, costruita considerando casualmente uno dei k = 5 vicini del record selezionato.

Lo SMOTE per il K-NN permette di raggiungere un'accuracy maggiore(da 0.87 a 0.91), mentre l'AUC rimane quasi invariata. Tale risultato è simile a quello ottenuto utilizzando la tecnica Random Oversampling. Il classificatore GaussianNB mostra delle performance lievemente inferiori, con accuracy pari a 0.97. La Logistic Regression rimane il classificatore più penalizzato, in termini di accuracy, dallo sbilanciamento dei dati.
Seppur in misura minore rispetto alla tecnica Random Oversampling, questa tecnica di bilanciamento oversampling non permette di migliorare le perfomance del Decision Tree sul test set. Il modello presenta un'accuracy inferiore dell'1% (0.88), così come l'AUC (Tabella 3.16).

	Accuracy	**AUC**
K-NN	0.91	0.913
GaussianNB	0.97	0.989
Logistic Regression	0.64	0.988
Decision Tree	0.88	0.847

Tabella 3.16: Performance dei modelli usando SMOTE

3.6.6 Adjust the Class Weights

Per la tecnica di bilancimento riguardante il peso delle classi, sono stati assegnati:

- peso = 1 per la classe Occupancy = 0,
- peso = 10 per la classe Occupancy = 1,

per dare maggiore importanza alla classe minore.
Tale tecnica è stata utilizzata sugli algoritmi Logistic Regression e Decision Tree, che permettono al loro interno di modificare il peso delle classi.
La Logistic Regression non mostra cambiamenti in termini di accuracy (0.64), mentre l'AUC aumenta da 0.959 a 0.988 come mostrato nella Tabella 3.17. Di contro, l'aggiustamento del peso delle classi permette al modello Decision Tree di raggiungere un livello di accuracy pari a 0.90 e un livello di AUC superiore dell'1.4%.

	Accuracy	**AUC**
Logistic Regression	0.64	0.988
Decision Tree	0.90	0.869

Tabella 3.17: Performance dei modelli usando Adjust Class Weights

Questa tecnica di bilanciamento con i pesi scelti permette in generale di migliorare le performance dei due modelli sul test set. Il Decision Tree vede un miglioramento maggiore in termini di accuracy; mentre il classificatore Logistic Regression presenta un miglioramento maggiore in termini di AUC.

3.6.7 Meta-Cost Sensitive Classifier

Per l'utilizzo del Meta-Cost Sensitive Classifier è stato importato il modulo *costcla*. Nello specifico, sono stati utilizzati i modelli:

- BayesMinimumRiskClassifier per gli algoritmi K-NN e GaussianNB;
- CostSensitiveLogisticRegression per l'algoritmo Logistic Regression;
- CostSensitiveDecisionTreeClassifier per l'algoritmo Decision Tree.

In particolare, il modello BayesMinimumRiskClassifier è un classificatore di rischio minimo sensibile ai costi ed example-dependent.
Per valutare il risparmio dell'utilizzo di Y(ovvero Occupancy) stimato sul vero valore di Y con matrice dei costi, è stata usata la metrica *savings_ score*, sapendo che la migliore performance è quella per cui *savings_ score = 1.*
La matrice dei costi utilizzata per K-NN e GaussianNB è rappresentata nella Tabella 3.18.

		Actual	
		Y	N
Predicted	Y	0	1
	N	5	0

Tabella 3.18: Weight Matrix usata sul Bayes Minim Risk

Utilizzando soltanto il classificatore K-NN si è ottenuto un punteggio di risparmio pari a 0.0348, mentre utilizzando sia il classificatore K-NN che il Bayes Minimum Risk si è ottenuto un punteggio di risparmio pari a 0.5328. L'accuracy del modello K-NN costruito sui dati fortemente sbilanciati, era inizialmente pari a 0.87; utilizzando, invece, il Bayes Minimum Risk si riesce ad ottenere un'accuracy pari a 0.93.
Per quanto riguarda il modello GaussianNB, utilizzando il Bayes Minim Risk, si è ottenuto un'accuracy e un punteggio di risparmio lievemente superiore pari a 0.9551 (contro un punteggio di 0.9498 con il solo GaussianNB).
La matrice dei costi utilizzata per Logistic Regression e Decision Tree è rappresentata nella Tabella 3.19.

		Actual	
		Y	N
Predicted	Y	0	8
	N	10	0

Tabella 3.19: Weight Matrix usata sul CostSensitiveLogisticRegression e CostSensitiveDecisionTreeClassifier

Utilizzando il modello CostSensitiveLogisticRegression con la matrice dei costi indicata, l'accuracy del modello Logistic Regression rimane invariata, ma aumenta l'AUC di quasi il 3%, come mostra la Tabella 3.20. Il modello CostSensitiveDecisionTreeClassifier permette al Decision Tree di ave-

re performance nettamente migliori sia in termini di accuracy ma soprattutto in termini di AUC. L'accuracy infatti aumenta da 0.89 a 0.91 e l'AUC adesso risulta pari al 98%.

	Accuracy	**AUC**
Logistic Regression	0.64	0.984
Decision Tree	0.91	0.981

Tabella 3.20: Performance dei modelli usando Meta-Cost Sensitive Classifier

In sintesi, i classificatori sensibili ai costi permettono di migliorare le performance di tutti i modelli considerati. Le prestazioni migliori si hanno soprattuto per il K-NN la cui accuracy aumenta del 6% e per il Decision Tree la cui AUC aumenta del 13.4%.

3.7 Linear Regression

In tutti i problemi di regressione lineare considerati, le misure di valutazione dei modelli sul test set considerate saranno:

- **Coefficiente di determinazione** R^2, per valutare quanto della variabilità di Y il modello è in grado di spiegare;
- **Mean Squared Error** MSE, per valutare l'errore quadratico medio tra i valori di Y osservati e quelli stimati dal modello di regressione;
- **Mean Absolute Error** MAE, per valutare anche l'errore assoluto tra i valori di Y osservati e quelli previsti.

3.7.1 2-dimensional Linear Regression

Il criterio secondo il quale sono state scelte le due variabili su cui costruire il modello di regressione bidimensionale è la maggiore correlazione. In particolare, l'attributo CO2 è stato scelto come variabile dipendente Y, essendo il livello di CO2 dipendente da tanti fattori, e l'attributo Light come variabile indipendente X.
Il modello ottenuto sui dati relativi al training set è risultato:

$$Y = 478.45966207 + 1.07168048 * X \quad (3)$$

Il coefficiente R^2 è pari a 0.558, quindi il modello spiega soltanto il 55.8% della variabilità della variabile CO2. Ciò è ben visibile nella Figura 3.17, in cui si vede che il modello di regressione non è un ottimo modello. L'MSE è pari a 37868.055, mentre l'MAE è 133.646.

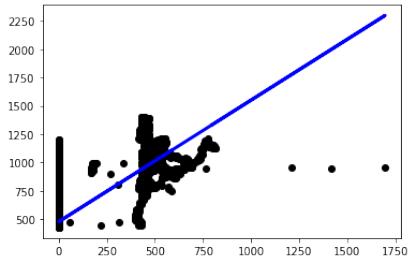

Figura 3.17: Grafico Light-CO2 e regressione lineare

3.7.2 Regolarizzazione con LASSO e Ridge

La regolarizzazione LASSO produce un modello praticamente identico alla regressione semplice:

$$Y = 478.46281352 + 1.07165412 * X \quad (4)$$

Il coefficiente R^2 e l'MAE presentano gli stessi valori riscontrati per la regressione lineare semplice; anche l'MSE potrebbe considerarsi tale perchè, sebbene pari a 37867.614, la differenza è davvero minima.
La regolarizzazione Ridge produce invece un modello esattamente uguale alla regressione semplice, con stesse performance.

3.7.3 Multiple Linear Regression

Il modello di regressione lineare multipla è stato ottenuto considerando sempre l'attributo CO2 come variabile dipendente Y e tutti gli attributi numerici come variabili indipendenti, aggiungendo anche la variabile binaria Occupancy (Tabella 3.21).

Attributo	**Coefficiente**
Temperature	120.634963
Humidity	24.4743004
Light	-0.0878925700
Occupancy	379.908918
Intercetta	-2580.7578275019355

Tabella 3.21: Coefficienti di regressione lineare multipla

Il modello di regressione lineare multipla ha performance nettamente migliori sul test set. Infatti, il coefficiente di determinazione R^2 indica che il modello spiega l'82.6% della variabilità di Y; l'MSE diminuisce notevolmente, passando da 37868.055 a 14871.703, così come l'MAE che adesso risulta pari a 78.730.

3.8 Conclusioni sul migliore classificatore

I modelli Logistic Regression, Categorical Naive Bayes e Decision Tree sono sicuramente da escludere perchè sono risultati i peggiori in termini di precisione, recall, f1-score e accuracy. Lo stesso vale per l'indicatore AUC. In termini di Cumulative Gain, la Logistic Regression e il Decision Tree sembravano essere degli ottimi modelli, ma, considerando le

altre misure di performance prese in esame, si ritiene che questi modelli non siano effettivamente ottimi.
La scelta, quindi, si restringe ai modelli K-NN e Gaussian Naive Bayes. Entrambi hanno mostrato delle ottime performance sul test set. Sebbene la misura di recall della classe Occupancy = 1 per il K-NN sia risultata lievemente superiore rispetto alla stessa per il Gaussian Naive Bayes, si ritiene che quest'ultimo modello sia da prediligere. Le motivazioni alla base di tale scelta sono essenzialmente le seguenti:

1. La curva Cumulative Gain presenta dei risultati migliori per il classificatore Gaussian Naive Bayes, dal momento che il gain per la classe Occupancy = 0 raggiunge prima il valore massimo pari a 1;
2. Il modello Gaussian Naive Bayes è più semplice e permette di definire più velocemente le classi del test set;
3. Il modello K-NN è un *lazy learner*.

4 Task 2 - Advanced Classifiers and Evaluation

4.1 Support Vector Machine

Per quanto riguarda la SVM, è stata implementata applicando al dataset iniziale lo StandardScaler sugli attributi numerici, lasciando invariate invece le variabili dummies. Questo ci ha permesso di risolvere i problemi di non convergenza dell'algoritmo. Per la ricerca dei migliori parametri per l'implementazione del Support Vector Classifier è stata utilizzata la GridSearchCV. Lo scoring sulle varie combinazioni dei parametri è stato fatto tenendo conto dell'accuracy misurata su un set di validazione di ampiezza 5. La messa a punto dei parametri è stata eseguite per ogni singola tipologia di Kernel Function, per rendere più chiara la differenza tra le differenti tipologie di Kernel. Le Kernel Function prese in considerazione sono state:

- Linear;
- Sigmoid;
- Polynomial;
- Radial basis function;

mentre i parametri investigati sono stati:

- C: [0.001, 0.01, 0.1, 1, 10, 100], che rappresenta il costo per l'errata classificazione ;
- gamma: [1,0.1, 0.01, 0.001,'scale','auto'], dove 'scale' utilizza come valore $1/(n\_features * varianza)$ e 'auto' utilizza come valore $1/n\_features$; questo è il parametro adottato dalle kernel functions per regolare la sua forma.

Le Figure 4.1 , 4.2, 4.3 e 4.4 mostrano le differenze che emergono utilizzando differenti Kernel Function.

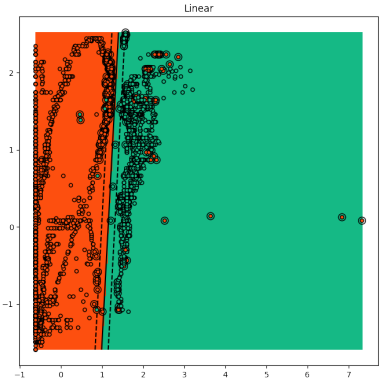

Figura 4.1: Grafico della SVM Linear

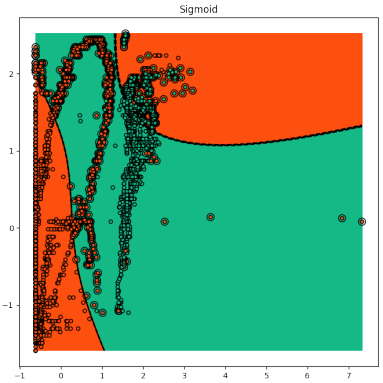

Figura 4.2: Grafico della SVM Sigmoid

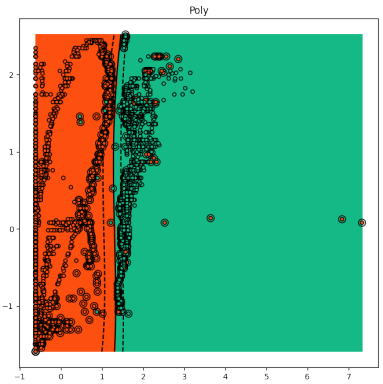

Figura 4.3: Grafico della SVM Poly

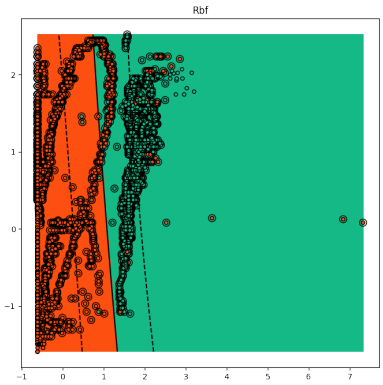

Figura 4.4: Grafico della SVM Rbf

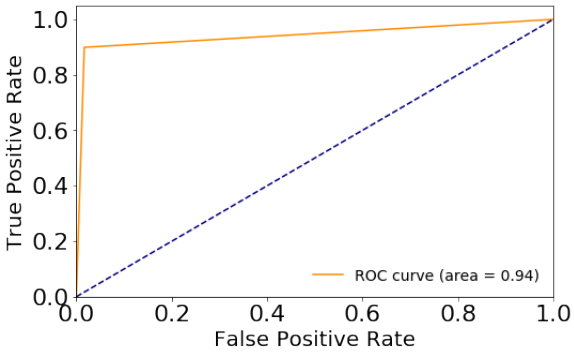

Figura 4.5: ROC curve - SVM

Come è possibile osservare dalla Tabella 4.1, l'accuracy massima (pari a 0.953) è stata ottenuta utilizzando il Kernel Trick = 'Polynomial', in particolare con i seguenti parametri: C = 1 e gamma = 'auto'.

Kernel	Linear	Sigmoid	Poly	Rbf
accuracy	0.92	0.92	0.953	0.92

Tabella 4.1: Accuracy per i modelli con diverse Kernel Functions

Nello specifico, la Tabella 4.2 riassume le performance del modello sul test set considerato. Il 94% dei record classificati come Occupancy = 0 appartiene a questa classe e il 98% dei record cheeffettivamente appartiene alla classe Occupancy = 0 è stato classificato come appartenente ad essa (recall = 98%). La precisione del modello per la classe 1 è maggiore (precision = 97%), ma la misura di recall è pari a 0.90. In generale, il modello sul test set presenta accuracy pari a 0.95.

	precision	recall	f1-score
0	0.94	0.98	0.96
1	0.97	0.90	0.93
accuracy			0.95

Tabella 4.2: Performance del modello SVM Poly sul test set

La ROC curve in figura 4.5 dimostra che il modello è in grado di predire abbastanza bene le classi di appartenenza dei record. Infatti l'area AUC è pari al 94%.

4.2 SVM e PCA

L'implementazione della SVM è stata svolta tenendo anche in considerazione la PCA sullo stesso dataset utilizzato in precedenza. Anche in questo caso, la messa a punto dei parametri è stata eseguita con la GridSearchCV. Si specifica, inoltre, che la riduzione della dimensionalità del dataset mediante PCA è stata effettuata utilizzando l'estensione KernelPCA al fine di ottenere una riduzione non lineare attraverso l'utilizzo del kernel. Questo approccio ha permesso di lavorare senza particolari problemi sulle Kernel Function non lineari ('Sigmoid', 'Poly' e 'Rbf').
La Tabella 4.3 mostra le accuracy ottenute sulle differenti Kernel Function.

Kernel	Linear	Sigmoid	Poly	Rbf
accuracy	0.898	0.893	0.781	0.901

Tabella 4.3: Performance del modello SVM sul test set ridotto con PCA

Come è possibile notare i risultati ottenuti non sono molto buoni rispetto a quelli ottenibili utilizzando la multidimensionalità del dataset preso in considerazione in precedenza. In Figura 4.6 è rappresentato il modello SVM costruito utilizzando il dataset ridotto che restituisce accuracy migliore.

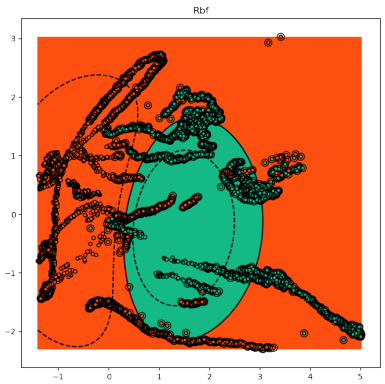

Figura 4.6: Grafico della SVM Rbf con PCA

La riduzione a due sole componenti implica una perdita di informazioni per cui i modelli non lineari vengono notevolmente penalizzati. Le componenti principali sono combinazioni lineari degli attributi, per cui risulta difficile modellarle utilizzando altre tipologie di kernel.

4.3 Neural Network

4.3.1 Single Perceptron

La rete neurale Single Percetron è stata eseguita tramite il classificatore Perceptron della libreria Scikit-Learn. I dati sono stati preventivamente normalizzati e per quanto riguarda il training del modello, i parametri del perceptron sono risultati i seguenti:

- max_iter= 40 ;
- tol = 0.001;
- eta0 = 0.01 .

Dopo aver fissato il numero di iterazioni massime, la tolleranza di uscita (tol) e il tasso di apprendimento (eta0), il modello ha restituito una accuracy molto alta, pari al 97% : ciò vuol dire che dopo l'addestramento il modello prevede e classifica correttamente il 97% degli esempi presenti nel test set.

4.3.2 Multilayer Perceptron

Il Multilayer Perceptron è stato implementato utilizzando la libreria Scikit-Learn, in particolare MLPClassifier. Come dimensione degli hidden layer si è pensato di utilizzare 128, 64 e 32. La scelta è stata dettata dalla visione di diversi forum come StakeOverFlow, i quali consigliavano proprio tali dimensioni come 'migliori'. Per la ricerca della migliore activation fuction, sono state passate in rassegna tutte le funzione presenti nella libreria: *'Tanh'*, *'Logistic'*, *'Relu'* e *'Identify'*.

Activation	Tahn	Logistic	Relu	Identify
Accuracy	0.9347	0.9786	0.9418	0.9786

Tabella 4.4: Accuracy per le varie Activation function

Come si evince dalla Tabella 4.4, la miglior activation function da utilizzare per la creazione del Multilayer Percetron è sia *'Logistic'* che *'Identify'*.

4.3.3 Deep Neural Network

Per quanto riguarda il Deep Neural Network, la rete neurale è stata sviluppata utilizzando la liberia Keras, in particolare i modelli Sequential e Dense. Il primo layer è stato impostato come Sequential e gli altri 3 modelli come Dense. Come ottimizzatore, è stato scelto il parametro *'ADAM'* perchè molto performante nel caso di dataset di grandi dimensioni (superiori a 100 records); come funzione di Loss, ossia la funzione obiettivo che il modello tenta di minimizzare, è stata scelta *'binary_ crossentropy'*.

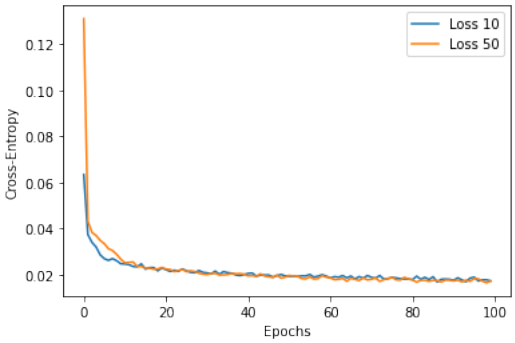

Figura 4.7: Difference batch size

Si è scelto di impostare le Epochs pari a 100, in modo da avere una valutazione più accurata, anche se dispendiosa per il tempo di calcolo, e infine si è valutato il modello in base alla dimensione del batch che varia da 10 a 50. La variazione della dimensione del batch non ha apportato grandi differenze, infatti i modelli sono alquanto simili come dimostrano l'accuracy e il loss della Tabella 4.5 sottostante.

	Loss 10	Loss 50
Accuracy	0.942589	0.930206
Loss	0.234624	0.253963

Tabella 4.5: Tabella dell'accuracy e del loss

Si è voluto valutare il modello anche estraendo dal training set dei records da utilizzare come validation set: l'80%, ossia 6514, delle istanze è stato usato per addestrare il modello e il restante 20%, ossia 1629 records, per validarlo.

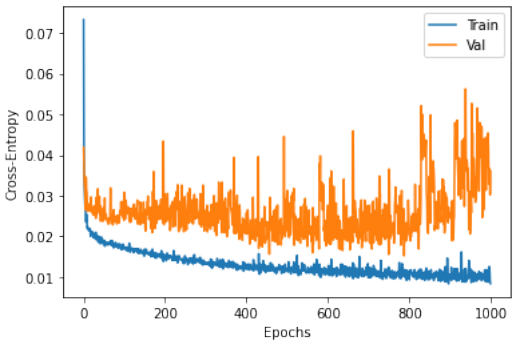

Figura 4.8: Valutazione della cross-entropy

Con il validation set, come era da aspettarsi, il modello risulta più accurato, infatti l'accuracy aumenta fino a 0.951595; per quanto riguarda il valore del Loss, esso arriva a 0.989468. Per evitare problemi di overfitting, si è deciso di applicare gli approcci di Early Stopping, L2 Regularization e Drop-Out e infine vedere tra questi 3 quale fosse il migliore in termini di accuratezza.

	Earl Stopping	L2 Regularization	Drop-Out
Accuracy	0.9557	0.6353	0.9711
Loss	0.224719	5.709756	0.380418

Tabella 4.6: Tabella dei risultati

In termine di accuratezza, il miglior risultato è stato riscontrato utilizzando l'approccio Drop-Out che consiste nel rimuovere randomicamente alcune connessioni durante il training.
Infine sono stati valutati gli iperparametri del modello Keras con obiettivo l'ottimizzazione. Per costruire il modello, anche in questo caso, è stato utilizzato la libreria Keras e per ricercare i migliori parametri si è stato utilizzato l'approccio RandomizedSearchCv in cui, per ogni iperparametro, viene definito un set di possibili valori discreti che esso può assumere, ricercando in maniera casuale le possibili combinazioni. I parametri da valutare sono stati i seguenti:

- n_layers = [1, 2, 3] ;
- h_dim = [32, 64, 128];
- activation = ['relu', 'linear', 'sigmoid', 'tanh'];
- optimizer = ['adagrad', 'adam'].

e il migliore risultato ottenuto è stato:

- n_layers = 2 ;
- h_dim = 64;
- activation = 'tanh';
- optimizer = 'adagrad'.

4.4 Ensemble Classifiers

4.4.1 RandomForest

Per la stima dei parametri del classificatore Random Forest è stato utilizzato l'approccio esaustivo GridSearchCV. Le performance degli iperparametri e del modello selezionati sono state valutate tenendo conto dell'accuracy e misurate su un set di validazione di ampiezza 5, seguendo il metodo 5-fold Cross Validation. I parametri investigati sono stati:

- le misure di impurità *criterion*, selezionando sia l'indice Gini che l'entropia;
- la massima ampiezza dell'albero *max_ depth*, investigata tra 1, 4, 6, 8 e 10;
- il minimo numero di record richiesti per dividere un nodo interno *min_ sample_ split*, selezionati tra 2, 4, 5, 10, 20, 50;
- il numero minimo di record richiesti in un nodo foglia *min_ sample_ leaf*, selezionati tra 1, 2, 4, 5, 10, 20, 35, 50;
- *n_ estimators* selezionati tra 1, 2, 5, 10, 20, 30, 40, 50, 55 e 100.

Il miglior modello è risultato avere i seguenti parametri: *criterion* = 'gini', *max_ depth* = 6, *max_ samples_ leaf* = 50, *min_ samples_ split* = 4 e *n_ estimators* = 5.
L'importanza degli attributi nella costruzione dell'albero è mostrata nella Figura 4.9, mentre in Figura 4.11 è raffigurato l'albero decisionale ottenuto.

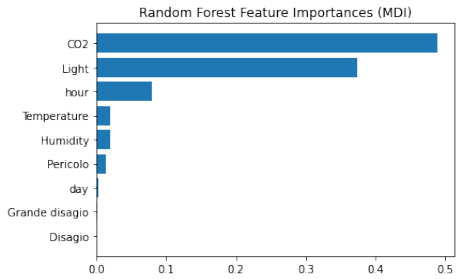

Figura 4.9: Feature importance - Random Forest Classifier

La Tabella 4.7 sintetizza le performance del modello scelto sul test set. Il 97% dei record classificati come Occupancy = 0 appartiene a questa classe e sempre il 97% dei record che effettivamente appartiene alla classe Occupancy = 0 è stato classificato come appartenente ad essa (recall = 0.97). La precisione del modello per la classe 1 è lievemente inferiore (precision = 95%), così come la misura di recall. In generale, il modello sul test set presenta accuracy pari a 0.96 .

	precision	recall	f1-score
0	0.97	0.97	0.97
1	0.95	0.95	0.95
accuracy			0.96

Tabella 4.7: Performance del modello Random Forest sul test set

L'AUC in Figura 4.10 è pari al 96%, quindi si può concludere che il modello è in grado di predire abbastanza bene quando la stanza d'ufficio è occupata o no.

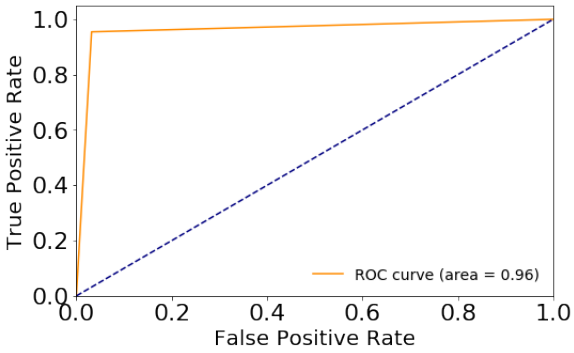

Figura 4.10: ROC curve - Random Forest

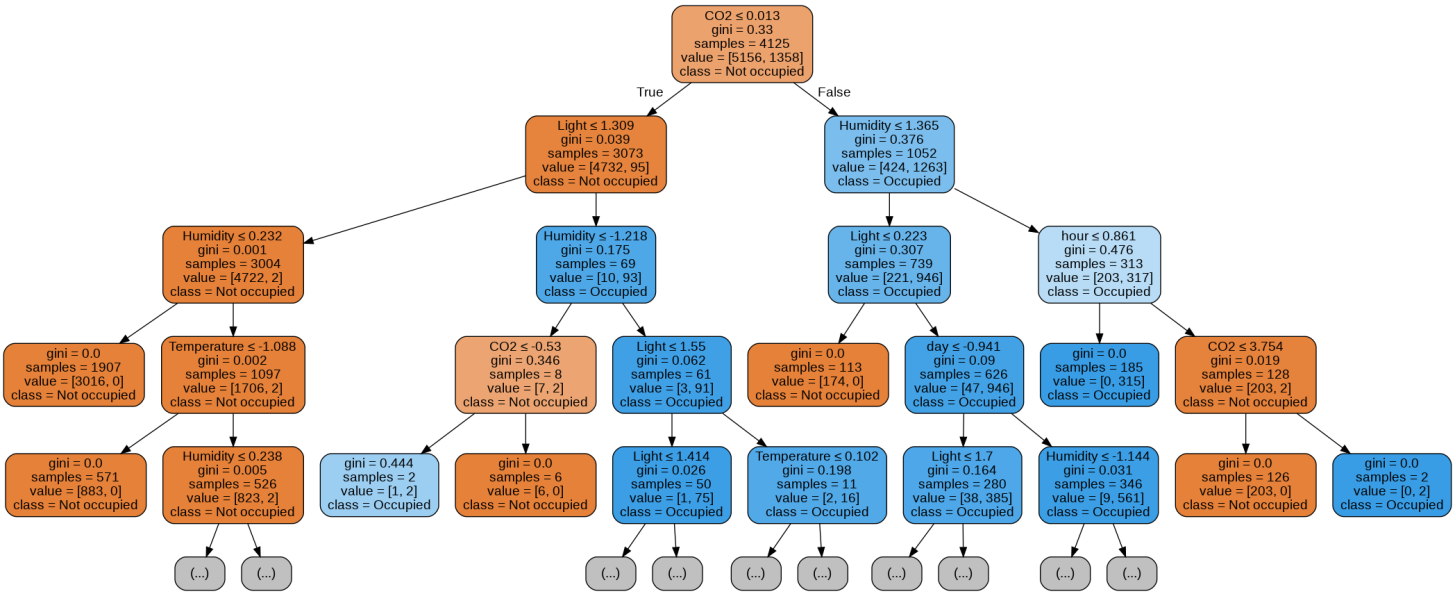

Figura 4.11: Random Forest tree

4.4.2 AdaBoost

L'AdaBoost è un classificatore adottato in concomitanza con altri algoritmi di classificazione. L'output di questi algoritmi viene combinato mediante una somma pesata per ottenere l'output finale. Nel nostro caso, verranno considerati due classificatori implementati in precedenza: il Decision Tree Classifier e il Random Forest Classifier.
La ricerca dei parametri migliori è stata eseguita utilizzando la GridSearchCV, impostando a priori i parametri del base classifier come i migliori parametri trovati precedentemente. Per quanto riguarda l'AdaBoost, gli iperparametri valutati sono stati:

- n_estimators: [1, 2, 5, 10, 20, 30, 40, 50, 55, 100] è il massimo numero di stimatori per il quale il boosting termina;
- learning_rate: [0.01, 0.05, 0.1, .5, 1.0] e indica la riduzione del contributo di ogni classificatore;
- algorithm: ['SAMME.R', 'SAMME'] indica il tipo di algoritmo adoperato. Il SAMME.R tipicamente converge più velocemente rispetto al SAMME ottenendo un errore sul test inferiore (meno iterazioni di boosting).

Tramite la GridSearchCV, i modelli ottimali trovati presentano i seguenti parametri:

- **Decision Tree**:
  - n_estimators: 100
  - learning_rate: 0.1
  - algorithm: 'SAMME'
- **Random Forest**:
  - n_estimators: 2
  - learning_rate: 0.05
  - algorithm: 'SAMME.R'

Nella Tabella 4.8 sono mostrate le performance del modello utilizzando come base classifier il Decision Tree, mentre nella Tabella 4.9 quelle relative al Random Forest Classifier. L'AdaBoost implementato utilizzando il Decision Tree presenta performance migliori rispetto allo stesso utilizzando, invece, il Random Forest: il primo infatti ha un'accuracy del 98%, mentre il secondoi del 96%.

	precision	recall	f1-score
0	1	0.97	0.98
1	0.95	1	0.97
accuracy			0.98

Tabella 4.8: Performance del modello AdaBoost con Decision Tree sul test set

	precision	recall	f1-score
0	0.96	0.97	0.97
1	0.95	0.93	0.94
accuracy			0.96

Tabella 4.9: Performance del modello AdaBoost con Random Forest sul test set

Confrontando le Figure 4.12 e 4.13 si vuole mettere in evidenza come il Decision Tree generalizzi molto bene, mentre la Random Forest risulta poco generica e crea problema di overfitting. Nelle stesse figure sono mostrate inoltre le distribuzioni dei punteggi delle decisioni per le due classi: i campioni con punteggio maggiore di 0 sono classificati come 1, gli altri come 0.

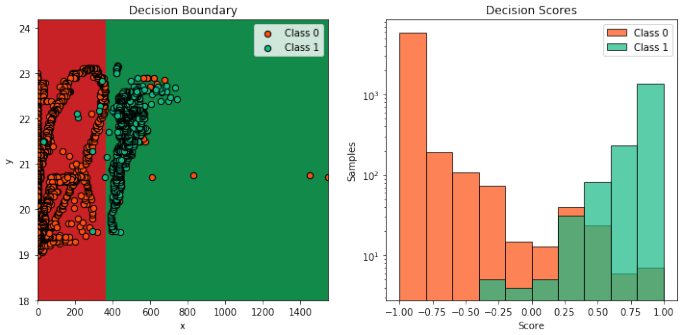

Figura 4.12: Grafico AdaBoost con Decision Tree

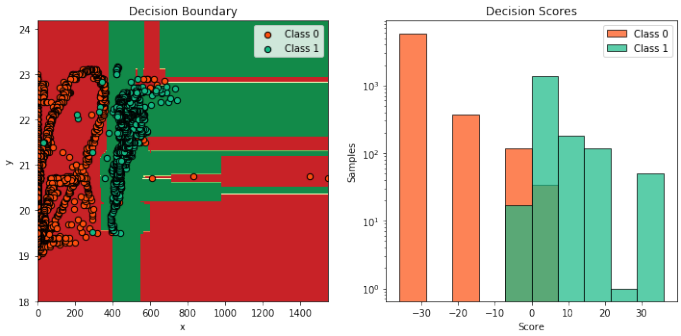

Figura 4.13: Grafico AdaBoost con Random Forest

In conclusione, il modello AdaBoost con il Decision Tree risulta un buon Ensemble Classifier che offre performance migliori rispetto a quello in cui è utilizzato il Random Forest. Come è possibile osservare anche dall'area AUC nella Figura 4.14, che è pari al 98%, l'AdaBoost con Decision Tree permette di classificare correttamente il 98% dei record presenti nel test set.

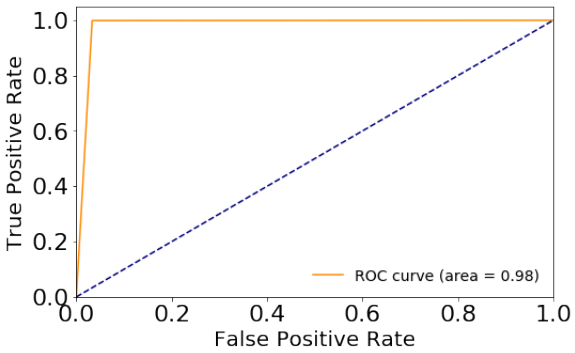

Figura 4.14: ROC curve - AdaBoost con Decision Tree Classifier

4.4.3 Bagging

Dal momento che il classificatore Decision Tree presenta performance migliori del classificatore Random Forest e ciò risulta anche dall'utilizzo dello stesso nell'AdaBoost, si è deciso di implementare il Bagging utilizzando proprio il modello Decision Tree costruito in precedenza. Mediante l'utilizzo dell'approccio esaustivo GridSearchCV, si è scelto di valutare i seguenti iperparametri, al fine di trovare il miglior modello in termini di accuracy:

- Il numero di stimatori *n_ estimators*: [1, 2, 5, 10, 20, 30, 40, 50, 55, 100]
- Il numero di campioni *max_ samples* da prelevare dal dataset di training per addestrare ciascuno stimatore di base: [1, 10, 20, 50, 100,200,300,400,500,800,1000]
- Il numero di attributi *max_ features* da utilizzare per il training: [1,2,3,4,5,6,7,8,9]

Ciò che ci aspettavamo era che il miglior modello fosse quello in cui il numero di campioni fosse più elevato e, in effetti, i parametri per cui si è ottenuto il modello migliore sono: *max_ samples* = 1000, *max_ features* = 5 ed *n_ estimators* = 20.

Come si può vedere nella Tabella 4.10, le performance del modello sul test set sono pressochè le stesse ottenute utilizzando la tecnica AdaBoost: il modello presenta un'accuracy pari a 0.98 e anche in termini di precisione e recall i risultati sono ottimi.

	precision	recall	f1-score
0	1	0.97	0.98
1	0.95	0.99	0.97
accuracy			0.98

Tabella 4.10: Performance del modello Bagging con Decision Tree sul test set

4.5 Conclusioni sul migliore classificatore

I modelli AdaBoost e Bagging in cui viene utilizzato il Decision Tree come base classifier sono risultati essere i migliori tra gli Ensemble Classifier presi in esame in questa sezione. Prestazioni molto elevate sono state ottenute anche utilizzando il MLPClassifier con activation function 'Logistic' e 'Identify'. Tuttavia, si ritiene che questi classificatori siano da ritenersi peggiori rispetto a quelli appena citati dal momento che:

1. hanno richiesto un tempo eccessivamente superiore per la costruzione del modello rispetto agli Ensemble Classifier;
2. le performance dei modelli sono pressochè identiche.

Volendo fare un confronto tra l'AdaBoost e il Bagging, si vuole sottolineare che soltanto l'AdaBoost determina i pesi dei records al fine di focalizzarsi su quelli più difficili da classificare e questo è sicuramente uno dei vantaggi che ci porta a considerare questo come il migliore dei classificatori analizzati. Inoltre, in generale, questo classificatore è meno suscettibile a problemi di overfitting. Di conseguenza, si ritiene che l'AdaBoost sia in definitva il miglior advanced classifier per il dataset considerato.