UNIVERSITÀ DEGLI STUDI DI SALERNO

DIPARTIMENTO DI INFORMATICA

DIPARTIMENTO DI ECCELLENZA

BEEHAIVE

buzz into the future

ANNO ACCADEMICO 2023/2024

Documentazione FIA - beehAIve

Carmine Boninfante (matr. 0512113309)

Andrea De Pasquale (matr. 0512114909)

Francesco Festa (matr. 0512113547)

Nicolò Gallotta (matr. 0512114639)

Sara Valente (matr. 0512114627)

Repository di GitHub: https://github.com/XJustUnluckyX/beehAIve

Indice

1 Introduzione

Al giorno d'oggi, il lavoro degli apicoltori è poco supportato da sistemi software affidabili e funzionanti che riescano a rappresentare un reale vantaggio alla salvaguardia delle api e alla sostenibilità dell'intero settore apistico. In particolare, non vi è alcun tipo di assistenza per far fronte ad uno dei più grandi fenomeni che minacciano l'intera sopravvivenza delle api: il **Colony Collapse Disorder (CCD)**. Quest'ultimo è un fenomeno per il quale le colonie subiscono una brusca diminuzione della popolazione per via dell'abbandono delle arnie o, peggio, della morte delle api [1].
A tal proposito, nasce la piattaforma *beehAIve* il cui scopo è dare supporto agli apicoltori, assistendoli nelle loro attività e nella diagnosi del CCD.

Il codice e il materiale utilizzato è reperibile alla repository di GitHub sopra citata, più precisamente all'interno della cartella src/ai.

2 Definizione del problema

2.1 Obiettivi

L'obiettivo di questo progetto è sviluppare un modello di Machine Learning che possa analizzare i dati raccolti da dispositivi IoT real-time per identificare potenziali casi di CCD all'interno delle arnie di un apicoltore. In particolare, tra i dati a nostra disposizione, la caratteristica più importante è la presenza/assenza dell'ape regina. Per poter ricavare questa informazione, abbiamo bisogno di analizzare l'audio prodotto dalle api presenti nell'arnia. Per fare ciò, il nostro scopo si estende a sviluppare anche una Rete Neurale Convoluzionale (CNN) che sia di supporto al modello summenzionato, riuscendo a valutare la presenza o l'assenza dell'ape regina attraverso gli spettrogrammi derivati dal suono del ronzio api.

2.2 Specifica PEAS

La specifica PEAS della nostra CNN è la seguente:

- **Performance**: le misure di prestazione adottate per valutare l'operato della CNN sono *Loss*, *Accuracy*, *Precision* e *AUC (Area Under the Curve)*;
- **Environment**: gli elementi che costituiscono l'ambiente sono degli spettrogrammi ottenuti dalle registrazioni audio delle arnie;
- **Actuators**: la CNN etichetterà gli spettrogrammi indicando la presenza o l'assenza dell'ape regina all'interno delle arnie;
- **Sensors**: la CNN riceve gli input necessari attraverso dei sensori IoT che catturano l'attività sonora delle arnie.

La specifica PEAS del nostro modello di Machine Learning è invece la seguente:

- **Performance**: le misure di prestazione adottate per valutare l'operato del modello sono *Accuracy*, *Precision*, *Recall* e *F1 Score*;
- **Environment**: gli elementi che costituiscono l'ambiente sono delle misurazioni di vari parametri eseguite sulle arnie;
- **Actuators**: il modello etichetterà le arnie come soggette al CCD oppure non soggette al CCD;
- **Sensors**: il modello riceve gli input necessari attraverso dei sensori IoT e dalla CNN, i quali forniscono misurazioni in tempo reale delle arnie.

2.2.1 Caratteristiche dell'ambiente

Le caratteristiche dell'ambiente sono le seguenti:

- **Parzialmente osservabile**: non conosciamo ogni singola caratteristica di un'arnia, poiché le misurazioni sono limitate;
- **Stocastico**: non sappiamo cosa può succedere all'ambiente nel momento in cui viene effettuata una predizione;
- **Episodico**: ogni misurazione può essere vista come un episodio unico, poiché non c'è correlazione tra le varie misurazioni/registrazioni;
- **Statico**: le caratteristiche dell'istanza di input e la relativa etichetta rimangono costanti durante la previsione;
- **Continuo**: le caratteristiche dell'ambiente possono assumere un numero infinito di valori all'interno di un determinato intervallo;
- **Multi-agente**: l'ambiente ospita due agenti, ovvero la CNN e il modello di Machine Learning, i quali cooperano per raggiungere lo stesso obiettivo.

2.3 Analisi del problema

Il nostro modello mira ad effettuare una diagnosi del CCD sulla base di determinate caratteristiche. Pertanto, si tratta di un'istanza di un problema di apprendimento supervisionato, più precisamente di **classificazione binaria**. Andremo quindi ad utilizzare e confrontare diverse tecniche di Machine Learning, in modo da identificarne la migliore sulla base delle metriche precedentemente considerate.

3 Rete Neurale

Iniziamo con la progettazione della CNN, in quanto questa sarà necessaria per valutare una delle *features* utili poi al nostro classificatore.

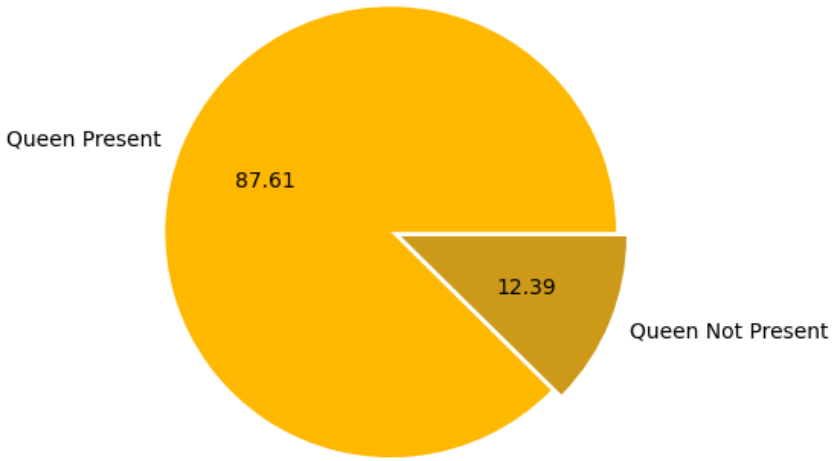
3.1 Data Understanding

I dati che sono stati utilizzati per l'addestramento della nostra CNN provengono da Kaggle. In particolare, abbiamo individuato un dataset contenente i file audio del ronzio delle api assieme alla relativa presenza o assenza dell'ape regina. Tale dataset è presente al seguente link: `https://www.kaggle.com/datasets/annajyang/beehive-sounds/data`.
Il dataset è composto da **1275 righe** e **23 colonne**. Alle finalità del nostro problema le colonne interessate sono **"*file name*"** e **"*queen presence*"**, che indicano rispettivamente il nome del file audio e la presenza oppure l'assenza della regina (1 se è presente, 0 altrimenti).

Oltre al dataset, abbiamo anche scaricato tutti i file audio riportati a parte. Tuttavia, facendo ciò abbiamo ottenuto **7100 tracce audio**, piuttosto che 1275. Questo è dovuto al fatto che ogni file audio è diviso in vari segmenti di un minuto ciascuno. Osservando i file, ci sono apparentemente sei segmenti audio per file, ciò significherebbe che una registrazione completa avrebbe una durata di sei minuti. Tuttavia, i conti non tornano, poiché $1275 \cdot 6 = 7650$. Questo significa che non tutti i file audio presentano lo stesso numero di segmenti. Di conseguenza, vale la pena indagare ulteriormente a riguardo e **stabilire un numero di segmenti comune** ad ogni registrazione.

Un' altra considerazione da fare riguarda il nome dei file all'interno della colonna "*file name*". In particolare, il nome di ogni file nella colonna presenta la forma *yy-mm-dd–hh-mm-ss_X.raw*, mentre il nome di ogni file scaricato da noi presenta la forma *yy-mm-dd–hh-mm-ss_X__segmentX.wav* dove "*segmentX*" rappresenta il numero del segmento relativo all'intero file audio. Si avverte quindi la necessità di **modificare i valori** di questa colonna nel nostro dataset.

Per quanto riguarda la classe da predire, osserviamone il bilanciamento:



Il grafico parla chiaro: vi è un **notevole sbilanciamento** causato dalla pronunciata disparità tra le istanze appartenenti alle due classi che indicano presenza e assenza dell'ape regina. Difatti, identifichiamo 1117 campioni in cui l'ape regina risulta presente e 158 campioni in cui l'ape regina risulta assente.

In merito ai valori nulli, le uniche colonne coinvolte sono poco rilevanti per affrontare il nostro problema, quindi questa è una problematica che ci riguarda ben poco.

Di seguito sono riportati i dati discussi finora, non ancora sottoposti a modifiche:

	device	hive number	date	hive temp	hive humidity	weather temp	weather humidity	...	rain	lat	long	file name	queen presence
0	1	5	2022-06-08 14:52:28	36.42	30.29	26.68	52	...	0	37.29	-121.95	2022-06-08--14-52-28_1.raw	1
1	1	5	2022-06-08 15:51:41	33.56	33.98	25.99	53	...	0	37.29	-121.95	2022-06-08--15-51-41_1.raw	1
2	1	5	2022-06-08 17:21:53	29.01	42.73	24.49	56	...	0	37.29	-121.95	2022-06-08--17-21-53_1.raw	0
3	1	5	2022-06-08 18:20:59	30.51	36.74	22.97	59	...	0	37.29	-121.95	2022-06-08--18-20-59_1.raw	0
4	1	5	2022-06-08 19:20:04	30.32	35.55	21.52	61	...	0	37.29	-121.95	2022-06-08--19-20-04_1.raw	0
...	...	...	...	...	...	...	...	...	...	...	...	...	...
1270	2	1	2022-07-15 11:18:39	50.84	11.99	23.58	55	...	0	37.29	-121.95	2022-07-15--11-18-39_2.raw	1
1271	2	1	2022-07-15 12:25:14	49.58	11.60	25.60	51	...	0	37.29	-121.95	2022-07-15--12-25-14_2.raw	1
1272	2	1	2022-07-15 13:25:58	45.83	15.36	26.49	49	...	0	37.29	-121.95	2022-07-15--13-25-58_2.raw	1
1273	2	1	2022-07-15 14:24:58	35.82	23.48	27.33	46	...	0	37.29	-121.95	2022-07-15--14-24-58_2.raw	1
1274	2	1	2022-07-15 15:28:21	31.55	27.90	26.90	45	...	0	37.29	-121.95	2022-07-15--15-28-21_2.raw	1

1275 rows × 23 columns

3.2 Data Preparation

Una volta individuati i problemi che affliggono il nostro dataset, procediamo ad applicare le soluzioni discusse.

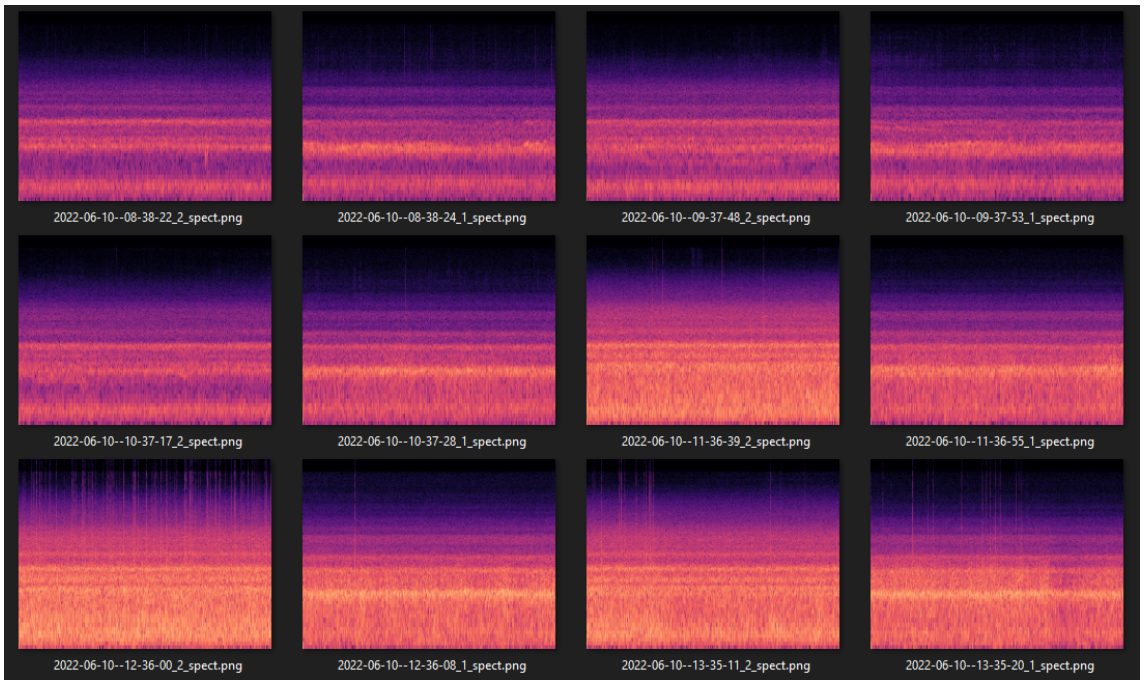
Come precedentemente spiegato, le uniche colonne che ci interessano all'interno del nostro dataset sono "*file name*" e "*queen presence*" per ricavare il file audio con la corrispettiva label. Di conseguenza, **abbiamo eliminato le colonne inutili**. Il seguente è il nostro dataset dopo aver effettuato questa operazione:

index	file name	queen presence
0	2022-06-08--14-52-28_1.raw	1
1	2022-06-08--15-51-41_1.raw	1
2	2022-06-08--17-21-53_1.raw	0
3	2022-06-08--18-20-59_1.raw	0
4	2022-06-08--19-20-04_1.raw	0
5	2022-06-08--20-19-13_1.raw	0
6	2022-06-08--21-18-22_1.raw	0
7	2022-06-08--22-17-32_1.raw	0
8	2022-06-08--23-16-44_1.raw	0
9	2022-06-09--00-15-56_1.raw	0
10	2022-06-09--01-15-08_1.raw	0
11	2022-06-09--02-14-24_1.raw	0
12	2022-06-09--03-13-35_1.raw	0
13	2022-06-09--04-12-49_1.raw	0
14	2022-06-09--05-12-03_1.raw	0
15	2022-06-09--06-11-16_1.raw	0
16	2022-06-09--07-10-32_1.raw	0
17	2022-06-09--08-09-59_1.raw	0
18	2022-06-09--09-09-25_1.raw	0
19	2022-06-09--10-08-51_1.raw	0
20	2022-06-09--11-08-21_1.raw	0
21	2022-06-09--12-07-53_1.raw	0
22	2022-06-09--13-07-21_1.raw	0
23	2022-06-09--14-06-06_1.raw	0
24	2022-06-09--15-05-10_1.raw	0
...	...	...

Oltre a ciò, considerando che il ronzio delle api è un suono tendenzialmente ritmico e difficilmente subisce forti variazioni, si è scelto di utilizzare solo **tre segmenti** per traccia audio. Questa decisione mira a semplificare il processo di creazione degli spettrogrammi senza perdere una quantità eccessiva di informazioni potenzialmente significative.
Inoltre, per uniformare la durata delle tracce audio, sono state eliminate dal dataset quelle di cui non si dispongono almeno tre segmenti. Infine, i segmenti appartenenti alla stessa registrazione sono stati uniti in un'unica traccia da tre minuti.

Eseguite le operazioni sopraelencate, da 7100 tracce audio siamo passati a 1195.

Successivamente, abbiamo ottenuto tutti gli **spettrogrammi** dei corrispettivi file audio. Tuttavia, questo processo si è rivelato particolarmente oneroso a livello computazionale, abbiamo quindi deciso di parallelizzare l'algoritmo su cinque thread, poiché questo è il massimo comune divisore utile di 1195. Risolto questo impedimento, abbiamo ottenuto gli spettrogrammi che rappresentano l'input della nostra rete neurale.
La seguente immagine rappresenta una piccola parte dei risultati che siamo riusciti ad ottenere:



Poiché vogliamo che il numero dei file audio corrisponda al numero di campioni nel dataset, abbiamo dovuto **eliminare le righe** i cui file audio non presentavano almeno tre segmenti e di cui non esiste la traccia audio corrispondente in primo luogo (riportati nel dataset, ma non presenti nella nostra cartella come .wav). Per farlo, abbiamo innanzitutto cambiato il nome dei file nel dataset in modo da farlo corrispondere al formato degli spettrogrammi prodotti precedentemente, poiché d'ora in poi ci concentreremo su questi. Quindi, dal precedente formato "*yy-mm-dd–hh-mm-ss_X.raw*" siamo passati ad un nuovo formato, ovvero "*yy-mm-dd–hh-mm-ss_X_spect.png*".
Eseguite queste operazioni, da 1275 campioni nel dataset siamo passati a 1195. Una volta fatto ciò, abbiamo effettuato un'**estrazione di un piccolo dataset pseudocasuale** che verrà utilizzato nella valutazione del nostro modello.
Infine, abbiamo preso atto dello sbilanciamento delle nostre classi. I possibili approcci che abbiamo considerato sono stati due:

- Effettuare un semplice ***undersampling* della classe maggioritaria**, ottenendo complessivamente 300 campioni, diviso in 150 in cui l'ape regina è presente e 150 in cui l'ape regina non è presente. Ovviamente questo comporterebbe un'enorme perdita di dati ma, come abbiamo detto precedentemente, i ronzii delle api sono generalmente molto simili tra loro per cui non comportano particolari differenze nei loro spettrogrammi. Per tale motivo, anche disponendo di pochi campioni, possiamo aspettarci prestazioni decenti dalla nostra CNN;
- Produrre più campioni **tagliando gli spettrogrammi a metà e trattandoli come due diversi campioni** [2], così da raddoppiarne il numero totale. Questo approccio sembra applicabile sulla carta, ma potrebbe portare la CNN ad un maggiore *overfitting* poiché i ronzii sono molto simili tra loro. Inoltre, nasce una nuova problematica nel momento in cui l'anomalia si trova solo in una delle due metà dello spettrogramma, portandoci a dei dati errati.

Poiché la seconda opzione è un notevole investimento di tempo per un riscontro minimo, abbiamo deciso di applicare la prima.

Prima di procedere, abbiamo dovuto **trasformare i nostri dati in un formato accettabile dalla rete neurale**, ovvero array NumPy. In particolare, durante la conversione delle immagini, è stato necessario eliminare una dimensione, che rappresenta il valore alfa (la trasparenza dell'immagine), il quale è costantemente impostato su 255 poiché i pixel sono sempre completamente opachi, trattandosi quindi di un'informazione superflua. Abbiamo proceduto in questo modo:

```
cnn_dataset = pd.read_csv("CNN_dataset.csv")
spectrogram_list = cnn_dataset["file name"].tolist()

# Creazione delle variabili per l'addestramento della CNN
X_train = []  # array contenente le immagini di ogni spettrogramma
y_train = np.asarray(cnn_dataset["queen presence"].tolist())  # array contenente le label

# Trasformazione delle immagini degli spettrogrammi in array NumPy da dare in input alla CNN
for file in spectrogram_list:
    image = Image.open(spectrograms_dir + "/" + file)
    image_array = np.asarray(image)[:, :, :-1]  # Rimozione di una dimensione dell'immagine (alpha)
    X_train.append(image_array)
X_train = np.asarray(X_train)
```

A questo punto, i dati sono pronti per il training e si può passare alla prossima fase.

3.3 Data Modeling

Durante la fase di training abbiamo usato una ***stratified k-fold*** *validation* con k = 5. Questa scelta è motivata dal fatto che, per k = 10, il tempo di esecuzione risultava essere sensibilmente lento e, considerando l'ampia applicazione della sperimentazione empirica degli iperparametri (che implica numerosi cicli di fit del modello), non potevamo permetterci un tempo di computazione così alto.

Per identificare il modello migliore, ci siamo basati su *Accuracy*, *Precision* e *AUC* e abbiamo sfruttato la *Binary Crossentropy*. In particolare, abbiamo scelto come metrica principale la *Precision*, poiché in questo contesto specifico **il sistema è particolarmente sensibile ai falsi positivi**. Quest'ultimi sono i casi in cui l'ape regina è assente ma il modello prevede che è invece presente nell'arnia, conducendoci inevitabilmente a potenziali previsioni sbagliate di CCD.

Una volta definite le nostre metriche, abbiamo proceduto con il testare empiricamente alcuni valori dei parametri e varie configurazioni di *layers* della nostra rete neurale. Per farlo, abbiamo sviluppato un piccolo sistema di *logging* che riportava, dopo ogni addestramento, la configurazione e le corrispettive valutazioni.
Il seguente è un estratto del log:

```
------------------
STATO CONFIGURAZIONE:
Pooling: MaxPooling, Dense_neurons: 128, Optimizer: adam, Epochs: 3, Batch_Size: 16

VALUTAZIONE TRAINING:
Training Loss: 0.3700, Training Accuracy: 91.6935%, Training Precision: 93.3614%, Training AUC: 0.9770

VALUTAZIONE TESTING:
Test Loss: 0.6727, Test Accuracy: 63.5484%, Test Precision: 65.6475%, Test AUC: 0.6843
------------------
STATO CONFIGURAZIONE:
Pooling: MaxPooling, Dense_neurons: 256, Optimizer: adam, Epochs: 3, Batch_Size: 16

VALUTAZIONE TRAINING:
Training Loss: 0.8751, Training Accuracy: 80.5645%, Training Precision: 85.2021%, Training AUC: 0.8691

VALUTAZIONE TESTING:
Test Loss: 0.8646, Test Accuracy: 64.8387%, Test Precision: 76.4121%, Test AUC: 0.6861
------------------
```

Grazie a questo processo, abbiamo individuato la configurazione per noi ottima, con delle relative considerazioni:

- **Pooling**: l'*Average Pooling* prestava tendenzialmente peggio rispetto al *Max Pooling*;
- **Numero di neuroni per layer**: un numero di neuroni inferiore a 256 conduce il modello all'*overfitting*, invece un numero maggiore comporta un costo computazionale elevato;
- **Optimizer**: altri optimizers oltre *Adam* presentano prestazioni pessime oppure causano l'esplosione del gradiente e di conseguenza anche della loss function;
- **Numero di epoche**: il numero ottimale di epoche si assesta tra tre e quattro, poiché un numero inferiore conduce all'*underfitting*, mentre un numero maggiore conduce molto velocemente all'*overfitting*;
- **Dimensione del batch**: una dimensione del batch minore di 16 non offriva buone prestazioni, mentre una maggiore rende l'algoritmo lento e da un costo computazionale elevato.

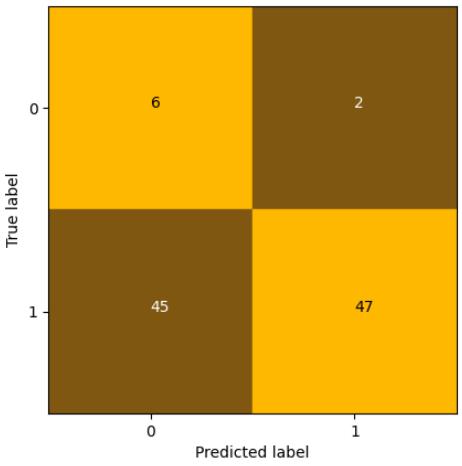
Dalla nostra sperimentazione empirica siamo inoltre giunti alla conclusione di dover utilizzare **tre layer convoluzionali**, di cui il primo con un *kernel* maggiore (le immagini sono piuttosto grandi), poiché ci portano ai risultati migliori.

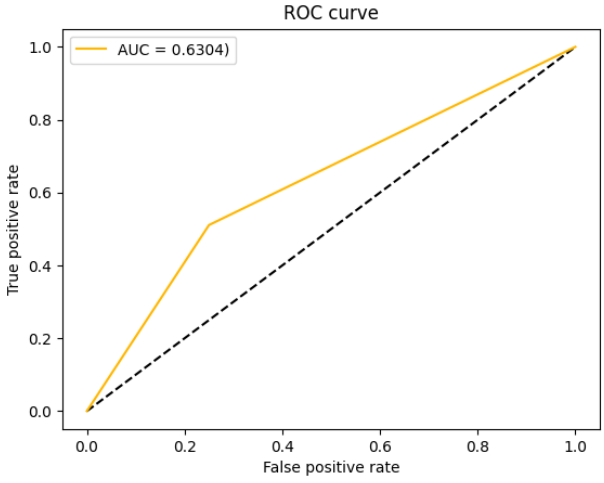
3.4 Evaluation

Con la configurazione summenzionata, i valori relativi ad ognuna delle metriche prima definite sono i seguenti:

- **Loss**: 0.7917;
- **Accuracy**: 53.0000;
- **Precision**: 95.9184;
- **AUC**: 0.7867.

Riportiamo di seguito la *confusion matrix* e la *ROC Curve*:





Ci rendiamo conto che le prestazioni della rete neurale da noi sviluppata non sono delle migliori, soprattutto per quanto riguarda l'accuratezza. Per poterla ottimizzare ulteriormente, sarebbe senza dubbio necessario un quantitativo maggiore di dati. Tuttavia, il quantitativo di dati riguardo questa particolare problematica è limitato, e volendo evitare l'applicazione della tecnica di *oversampling* discussa precedentemente, non abbiamo avuto alternative. Inoltre uno studio più attento e meticoloso della teoria delle reti neurali potrebbe far emergere concetti a noi attualmente non del tutto chiari, che potrebbero dar vita a nuove intuizioni per migliorare il lavoro svolto.

4 Classificatore

Una volta conclusa la valutazione della nostra CNN, siamo ora pronti a muoverci nella successiva fase, dedicata alla progettazione del classificatore per diagnosticare il CCD.

4.1 Data Understanding & Data Preparation

Il dataset utilizzato per l'addestramento del nostro classificatore è lo stesso utilizzato dalla nostra CNN, reperibile al link già citato in precedenza: https://www.kaggle.com/datasets/annajyang/beehive-sounds/data.

Prima di procedere, riteniamo necessario comprendere a pieno il problema da un punto di vista più tecnico, così da poter individuare le *features* migliori possibili che possano garantire la maggiore potenza predittiva allo scopo della previsione del CCD.
A tal proposito, è stato di estrema utilità lo studio della letteratura scientifica in merito. In particolare, abbiamo estrapolato le seguenti informazioni:

- Come già anticipato, **la regina deve essere presente** all'interno dell'arnia, poiché la sua assenza è il più importante indicatore di cattiva salute dell'arnia
- La **temperatura percepita all'interno dell'arnia** dev'essere compresa in un range che va da 33°C a 36°C. Preferiamo la temperatura percepita rispetto alla temperatura effettiva poiché le api portano il loro ambiente nel range ottimale attraverso il cambiamento dell'umidità (difatti, quando decidono di aumentare l'umidità, esse portano delle particelle d'acqua all'interno dell'arnia) [3];
- La **differenza tra la temperatura percepita interna rispetto alla temperatura percepita esterna all'arnia** è altrettanto importante per affrontare il nostro problema. Questo è dovuto al fatto che, come abbiamo suggerito al punto precedente, le api si impegnano per rimanere nel range ottimale. Di conseguenza, nel momento in cui la temperatura è anomala, è bene che questa differenza non sia troppo vicino allo zero poiché significherebbe che le api non stanno reagendo di conseguenza, il che significa che l'arnia potrebbe essere potenzialmente abbandonata;
- Un'ultima caratteristica importante è il **peso dell'arnia**, il quale dovrebbe essere compreso in un range che va da 25kg a 135kg. Un peso inferiore suggerirebbe che l'arnia è potenzialmente abbandonata o in via di abbandono, data la scarsa presenza di api e viveri. D'altra parte, un peso maggiore suggerirebbe che vi è un'eccessiva quantità di viveri all'interno dell'arnia, il che suggerisce che questa è potenzialmente abbandonata. [4][5]

Questi quattro aspetti rappresenteranno quindi le *features* del nostro problema.

Osservando il dataset, tuttavia, ci vengono incontro quelle che sono le prime e le più grandi problematiche che abbiamo dovuto necessariamente affrontare:

- **Non è presente la colonna relativa alle temperature percepite esterne ed interne all'arnia**. A tal proposito, abbiamo calcolato noi stessi i valori attraverso le misurazioni di temperatura e umidità provenienti dal dataset per ogni specifica arnia e dell'esterno;
- **Non è presente la colonna relativa al peso delle arnie**. Purtroppo, non abbiamo potuto risolvere questo problema, in quanto non potevamo applicare alcuna tecnica di *data imputation* su tale caratteristica, essendo questa non correlata agli altri dati a nostra disposizione. Per questa ragione non avevamo alternative se non ignorare il peso all'interno del nostro modello. Questo ha inevitabilmente dato vita ad un modello sub-ottimo che non rappresenta nel migliore dei modi una soluzione applicabile nel mondo reale;
- **Non è presente la colonna relativa alla presenza del CCD**, la quale è la variabile dipendente che il modello deve predire. In questo caso siamo stati costretti a risolvere il problema, definendo un'**euristica** concepita tramite lo studio della lettura scientifica. Per fare ciò, è stato però necessario anticipare qualche step della *Data Preparation*.

Alle finalità del nostro problema quindi, le colonne del dataset che sono di nostro interesse sono "*hive temp*", "*hive humidity*", "*weather temp*", "*weather humidity*" e "*queen presence*".

Prima di definire e applicare l'euristica e ottenere la colonna relativa al rischio di CCD, abbiamo verificato la presenza di eventuali **valori nulli** all'interno del nostro dataset, poiché potrebbero compromettere i calcoli da svolgere. Per nostra fortuna, le colonne i cui valori sono nulli sono di poco interesse, se non per quattro istanze della colonna "*weather temp*". Poiché sono pochissime righe in relazione all'intero dataset, non aveva senso applicare una tecnica di *data imputation*, è bastato rimuoverle.

Successivamente , abbiamo calcolato la temperatura percepita interna ed esterna utilizzando l'indice Humidex [6] servendoci delle seguenti formule:

$$e = (6.112 \cdot 10^{\frac{7.5 \cdot T}{237.7+T}} \cdot \frac{U}{100})$$

$$H = T + (\frac{5}{9} \cdot (e - 10))$$

dove T è la temperatura, U è l'umidità, e è la tensione di vapore dell'aria e H è la temperatura percepita. Questo calcolo non ha rappresentato un ostacolo poiché, come abbiamo già detto, è stato possibile ricavare la temperatura e l'umidità dal dataset di partenza e da queste calcolare la tensione di vapore dell'aria.

Una volta fatto questo, abbiamo calcolato le *features* sopra citate e abbiamo proseguito eliminando tutte le colonne non utili alle finalità del nostro problema, comprese quelle utilizzate per i calcoli, così da avere un dataset più pulito. Il risultato è il seguente:

index	queen presence	apparent hive temp	apparent temp diff
0	1	41.07	9.86
1	1	37.78	7.470000000000002
2	0	32.95	4.470000000000002
3	0	33.85	7.260000000000002
4	0	33.28	8.630000000000003
5	0	31.94	10.040000000000003
6	0	27.5	8.18
7	0	25.96	7.75
8	0	24.76	7.350000000000001
9	0	24.38	7.800000000000001
10	0	23.81	8.44
11	0	23.67	8.450000000000001
12	0	23.4	8.919999999999998
13	0	22.69	8.820000000000002
14	0	22.42	8.950000000000001
15	0	24.12	10.97
16	0	27.57	12.65
17	0	29.45	11.629999999999999
18	0	33.25	13.07
19	0	36.37	13.479999999999997
20	0	38.63	11.970000000000002
21	0	42.67	13.23
22	0	43.09	11.460000000000004
23	0	40.48	7.909999999999997
24	0	38.95	5.790000000000006
...	...	...	...

A questo punto, finalmente, abbiamo potuto applicare la nostra euristica di *labeling*, che abbiamo deciso di chiamare ***Euristica di Aristeo***, il dio greco dell'apicoltura. A tal proposito, spieghiamo come abbiamo ragionato per stabilire i valori ottimali di ogni nostra *feature*:

- **Presenza della regina**: se la regina non è presente nell'arnia, probabilmente qualcosa di molto grave sta avvenendo al suo interno. Infatti, questo è il maggior indicatore di possibile CCD e più in generale di cattiva salute dell'arnia. Se invece la regina è presente, ignoriamo questo indicatore poiché, da questo punto di vista, l'arnia è in uno stato di salute ottimale;
- **Temperatura percepita interna all'arnia**: l'intervallo ottimale è compreso tra 33 e 36, ma poiché necessitiamo di bande di guardia, consideriamo 32 e 37. In particolare, la temperatura alta è considerata più critica di quella bassa;
- **Differenza tra temperatura percepita esterna e interna all'arnia**: dovrebbe avere un valore il più lontano possibile dallo 0, poiché significherebbe che la temperatura esterna è molto simile alla temperatura interna. Ciò suggerirebbe le api non sono in condizioni di intervenire o che l'arnia è stata probabilmente abbandonata o in via di abbandono. Questo valore dovrebbe essere considerato solo nel momento in cui la temperatura interna sia fuori dall'intervallo prestabilito.

L'euristica, nello specifico, produce una probabilità in base a quanto appena definito. Dopodiché, tale probabilità verrà "lanciata" per produrre il *labeling* effettivo. Questo è stato fatto poiché il CCD non è un fenomeno prettamente deterministico e quindi abbiamo voluto creare una leggera perturbazione all'interno dei dati, nella speranza che il modello impari al meglio le relazioni nascoste all'interno di ciò che è stato individuato nella letteratura scientifica. Inoltre, questo è un modo per dare senso all'applicazione del Machine Learning, in quanto l'euristica non è applicabile a ritroso.

Di seguito il codice eseguito per effettuare il *labeling* del dataset:

```
def AristeoHeuristic(dataset):
    ccd_list = []

    # Iterazione sull'intero dataset
    for index, row in dataset.iterrows():

        # Calcolo inerente alla temperatura interna
        internal_temp_coeff = 0
        hive_temp = row["apparent hive temp"]

        if hive_temp < 32:  # Se la temperatura è al di sotto del range prestabilito...
            # Computazione del coefficiente in base a quanto la temperatura dista dall'estremo inferiore del range,
            # moltiplicato successivamente per una costante
            internal_temp_coeff = (32 - hive_temp) * 1.5
        elif hive_temp > 37:   # Se la temperatura è al di sopra del range prestabilito...
            # Computazione del coefficiente in base a quanto la temperatura dista dall'estremo superiore del range,
            # moltiplicato successivamente per una costante maggiore rispetto alla costante precedente, poiché
            # il rischio è maggiore all'aumento della temperatura
            internal_temp_coeff = (hive_temp - 37) * 2.5
        else:  # Se la temperatura si trova all'interno del range prestabilito...
            # Coefficiente nullo, poiché tale valore dovrebbe essere considerato solo nel momento in cui la
            # temperatura si trovi al di fuori del range
            internal_temp_coeff = 0


        # Calcolo inerente alla differenza di temperatura
        temp_diff_coeff = 0

        if hive_temp >= 32 and hive_temp <= 37:  # Se la temperatura si trova all'interno del range prestabilito...
            # Coefficiente nullo, poiché tale valore dovrebbe essere considerato solo nel momento in cui la
            # temperatura si trovi al di fuori del range
            temp_diff_coeff = 0
        else:  # Se la temperatura non si trova all'interno del range prestabilito...
            # Più la differenza di temperatura è vicina allo 0, maggiore sarà il valore del coefficiente
            temp_diff_coeff = 40 - abs(row["apparent temp diff"] * 0.8)


        # Calcolo inerente alla presenza della regina
        queen_coeff = (1 - row["queen presence"]) * 150  # Pari a 0 se la regina è presente, 150 altrimenti


        # Calcolo della probabilità sommando i tre coefficienti ottenuti al numeratore e i relativi "casi peggiori" al
        # denominatore. In altri termini, al denominatore sono riportati i pesi più alti
        ccd_probability = (internal_temp_coeff + temp_diff_coeff + queen_coeff) / (150 + 60 + 40)

        # Poichè l'euristica non è assoluta, abbiamo deciso di "lanciare" questa probabilità così da causare un
        # minimo di perturbazione all'interno della label. Così facendo, creiamo un'euristica non deterministica
        # che dà senso all'applicazione del Machine Learning
        random_num = random.random()
        ccd = random_num <= ccd_probability
        ccd_list.append(int(ccd))

    dataset["CCD"] = pd.Series(ccd_list)
```

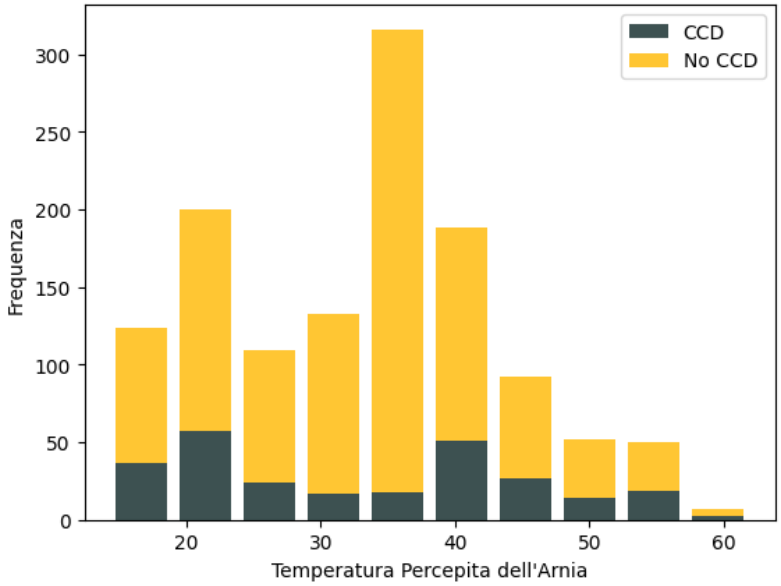
Il seguente è il dataset dopo aver applicato la nostra euristica:

index	queen presence	apparent hive temp	apparent temp diff	CCD
0	1	41.07	9.86	1
1	1	37.78	7.470000000000002	0
2	0	32.95	4.470000000000002	1
3	0	33.85	7.260000000000002	0
4	0	33.28	8.630000000000003	1
5	0	31.94	10.040000000000003	1
6	0	27.5	8.18	1
7	0	25.96	7.75	1
8	0	24.76	7.350000000000001	0
9	0	24.38	7.800000000000001	1
10	0	23.81	8.44	0
11	0	23.67	8.450000000000001	0
12	0	23.4	8.919999999999998	1
13	0	22.69	8.820000000000002	1
14	0	22.42	8.950000000000001	1
15	0	24.12	10.97	1
16	0	27.57	12.65	0
17	0	29.45	11.629999999999999	0
18	0	33.25	13.07	0
19	0	36.37	13.479999999999997	0
20	0	38.63	11.970000000000002	1
21	0	42.67	13.23	1
22	0	43.09	11.460000000000004	0
23	0	40.48	7.909999999999997	0
24	0	38.95	5.790000000000006	1
...	...	...	...	...

Una volta ottenuto il dataset etichettato, abbiamo potuto continuare con la fase di *Data Understanding*. Per prima cosa, abbiamo controllato le **distribuzioni delle *features* in relazione alla nostra *label*** appena ottenuta. Ci aspettiamo di trovare risultati simili a quanto studiato all'interno degli articoli scientifici sul tema, in cui viene riportato che circa il 25% delle arnie soffre di CCD [7][8] e le distribuzioni dei valori seguono quanto definito precedentemente.

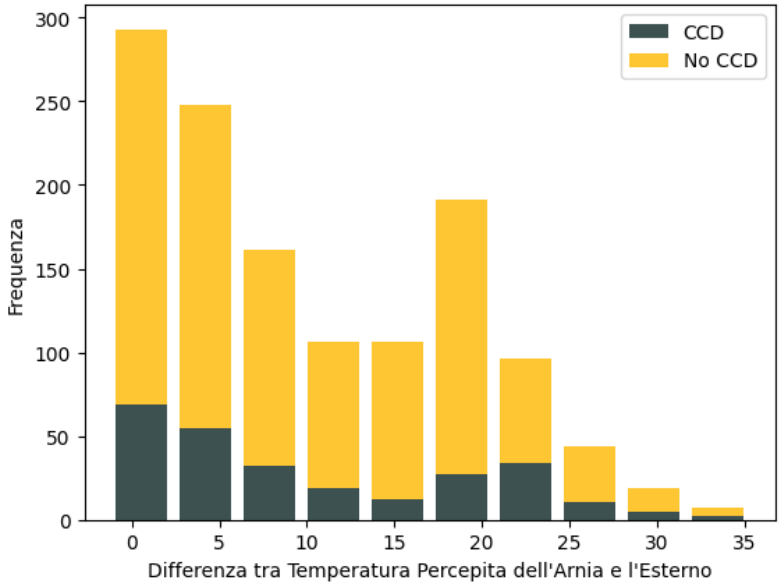
Quello che faremo sarà quindi visualizzare i grafici relativi ad ogni feature a nostra disposizione, in modo da visualizzare graficamente la loro distribuzione in base alla *label* ottenuta.

Iniziamo riportando l'istogramma della temperatura percepita interna all'arnia in relazione alla presenza o all'assenza del CCD:



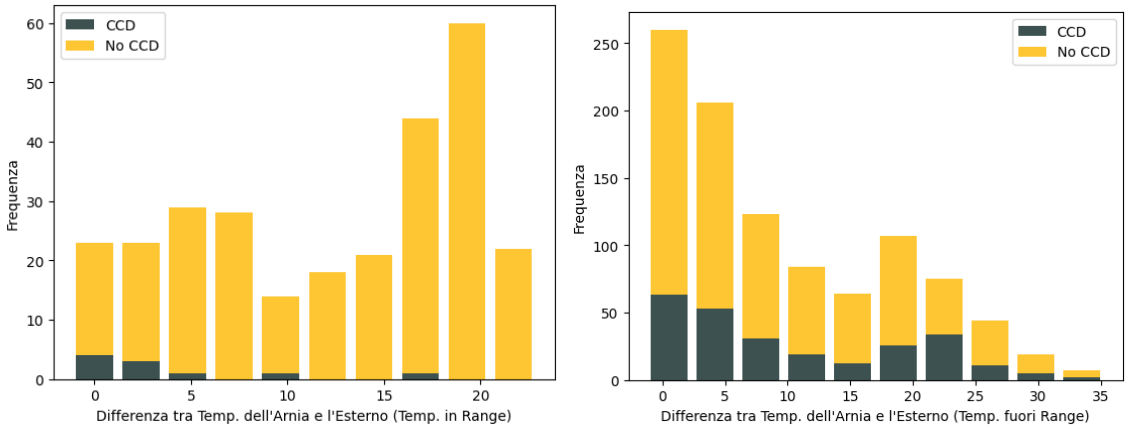
Come ci aspettavamo, **al centro della distribuzione la percentuale dei casi di CCD è minore rispetto agli estremi**, dove la percentuale di casi positivi è maggiore. Questo è realistico poiché più la temperatura dell'arnia si trova all'interno dell'intervallo prestabilito, maggiori sono le probabilità che le api siano in salute e, di conseguenza, minore è la probabilità che l'arnia sia soggetta al CCD.

Vediamo ora l'istogramma inerente alla differenza tra la temperatura percepita interna ed esterna all'arnia:



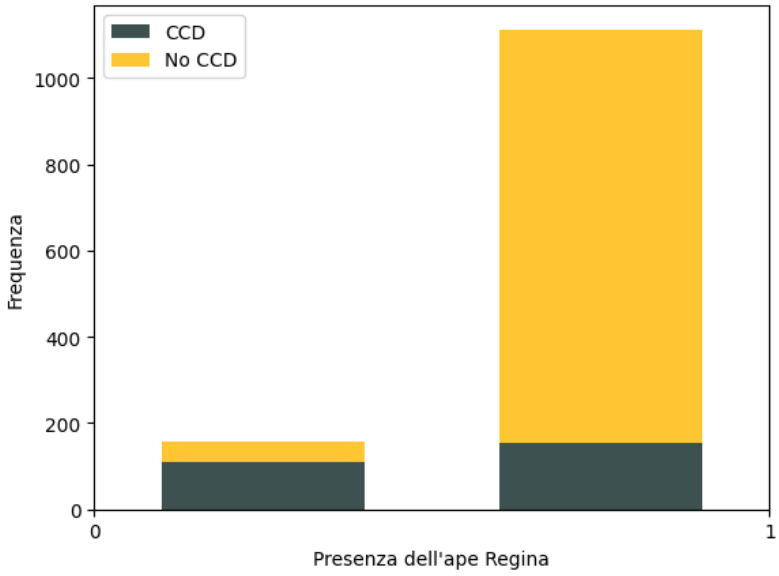
Il grafico ci riporta, come sospettavamo, una **frequenza maggiore di CCD quando ci avviciniamo allo 0**. C'è però da dire che l'euristica ragiona anche in relazione alla temperatura interna (infatti se ci troviamo nel range, la differenza tra interno ed esterno non viene mai considerata).

Vediamo ora gli istogrammi di nuovo inerenti alla differenza tra la temperatura percepita interna ed esterna all'arnia, ma nel momento in cui la temperatura interna si trova rispettivamente nel range prestabilito e al di fuori del range prestabilito:



Come immaginavamo, **quando la temperatura si trova nel range abbiamo pochi casi di CCD**, distribuiti maggiormente quando siamo vicini allo 0. Analogamente, **quando la temperatura si trova al di fuori del range abbiamo molti più casi di CCD**, in particolare quando ci troviamo verso gli estremi della distribuzione. Questo è dovuto probabilmente al fatto che una differenza alta in aggiunta ad una temperatura fuori range causa molto stress alle api, mentre una differenza tendente allo 0 trova spiegazione in quanto detto prima.

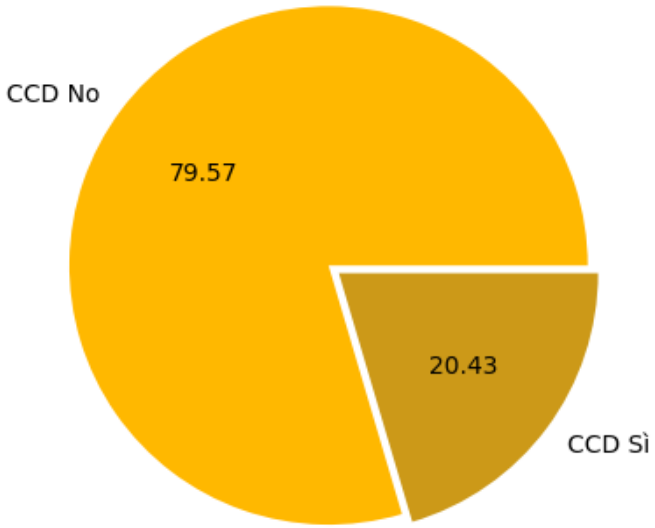
Infine, vediamo la frequenza del CCD in relazione alla presenza o all'assenza della regina:



In linea con le nostre aspettative, **quando la regina è assente la frequenza dei casi di CCD è significativamente più alta** in proporzione alle misurazioni, difatti questo è il fattore più incidente anche all'interno dell'euristica.

Dunque, da queste considerazioni, abbiamo definito la nostra euristica soddisfacente, oltre ad aver già delineato le relazioni tra *features* e *label*. Il passo successivo consiste nell'**estrazione di una parte del dataset, in maniera pseudocasuale, da utilizzare nella fase di validazione del modello**, proprio come abbiamo fatto per la CNN. Tale operazione è stata svolta in questo momento perché non volevamo mostrare questi dati all'algoritmo, per poterlo testare in maniera corretta.

A questo punto, abbiamo controllato il bilanciamento della classe da predire. Osserviamo il seguente grafico a torta:



Come previsto, il grafico è in linea con quanto detto prima, infatti abbiamo un **forte sbilanciamento** all'interno del nostro dataset. Per affrontare tale problematica, abbiamo confrontato due opzioni:

- Effettuare **esclusivamente un *undersampling* della classe maggioritaria** sino a raggiungere lo stesso numero delle istanze della classe minoritaria. Questa soluzione è molto semplice, tuttavia applicandola avremmo tra le mani un basso quantitativo di dati;
- Effettuare **sia *undersampling* della classe maggioritaria che *oversampling* della classe minoritaria**. In particolare, cercheremmo di ottenere 500 istanze per ognuna delle due classi, così da avere un dataset perfettamente bilanciato. Questa soluzione sarebbe ideale, tuttavia ciò rappresenta una sfida di maggiore difficoltà poiché la sua applicazione è più impegnativa e dovremmo stare attenti a non perturbare la distribuzione dei nostri dati.

Per individuare l'opzione migliore, abbiamo deciso di implementare e testare entrambe.

La **prima soluzione**, l'esclusiva applicazione dell'*undersampling*, non richiede particolari ragionamenti. La applichiamo blandamente e ne salviamo il risultato, così da testarlo nella prossima fase.

Passiamo alla **seconda soluzione**, che non risulta essere semplice come la prima, cominciando dall'*oversampling*. Poiché il quantitativo di dati in merito all'argomento è molto limitato, abbiamo deciso di optare per la tecnica di ***Data Synthesis***, ovvero il processo di generazione di dati artificiali, quindi non direttamente provenienti da una raccolta di dati del mondo reale. In particolare, abbiamo generato nuovi campioni basandoci su quelli già esistenti nel nostro dataset. Riguardo a ciò, abbiamo diligentemente preservato un livello di realismo nelle nuove istanze generate, introducendo delle variazioni pseudo-casuali e controllate. Per farlo, abbiamo stabilito dei template per generare dei valori realistici e basati sulle misurazioni e proporzioni sviluppate e osservate precedentemente:

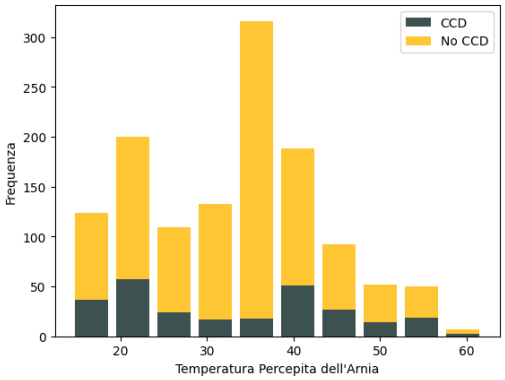
- **Presenza della regina**: osservando la frequenza del CCD in relazione alla presenza o all'assenza della regina, i campioni la cui label del CCD è "True" presentano una regina assente nell'80% dei casi. Per questo motivo, i nuovi campioni generati avranno la colonna relativa alla presenza della regina di valore pari a 0 con una probabilità dell'80%;
- **Temperatura percepita interna all'arnia**: innanzitutto, dobbiamo stabilire le probabilità in cui la temperatura percepita della nuova istanza generata cada al di fuori oppure all'interno del range prestabilito. Poiché è ovviamente più grave che la temperatura si trovi fuori dal range, la probabilità di generare una temperatura sballata sarà più alta. Successivamente, per quanto riguarda i valori da generare effettivi, al corrispettivo estremo del range superato aggiungeremo o sottrarremo rispettivamente un numero decimale da 0 a 15. Avremmo potuto andare oltre allargando questo intervallo di valori, e da un certo punto di vista sarebbe stato vantaggioso, poiché più ci allontaniamo dal range ottimale e maggiore è la possibilità di ottenere un'istanza con CCD positivo. Tuttavia, un intervallo più grande potrebbe generare numeri troppo vicini agli estremi, quindi più lontani dalle istanze che popolano la maggior parte del dataset. In altri termini, non avremmo ottenuto risultati realistici e conformi a ciò che è stato stabilito in precedenza;
- **Differenza tra temperatura percepita esterna e interna all'arnia**: questo valore può cambiare in base alla temperatura percepita interna all'arnia che abbiamo appena calcolato. In particolare, se la temperatura generata si trova nel range ottimale prestabilito, questo valore oscillerà tra 0 e 1 oppure tra 2 e 25, con una probabilità maggiore di rientrare nel primo intervallo poiché più grave. Altrimenti, se la temperatura generata è fuori dal range, questo valore potrà oscillare tra 0 e 10, 11 e 20 oppure 21 e 25. La probabilità che il valore rientri nel primo, secondo o terzo intervallo segue esattamente l'ordine in cui sono stati dichiarati (P(0-10) > P(11-20) > P(21-25)). Questo perché, come abbiamo già definito, più la differenza di temperatura è vicina allo zero e maggiore sarà il rischio di CCD. Questi intervalli sono stati stabiliti seguendo lo stesso ragionamento sviluppato al passo precedente.

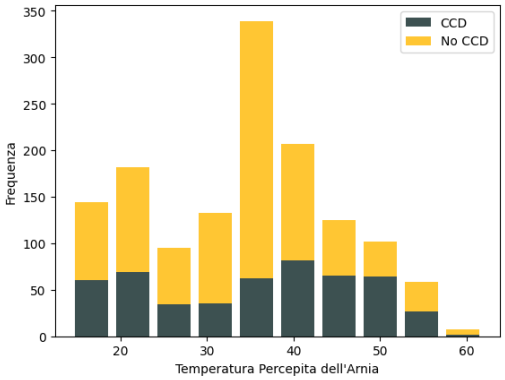
Dopo aver generato i tre valori di un'istanza, abbiamo riapplicato l'euristica per ottenerne la relativa *label*.

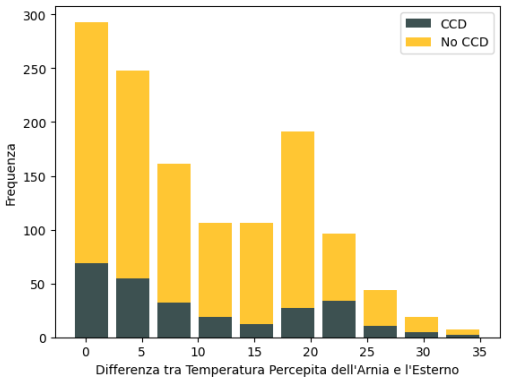
Per capire meglio il nostro ragionamento, ne mostriamo l'applicazione pratica attraverso il seguente codice:

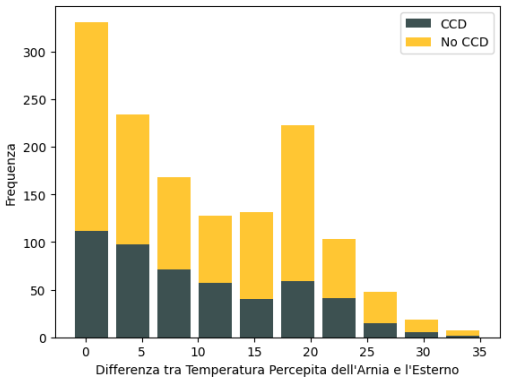
```
# Definizione del numero di righe del dataset in cui il CCD ha valore 1
CCD_rows = dataset["CCD"].value_counts()[1]

while CCD_rows < 500:

    # Calcolo della presenza dell'ape regina
    if random.random() >= 0.80:
        queen_mock = 0
    else:
        queen_mock = 1

    # Calcolo della temperatura percepita
    if random.random() >= 0.60:  # Probabilità del 40% che la temperatura sia al di sopra del range
        temp_mock = 37 + random.randint(0, 15) + round(random.random(), 2)  # Valore al di sopra del range da 0 a 15
    elif random.random() <= 0.40:  # Probabilità del 40% che la temperatura sia al di sotto del range
        temp_mock = 32 - random.randint(0, 15) - round(random.random(), 2)  # Valore al di sotto del range da 0 a 15
    else:  # Probabilità del 20% che la temperatura sia all'interno del range
        temp_mock = random.randint(32, 37) + round(random.random(), 2)  # Valore all'interno del range da 32 a 36

    # Inutile continuare poiché valori del genere genererebbero una probabilità di CCD pari a 0
    if 32 <= temp_mock <= 37 and queen_mock == 1:
        continue

    # Calcolo della differenza tra temperatura interna ed esterna
    if 32 <= temp_mock <= 37:  # Se la temperatura generata è in range...
        if random.random() >= 0.30:  # Probabilità del 70% che la differenza di temperatura sia tra 0 e 1
            temp_diff_mock = random.randint(0, 1) + round(random.random(), 2)
        else:  # Probabilità del 30% che la differenza di temperatura sia tra 2 e 25
            temp_diff_mock = random.randint(2, 25) + round(random.random(), 2)
    else:  # Se la temperatura generata è fuori range...
        if random.random() >= 0.40:  # Probabilità del 60% che la differenza di temperatura sia tra 0 e 10
            temp_diff_mock = random.randint(0, 10) + round(random.random(), 2)
        elif random.random() <= 0.10:  # Probabilità del 10% che la differenza di temperatura sia tra 21 e 25
            temp_diff_mock = random.randint(21, 25) + round(random.random(), 2)
        else:  # Probabilità del 30% che la differenza di temperatura sia tra 11 e 20
            temp_diff_mock = random.randint(11, 20) + round(random.random(), 2)

    # Creazione di un dizionario con 3 chiavi associate ai 3 valori calcolati in precedenza
    row = {"queen presence": [queen_mock], "apparent hive temp": [temp_mock], "apparent temp diff": [temp_diff_mock]}
    # Conversione del dizionario in un DataFrame
    row = pd.DataFrame.from_dict(row)

    # Applicazione della nostra euristica
    AristeoHeuristic(row)

    # Concatenazione dei DataFrame per ottenere il dataset di campioni con CDD = True
    if int(row["CCD"]) == 1:
        dataset = pd.concat([dataset, row], axis=0)
        CCD_rows+=1
```

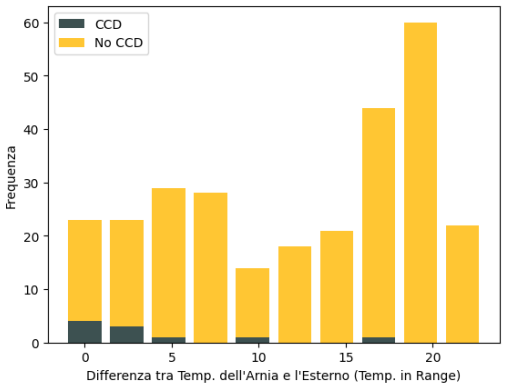
Per verificare la correttezza della tecnica applicata, visualizziamo nuovamente i grafici inerenti ad ognuna delle tre features per controllare che le distribuzioni siano rimaste simili alle precedenti. Sulla sinistra troviamo i grafici già visti in precedenza, mentre sulla destra gli stessi grafici dopo l'applicazione dell'*oversampling*:

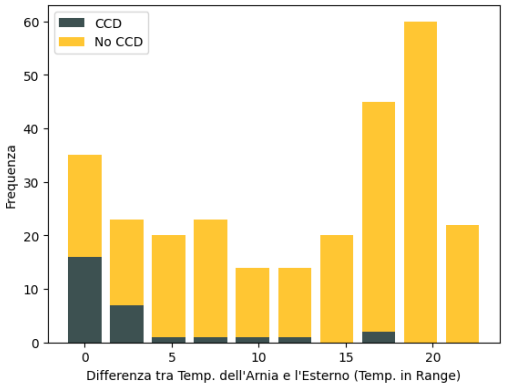


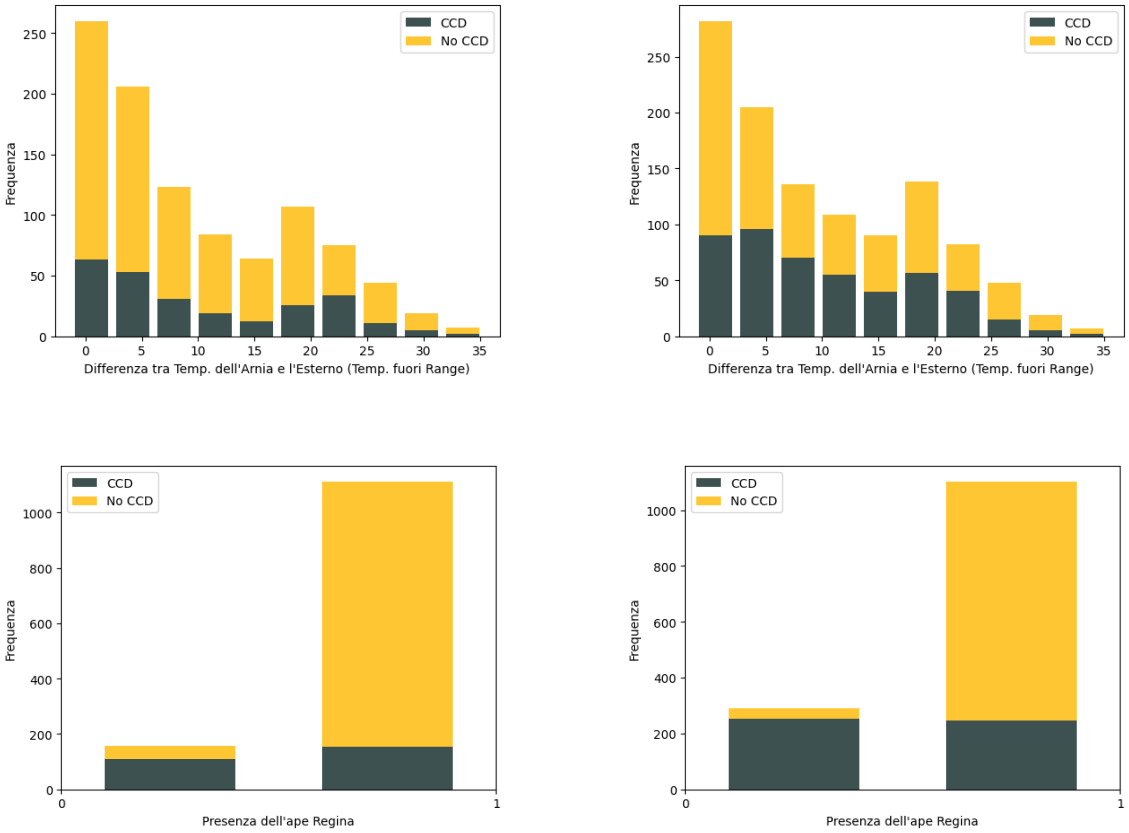












Possiamo vedere come la distribuzione della *label* del CCD non è cambiata notevolmente. Possiamo quindi ritenerci soddisfatti dei risultati ottenuti, poiché realistici e conformi al dataset di partenza.

Per rendere i dati pronti per il training, abbiamo concluso con l'*undersampling* della classe maggioritaria e la conversione dei dati in un formato leggibile ai modelli che andremo ad addestrare e testare. In particolare, come per la CNN, abbiamo convertito i dati in array NumPy, questa volta su due training set separati, di cui uno con oversampling e l'altro con solo undersampling.

```
X_train_US = ccd_dataset_US[["queen presence", "apparent hive temp", "apparent temp diff"]].to_numpy()
y_train_US = ccd_dataset_US[["CCD"]].to_numpy()
X_train_OS = ccd_dataset_OS[["queen presence", "apparent hive temp", "apparent temp diff"]].to_numpy()
y_train_OS = ccd_dataset_OS["CCD"].to_numpy()
```

4.2 Data Modeling

Una volta che i dati sono stati individuati e resi pronti all'utilizzo, non ci rimane che impiegarli per l'addestramento del nostro modello.
Poiché la nostra è un'istanza di un problema di classificazione binaria, abbiamo deciso di utilizzare, testare e confrontare i seguenti algoritmi: ***Decision Tree***, ***Random Forest***, ***Naive Bayes*** e ***K-Nearest Neighbors***.

Durante la fase di training utilizzeremo, come per la CNN, una ***stratified k-fold validation***, questa volta con k = 10, per diagnosticare eventuali problemi di *overfitting* e le performance generali dei modelli. Inoltre, ci teniamo a ricordare che utilizzeremo due dataset differenti: uno generato dall'esclusiva applicazione dell'*undersampling*, l'altro generato dall'applicazione dell'*oversampling* e *undersampling*.

Per capire quale dei due dataset e quale modello vale la pena utilizzare, per la fase di valutazione abbiamo analizzato una per una le metriche ottenute:

Algoritmo	Accuracy US	Accuracy OS	Precision US	Precision OS	Recall US	Recall OS	F1 Score US	F1 Score OS
Decision Tree	0.6629	0.7030	0.6701	0.7050	0.6609	0.704	0.6625	0.7034
Random Forest	0.6863	0.7390	0.6916	0.7540	0.6832	0.7140	0.6850	0.7322
Naive Bayes	0.7140	0.7290	0.9197	0.9139	0.4759	0.508	0.6175	0.6514
K-Nearest Neighbors	0.6015	0.6760	0.6036	0.7035	0.5819	0.6160	0.5911	0.6544

Dai risultati ottenuti, si evince chiaramente che lavorare con il dataset ottenuto dall'applicazione dell'***oversampling*** **produce risultati migliori**. Pertanto, abbiamo proceduto con l'utilizzo di tale dataset per l'addestramento finale.

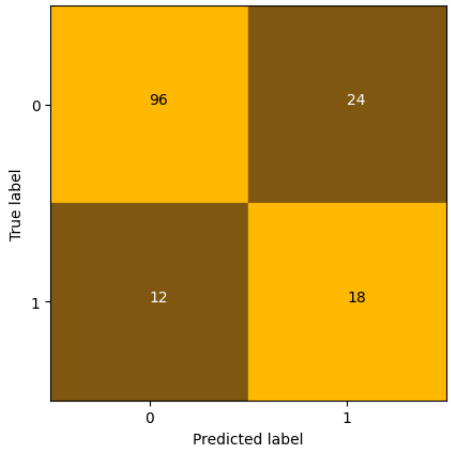
Oltre a ciò, osservando i risultati ottenuti, risulta evidente che il classificatore con le migliori prestazioni, facendo particolare attenzione alla *Recall* dati i notevoli impatti che i falsi negativi hanno sul nostro sistema (CCD non segnalato quando l'arnia, in realtà, ne è a rischio), è il ***Random Forest***, che risulta leggermente migliore rispetto al *Decision Tree*. Tuttavia, anche ***Naive Bayes*** non è da sottovalutare, poiché pur avendo una *Recall* più bassa presenta una *Precision* ottima. Vale la pena, quindi, osservare entrambi.
A tal proposito, abbiamo cercato di ottimizzarne le prestazioni ricorrendo al *model tuning*. In particolare, abbiamo utilizzato per entrambi la ***Grid Search*** per trovare la miglior configurazione di iperparametri alle finalità del nostro problema e produrre dei modelli ottimizzati. Avendo ora due modelli candidati, siamo passati alla fase di Evaluation, in cui abbiamo confrontato le loro performance.

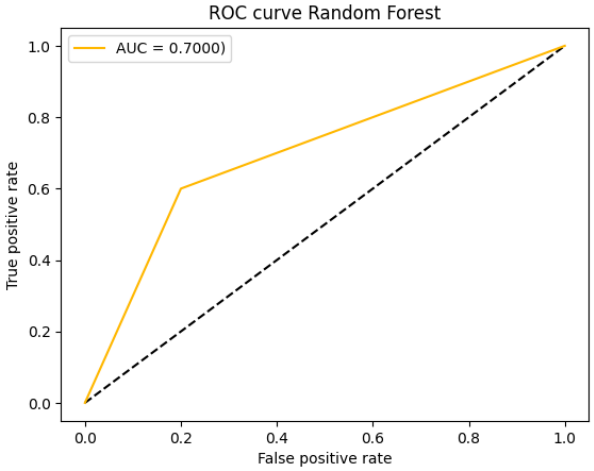
4.3 Evaluation

Per quanto riguarda il *Random Forest* ottimizzato, i valori ottenuti relativi ad ognuna delle metriche sono i seguenti:

- **Accuracy**: 0.76;
- **Precision**: 0.43;
- **Recall**: 0.60;
- **F1 Score**: 0.50.

Riportiamo di seguito la *confusion matrix* e la *ROC Curve*:

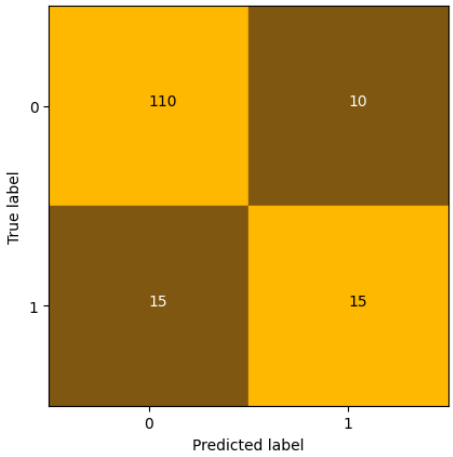


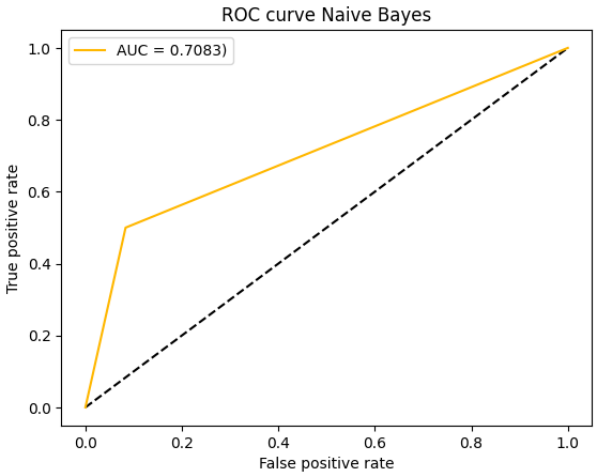


Per quanto riguarda *Naive Bayes* ottimizzato invece, i valori ottenuti relativi ad ognuna delle metriche sono i seguenti:

- **Accuracy**: 0.83;
- **Precision**: 0.60;
- **Recall**: 0.50;
- **F1 Score**: 0.54.

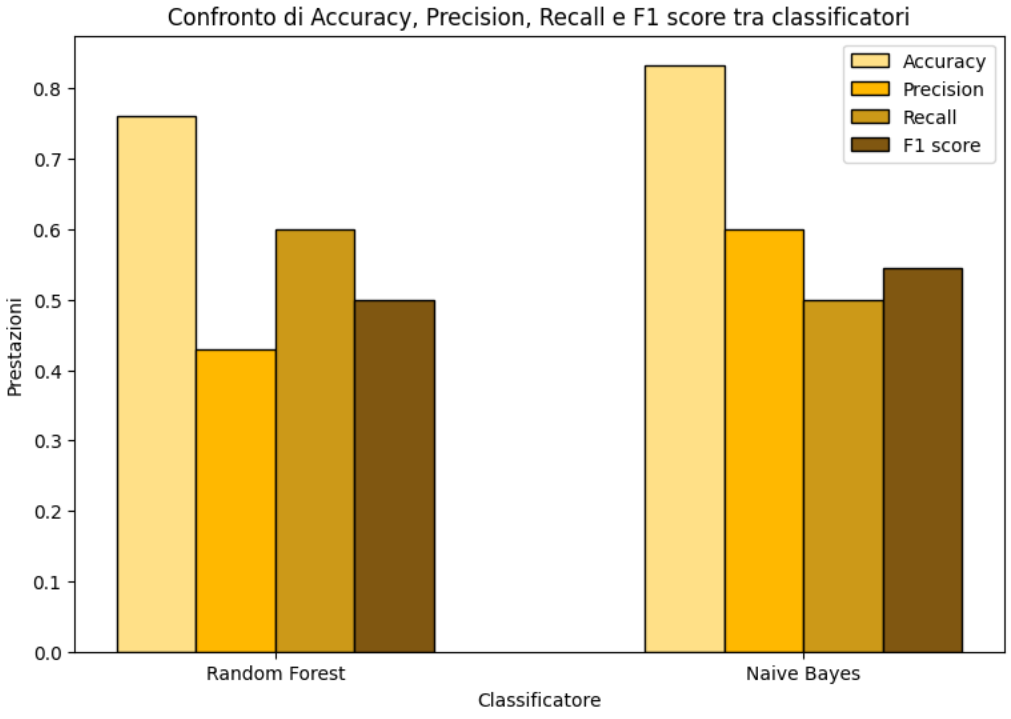
Riportiamo di seguito la *confusion matrix* e la *ROC Curve*:





Osserviamo che le prestazioni di *Naive Bayes* sono tendenzialmente migliori da tutti i fronti rispetto al *Random Forest*, ad eccezione della *Recall*, dove abbiamo prestazioni inferiori.

Per toglierci ogni dubbio, visualizziamo le metriche di entrambi i classificatori su un plot, così da osservare graficamente i diversi risultati ottenuti da ogni modello:



Da questo grafico possiamo trarre le nostre considerazioni:

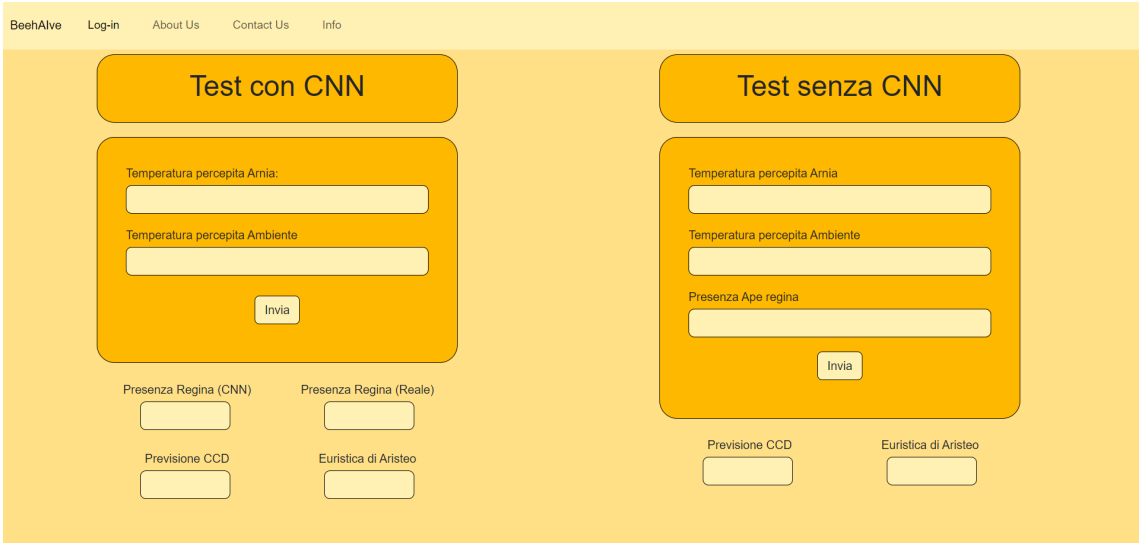
- *Random Forest* presenta dei risultati mediamente peggiori, tuttavia ha una *Recall* più alta, che è di particolare interesse al nostro problema;
- *Naive Bayes* presenta dei risultati mediamente migliori, tuttavia ha una *Recall* leggermente inferiore;

Dopo un'attenta analisi, basata anche sull'usabilità del modello, si è deciso di procedere con *Naive Bayes*. Questo perché abbiamo ritenuto opportuno prendere un modello che mediamente effettua meno predizioni errate, poiché risulterebbe frustrante per un apicoltore trovarsi di fronte ad un gran numero di falsi allarmi, e la percentuale di *Recall* persa non è così significativa da andare a discriminare completamente questa scelta.

5 Deployment

Abbiamo finalmente ottenuto i nostri modelli, quindi ora vanno messi in funzione così che possano supportare gli apicoltori nella lotta contro il CCD. Per fare ciò, abbiamo optato per l'utilizzo di ***Flask***, in modo da stabilire una comunicazione attraverso HTTP. Questa scelta ci permette inoltre di racchiudere il sistema in un modulo importabile in altri sistemi software, oltre che metterli in comunicazione con sensori IoT.

Per testare il modello ci siamo appoggiati all'architettura già esistente della piattaforma web beehAIve, sviluppata con Java Spring, realizzando un piccolo *driver*. Per comunicare con *Flask* abbiamo implementato un ***Adapter*** che converte il nostro input processato in Java in formato JSON, prima di inviarlo attraverso HTTP al modello, per poi ricevere la sua risposta e riconvertirla in un formato leggibile dall'utente.



6 Conclusioni

Ora che i modelli sono conclusi, vi sono delle ultime considerazioni da fare. Seppur le prestazioni non sono ottimali, e in generale il modello risultante è sub-ottimo per definizione (stiamo escludendo il peso dell'arnia, di cui non avevamo il dato, e il labeling è basato su un'euristica da noi definita) possiamo ritenerci estremamente soddisfatti del lavoro svolto. Abbiamo avuto modo di applicare numerose tecniche che si sono rivelate utili nell'ottenere, anche se di poco, un modello migliore. In particolare, abbiamo avuto la possibilità di approfondire argomenti come la Data Synthesis, basata su nostre intuizioni, osservazioni e studi, rivelatesi poi corrette, la creazione degli spettrogrammi, le reti neurali convoluzionali, il *tuning* con *Grid Search* e molto altro ancora, oltre che affinare le nostre abilità da programmatori. Volendo migliorare il lavoro svolto, sicuramente sarebbe necessario avere un dataset etichettato da degli esperti e non da un'euristica, oltre ad avere a disposizione la colonna relativa al peso dell'arnia, entrambe cose che sono purtroppo fuori dal nostro controllo (difatti il problema del CCD, seppur ben documentato nella letteratura scientifica, manca di dataset affidabili su cui lavorare).
Faremo grande tesoro di questa esperienza, di cui ci riteniamo molto orgogliosi.

Riferimenti bibliografici

[1] Wikipedia contributors. Colony collapse disorder, 2024.

[2] Vinicius D. Valerio, Rodolfo M. Pereira, Yandre M. G. Costa, Diego Bertolini, and Carlos N. Silla Jr. A resampling approach for imbalanceness on music genre classification using spectrograms. 2018.

[3] H. F. Abou-Shaara, A. A. Owayss, Y. Y. Ibrahim, and N. K. Basuny. A review of impacts of temperature and relative humidity on various activities of honey bees. *Insectes Sociaux*, 2017.

[4] Wayne Esaias and Howard County Maryland. Protocol for scale hive measurements of the honey bee nectar flow. 2006.

[5] William G. Meikle, Niels Holst, Théotime Colin, Milagra Weiss, Mark J. Carroll, Quinn S. McFrederick, Andrew B, and Barron. Using within-day hive weight changes to measure evironmental effects on honey bee colonies. *PLoS ONE*, 2018.

[6] Nimbus. L'indice di calore, quando l'umidità aumenta la sensazione di calore, 2003.

[7] Peyton Ferrier, Randal R. Rucker, Walter N. Thurman, and Michael Burgett. Economic effects and responses to changes in honey bee health. 2018.

[8] Dennis vanEngelsdorp, Jerry Hayes Jr, Robyn M Underwood, and Jeffery S Pettis. A survey of honey bee colony losses in the united states, fall 2008 to spring 2009. *Journal of Apicultural Research*, 2015.