

ModuloFIA: CardIACA

Giovanni Casaburi, Salvatore Basilicata,

Primo Vinicio Calabrese

January 2024

Indice

1 DEFINIZIONE DEL PROBLEMA	3
1.1 Introduzione	3
1.2 Obiettivi	3
1.3 Specifica dell'Ambiente	3
1.3.1 Specifica PEAS	3
1.4 Caratteristiche dell'Ambiente	4
2 ANALISI DEI DATI	5
2.1 Introduzione al Dataset	5
2.2 Analisi dei Dati	7
2.3 Analisi delle Relazioni tra i Dati	8
2.3.1 Variabili Numeriche	8
2.3.2 Variabili Categoriche	10
2.3.3 Matrice di correlazione	12
2.4 Facilità della reperibilità dei dati	13
3 Preparazione dei dati	14
3.1 Gestione degli Outliers	14

3.2	Feature Scaling	15
3.3	Data Balancing	17
4	Data Modeling	18
4.1	Definizione del tipo di problema	18
4.2	Configurazione per la validazione del modello	18
4.3	KNN (K-nearest neighbors)	19
4.3.1	Scelta del K	20
4.4	Support Vector Machine	23
4.5	Albero Decisionale	24
4.6	Validazione	26
4.6.1	Un Passo Avanti nella Validazione	26
4.6.2	Risultati di Validazione	27
4.7	Validazione con un Dataset più "Semplice"	27
5	Deployment	28
5.1	Implementazione Web	28
5.2	Aggiornamenti Dinamici	30
5.3	Fallback del modello	30
6	I Modelli non sono Medici	30
6.1	La precisione del modello nel mondo reale	31
6.2	Conclusioni	32

1 DEFINIZIONE DEL PROBLEMA

1.1 Introduzione

Il progetto CardIAca nasce dalla necessità di facilitare gli utenti nel ricevere in modo rapido e semplice una valutazione del rischio di un attacco cardiaco. Attualmente, per ottenere questa risposta, bisogna sottoporsi a numerose analisi mediche e poi attendere settimane per ottenere un riscontro. CardIAca si pone l'obiettivo di eliminare questo inconveniente, che, seppur semplice, è ciò che disturba maggiormente una persona. Il codice sarà reperibile sulla repo di [GitHub](#)

1.2 Obiettivi

Lo scopo di questo progetto è sviluppare un modulo di Intelligenza Artificiale basato su un modello di Machine Learning che, analizzando i dati ricevuti, sarà in grado di predire se una persona potrebbe essere soggetta a un attacco cardiaco. In ambito medico, l'elettrocardiogramma rappresenta una fonte affidabile per ottenere queste informazioni, ma talvolta non è l'unico elemento necessario per prevenire un attacco cardiaco, poiché la situazione può essere più complessa.

1.3 Specifica dell'Ambiente

1.3.1 Specifica PEAS

- **Perfomance:** La misura di prestazione utilizzata per valutare l'operato dell'agente sarà basata sulla correttezza delle diagnosi di attacco cardiaco effettuate.
- **Environment:** l'ambiente in cui opera l'agente sarà una piattaforma web per l'ambito sanitario nella quale potrà interagire con l'utente.
- **Actuators:** Gli attuatori che consentiranno all'agente di eseguire azioni saranno costituiti da semplici metodi in grado di effettuare inferenze sui dati e fornire un riscontro successivamente.

- **Sensors:** I sensori a disposizione dell'agente, che riceveranno gli input, saranno rappresentati da caselle di testo sul lato utente, i cui dati verranno inseriti in appositi contenitori sul lato dell'agente.

1.4 Caratteristiche dell'Ambiente

- **Parzialmente osservabile:** All'agente sono disponibili solo alcune informazioni utili per effettuare la diagnosi, mentre altre non sono accessibili.
- **Deterministico:** Lo stato successivo dell'ambiente dipenderà solamente dalle azioni compiute dall'agente, poiché sarà l'unico a poter agire liberamente durante la predizione.
- **Sequenziale:** Le azioni successive dell'agente dipenderanno da quelle compiute in precedenza, poiché sono basate sulle inferenze dai dati con cui esiste una forte dipendenza.
- **Discreto:** Il numero di azioni disponibili per l'agente sarà limitato, principalmente perché dovrà rispondere con un **Sì** o **No** in base alle informazioni ottenute dai dati.
- **Statico:** Mentre l'agente prende decisioni, l'ambiente in cui opera rimane immutato.
- **Singolo-Agente:** L'ambiente in cui opera l'agente non prevede la presenza di altri agenti che possano ostacolarlo o aiutarlo durante la predizione dell'attacco cardiaco.

2 ANALISI DEI DATI

2.1 Introduzione al Dataset

Il dataset è stato ottenuto in formato tabellare csv da una fonte che ha raccolto i dati medici dei pazienti con sospetta malattia. Il dataset contiene 303 osservazioni e 14 caratteristiche, di cui 13 sono indipendenti e una è la variabile di target. Le caratteristiche sono le seguenti:

- **Age**: età del paziente
- **sex**: sesso (1 maschio, 0 femmina)
- **cp**: tipo di dolore al torace (0 = angina tipica, 1 = angina atipica, 2 = dolore non anginoso, 3 = asintomatico)
- **trtbps**: pressione sanguigna a riposo in mmHg
- **chol**: livello di colesterolo in mg/dl
- **fbs**: livello di glucosio nel sangue a digiuno (1 = superiore a 120 mg/dl, 0 = inferiore a 120 mg/dl)
- **restecg**: risultato dell'ecocardiogramma a riposo (0=normale, 1= anomalia dell'onda ST-T, 2 = ipertrofia ventricolare sinistra)
- **thalacc**: frequenza cardiaca massima raggiunta in bpm
- **exng**: angina indotta dall'esercizio (1=sì, 0 = no)
- **oldpeak**: depressione del segmento ST indotta dall'esercizio rispetto al riposo
- **slp**: pendenza del segmento ST di picco durante l'esercizio (0 = ascendente, 1 = piatta, 2 = discendente)
- **caa**: numero di vasi principali colorati dalla fluoroscopia (da 0 a 3)
- **thall**: tipo di talassemia (0 = null, 1 = difetto fisso, 2 = normale, 3 = difetto reversibile)

- **output**:diagnosi della malattia cardiaca
(1 = >50% di diametro stenosi. Maggiore probabilità di malattia cardiaca,
0 = <50% di diametro stenosi. Minor probabilità di malattia cardiaca)

Si può notare che all'interno del dataset compaiono soltanto interi e float. Inoltre, nel dataset **non ci sono dati mancanti**, pertanto i dati sono già **strutturati**.

2.2 Analisi dei Dati

Analizzando le distribuzioni delle caratteristiche principali, si osserva che la variabile "età" ha una forma pressoché normale (a), con una moda intorno ai 50-60 anni. La variabile "sesso", invece, mostra una prevalenza di uomini (con valore 1) rispetto alle donne (b). La variabile "livello di colesterolo" segue una distribuzione normale (c), così come la "pressione sanguigna a riposo" (d), entrambe con una media nella norma della popolazione generale.

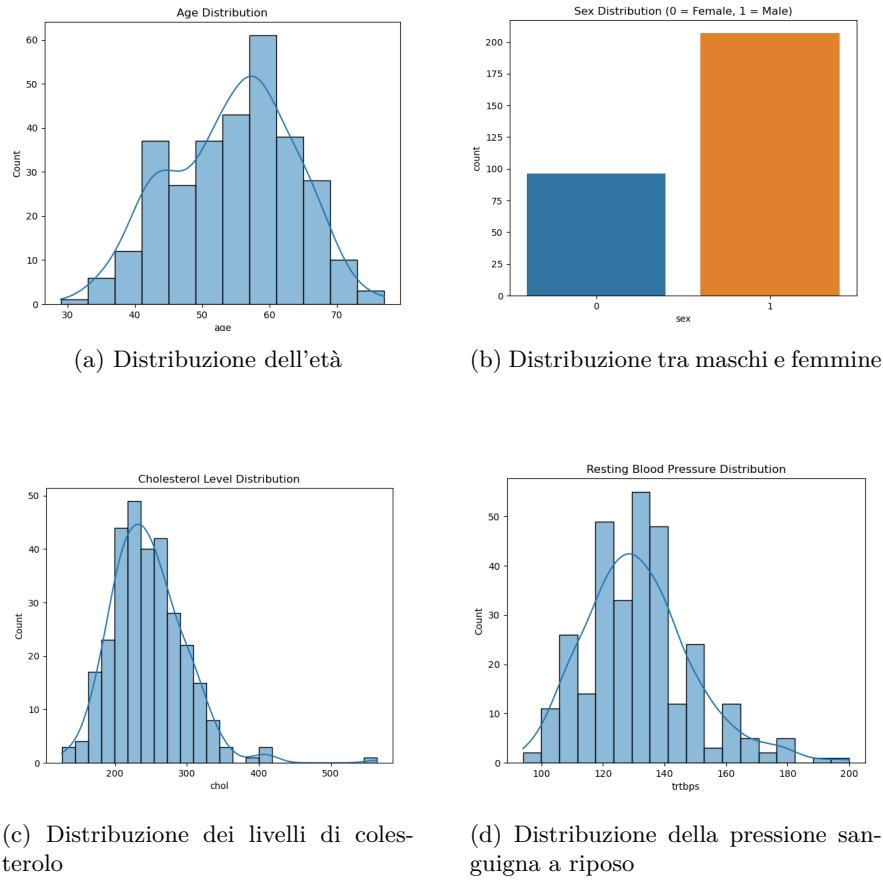


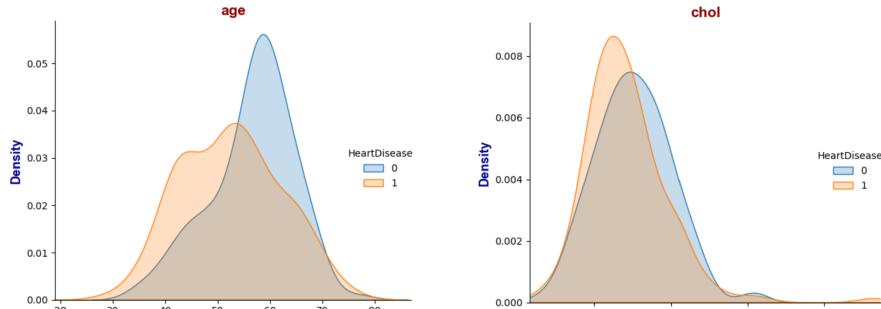
Figure 1: Analisi dei Dati - Distribuzione delle Caratteristiche Principali

2.3 Analisi delle Relazioni tra i Dati

In questa sezione, ci concentriamo sull'analisi bivariata tra le variabili indipendenti (numeriche o categoriche) e quella dipendente.

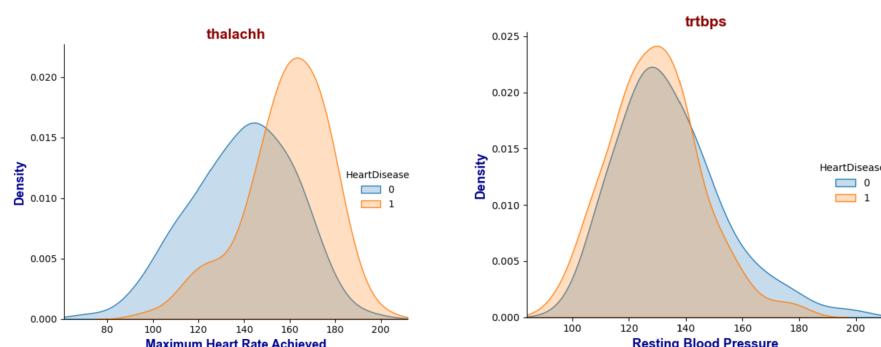
2.3.1 Variabili Numeriche

Per visualizzare i grafici, abbiamo scelto di utilizzare i KDE plot (Kernel Density Estimate) per le variabili numeriche. Questi grafici mostrano la densità della variabile indipendente per ogni valore target. Se le due curve sono ben separate e non si sovrappongono, ciò suggerisce una correlazione con il target.



(a) Correlazione dell'età

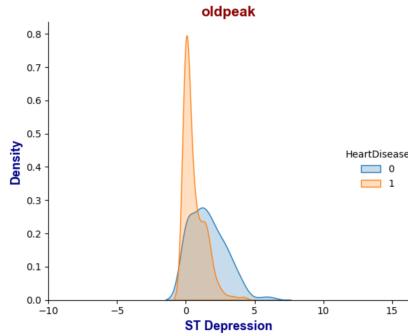
(b) Correlazione del colesterolo



(c) Correlazione della massima frequenza cardiaca

(d) Correlazione della pressione sanguigna a riposo

Figure 2: Analisi Bivariata - Stima Kernel di densità (KDE)



(a) Correlazione della depressione segmento ST

Figure 3: Analisi Bivariata - Stima Kernel di densità (KDE)

Dall'analisi dei grafici e dei coefficienti di Pearson, si possono trarre le seguenti conclusioni:

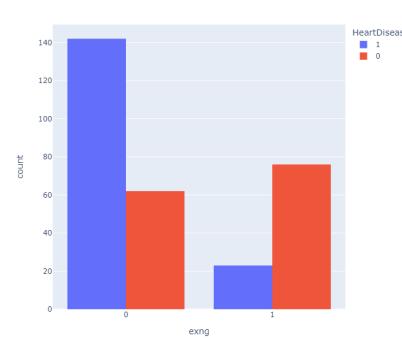
- **Age, Trtbps e Chol:** mostrano una correlazione molto bassa e negativa. Si può notare che il rischio di malattia cardiaca diminuisce dopo i 55 anni, contrariamente alla credenza comune.
- **Thalach:** presenta una correlazione positiva discreta. Il rischio di malattia aumenta con l'aumentare della frequenza cardiaca massima raggiunta.
- **Oldpeak:** mostra una correlazione negativa discreta. Quando il valore è compreso tra 0 e 1,5, si osserva un aumento significativo della probabilità di malattia.

2.3.2 Variabili Categoriche

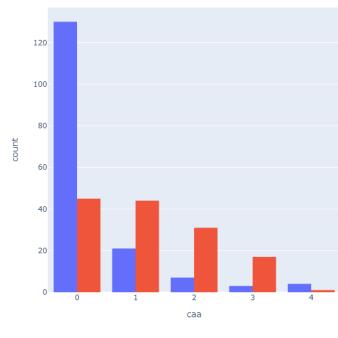
Per le variabili categoriche, abbiamo utilizzato grafici a barre, i quali mostrano il numero di osservazioni della variabile per i due valori della variabile target. Se le due barre sono molto diverse in altezza, ciò suggerisce una correlazione con il target.



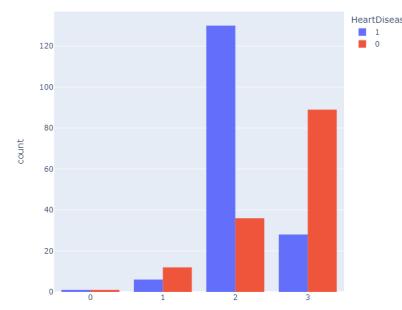
(a) Correlazione del sesso



(b) Correlazione affaticamento

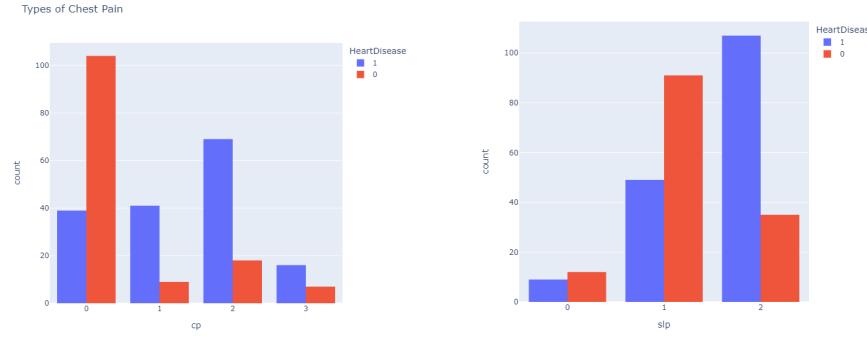


(c) Correlazione dei vasi sanguigni col-
orati



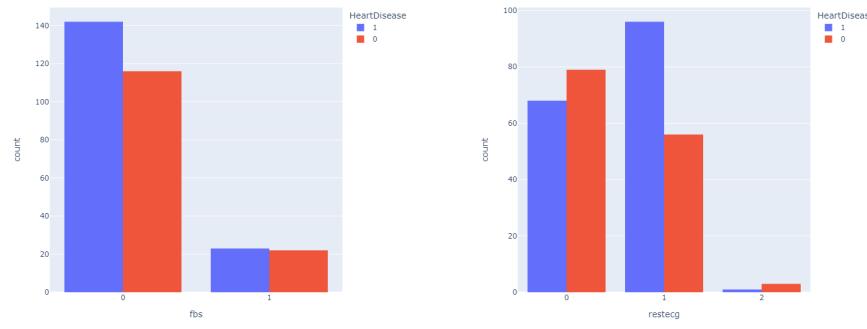
(d) Correlazione della talassemia

Figure 4: Analisi dei Dati - Distribuzione delle Caratteristiche Principali



(a) Correlazione del dolore al petto

(b) Correlazione del segmento ST



(c) Correlazione dei livelli di glucosio nel sangue a digiuno

(d) Correlazione del risultato dell'ecg a riposo

Figure 5: Analisi Bivariata - Distribuzione delle Caratteristiche Principali

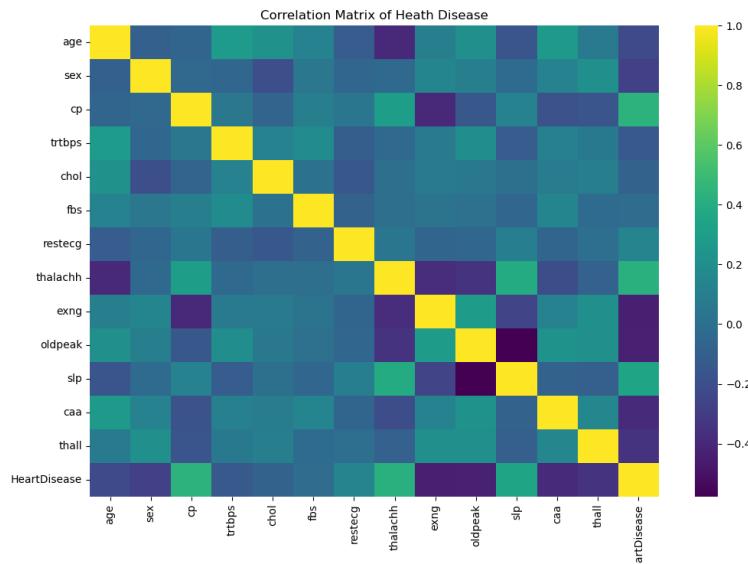
- Le variabili **Sex**, **Exng**, **Ca** e **Thal** hanno una correlazione negativa e moderata con il Target, con valori compresi tra -0.280937 e -0.436757. Ciò indica che il rischio di attacco cardiaco diminuisce all'aumentare del valore della variabile categorica. Ad esempio, il rischio di attacco cardiaco è maggiore per i pazienti di sesso femminile, per coloro che non presentano angina indotta dall'esercizio, che hanno zero vasi principali colorati dalla fluoroscopia, o che presentano un difetto reversibile.
- Le variabili **Cp** e **Slp** hanno una correlazione positiva, sia bassa che mod-

erata, con il Target, con valori compresi tra 0.137230 e 0.433798. Ciò indica che il rischio di attacco cardiaco aumenta all'aumentare del valore della variabile categorica. Ad esempio, il rischio di attacco cardiaco è maggiore per i pazienti che presentano un tipo di dolore toracico diverso da quello asintomatico o che hanno una pendenza del segmento ST in salita.

- Le variabili **Fbs** e **Restecg** mostrano una correlazione bassissima con il Target, con un valore vicino allo 0. Ciò indica che il rischio di attacco cardiaco non dipende molto dal livello di zucchero nel sangue a digiuno o dal risultato dell'ecocardiogramma a riposo.

2.3.3 Matrice di correlazione

Utilizziamo una matrice per misurare il grado di correlazione tra ogni coppia di variabili. In particolare, mostriamo i **coefficienti di Pearson**, i quali variano tra 1 e -1. Un valore di 1 o -1 indica una correlazione perfetta, mentre 0 indica



assenza di correlazione. Si nota che la frequenza cardiaca massima (thalach) è moderatamente correlata in negativo con l'età, mentre l'età e la pressione sanguigna hanno una leggera correlazione. La correlazione più significativa si

osserva tra oldpeak e slope (0.58). Se ci fossero state due **caratteristiche altamente correlate, sarebbe stato opportuno ridurre la dimensionalità** unendo le due in una sola. Tuttavia, con un grado di correlazione massimo di 0.58, tale necessità non sussiste.

2.4 Facilità della reperibilità dei dati

Considerando la natura complessa dei dati di tipo sanitario e l'obiettivo di rendere accessibile il modello ai medici per utilizzarlo sui dati di un paziente "medio", è naturale porsi le seguenti domande: "Quanto è realisticamente possibile che un paziente abbia questi dati nel mondo reale?", "**Quanto sono facilmente reperibili questi dati per un paziente?**", "Quali dati conviene reperire?", "Quali analisi conviene fare per ottenere una maggiore predizione?"

È importante considerare questi problemi fin da ora, poiché ciò permette di porre le basi per verificare, durante la fase di modellazione, l'esistenza di eventuali **leaky predictors** (caratteristiche significative per il problema, ma difficilmente disponibili in ambienti reali) e di minimizzare il problema di data leakage.

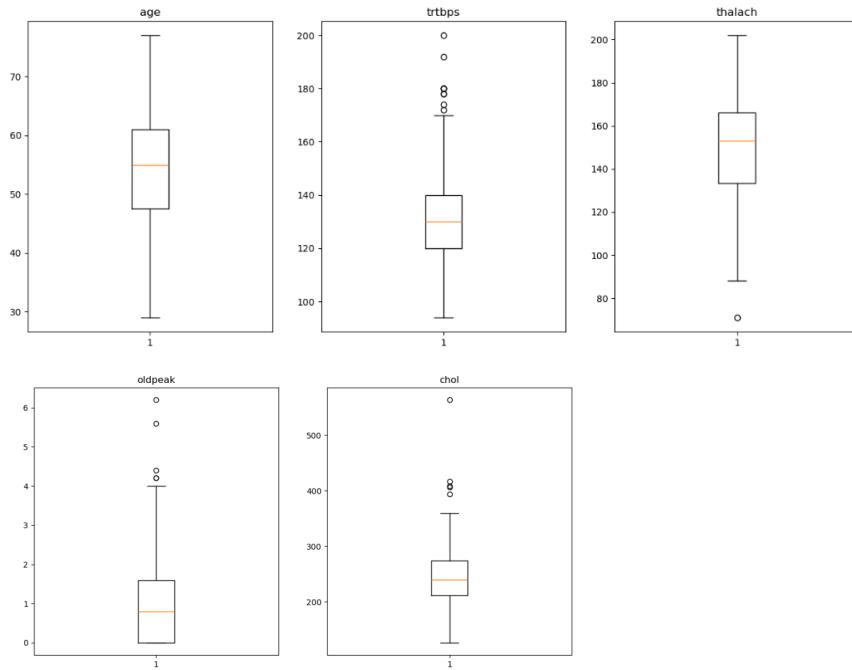
Alcuni dati sono più comuni e **facilmente reperibili** anche da dispositivi domestici, come l'età, il sesso, la pressione sanguigna, il colesterolo, il glucosio nel sangue, la frequenza cardiaca, il tipo di dolore al torace, l'ecocardiogramma, la depressione del segmento ST e l'angina. Altri dati sono più rari e **richiedono esami specifici e invasivi**, come la pendenza del segmento ST, il numero di vasi principali colorati e il tipo di talassemia. Questi ultimi dati sono solitamente raccolti solo in presenza di sospetti di malattia cardiaca. Nella fase di data modeling, valuteremo e confronteremo le performance dei modelli con e senza questi dati per valutare se il loro impatto è significativo per le performance.

3 Preparazione dei dati

In questa sezione procederemo a rimuovere gli outliers, effettuare la normalizzazione e bilanciare i dati. **Non è necessario ricorrere alle tecniche di Data Imputation** poiché non ci sono dati mancanti.

3.1 Gestione degli Outliers

Poiché molte tecniche di normalizzazione dei dati, come la min-max normalization, sono fortemente influenzate dagli outliers, abbiamo deciso di gestirli. Per identificare gli outliers, abbiamo utilizzato i boxplot, che mostrano la distribuzione dei dati in un range. Gli outliers sono rappresentati dai valori che si trovano al di fuori di questo range, cioè al di fuori degli ultimi "segmenti orizzontali", estendendosi dal primo al terzo quartile.



Possiamo trarre le seguenti conclusioni: l'età non presenta valori anomali e ha una distribuzione simmetrica. La variabile Trtbps mostra alcuni valori anomali e segue una distribuzione normale. Thalach presenta un valore anomalo e una

distribuzione leggermente asimmetrica. Oldpeak ha alcuni valori anomali e una distribuzione leggermente asimmetrica. Procediamo a **modificare i valori estremi degli outliers rendendoli meno estremi**, facendoli rientrare negli estremi del range attraverso la Winsorization.

```

1   #Rimozione outliers
2   winsorize_percentile_oldpeak=(stats.percentileofscore(
3     df[ 'oldpeak' ],4)/100)
4   old_peak_winsorize=winsorize(df[ 'oldpeak' ],(0,(1-
5     winsorize_percentile_oldpeak)))
6   df[ 'oldpeak' ]=old_peak_winsorize
7
8
9
10  winsorize_percentile_trtbps=(stats.percentileofscore(
11    df[ 'trtbps' ],170)/100)
12  trtbps_winsorize=winsorize(df[ 'trtbps' ],(0,(1-
13    winsorize_percentile_trtbps)))
14  df[ 'trtbps' ]=trtbps_winsorize
15
16
17  winsorize_percentile_chol=(stats.percentileofscore(df
18    [ 'chol' ],360)/100)
19  chol_winsorize=winsorize(df[ 'chol' ],(0,(1-
20    winsorize_percentile_chol)))
21  df[ 'chol' ]=chol_winsorize

```

Listing 1: Rimozione Outliers

3.2 Feature Scaling

Poichè si è visto che la distribuzione delle caratteristiche segue una forma, nella maggior parte dei casi normale, e avendo gestito gli outliers, abbiamo optato per normalizzare i dati mediante la min-max normalization, anziché utilizzare la z-score normalization. Di seguito, viene mostrato come è stato eseguito il processo

```

1 import numpy as np
2
3 #Per ogni valore abbiamo effettuato questo processo:
4 #Normalizziamo un certo valore "value"
5 prev=df[ 'value ']
6 min_prev=prev .min()
7 max_prev=prev .max()
8
9 normale=(prev-min_prev )/(max_prev-min_prev )
10 df[ 'value ']=normale

```

Listing 2: Min-Max Normalization

Per le variabili categoriche, abbiamo adottato un approccio diverso rispetto alla min-max normalization. Per scalare adeguatamente tali variabili, abbiamo utilizzato la codifica OneHotEncode. Nonostante siano espressi numericamente, **l'ordine di queste variabili non influenza la predizione della variabile target**. Creando una nuova colonna per ogni possibile valore di ciascuna variabile, **conserviamo la natura categorica** delle informazioni, consentendo al modello di comprendere meglio la natura di ogni caratteristica **senza attribuire erroneamente un ordine** al valore della caratteristica o creare relazioni artificiali inesistenti. Ad esempio, per la variabile thall, il modello potrebbe notare che thall=3 (difetto reversibile) sia un valore più importante rispetto a 2 (normale) e quindi attribuire meno importanza ai valori più bassi. Invece, thall=1 indica un difetto fisso, che è il più grave.

```

1
2 #Normalizziamo le categoriche con OneHotEncode
3 import pandas as pd
4
5 colonne_cat=[ 'sex ', "cp ", 'fbs ', 'restecg ', 'exng ', 'slp ', 'caa
6 , 'thall ']

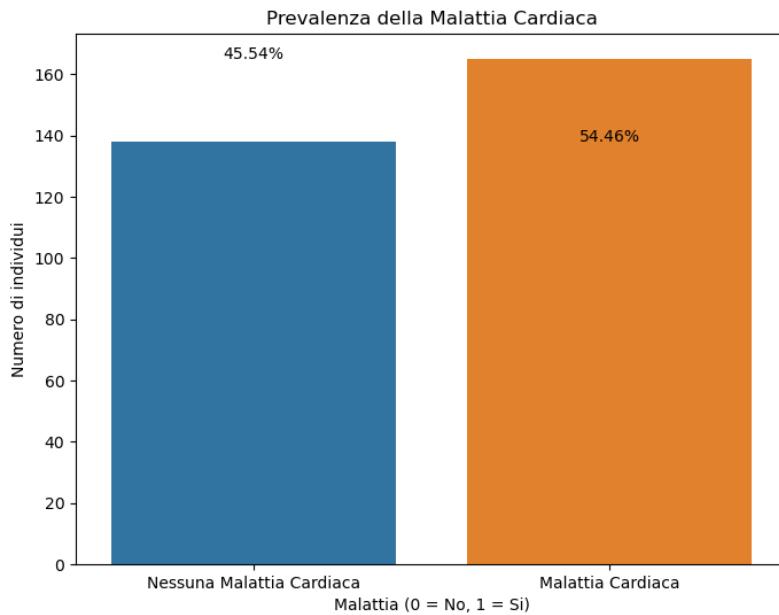
```

```
7 df=pd.get_dummies(df,columns=colonne_cat, prefix=
     colonne_cat)
8 print(df)
```

Listing 3: OneHotEncode

3.3 Data Balancing

Esaminando il grafico, abbiamo notato che la suddivisione attuale è adeguatamente equa (45%-55%), per cui non è necessario intraprendere ulteriori passi per bilanciare ulteriormente il dataset. Nello specifico, ci sono 165 positivi e 135 negativi, di seguito viene mostrato un grafico che illustra la distribuzione:



4 Data Modeling

4.1 Definizione del tipo di problema

Il problema che si considera in CardIACA è un problema di apprendimento supervisionato, nello specifico di classificazione binario perché la variabile target assume solo due valori (Sì e No). Come algoritmi, abbiamo utilizzato il Decision tree visto a lezione e due algoritmi molto performanti: KNN e SVM.

4.2 Configurazione per la validazione del modello

Per la validazione del modello, abbiamo optato per l'utilizzo della **Stratified k-fold validation**, suddividendo il dataset di partenza in sette gruppi. Per implementare questa tecnica, ci siamo avvalsi della libreria scikit-learn fornita da Python:

```
1 from sklearn.model_selection import StratifiedKFold  
2 kfold= StratifiedKFold(n_splits=7, shuffle=True ,  
random_state=25)
```

Listing 4: Stratified k-fold validation

Impostiamo 'shuffle=True' per garantire il mescolamento casuale dei dati prima della suddivisione. successivamente, configuriamo anche il parametro 'random-state', in modo tale che, rieseguendo il codice, otterremo la stessa suddivisione casuale per risultati riproducibili. Si è scelto $k=7$ perché addestrando varie volte il modello abbiamo notato che un k troppo grande, oltre ad aumentare la complessità computazionale, fa sì che non si riesca a "stratificare" e creare gruppi rappresentativi della popolazione, perché i gruppi creati sono troppo piccoli. Questo comporta una varianza eccessivamente diversa tra le performance di ogni fold.

4.3 KNN (K-nearest neighbors)

K-Nearest Neighbors (KNN) è un algoritmo di classificazione che assegna un'etichetta ad un'istanza nuova considerando le etichette dei k vicini più prossimi, dove k è un parametro scelto dall'utente, alla fine verrà assegnata l'etichetta più frequente fra i vari k vicini, scelti in base ad una metrica di distanza.

Per implementare questo algoritmo abbiamo sempre fatto uso di scikit-learn, di seguito il codice completo:

```

1     acc_log=[]
2     for fold , (train_index , val_index) in enumerate(kfold
. split(X=df , y=predict)):
3
4         X_train=df.loc[train_index , train]
5         Y_train=df.loc[train_index , 'HeartDisease']
6
7         X_val=df.loc[val_index , train]
8         Y_val=df.loc[val_index , 'HeartDisease']
9
10        modello = KNeighborsClassifier(n_neighbors=10)
11        modello . fit (X_train , Y_train)
12        y_pred=modello . predict (X_val)
13        print(f"The fold is : {fold + 1} : ")
14        print(classification_report(Y_val , y_pred))
15        acc = roc_auc_score(Y_val , y_pred)
16        acc_log.append(acc)
17        print(f"The accuracy for Fold {fold + 1} : {acc}")
18        matrice=confusion_matrix(Y_val , y_pred)
19        print(f"The confusion matrix for Fold {fold + 1} ")
20        print(matrice)

```

Listing 5: K-nearest neighbors

4.3.1 Scelta del K

Una fase cruciale è la scelta del parametro K, determinante nella classificazione di una nuova istanza. A tal fine, abbiamo impiegato un **algoritmo genetico per trovare il valore ottimale di K**. La funzione di fitness utilizzata si basa sulla precision del predittore. Perché Precision e non Recall?

Il **recall è una metrica invariante rispetto alla prevalenza** dei positivi, ad esempio l'affermazione "su un totale di 5 malati ne predice 4 correttamente positivi" è valida sia se chi ha la malattia è il 50% della popolazione (ambiente di testing), sia se è solo il 5%. Invece, l'affermazione "su un totale di 5 predizioni positive, 4 hanno realmente la malattia" è **dipendente dalla prevalenza** dei malati: se ci sono tanti malati, come nel caso del nostro dataset bilanciato, è più facile indovinare ed avere un'alta precisione. Dato che in ambienti reali i malati saranno una frazione molto piccola, il peso dei falsi positivi sarà molto maggiore: **affermare che una persona sia positiva sarà più rischioso**. E' per questo che puntiamo a massimizzare la precisione: in ambiente reale tenderà inevitabilmente a diminuire (Approfondimento nella [sezione 6](#)).

Di seguito il codice per la scelta di K con algoritmo genetico:

```

1 import random
2 import math
3 from deap import base, creator, tools, algorithms
4 from sklearn.neighbors import KNeighborsClassifier
5 from sklearn.metrics import precision_score
6 from sklearn.model_selection import train_test_split
7 from flaskDir.source.ModuloIA.DataPreparation import df,
     predict, train
8
9 # Definire la funzione di fitness (precision del modello
# KNN)
10 def fitness_function(K):

```

```

11     knn = KNeighborsClassifier(n_neighbors=K)
12     knn.fit(X_train, y_train)
13     y_pred = knn.predict(X_val)
14     precision = precision_score(y_val, y_pred)
15     return precision
16
17 # Definire la funzione obiettivo per DEAP (massimizzare
18 # la precisione)
18 def objective_function(chromosome):
19     K = chromosome[0]# Un solo gene rappresenta il valore
20     di K
21     return fitness_function(K)
21
22 #Crossover personalizzato
23 def crossover(parent1, parent2):
24     child=tools.cxBlend(parent1, parent2, alpha=0.5)
25     child[0][0]=math.ceil(child[0][0])
26     if child[0][0]<0:
27         child[0][0]=abs(child[0][0])
28     elif child[0][0]==0:
29         child[0][0]=random.randint(1,40)
30     return child
31
32 def mutation(chromosome):
33     newChromosome=tools.mutUniformInt(chromosome
34     ,5,40,0.1)
35     newChromosome[0][0]=math.ceil(newChromosome[0][0])
36     return newChromosome
37
37 if __name__ == "__main__":

```

```

38
39      # Suddividere i dati
40      trainset=df[train]
41      X_train, X_val, y_train, y_val = train_test_split(
42          trainset, predict, test_size=0.2, random_state=42)
43
44      # Inizializzare DEAP
45      creator.create("FitnessMax", base.Fitness, weights
46                      =(1.0,))
47      creator.create("Individual", list, fitness=creator.
48                      FitnessMax)
49
50      toolbox = base.Toolbox()
51      toolbox.register("attr_int", random.randint, 1, 40)
52      toolbox.register("individual", tools.initRepeat,
53                      creator.Individual, toolbox.attr_int, n=1)
54      toolbox.register("population", tools.initRepeat, list
55                      , toolbox.individual)
56      toolbox.register("evaluate", objective_function)
57      toolbox.register("mate", crossover)
58      toolbox.register("select", tools.selTournament,
59                      tournsize=4)
60      toolbox.register("mutate", mutation)
61
62
63      # Eseguire l'algoritmo genetico con DEAP
64      population = toolbox.population(n=50)
65      algorithms.eaMuPlusLambda(population, toolbox, mu
66                      =125, lambda_=250, cxpb=0.7, mutpb=0.3, ngen=20, stats
67                      =None, halloffame=None, verbose=True)

```

```

60
61     # Trovare il miglior individuo
62     best_individual = tools.selBest(population, k=1)[0]
63     best_K = best_individual[0]

```

Listing 6: Algoritmi Genetici

4.4 Support Vector Machine

Support Vector Machine (SVM) è un algoritmo di apprendimento supervisionato utilizzato sia per problemi di classificazione che di regressione. L'obiettivo principale di SVM è trovare un iperpiano ottimale che separi le varie istanze in maniera efficace, è come se tracciasse una linea che separe quest'ultime. Su quest'algoritmo l'attenzione si è focalizzata sulla scelta del Kernel, il quale consente ad SVM di trattare i dati in maniera molto efficace anche se essi non sono facilmente separabili mediante una linea nello spazio. Dalle precedenti analisi effettuati sui dati si è optato per un kernel lineare perché i dati sono linearmente separabili.

```

1     acc_log=[]
2     for fold, (train_index, val_index) in enumerate(kfold
. split(X=df, y=predict)):
3
4         X_train=df.loc[train_index,train]
5         Y_train=df.loc[train_index,'HeartDisease']
6
7         X_val=df.loc[val_index,train]
8         Y_val=df.loc[val_index,'HeartDisease']
9
10        modello2 = SVC(kernel='linear')
11        modello2.fit(X_train,Y_train)
12        y_pred=modello2.predict(X_val)

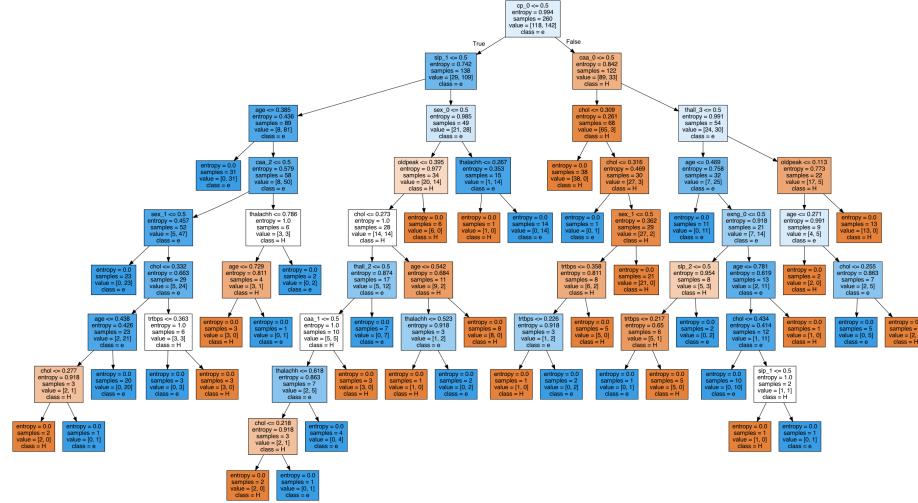
```

```
13     print(f"The fold is : {fold + 1} : ")
14     print(classification_report(Y_val, y_pred))
15     acc = roc_auc_score(Y_val, y_pred)
16     acc_log.append(acc)
17     print(f"The accuracy for Fold {fold + 1} : {acc}%")
18
19     matrice=confusion_matrix(Y_val, y_pred)
20     print(f"The confusion matrix for Fold {fold + 1}")
21
22     print(matrice)
```

Listing 7: Support Vector Machine

4.5 Albero Decisionale

L'albero decisionale basato su entropia non sarà tra i modelli più performanti se confrontato con KNN e SVM, ma lo abbiamo comunque scelto per la facilità di interpretazione dei risultati. Infatti, tramite la struttura ad albero si riesce ad avere un eccellente livello di explainability. Ecco la struttura del nostro albero:



Di seguito il codice:

```

1     acc_log=[]
2     print("DC")
3     for fold , (train_index , val_index) in enumerate(kfold
. split(X=df , y=predict)):
4
5         X_train=df.loc[train_index , train]
6         Y_train=df.loc[train_index , 'HeartDisease']
7
8         X_val=df.loc[val_index , train]
9         Y_val=df.loc[val_index , 'HeartDisease']
10
11        modello3 = DecisionTreeClassifier(criterion='
entropy')
12        modello3 . fit (X_train , Y_train)
13        y_pred=modello3 . predict (X_val)
14        print(f "The fold is : {fold + 1} : ")
15        print(classification_report(Y_val, y_pred))
16        acc = roc_auc_score(Y_val, y_pred)
17        acc_log.append(acc)
18        print(f "The accuracy for Fold {fold + 1} : {acc}" )
19        matrice=confusion_matrix(Y_val, y_pred)
20        print(f "The confusion matrix for Fold {fold + 1}" )
21        print(matrice)

```

Listing 8: Albero decisionale

4.6 Validazione

Per validare i modelli utilizziamo la **Stratified K Fold Cross Validation**. Confrontiamo la matrice di confusione media per ogni modello:

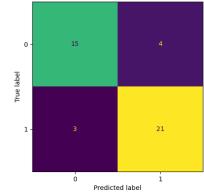


Figure 6: KNN (84% Precision, 88% Recall, 83% Accuracy)

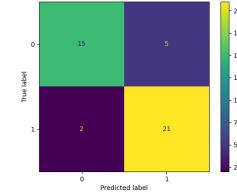


Figure 7: SVM (81% Precision, 91% Recall, 83% Accuracy)

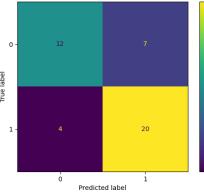


Figure 8: Decision Tree (74% Precision, 83% Recall, 74% Accuracy)

4.6.1 Un Passo Avanti nella Validazione

Oltre alla matrice di confusione, abbiamo deciso di tenere in considerazione, per la validazione dei vari modelli, anche del ROC AUC score, una metrica che rappresenta il trade off tra Sensitività, cioè il Recall, e Specificità, cioè $P(\text{Negativo}|\text{Sano})$: Notiamo che KNN e SVM si comportano mediamente allo stesso modo, con un AUC score intorno all'83-84%, mentre il Decison Tree resta indietro con uno score intorno al 74%.

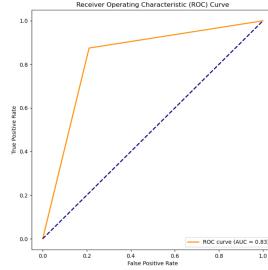


Figure 9: KNN

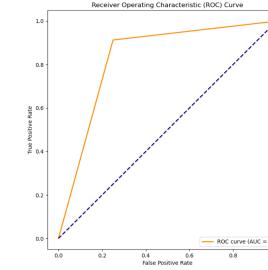


Figure 10: SVM

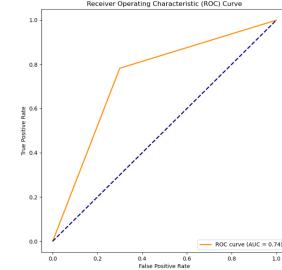


Figure 11: Decision Tree

stesso modo, con un AUC score intorno all'83-84%, mentre il Decison Tree resta indietro con uno score intorno al 74%.

4.6.2 Risultati di Validazione

Considerando che si sta usando la cross validation e che quindi non tutti i dati vengono usati nell'addestramento, crediamo che in ambiente reale, essendo allenato su tutti i dati, riuscirebbe ad avere performance leggermente migliori. Abbiamo confrontato tre algoritmi con varie metriche di valutazione, e abbiamo potuto concludere che il **Decision Tree è il meno performante**, mentre **KNN e SVM** si comportano entrambi in modo **ottimale**. Quale utilizzare tra questi due? Per scegliere abbiamo considerare un aspetto che spesso viene trascurato: la complessità computazionale. L'allenamento di SVM impiega $O(n^3)$, e **tempo lineare** per fare predizioni. Invece KNN non ha una fase di allenamento, ma **per ogni predizione ha bisogno di fare un'inferenza in** $O(n * d)$, molto più complesso: con dataset molto grandi KNN non ha utilizzi pratici, la complessità del database cresce inesorabilmente nel tempo e **ci vorrebbe troppo tempo** e troppa potenza computazionale per effettuare le predizioni. Per la natura del nostro dominio, ci troveremo molto spesso ad effettuare predizioni e raramente avremo bisogno di rieffettuare l'addestramento. Pertanto, nella fase di Deployment utilizzeremo SVM.

4.7 Validazione con un Dataset più "Semplice"

Nella [sezione 2.4](#) abbiamo visto come non tutti i dati siano facili da ottenere per il paziente medio. Di conseguenza, abbiamo valutato le performance del modello su un dataset in cui avevamo rimosso thall, caa e slp, le variabili "problematiche". Abbiamo riscontrato una moderata diminuzione delle performance, del 6-7% (indipendentemente dall'algoritmo) rispetto alle performance sui dataset normali. Con queste informazioni cercheremo di trovare un **compromesso tra flessibilità del modello e accuratezza** in fase di Deployment.

5 Deployment

Il modello sarà reso disponibile su una piattaforma web chiamata MediCare, il cui scopo principale è gestire le prenotazioni in ambito sanitario. Il modello avrà una sezione dedicata a parte dove l'utente dovrà compilare un modulo, il quale sarà inviato al controller apposito che sarà incaricato d'inoltrare la richiesta al modello vero e proprio. Una volta fatte tutte le operazioni sui dati, il modello restituirà un riscontro che sarà in seguito mostrato all'utente

5.1 Implementazione Web

Crediamo che anche il design delle pagine sia importante per "attirare l'attenzione" del paziente, così abbiamo creato una pagina moderna con l'aiuto del framework Tailwind CSS:

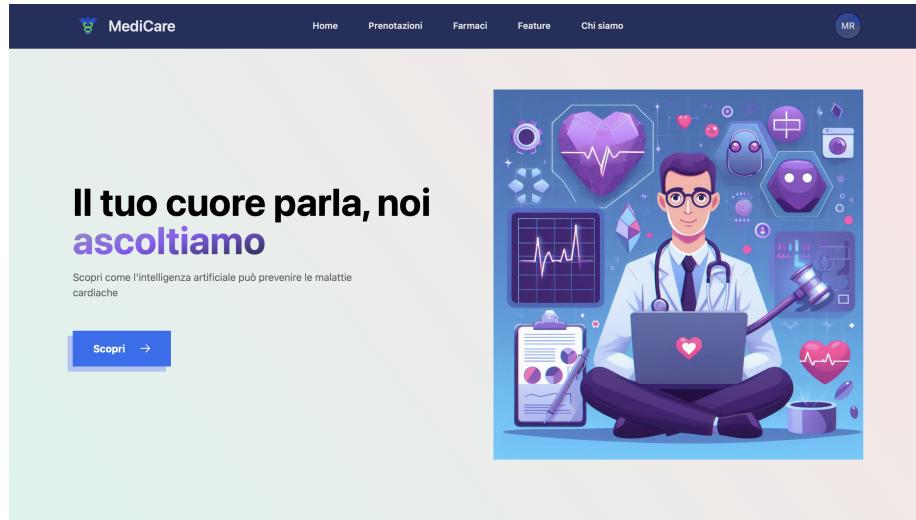


Figure 12: Pagina Introduttiva

Inserisci i dati

Età del paziente: Pressione sanguigna a riposo (in mmHg):

Frequenza cardiaca massima raggiunta: Livello di colesterolo (in mg/dl):

Sesso: Angina indotta dall'esercizio:
Select Select

Tipo di Talassemia: Numero di vasi principali colorati dalla fluoroscopia:
Select Select

Livello di glucosio nel sangue a digiuno: Risultato dell'ecocardiogramma a riposo:
Select Select

Tipo di dolore al torace: Pendente del segmento ST durante l'esercizio:
Select Select

Depressione del segmento ST indotta dall'esercizio rispetto al riposo:

Ottieni il risultato

Un processo all'avanguardia
Elaboriamo i dati dei pazienti nel rispetto della privacy e con la massima sicurezza

Inserisci i dati di un paziente. Se il paziente non ha quei dati, e se lo si ritiene opportuno, si procede a fare le analisi necessarie per ottenerne quei dati.

Il sistema elabora i dati tramite sistemi di intelligenza artificiale.

I risultati vengono mostrati al medico, che decide il da farsi in base all'esito.

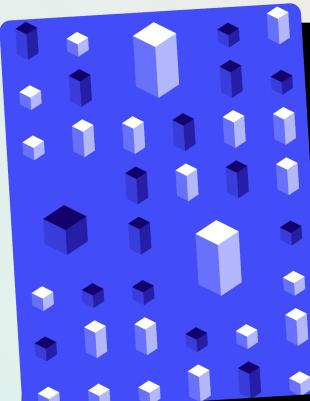


Figure 13: Form per inserire i dati

5.2 Aggiornamenti Dinamici

Essendo un Medicare un sito in cui vengono conservati tutti i dati sanitari dei paziente, **qualora** a un paziente venisse **diagnosticata una malattia cardiaca**, le sue informazioni anonime (età, peso, etc) verranno aggiunte al training-set. Poiché il modello scelto è basato su SVM, la cui complessità di addestramento è molto alta($O(n^3)$), abbiamo scelto di effettuare il training manualmente ad intervalli temporali e non ad ogni diagnosi di malattia, questo permette al modello di essere continuamente addestrato, mantenendo comunque un impatto relativamente basso sulle performance del sistema.

5.3 Fallback del modello

Come abbiamo visto nella [sezione 4.7](#), rimuovere quelle caratteristiche che per il paziente sono più difficili da reperire porta ad una riduzione del 6-7% delle performance, per un'accuratezza dell'76-77% (per SVM). Sebbene questo valore sia comunque buono, abbiamo preferito **mantenere entrambi i modelli**: uno più accurato ma meno flessibile e uno più flessibile ma meno accurato. Il form per effettuare la predizione è quindi costituito da 9 campi obbligatori e 3 opzionali che richiedono test più specifici ma che permetterebbero di avere un risultato più accurato. In seguito avviene la verifica dei campi opzionali a livello di backend: se almeno uno è null, si procede ad effettuare la predizione con il modello meno accurato.

6 I Modelli non sono Medici

Abbiamo identificato un elemento chiave che modifica radicalmente il contesto: la prevalenza della condizione nel mondo reale, o più in generale, le probabilità a priori della presenza della condizione.

Immaginiamo di avere un modello con 100% di recall e 98% di specificità per una malattia che colpisce il 2% della popolazione, sembra ottimo, no? Purtroppo non è così, infatti, se si risulta positivi, c'è solo il 50% di probabilità di essere

veramente malati. I motivi che ci spingono a credere ciò che viene affermato nel titolo, verranno spiegato nei successivi paragrafi.

6.1 La precisione del modello nel mondo reale

Per calcolare la precisione del modello in ambiente reale, assumiamo una prevalenza delle malattie cardiache del 5% e usiamo il teorema di bayes:

$$\begin{aligned}
 P(\text{Malato}|\text{Positivo}) &= \frac{TP}{TP + FP} = \\
 &= \frac{P(\text{Malato})P(\text{Positivo}|\text{Malato})}{P(\text{Malato})P(\text{Positivo}|\text{Malato}) + P(\text{NoMalato})P(\text{Positivo}|\text{NoMalato})} \\
 &= \frac{(\text{Precedente})(\text{Recall})}{(\text{Precedente})(\text{Recall}) + (1 - \text{Precedente})(\% \text{Falsi Positivi})} \\
 &= \frac{(0.05)(0.91)}{(0.05)(0.91) + (1 - 0.05)(0.25)} = 16\%
 \end{aligned} \tag{1}$$

Infatti, su 1000 persone, di cui 50 sono malati, dovremmo predirne correttamente 45 positivi (Recall 91%). I falsi positivi sono 23.5 (il nostro modello ha il 25% di False Positive Ratio, cioè i falsi positivi sul totale dei negativi, che nel nostro caso è 950). Calcoliamo infine la precisione, $\frac{(45)}{45+23.5} = 16\%$

Abbiamo imparato un'altra lezione a caro prezzo:

Come fanno i test medici ad essere precisi? In base a che cosa affermano la presenza della malattia? Il punto è che la precisione è così bassa perché **gli asintomatici non andrebbero testati**, perché la loro probabilità iniziale di avere la malattia è troppo bassa, oppure se uno di loro fosse affetto dalla malattia, la mancanza dei sintomi renderebbe difficile il rilevamento da parte di macchine (le macchine imparano ma non ragionano), dunque non è solo la prevalenza, ma anche la probabilità di avere la malattia pre-test (Precedente),

che dipende dal contesto iniziale e varia di persona in persona (eventuali sintomi, fattori genetici, etc).

6.2 Conclusioni

Il nostro obiettivo era di costruire un modello che permettesse di identificare se una persona fosse malata tramite ciò che abbiamo appreso a lezione. Sebbene l'ambito in cui opera l'agente è molto delicato e insidioso, siamo riusciti ad progettarlo in maniera tale da avere delle prestazioni sub-ottimali. Un aspetto chiave che ci ha colpiti è l'effetto del bilanciamento del dataset nel mondo reale. Per addestrare "bene" il modello è necessario che metà della popolazione sia malata e l'altra metà sia sana. Purtroppo questo fa sì che la precisione, che in fase di addestramento era altissima, **nel mondo reale cala drasticamente** a causa della bassa prevalenza dei pazienti malati. L'illusione della precisione colpisce tutti i test in cui la prevalenza dei malati reali è una frazione dei pazienti sani. Inoltre, ci siamo posti una domanda che non viene risposta in fase di validazione: Quante sono le probabilità di essere veramente positivo se si è risultati positivi? Al momento del training e della validazione, i positivi e i negativi sono bilanciati ma quando il modello sarà a chiamato ad agire nel mondo reale si troverà di fronte una situazione sbilanciata, il numero di persone realmente positive è enormemente minore rispetto a quelle negative, quindi il modello non si comporterà come nella fase di addestramento. È emerso chiaramente che tali modelli non forniscono un riscontro definitivo sulla presenza o assenza della malattia, ma aggiornano di un fattore costante la probabilità iniziale.