UNIVERSITÀ DEGLI STUDI
DI SALERNO

UNIVERSITÀ DEGLI STUDI DI SALERNO
DIPARTIMENTO DI INFORMATICA
Corso ”Fondamenti di Intelligenza Artificiale” A.A. 2023
Prof. F.Palomba

Documentazione di progetto
A.A. 2022/2023

Scheduler intelligente ”ChemoSmart”

Ciro Troiano, Antonio Nappi, Mihail Purice , Giuseppe Basile

Sommario

Scheduler intelligente ”ChemoSmart”

Ciro Troiano, Antonio Nappi, Mihail Purice, Giuseppe Basile

Febbraio 2023

1 Introduzione

1.1 Sistema attuale

Ad oggi la schedulazione di appuntamenti chemioterapici si basa principalmente su delle euristiche volte ad evitare lo spreco di farmaci le cui dosi possono realizzare più di un chemioterapico e assecondare i bisogni dei pazienti.

1.2 Sistema proposto

ChemoSmart è un sistema che mira ad ottimizzare la schedulazione degli appuntamenti chemioterapici fornendo supporto al personale addetto. In particolare, il modulo di IA di ChemoSmart, che rappresenta il ”nucleo” del sistema stesso, è uno agente che mira ad ottimizzare e ”personalizzare” la schedulazione degli appuntamenti, fornendo una sistema di priorità realizzato sulla base della storia medica di un paziente.

1.3 Link utili

Qui presente il link alla repository di github a cui è possibile accedere e visualizzare l'implementazione del modello e la realizzazione dei grafici che saranno utilizzati nella sezione *4.3* Caratteristiche del dataset e Feature Selection.
Link della repository del modello: https://github.com/ImCirot/ChemoSmart-AI

Il link al progetto Chemosmart sul quale il modello è stato utilizzato:
Link della repository: https://github.com/alessandro-bergamo/chemosmart

Ed infine, il link del dataset utilizzato per il training e testing del modello
Link dataset: https://www.kaggle.com/datasets/rishidamarla/cancer-patients-data

2 Descrizione dell'agente

2.1 Obiettivo

L'obiettivo dell'agente è quello di classificare i pazienti in diverse categorie di "rischio" e fornire loro una priorità utile per definire una schedulazione che possa favorire i pazienti che necessitano di cure immediate. Tale priorità verrà stabilita con l'ausilio di alcuni dati fondamentali della storia clinica del paziente come:

- Età;
- Uso di alcolici;
- Alimentazione sbilanciata;
- Obesità.

Questi sono solo alcune delle caratteristiche che l'agente utilizzerà per classificare un paziente. Ulteriori informazioni saranno disponibili nella sezione *4.2 Il dataset*.

2.2 Specifica PEAS

Di seguito la specifica PEAS dell'agente:

- **Performance**: la misura di performance dell'agente è la sua capacità di avvicinarsi quanto più possibile ad una classificazione dei pazienti sulla base delle caratteristiche comuni che identificano un grado di "rischio" maggiore o minore di un altro;
- **Enviroment**: l'ambiente in cui opera l'agente è determinato da un numero finito di pazienti con i propri indici.
  L'ambiente è:
  - **Osservabile**, l'ambiente di lavoro dell'agente può essere completamente osservato dai sensori di quest'ultimo in quanto l'agente ha sempre a disposizione tutte le informazioni dei pazienti;
  - **Deterministico**, l'ambiente viene modificato dal solo operato dell'agente in quanto solo quest'ultimo può decidere come classificare un paziente;
  - **Episodico**, la scelta dell'agente non è condizionata dalle scelte "precedenti";
  - **Dinamico**, l'ambiente potrebbe variare anche senza azioni da parte dell'agente in quanto gli indici dei pazienti, e quindi le caratteristiche a cui l'agente può accedere, possono essere modificati;
  - **Discreto**, l'ambiente presenta un numero limitato di percezioni e stati distinti che è possibile stabilire completamente data la natura dell'entità paziente;
  - **Singolo**, nell'ambiente opera unicamente l'agente descritto.
- **Actuators**: l'attuatore fondamentale dell'agente è la stato attuale del paziente sottoforma di indici di valutazione;
- **Sensors**: Il sensore dell'agente è rappresentato dal bottone "schedula terapia" che permette di inizializzare la schedulazione della terapia del dato paziente fornendo all'agente il paziente per ottenerne una priorità utile alla schedulazione.

3 Analisi del problema

Data la natura ”delicata” dell’ambiente di lavoro dell’agente e dalla presenza di euristiche fondamentali e da tempo note, che rappresentano uno standard per la realizzazione di una schedule per una terapia chemioterapica si è pensato di realizzare un ulteriore ”specializzazione” degli appuntamenti senza intaccare le euristiche appena descritte. Si è, quindi, pensato di ricorrere a tecniche di machine learning per fornire un sistema di priorità per pazienti che necessitano cure in maniera tempestiva per poter, idealmente, ridurre i danni e l’avanzamento dello stato di malattia in presenza di un quadro clinico particolare. Quest’ultima affermazione sposa il concetto di **classificazione** ed è quindi stato un approccio quasi istantaneo alla soluzione del problema.

4 Raccolta, analisi e processing dei dati

4.1 Raccolta dei dati

Data la natura dell’ambiente medico e del contesto scolastico per cui tale agente è stato realizzato, è impossibile, senza licenze di stato o enti privati, ottenere un dataset ufficiale utile alla realizzazione di quest’ultimo.
La scelta, quindi, ricade nell’utilizzare dataset ad-hoc per scopi scolatistici. Tramite l’utilizzo ***Kaggle***, piattaforma dedita alla raccolta di dataset utili per il machine learning in ogni ambito, abbiamo trovato un dataset che sposa l’idea iniziale del progetto.

4.2 Il dataset

Il dataset è composto da **1000** pazienti. I dati presenti all’interno del dataset sono degli indici che descrivono alcune caratteristiche della situazione medica del paziente. Tali indici, da una prima analisi del dataset, seguono una codifica decimale presentando valori nel range **[1,9]**, dove più basso è l’indice ”migliore” e la condizione di tale caratterisica. Di seguito, viene fornita una traduzione in italiano e una descrizione del dataset con una spiegazione per ogni indice.

- **idPaziente**, id univoco del paziente;
- **Età**, età del paziente;
- **Sesso**, rappresentata tramite codifica decimale dove **”1”** corrisponde a **”Uomo”** e **”2”** corrisponde a **”Donna”**;
- **Qualità dell’aria**, questo indice rappresenta la qualità dell’aria nel contesto abitativo del paziente;
- **Uso di alcolici**, indice che rappresenta il consumo, più o meno frequente, di alcolici;
- **Allergie**, indice che rappresenta la presenza di una o più allergie a medicinali e ambientali;
- **Esposizione al rischio in ambiente lavorativo**, indice utilizzato per descrivere la possibile esposizione a rischi in ambiente lavorativo. Maggiore è l’indice maggiore è la possibilità che il paziente sia esposto a gas nocivi, metalli, sostanze che possono danneggiare, a lungo andare, la salute di quest’ultimo;
- **Indice dei fattori di rischio familiare**, indice utilizzato per indicare la predisposizione del paziente a tale quadro clinico, sulla base di situazioni ”simili” nel contesto familiare;

- **Problemi respiratori**, utilizzato per indicare se il paziente presenta un qualsiasi tipo di problema respiratorio e in quale stato;
- **Alimentazione scorretta**, indica un'alimentazione sbilanciata povera di nutrienti;
- **Obesità**, indica lo stato di obesità del paziente;
- **Esposizione al fumo attivo**, indice usato per definire un paziente fumatore e la frequenza;
- **Esposizione al fumo passivo**, indice usato per indicare l'esposizione di un paziente a contesti di fumo passivo;
- **Dolori localizzati**, indice usato per indicare la presenza e l'intensità di dolori localizzati;
- **Emottisi**, indica se il paziente presenta tosse con sangue e la sua frequenza;
- **Astenia**, indice usato per stabilire la presenza di senso di affaticamento in rapporto agli sforzi necessari a raggiungere tale condizione;
- **Perdita di peso**, indica se il paziente è soggetto a perdite di peso e in che relazione;
- **Dispnea**, presenza o meno di respiro corto, affannato anche in contesti di non attività;
- **Respiro sibilante**, presenza o meno di respiro sibilante e la frequenza;
- **Disfagia**, presenza di problemi alla deglutizione e la sua estensione;
- **Dita di Ippocrate**, presenza o meno delle dita di Ippocrate, condizione clinica usata come allarme per problemi polmonari;
- **Sensibilità all'ammalarsi**, sensibilità e frequenza del paziente all'ammalarsi;
- **Tosse secca**, presenza e frequenza di tosse secca;
- **Russamento**, presenza, frequenza e stato di russamento a riposo;
- **Priorità**, rappresenta il livello di priorità del paziente.

4.3 Caratteristiche del dataset e Feature Selection

Di seguito sono fornite alcune caratteristiche pregnanti del dataset, ottenute da uno studio necessario per la definizione delle cosiddette caratteristiche *feature*.
Un primo grafico ci mostra la distribuzione dei casi per priorità all'interno del dataset:

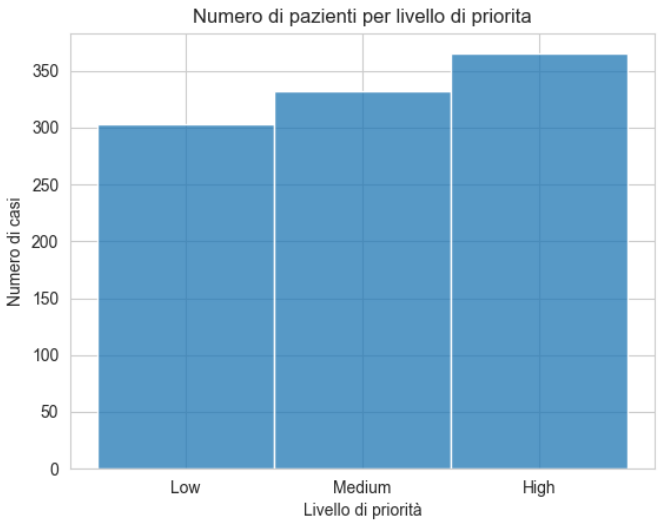

Fig. Distribuzione dei casi nelle diverse priorità

Viene evidenziato, quindi, come il numero di casi per ogni priorità sia ragionevolmente equo, ma non bilanciato, passando dai 300 casi per priorità bassa ai 350+ casi per priorità alta.
Vengono forniti, ora, alcuni dei rapporti più interessanti fra le caratteristiche del dataset e la priorità ad esso associata per poter stabilire in che modo, ogni caratteristica esaminata, ha contribuito alla scelta della priorità. Un primo rapporto è la distribuzione dei casi in base all'età:

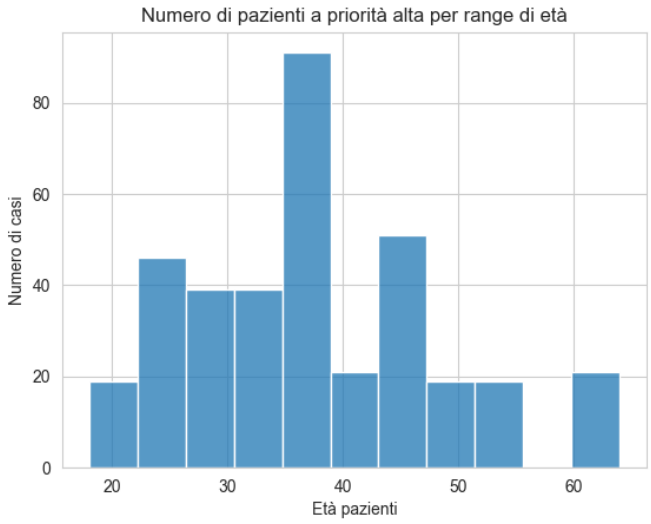

Fig.1 Distribuzione dei casi a priorità alta nel range età dei pazienti

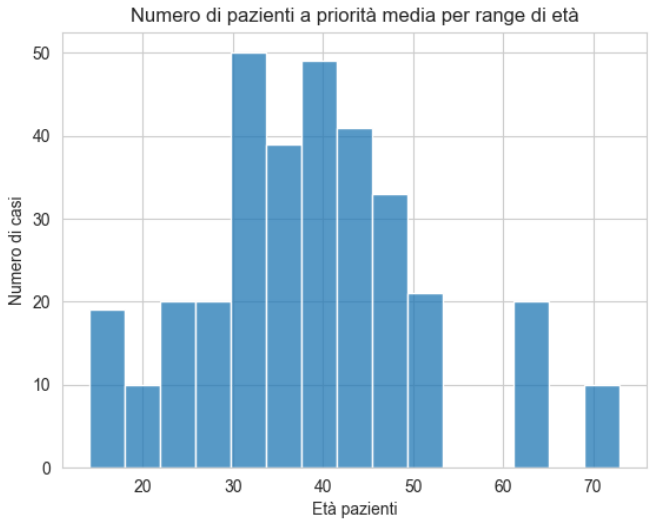

Fig.2 Distribuzione dei casi a priorità media nel range età dei pazienti

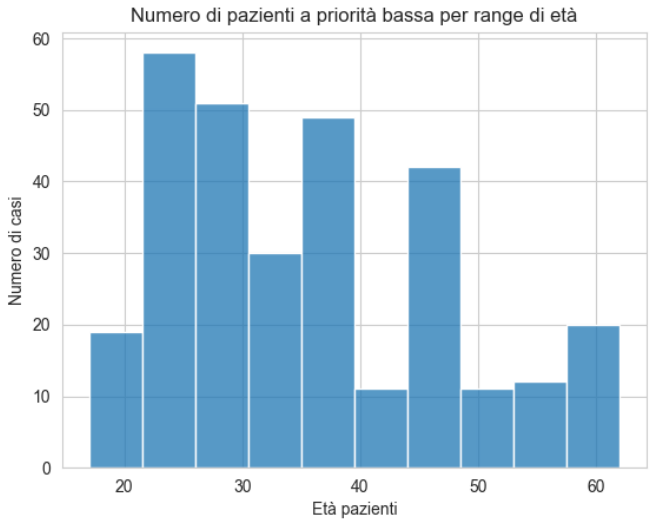

Fig.3 Distribuzione dei casi a priorità bassa nel range età dei pazienti

Possiamo vedere come i casi, per priorità alta e media, presenti nel dataset, si concentrino nel range di età compreso fra i 30 e 45-50 anni. Nel caso di priorità bassa invece abbiamo un'altra concentrazione fra i 20+ e i 40 anni. Esaminiamo ora la distribuzione delle priorità del dataset in rapporto all'uso di alcolici da parte dei pazienti:

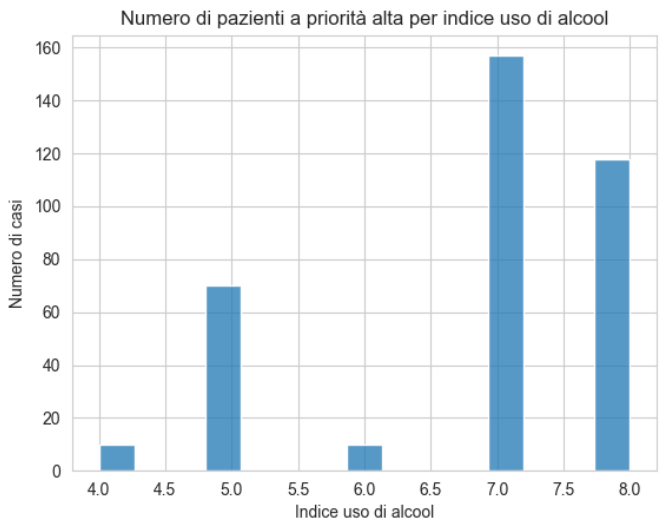

Fig.1 Distribuzione dei casi a priorità alta in rapporto al consumo di alcolici

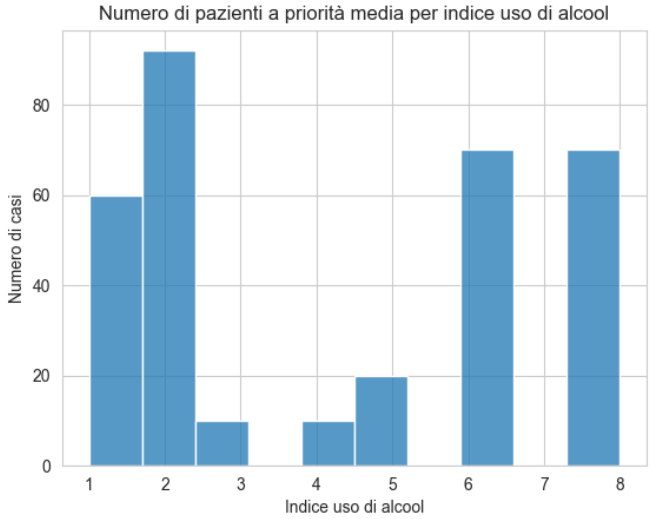

Fig.2 Distribuzione dei casi a priorità media in rapporto al consumo di alcolici

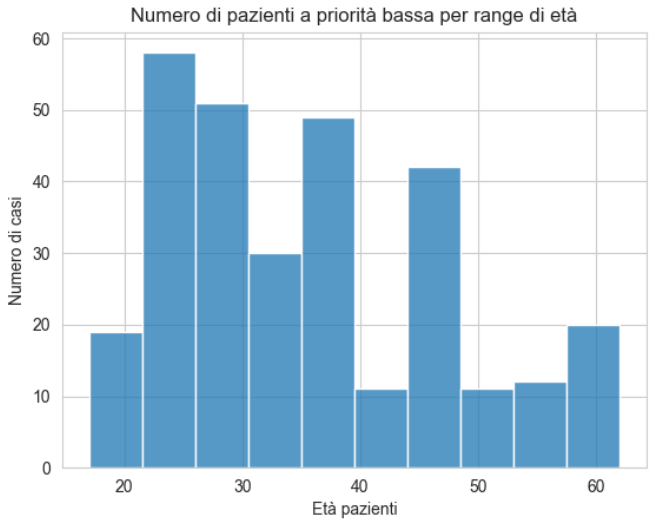

Fig.3 Distribuzione dei casi a priorità bassa in rapporto al consumo di alcolici

Viene evidenziato come, naturalmente, un maggior consumo, o addirittura abuso, di alcolici in-

fluenzi di molto il quadro clinico del paziente portando, nel caso di abuso di alcolici, ad essere classificati come pazienti a priorità alta.
Esaminiamo ora la distribuzione delle priorità in rapporto all'indice del fattore di rischio familiare:

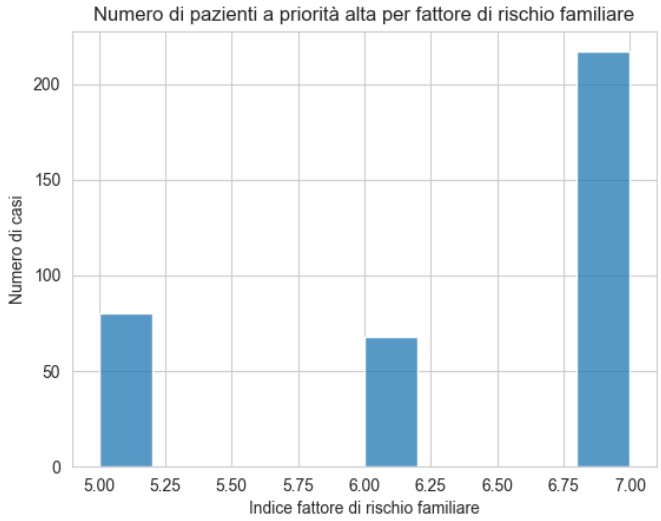

Fig.1 Distribuzione dei casi a priorità alta in rapporto all'indice del fattore di rischio familiare

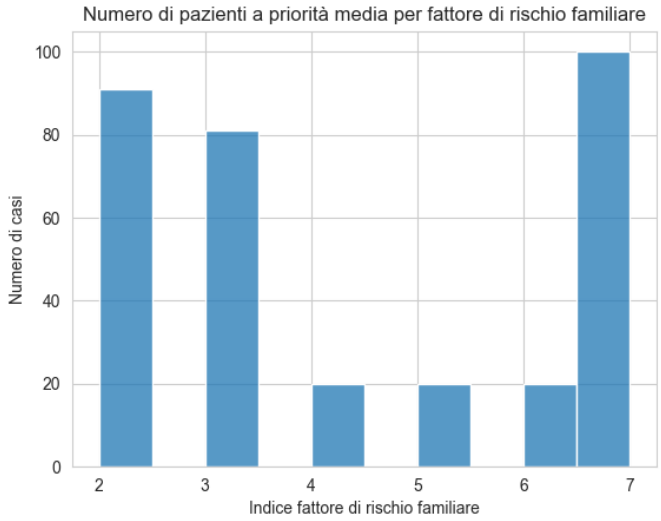

Fig.2 Distribuzione dei casi a priorità media in rapporto all'indice del fattore di rischio familiare

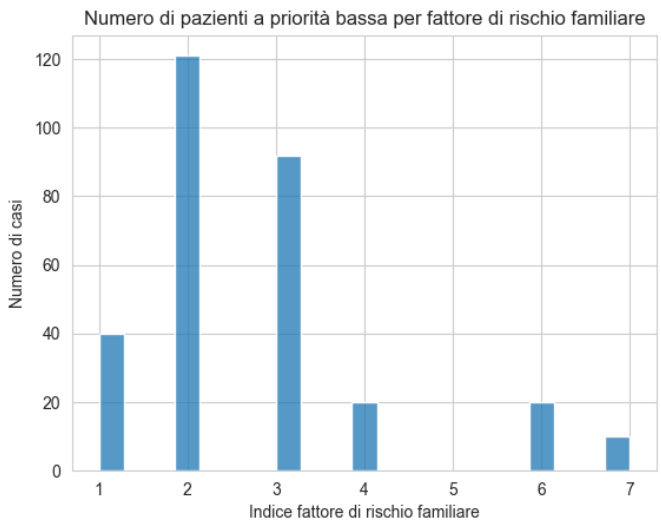

Fig.3 Distribuzione dei casi a priorità bassa in rapporto all'indice del fattore di rischio familiare

Ancora, possiamo osservare che, per priorità alta, il fattore di rischio familiare influisca molto sulla scelta di tale priorità.

Analizziamo, in seguito, la distribuzione delle priorità in rapporto all'esposizione al fumo attivo (paziente fumatore):

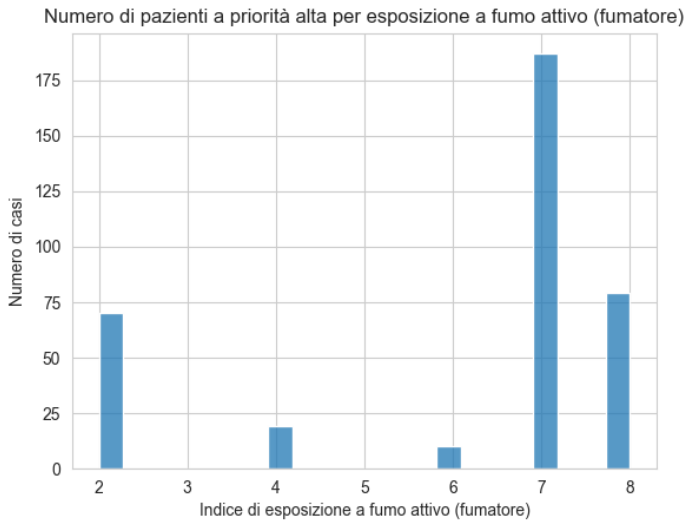

Fig.1 Distribuzione dei casi a priorità alta in rapporto all'esposizione al fumo attivo

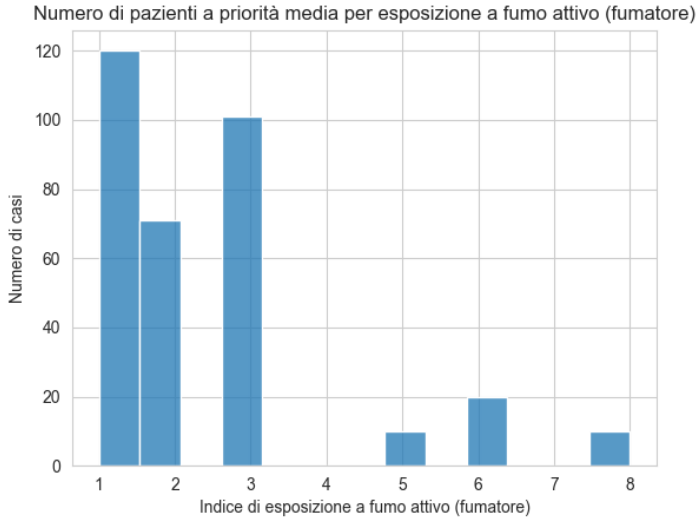

Fig.2 Distribuzione dei casi a priorità media in rapporto all'esposizione al fumo attivo

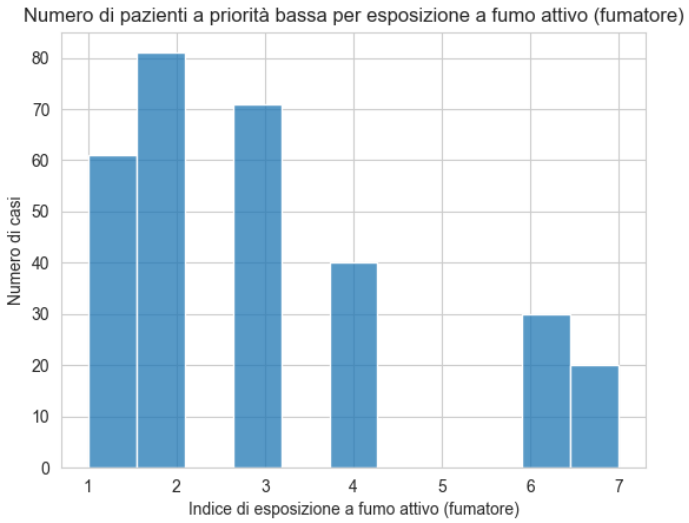

Fig.3 Distribuzione dei casi a priorità bassa in rapporto all'esposizione al fumo attivo

Anche in questo caso possiamo osservare come l'esposizione al fumo attivo (paziente fumatore) condizioni molto scelta di priorità di un paziente.
Possiamo concludere, dunque, che qualsiasi caratteristica del dataset condiziona, per valori alti, la scelta della priorità. Queste informazioni sono facilmente reperibili da una cartella clinica di un paziente e, per questo, si è scelto di utilizzare come caratteristiche *feature* tutte le caratteristiche possibili, ad esclusione del **sesso**, considerato "superfluo" data la presenza di tali caratteristiche maggiormente esplicite nella definizione della priorità e **PazienteId**, completamente superfluo ai fini della predizione.

Infine, tutti i grafici sopra utilizzati, sono disponibili nella cartella *"plots"* e sono stati realizzati, mediante l'uso di **Python** e le librerie **Seaborn e Matplotlib** nel file *"plotting_data.py"*, presenti nella repository del progetto.

5 Il modello

5.1 Scelte progettuali

È stato in precedenza discusso come il problema, da un punto di vista di modello di machine learning, sia un problema di **classificazione**. Si è scelto quindi di valutare modelli e confrontare i loro risultati sulla base degli algoritmi più utilizzati per modelli di classificazione. In particolare verrano descritti ed utilizzati gli algoritmi **Naive Bayes** e **Alberi decisionali**.

5.2 Naive Bayes

Il classificatore Naive Bayes si basa sull'utilizzo del teorema di Bayes, da cui prende il nome.
Il teorema di Bayes:

$$P(A|B) = \frac{P(B|A)P(A)}{(P(B)}$$

Che permette di calcolare la probabilità di un evento condizionato.// Il processo di esecuzione dell'algoritmo avviene in **tre** step:

1. **Calcolo della probabilità della classe:** Le probabilità di classe sono semplicemente le frequenze delle istanze che appartengono a ciascuna classe divisa per il numero totale di istanze;

2. **Calcolo della probabilità condizionata:** si applica il teorema di Bayes, per determinare le probabilità condizionate delle caratteristiche del problema;

3. **Decisione:** identificata nella classe che ottiene il valore di probabilità più elevato.

Il classificatore Naive Bayes, d'altro canto, non tiene conto del significato di una combinazione di valori ma assume e predice sulla base delle probabilità ottenute durante il calcolo. Questa caratteristica verrà tenuta in considerazione nella scelta definitiva del modello presente alla sezione *5.6 Scelta del modello*.

5.3 Alberi decisionali

L'algoritmo mira a realizzare un albero dove i nodi rappresentano un sotto-insieme delle caratteristiche del problema e gli archi rappresentano delle decisioni.
L'algoritmo punta a realizzare l'albero e le decisioni sulla base di regole inferite dall'analisi delle caratteristiche del problema e come esse si leghino alla caratteristica da predire.
L'algoritmo si basa su dei semplici step:

1. **Posiziona la migliore caratteristica del training set come radice dell'albero**, un passaggio fondamentale che verrà illustrato subito dopo;

2. **Divide il training set in sotto-insiemi**, ogni sotto-insieme dovrebbe essere composto di valori simili per una certa caratteristica

3. **Ripete passaggio 1 e 2**, fino al raggiungimento di un nodo foglia in ogni sotto-albero.

Abbiamo evidenziato come il primo punto sia un passaggio importante, in quanto, scegliendo una caratteristica in grado di dividere l'insieme di test in sotto-insiemi omogenei influisce di molto sulle prestazioni e correttezza dell'algoritmo.
Tale scelta viene realizzata utilizzando la metrica dell'**Information Gain**.
L'Information Gain ci permette di stabilire il grado di "purezza" di una caratteristica, ovvero, quanto tale attributo sarà in grado di dividere adeguamente il dataset.
Fondamentale è il concetto di **entropia**, che misura quanto un messaggio è ambiguo e difficile da capire.
La formula dell'entropia è la seguente:

$$H(D) = \sum_{c} p(c) * \log_2 p(c)$$

Tale formula è usata nella formula dell'Information Gain:

$$Gain(D, A) = H(D) - \sum_{v \in values(A)} \frac{|D_v|}{|D|} * H(D_v)$$

Dove:

- D è l'entropia del dataset;
- D_v è il sottoinsieme di D per cui l'attributo A ha valore v;
- $|D_v|$ è il numero di elementi di D_v;
- $|D|$ è il numero di elementi del dataset.

5.4 Training del modello

Il modello è stato addestrato e testato usando la tecnica della *K-Fold Validation*, che permette di addestrare e testare il modello in maniera iterativa dividendo il dataset in k sottogruppi, dove k-1 sottogruppi verranno utilizzati per il training del modello e il restante gruppo come test.
Questo strategia, ripetuta per tutti i k gruppi, ci permette di fornire delle stime più accurate del modello non limitandosi ad addestrare e testare il modello una singola volta.
Viene letto il dataset e impostata la variabile *feature*, contenente tutti i parametri sulla quale il modello basa la propria predizione, infine, impostate la variabile *X*, che contiene tutte le righe del dataset sulla base delle sole colonne stabilite dalla variabile *feature* e la variabile *y*, che contiene i valori di tutte le righe del dataset della sola variabile da predire indicata dalla colonna *Level*:

```
def training_and_testing_models():
    #lettura dataset
    df = pd.read_csv('./dataset.csv')

    #drop delle caratteristiche inutili al modello
    df.drop('PatientId', inplace=True ,axis=1)
    df.drop('sesso', inplace=True ,axis=1)

    #setting delle feature
    features = ['eta', 'indice_inquinamento_ambientale','indice_uso_alcolici',
    'grado_allergia','grado_rischio_lavorativo',
    'indice_fattori_rischio_familiare','indice_malattie_croniche',
    'indice_alimentazione_scorretta','indice_obesita',
    'grado_esposizione_fumo_attivo','grado_esposizione_fumo_passivo',
    'indice_dolori_localizzati','indice_emottisi','indice_astenia',
    'indice_perdita_peso','indice_dispnea','indice_respiro_sibilante',
    'indice_disfagia','stato_dita_di_Ippocrate','stato_immunodepressione',
    'indice_tosse_secca','indice_russamento']

    #si trasforma il dataframe in un array per poter eseguire lo split
    #della k-fold validation
    df_array = np.array(df)

    #si salva in X il dataframe sulle sole colonne feature
    X = df[features]

    #setting della variabile da predire
    y = df['priorita']
```

È stato scelto k=10, permettendo una training quanto più esaustivo possibile.

```
    kf = KFold(n_splits=10)
```

Vengono inizializzati i due modelli:

```
    clf = DecisionTreeClassifier()
    gnb = GaussianNB()
```

Entrambi i modelli sono stati addestrati e valutati contemporaneamente utilizzando gli stessi gruppi ad ogni iterazione.

Di seguito lo snippet di codice che realizza il training tramite la strategia *K-Fold Validation* ed, infine, invoca la funzione per valutare i risultati ottenuti dalla k-esima iterazione:

```
    #inizializzo contatore utile per identificare iterazione del metodo kfold
    i = 0

    #ciclo per la kfold validation
    for train_index, test_index in kf.split(df_array):
        i = i+1

        #si ottengono gli array di training e test sulla base degli indici
        #ottenuti dall'iterazione k attuale
        X_train, X_test = X.loc[train_index], X.loc[test_index]
        y_train, y_test = y.loc[train_index], y.loc[test_index]

        #si allenano i due moduli tramite il set di training
        #dell'iterazione k attuale
        clf.fit(X_train,y_train)
        gnb.fit(X_train,y_train)

        #vengono stampate metriche di valutazione
        #dei due modelli sul test dell'iterazione k
        validation(clf,X_test,y_test,i,"clf")
        validation(gnb,X_test,y_test,i,"gnb")
```

5.5 Testing del modello

Il testing del modello è stato effettuato per ogni iterazione della strategia *K-Fold Validation*. Ad ogni iterazione viene calcolata la matrice di confusione del modello e le metriche più importanti per una corretta valutazione del modello. I risultati vengono iterativamente salvati su file. Di seguito, la funzione utilizzata iterativamente per calcolare le metriche utili alla valutazione del modello:

```
def validation(model,X_test, y_test,i,model_type):
    #viene usato il test set per valutare il classificatore
    pred = model.predict(X_test)

    #calcoliamo matrice di confusione
    matrix = confusion_matrix(y_test, pred)

    #calcoliamo metriche di valutazione
    report = classification_report(y_test, pred)

    if i == 1:
        open_type = "w"
    else:
        open_type = "a"

    #scriviamo su un file matrice di confusione ottenuta
    with open(f"./reports/matrix_report_{model_type}.txt",open_type) as f:
        f.write(f"{i} iterazione:\n")
```

```
        f.write(f"Matrice di confusione:\n")
        f.write(str(matrix))
        f.write('\n')

    #scriviamo su un file le metriche di valutazione ottenute
    with  open(f"./reports/metrics_report_{model_type}.txt",open_type) as f:
        f.write(f"{i} iterazione:\n")
        f.write("\nMetriche di valutazione:")
        f.write(str(report))
        f.write('\n')
```

Inoltre, bisogna evidenziare come i risultati vengano salvati su dei file appositi per ogni modello e metrica.
Otteniamo quindi **4** file:

- ***matrix_report_clf.txt***, contiene le matrici di confusione generate ad ogni test del modello *Decision Tree*;
- ***matrix_report_gnb.txt***, contiene le matrici di confusione generate ad ogni test del modello *Naive Bayes*;
- ***metrics_report_clf.txt***, contiene le metriche calcolate per ogni test del modello *Decision Tree*;
- ***metrics_report_gnb.txt***, contiene le metriche calcolate per ogni test del modello *Naive Bayes*.

5.6 Scelta del modello

Per ogni modello ora analizzeremo il risultato ”migliore” e quello ”peggiore” ottenuti nelle k iterazioni del test. Il modello *Decision Tree* ha riportato:

```
    Miglior risultato
    Metriche di valutazione:
                precision    recall  f1-score   support

        High       1.00      1.00      1.00        36
         Low       1.00      1.00      1.00        31
      Medium       1.00      1.00      1.00        33

    accuracy                           1.00       100
   macro avg       1.00      1.00      1.00       100
weighted avg       1.00      1.00      1.00       100

    Peggior risultato
    Metriche di valutazione:
                precision    recall  f1-score   support

        High       1.00      1.00      1.00        36
         Low       1.00      1.00      1.00        31
      Medium       1.00      1.00      1.00        33

    accuracy                           1.00       100
   macro avg       1.00      1.00      1.00       100
weighted avg       1.00      1.00      1.00       100
```

Da questo estratto delle metriche notiamo come il modello abbia ottenuto, per ogni iterazione, risultato **1.00 (100%)** per ogni metrica. Questi risultati ci permettono di dire che il modello non lavora bene con i dati o non è realmente adatto al problema poiché, nel mondo dell’intelligenza artificiale, non esiste un modello che possa essere definito ”perfetto” e che produca predizioni ”perfette”.

Valutiamo ora i risultati del modello *Naive Bayes*:

```
Miglior risultato
Metriche di valutazione:
              precision    recall  f1-score   support

        High       0.87      1.00      0.93        40
         Low       1.00      0.90      0.95        30
      Medium       0.93      0.83      0.88        30

    accuracy                           0.92       100
   macro avg       0.93      0.91      0.92       100
weighted avg       0.93      0.92      0.92       100

Peggior risultato
Metriche di valutazione:
              precision    recall  f1-score   support

        High       0.79      0.94      0.86        35
         Low       1.00      0.85      0.92        39
      Medium       0.76      0.73      0.75        26

    accuracy                           0.85       100
   macro avg       0.85      0.84      0.84       100
weighted avg       0.86      0.85      0.85       100
```

Possiamo quindi dire che il modello abbia ottenuto valori compresi fra lo **0.93 (93%)** e lo **0.85 (85%)**.
Quest'ultimo modello rappresenta una soluzione chiaramente "migliore" del modello che presentava una predizione "perfetta" ed è per questo il modello basato su Naive Bayes è stato scelto come soluzione.

6 Implementazione

6.1 Serializzazione del modello

Il modello è stato serializzato tramite la libreria ***pickle*** per poter essere utilizzato in maniera rapida in qualunque contesto sia necessario. Di seguito uno snippet del codice utilizzato per serializzare il modello:

```
import pickle

def serialize(model):
#indichiamo il nome del file in cui verra salvato il modello
file_name = 'trained_model.sav'

#effettuiamo il dump del modello nel file
pickle.dump(model, open(file_name,"wb"))
```

6.2 Integrazione con il sistema

Il primo passo è stato realizzare una funzione lato web-app che permetteva l'ottenimento del paziente e il salvataggio di quest'ultimo in un file CSV apposito, utile al modello per la definzione della predizione.

```
try{
    await axios.get('http://localhost:3007/pazienti/' + id)
    .then(function (response) {
        let paziente = response.data

        converter.json2csv(paziente, (err,csv) => {
            if(err) {
                throw err
            }

            try{
                fs.writeFile('../modello_FIA/patient_to_predict.csv',
                csv, (err,data) => {
                    if(err){
                        console.log(err)
                        return;
                    }
    //altro codice...
```

Continuando, viene richiamato lo script Python per la predizione utilizzando la libreria standard di Node.js per la creazione di sotto-processi e viene permesso al chiamante di "ascoltare" lo stdout dello script appena chiamato.

```
    let priorita
    const python = spawn("python",["../modello_FIA/ml_predict.py",
    "../modello_FIA/patient_to_predict.csv"])
    python.stdout.on("data", function(data) {
        priorita = data.toString()
    })

    //altro codice...
```

La funzione presente nello script Python, deserializza prende in input il nome del file CSV contenente i dati del paziente, li estrae tramite libreria ***pandas***, de-serializza il modello addestrato in precedenza e lo utilizza per effettuare la predizione, restituendo il valore di quest'ultima al chiamante.

```
def predict(file_name):
    #viene deserializzato il modello
    loaded_module = pickle.load(open('../Modello_FIA/trained_model.sav','rb'))
    #lettura del file csv contenente i dati del paziente da predire
    patient = pd.read_csv(file_name)

    #drop delle colonne non necessarie alla predizione
    #come id, nome, cognome, ...
    patient.drop(['_id','nome','cognome','cf','sesso',
```

```
    'dataNascita','telefono','email','priorita'], inplace=True, axis=1)

    #predizione e restituzione del valore predetto al chiamante
    pred = loaded_module.predict(patient)

    #viene stampato sul
    if(pred[0] == "High"):
        print("Alta")
    elif(pred[0] == "Medium"):
        print("Media")
    elif(pred[0] == "Low"):
        print("Bassa")

if __name__ == "__main__":
    #quando viene eseguito lo script viene richiamata
    #la funzione per la predizione passando
    #come argomento da linea di comando
    #il nome del file in cui sono memorizzati i dati del paziente
    predict(sys.argv[1])
```

Una volta terminato lo script, il risultato è presente nella variabile *"priorita"* che viene mandata alla prossima chiamata del sistema.

```
    python.on("close", (code) => {
        res.render(__dirname +
        "/views/schedulazione", {priorita: priorita})
    })
```

7 Il futuro del progetto

Chiaramente, il modello è stato realizzato sulla base di un dataset pubblico che prevede solo alcune delle caratteristiche che potrebbero influenzare la scelta di una priorità per un paziente. In futuro il modello potrebbe essere riadattato tramite retraining sulla base di un dataset ospedaliero "reale" fornito in collaborazione con ospedali che intendono collaborare ed utilizzare il sistema **ChemoSmart**. Allo stesso tempo il sistema, ad oggi, non è in grado di modificare dinamicamente i valori su cui il modello basa la propria predizione, in release future il sistema prevederà la possibilità di modificare questi parametri e permettere una nuova predizione sulla base dei nuovi dati aggiornati allo stato corrente di salute del paziente.