UNIVERSITÀ DEGLI STUDI
DI SALERNO

Corso di Laurea in Informatica

Medi-AI

Autori:
Gisolfi Andrea
Favale Giacomo
Merola Antonio
Nigro Giovanni

Docente:
Prof. Palomba Fabio

Contents

1 Introduzione: Sistema attuale e Sistema proposto

L'intelligenza artificiale è sempre più integrata nella nostra vita quotidiana. Esempi come ChatGPT, DALL-E e tecnologie simili offrono un supporto significativo, consentendo conversazioni fluide, creazione e manipolazione di immagini, e altro ancora. Con le loro avanzate capacità, questi strumenti stanno trasformando il modo in cui interagiamo con la tecnologia, aprendo nuove possibilità in molti settori.
Nel campo medico, l'intelligenza artificiale sta svolgendo un ruolo fondamentale nel fornire supporto agli utenti. In questo contesto, spicca **MediAI**, un assistente medico virtuale in grado non solo di fornire diagnosi non ufficiali, ma anche di valutare la gravità di una situazione specifica. Ciò non solo evita affollamenti inutili nei centri medici, ma fornisce agli utenti indicazioni e consigli, contribuendo a garantire maggiore sicurezza e tranquillità.
La presenza di tecnologie come MediAI si traduce in un accesso più rapido e affidabile alle informazioni mediche, consentendo alle persone di ottenere indicazioni importanti sulla propria salute senza dover necessariamente recarsi fisicamente in un centro medico. Questo non solo ottimizza l'uso delle risorse sanitarie, ma permette anche agli individui di prendersi cura della propria salute in modo più consapevole e tempestivo.

2 Definizione del problema

2.1 Obiettivi

Lo scopo del progetto consiste nella realizzazione di **un modulo di intelligenza artificiale**, il quale dovrà essere integrato ad una applicazione, che permetta agli utenti di **interagire** tramite **input testuali** rappresentanti i **sintomi**.
Il bot consente quindi agli utenti di inserire i singoli sintomi, di ricevere domande relative ai vari sintomi inseriti e, partendo da queste, di ricevere una **diagnosi preliminare**.

2.2 Specifica PEAS

Di seguito viene riportata le informazioni riguardante la **specifica PEAS**:

- **Performance**: La misura di prestazione si basa, oltre che sui tempi di risposta dell'agente, sulla correttezza della diagnosi fornita
- **Environment**: L'ambiente su cui opera il bot è costituito dalla sequenza di input forniti dall'utente sulla quale poi esso esegue la ricerca
- **Actuators**: L'agente effettua operazioni sull'ambiente fornendo una diagnosi all'utente
- **Sensors**: L'agente percepisce l'ambiente tramite l'input del sintomo e le risposte a sintomi associati forniti dal bot.

2.2.1 Caratteristiche dell'ambiente

L'ambiente su cui il bot opera risulta:

- **Parzialmente Osservabile**: Il bot permette l'inserimento di un singolo sintomo per volta, fornendo poi domande riguardanti eventuali sintomi correlati

- **Deterministico**: Le domande che il bot fornisce in merito a sintomi correlati dipendono dall'input inserito e dalle risposte che vengono fornite ogni volta
- **Sequenziale**: L'agente tiene traccia del sintomo inserito e delle risposte fornite ad ogni passo per altre evetntuali richieste o per fornire una risposta.
- **Statico**: L'agente effettua la ricerca sul singolo sintomo inserito, di conseguenza l'ambiente rimane statico ad ogni passo della sua deliberazione
- **Discreto**: L'ambiente fornisce un numero limitato di richieste sulle quali poi basarsi per fornire una diagnosi insieme a dei consigli su cosa fare
- **Singolo Agente**: L'ambiente consente la presenza di un singolo agente, ovvero il bot con cui si sta avendo la conversazione.

2.3 Analisi del problema

Il problema preso in esame è un'istanza di **apprendimento supervisionato**; più in particolare è un problema di **classificazione**. In base a degli input forniti dall'utente rappresentanti sintomi, il bot va a fornire una diagnosi preliminare, composta da un possibile riscontro medico, con annesse indicazioni riguardante la gravità. Per poter effettuare tali operazioni si è deciso di sfruttare metodologie **machine learning**, rendendo non solo l'interazione meno meccanica, ma fornendo una maggiore sicurezza sulle risposte in quanto non generate staticamente ma tramite inferenze logiche. Data la natura del problema, le possibile strategie utilizzabili si riduceano principalmente ad Alberi Decisionali e Naive Bayes. Abbiamo scelto la strada degli alberi decisionali, spiegando nella sezione 5.2.2 anche le motivazioni di tale scelta.

3 Data understanding

3.1 Data acquisition

Per la realizzazione del modello di machine learning, abbiamo condotto ricerche online alla ricerca di dataset preesistenti idonei al nostro obiettivo. Dopo un'attenta valutazione e svariate ricerche, abbiamo individuato un insieme di dataset particolarmente adatti provenienti da **Kaggle**, una piattaforma rinomata per la distribuzione di dataset adatti a diversi ambiti e problemi. Tra questi, la scelta è ricaduta su "**Disease Symptom Prediction**".

3.2 Data examination

I dataset contengono informazioni dettagliate riguardanti la descrizione delle malattie, le relative precauzioni e i sintomi associati a ciascuna malattia specifica. In totale i dataset utilizzati sono 5:

- **"dataset.csv"**: questo è il dataset principale contenente l'insieme delle mallattie con i relativi sintomi associati;
- **"Training.csv"**: dataset di training composto da una colonna per ogni possibile sintomo e una per la predizione. Nella cella relativa ad un sintomo è presente il valore 1 o 0 a seconda se il sintomo è legato alla malattia predetta o meno;
- **"symptom_Description.csv"**: il seguente dataset riporta per ogni malattia una sua descrizione;

- **"symptom_precaution.csv"**: il seguente dataset riporta per ogni malattia le relative precauzioni;
- **"Symptom_severity.csv"**: il seguente dataset riporta la gravità, espressa con interi che vanno da 1 a 10, di ogni sintomo.

3.3 Data exploration

Di seguito riportiamo un'immagine raffigurante una piccola parte del dataset principale:

Malattia	Sintomo 1	Sintomo 2	Sintomo 3	Sintomo 4
AIDS	muscle_wasting	patches_in_throat	high_fever	extra_marital_contacts
Diabetes	fatigue	weight_loss	restlessness	lethargy
Gastroenteritis	vomiting	sunken_eyes	dehydration	diarrhoea
Bronchial Asthma	fatigue	cough	high_fever	breathlessness
Hypertension	headache	chest_pain	dizziness	loss_of_balance
Migraine	acidity	indigestion	headache	blurred_and_distorted_vision
Cervical spondylosis	back_pain	weakness_in_limbs	neck_pain	dizziness
Paralysis (brain hemorrhage)	vomiting	headache	weakness_of_one_body_side	altered_sensorium
Jaundice	itching	vomiting	fatigue	weight_loss
Malaria	chills	vomiting	high_fever	sweating
Chicken pox	itching	skin_rash	fatigue	lethargy
Dengue	skin_rash	chills	joint_pain	vomiting
Typhoid	chills	vomiting	fatigue	high_fever
hepatitis A	joint_pain	vomiting	yellowish_skin	dark_urine
Hepatitis B	itching	fatigue	lethargy	yellowish_skin
Hepatitis C	fatigue	yellowish_skin	nausea	loss_of_appetite
Hepatitis D	joint_pain	vomiting	fatigue	yellowish_skin
Hepatitis E	joint_pain	vomiting	fatigue	high_fever
Alcoholic hepatitis	vomiting	yellowish_skin	abdominal_pain	swelling_of_stomach
Tuberculosis	chills	vomiting	fatigue	weight_loss
Common Cold	continuous_sneezing	chills	fatigue	cough
Pneumonia	chills	fatigue	cough	high_fever
Dimorphic hemmorhoids(piles)	constipation	pain_during_bowel_movements	pain_in_anal_region	bloody_stool
Heart attack	vomiting	breathlessness	sweating	chest_pain
Varicose veins	fatigue	cramps	bruising	obesity

Nota: L'immagine non riporta tutte le colonne effettive del dataset, in quanto renderebbe impossibile la lettura del suo contenuto.

4 Data preparation

4.1 Data cleaning

Con **data cleaning** si intende l'insieme di procedure atte a far si che l'insieme di dati in analisi sia scevro da duplicati, dati mancanti, dati inconsistenti e in generale tutto ciò che può compromettere la qualità stessa dei dati. **Data Imputation**.

```
RangeIndex: 4920 entries, 0 to 4919
Data columns (total 18 columns):
 #   Column      Non-Null Count  Dtype
---  ------      --------------  -----
 0   Malattia    4920 non-null   object
 1   Sintomo 1   4920 non-null   object
 2   Sintomo 2   4920 non-null   object
 3   Sintomo 3   4920 non-null   object
 4   Sintomo 4   4572 non-null   object
 5   Sintomo 5   3714 non-null   object
 6   Sintomo 6   2934 non-null   object
 7   Sintomo 7   2268 non-null   object
 8   Sintomo 8   1944 non-null   object
 9   Sintomo 9   1692 non-null   object
 10  Sintomo 10  1512 non-null   object
 11  Sintomo 11  1194 non-null   object
 12  Sintomo 12  744 non-null    object
 13  Sintomo 13  504 non-null    object
 14  Sintomo 14  306 non-null    object
 15  Sintomo 15  240 non-null    object
 16  Sintomo 16  192 non-null    object
 17  Sintomo 17  72 non-null     object
```

Esaminando la foto qui sopra riportata, relativa al dataset principale, emerge un considerevole numero di dati mancanti. Questo si deve alla struttura del dataset, che presenta una colonna per le malattie e 17 colonne dedicate ai sintomi associati a ciascuna malattia. Tuttavia, poiché diverse malattie possono manifestare un numero variabile di sintomi, molte delle colonne relative ai sintomi rimangono vuote. Questa peculiarità rende impraticabile l'uso di metodi di imputazione dati convenzionali, poiché l'applicazione di tali tecniche comporterebbe la perdita di informazioni cruciali contenute in alcune colonne relative a specifiche malattie o potrebbe addirittura compromettere interi record correlati a una determinata patologia.
Allo stesso tempo, l'impiego di tecniche più avanzate, come la **most frequent imputation** o l'**imputazione deduttiva**, risulta altrettanto inadatto. L'adozione di queste strategie comporterebbe l'inclusione di sintomi in malattie che non li presentano, generando un'evidente inconsistenza nell'intero dataset.

index ▼	itching	skin_rash	nodal_skin_eruptions	continuous_sneezing	shivering	chills	joint_pain	stomach_pain	acidity	ulcers_on_tongue	muscle_wasting
4919	0	1	0	0	0	0	0	0	0	0	0
4918	0	1	0	0	0	0	1	0	0	0	0
4917	0	0	0	0	0	0	0	0	0	0	0
4916	0	1	0	0	0	0	0	0	0	0	0
4915	0	0	0	0	0	0	0	0	0	0	0
4914	0	0	0	0	0	0	0	0	0	0	0
4913	0	0	0	0	0	0	1	0	0	0	0
4912	0	0	0	0	0	0	0	0	0	0	0
4911	0	0	0	0	0	0	0	0	0	0	0
4910	0	0	0	0	0	0	0	0	0	0	0
4909	0	0	0	0	0	0	0	0	0	0	0
4908	0	0	0	0	0	0	0	0	0	0	0
4907	0	0	0	0	0	0	0	0	0	0	0
4906	0	0	0	0	0	1	0	0	0	0	0
4905	0	0	0	1	0	1	0	0	0	0	0
4904	0	0	0	0	0	1	0	0	0	0	0
4903	0	0	0	0	0	0	0	0	0	0	0
4902	0	0	0	0	0	0	1	0	0	0	0
4901	0	0	0	0	0	0	1	0	0	0	0
4900	0	0	0	0	0	0	0	0	0	0	0
4899	1	0	0	0	0	0	0	0	0	0	0
4898	0	0	0	0	0	0	1	0	0	0	0

Nell'immagine sopra, è possibile osservare un estratto del contenuto del dataset denominato "Training.csv". Dai dati visualizzati, si nota che il dataset presenta un totale di 4920 voci (indicizzate da 0 a 4919). Tuttavia, durante l'analisi è emerso che il dataset conteneva un significativo numero di voci duplicate, una condizione che potenzialmente avrebbe potuto causare problemi di **data leakage** o **overfitting** nell'addestramento dei modelli. Di conseguenza, si è proceduto con la pulizia del dataset, eliminando tutte le voci duplicate. Questa operazione ha portato ad un numero molto ridotto di voci, come indicato nell'immagine sottostante:

index ▼	itching	skin_rash	nodal_skin_eruptions	continuous_sneezing	shivering	chills	joint_pain	stomach_pain	acidity	ulcers_on_tongue	muscle_wasting	vomiting
303	0	1	0	0	0	0	0	0	0	0	0	0
302	0	1	0	0	0	0	0	0	0	0	0	0
301	0	1	0	0	0	0	0	0	0	0	0	0
300	0	1	0	0	0	0	0	0	0	0	0	0
299	0	0	0	0	0	0	0	0	0	0	0	0
298	0	1	0	0	0	0	0	0	0	0	0	0
297	0	1	0	0	0	0	1	0	0	0	0	0
296	0	1	0	0	0	0	1	0	0	0	0	0
295	0	1	0	0	0	0	1	0	0	0	0	0
294	0	1	0	0	0	0	1	0	0	0	0	0
293	0	1	0	0	0	0	0	0	0	0	0	0
292	0	0	0	0	0	0	1	0	0	0	0	0
291	0	1	0	0	0	0	1	0	0	0	0	0
290	0	0	0	0	0	0	0	0	0	0	0	0
289	0	0	0	0	0	0	0	0	0	0	0	0
288	0	0	0	0	0	0	0	0	0	0	0	0
287	0	0	0	0	0	0	0	0	0	0	0	0
286	0	0	0	0	0	0	0	0	0	0	0	0

5 Data Modeling

5.1 Lettura e operazioni sui dataset

Le operazioni iniziali riguardano la **lettura** del dataset di **training**, utilizzato sia per il training che per il testing tramite un apposito split. Si procede selezionando le **feature indipendenti** da quelle **dipendenti** sfruttando la naturale divisione in colonne. Il passo successivo coinvolge la definizione delle variabili di training da utilizzare nel modello, ossia **x_train** e **y_train**. Per la variabile dipendente, si utilizza un encoder, il **Label Encoder**, che viene prima allenato e poi impiegato per l'encoding da formato testuale a formato numerico. Questo passaggio non è essenziale per il modello in sé, ma risulta cruciale per successive operazioni che sfruttano tale encoding.

5.2 Scelta dell'algoritmo

Dopo le operazioni preliminari, si passa alla definizione del **classificatore**. A tale scopo si è deciso di comparare alcuni classificatori e valutare quale di questi presenta risultati migliori. In praticolare, i classificatori presi in cosiderazione sono: **Decision Tree** e **Naive Bayes**.

5.2.1 Decision Tree

L'algoritmo mira a creare un albero i cui nodi rappresentano un sotto-insieme di caratteristiche del problema e i cui archi rappresentano delle decisioni. L'obiettivo di un albero decisionale è predire il valore di una variabile target apprendendo semplici regole di decisione inferite dai dati di training. La procedura per la creazione dell'albero inzia con la selezione della caratteristica migliore, utilizzando l'**information gain** e l'**entropia**, che meglio dividerà il dataset in modo omogeneo. Individuata la caratteristica, il dataset verrà diviso sulla base dell'algoritmo e infine viene ripetuto il processo su tutti i sottoinsiemi fin quando non siano **puri**.

5.2.2 Naive Bayes

L'algoritmo considera le caratteristiche della nuova istanza da classificare e calcola la probabilità che queste facciano parte di una classe tramite l'applicazione del teorema di Bayes. L'algoritmo è chiamato **naive** poiché assume che le caratteristiche non siano correlate l'una all'altra. Di conseguenza, l'algoritmo non andrà a valutare in fase di classificazione la potenziale utilità data dalla combinazione di più caratteristiche.

5.2.3 Confronto tra modelli

Confrontiamo i risultati ottenuti da DecisionTree e Naive Bayes:

	Precision	Accuracy	Recall	F1-score
Decision Tree	0.7085	0.7128	0.6288	0.6411
Naive Bayes	0.8415	0.8228	0.8414	0.8214

Prima di analizzare la tabella è necessario capire cosa questi valori indicano:

- La **Precision** ci indica la proporzione delle malattie classificate come corrette che sono effettivamente corrette; maggiore è il valore, minore sarà il numero di falsi positivi del modello.
- L'**Accuracy** indica la proporzione delle predizioni corrette (true positive e true negative) rispetto al totale delle predizioni effettuate dal modello

- La **Recall** ci indica la proporzione di malattie correttamente identificate rispetto al totale delle malattie corrette presenti nel dataset; più alto è il valore, maggiore sarà la capacità del modello di individuare correttamente la malattia.
- La **F1-score** infine, altro non è che la media armonica tra precision e recall.

Dalla tabella possiamo quindi vedere come il Naive Bayes sia a tutti gli effetti migliore. Perché abbiamo scelto allora il Decision Tree? I fattori principali risiedono nella **explainability**. Di natura, il modello continuerà ad essere aggiornato e migliorato, facendo si che le prestazioni migliorino nel tempo. Poiché stiamo lavorando nel campo medico, avere un'alta explainability, e quindi essere in grado di spiegare **perché** una determinata scelta sia stata presa, è fondamentale al fine di aumentare la fiducia dell'utente nel modello. Infine, come precedentemente affermato nella sezione 5.2.2, Naive Bayes non va a considerare la correlazione tra le features, caratteristica fondamentale per la generazione di una diagnosi accurata in quanto, banalmente, stessi sintomi possono appartenere a diverse malattie.

5.3 Funzioni di supporto al modello

Il modello è ora allenato. Sono necessarie funzioni di supporto che consentano di sfruttare le informazioni fornite dal bot e le rendano accessibili agli utenti esterni, recuperando i dati dai file associati ***symptom_Description.csv***, ***symptom_precaution.csv*** e ***Symptom_Severity.csv***. Tra le funzioni principali troviamo:

- **getSeverityDict**: permette di ottenere informazioni sulla severità dei sintomi utilizzando il file *symptom_severity.csv*.

```
def getSeverityDict():
    global severityDictionary
    with open('MasterData/Symptom_severity.csv') as csv_file:
        csv_reader = csv.reader(csv_file, delimiter=',')
        try:
            for row in csv_reader:
                if len(row) >= 2:
                    _diction = {row[0]: int(row[1])}
                    severityDictionary.update(_diction)
        except Exception as e:
            logging.error(f"Si è verificato un errore imprevisto: {str(e)}")
```

- **tree_to_code**: è la funzione chiave dell'intero progetto; si occupa di scorrere i vari nodi presenti nell'albero basandosi sulle informazioni inserite dall'utente e, attraverso la funzione interna **diagnose**, fornisce la diagnosi preliminare.

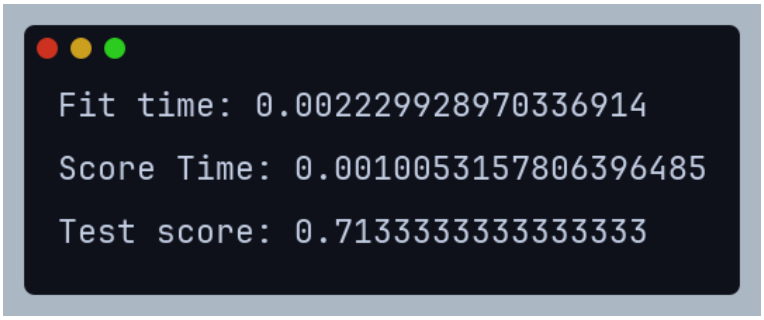
```
def tree_to_code(tree, feature_names):
    global num
    tree_ = tree.tree_
    feature_name = [
        feature_names[i] if i != _tree.TREE_UNDEFINED else "undefined"
        for i in tree_.feature
    ]

    chk_dis = ",".join(feature_names).split(",")

    while True:
        print("Che sintomo stai riscontrando?\t")
        disease_input = input("")
        disease_input.replace( _old: " ",  _new: "_")
        conf, cnf_dis = check_pattern(chk_dis, translator.translate(disease_input, src="it").text)
        if conf == 1:
            print("Ho trovato i seguenti sintomi in base alla tua risposta: ")
            for num, it in enumerate(cnf_dis):
                print(num, ")", translator.translate(it.replace("_", " "), dest="it").text)
            if num != 0:
                print(f"Che sintomo in particolare? (0 -{num}): ", end="")
                conf_inp = int(input(""))
            else:
                conf_inp = 0
            disease_input = cnf_dis[conf_inp]
            break
        else:
            print("Inserisci un sintomo valido.")

    while True:
        try:
            num_days = int(input("Da quanti giorni? "))
            break
        except:
            print("Inserisci un input valido.")
```

```
def diagnose(node, depth):
    if tree_.feature[node] != _tree.TREE_UNDEFINED:
        name, threshold = feature_name[node], tree_.threshold[node]
        val = 1 if name == disease_input else 0

        next_node = tree_.children_left[node] if val <= threshold else tree_.children_right[node]
        diagnose(next_node, depth + 1)
    else:
        present_disease = print_disease(tree_.value[node])
        symtomps_given = reduced_data.columns[reduced_data.loc[present_disease].values[0].nonzero()]
        print("Stai sperimentando qualche sintomo tra questi?")
        symptoms_exp = []
        for sym in list(symtomps_given):
            if sym != disease_input:
                print(translator.translate(sym.replace("_", " "), dest="it").text + "?", end=" ")
                while True:
                    inp = input("")
                    if inp == "si" or inp == "no":
                        break
                    else:
                        print("Dai una risposta valida, i.e. (si/no): ", end="")
                    if inp == "si":
                        sym.replace(" ", "_")
                        symptoms_exp.append(sym)


        disease = translator.translate(present_disease[0], dest="it").text
        diagnosis_text = f"\nIn base alle mie ricerche potresti avere {disease}"
        print(translator.translate(diagnosis_text, dest="it").text)
        print("\n" + translator.translate(descriptionDictionary[present_disease[0]], dest="it").text)
        print("Prendi le seguenti precauzioni:")
        for i, precaution in enumerate(precautionDictionary[present_disease[0]], 1):
            precaution = translator.translate(precaution, dest="it").text
            print(f"{i}) {precaution}")
diagnose( node: 0,  depth: 1)
```

6 Evaluation

Una volta implementato, inizia la fase di **validazione del modello**. Per questa operazione, abbiamo adottato la tecnica della **Cross Validation** o **Convalida Incrociata**. Questa strategia prevede la suddivisione del set in analisi in **k fold o porzioni**, ognuna delle quali viene considerata, in modo iterativo, come testing set, lasciando i restanti k-1 come training set. Attraverso la cross-validation otteniamo una visione delle prestazioni

del modello. In particolare, la funzione **cross_validate** ci fornisce informazioni sulle tempistiche di calcolo del modello, nonché sul punteggio ottenuto. Vediamo come il modello risulta apprendere molto velocemente con prestazioni buone, portando ad un test score di 0.71/1.0.



	Precision	Accuracy	Recall	F1-score
Decision Tree	0.7085	0.7128	0.6288	0.6411

Le prestazioni che possiamo osservare qui sono ampiamente impattate dal numero di elementi appartenenti al set. Come visto nella sezione 4.1, il set che inizialmente presentava quasi 5000 elementi, dopo una semplice operazione di eliminazione dei duplicati se ne ritrovava con soli 300. Ciò implica che il modello non è in grado di apprendere al meglio pattern sui dati stessi, portando ad un importante abbassamento delle prestazioni.

7 Deployment

Medi-AI è parte integrante di MediCare, applicazione in sviluppo che si propone di fare da intermediario tra popolazione e struttura sanitaria italiana. Il bot è inserito all'interno dell'applicazione con l'idea di fornire una rapida diagnosi preliminare, donando all'utente un senso di rapida risposta e tranquillità oltre che fornire consigli eventualmente fosse necessaria una visita fisica dal medico. In questi casi, la struttura MediCare fornisce la possibilità di ricerca degli ospedali non che della prenotazione ad uno specifico medico.
Tuttavia, per **questioni prettamente tempistiche**, non è stato possibile effettuare l'integrazione del bot all'interno dell'applicazione. Per tale motivo, a puro scopo dimostrativo, è stata utilizzata l'interfaccia da terminale.