UNIVERSITÀ DEGLI STUDI DI SALERNO

DIPARTIMENTO DI INFORMATICA

Tesina per il progetto di

Fondamenti di Data Sciene e Machine Learning

Medical Dialogue System

Annunziata ELEFANTE Mat. 0552501100

Benedetta COCCARO Mat. 0522501081

ANNO ACCADEMICO 2020/2021

Indice

Capitolo 1

Introduzione

Viviamo ormai in un'epoca in cui la tecnologia è protagonista. Ogni giorno usufruiamo di servizi che semplificano le attività della nostra vita, dalle più semplici e frivole alle più complesse e rilevanti. Tra queste ultime possiamo prendere in considerazione i numerosi sistemi che sono stati realizzati in questi anni, in ambito medico e clinico, con l'obiettivo di fornire aiuto e supporto a persone, medici e pazienti. Questo settore che è stato appena introdotto è meglio conosciuto come l'*eHealth* (Salute Digitale) o *telemedicina*. Il fine che porta avanti lo studio in questo campo è la realizzazione di nuove tecnologie informatiche e di telecomunicazione a vantaggio della salute umana che includano, oltre alle cartelle cliniche elettroniche e agli strumenti di telemedicina, l'impiego di tecnologie wireless mobili (*mHealth*) e di dispositivi indossabili (*wearables*). Il ruolo costituito da questo nuovo ambito *tech* della medicina oggi è considerato fondamentale per conseguire la copertura sanitaria universale (*Universalhealthcoverage*, *UHC*) che fa parte dei *SustainableDevelopmentGoals* che le Nazioni Unite puntano a raggiungere entro il 2030 e che mira alla riduzione del tasso di mortalità materna e infantile, alla scomparsa delle epidemie di AIDS e soprattutto alla copertura sanitaria per ogni individuo.

In questo documento verrà presentata e descritta una nuova tecnologia appartenente all'ambito della *telemedicina*. Il sistema che è stato realizzato è un Chatbot, il cui obiettivo è quello di automatizzare le diagnosi a distanza in

ambito medico.

Il sistema implementato, attraverso un dialogo con i pazienti, ha come fine quello di effettuare la diagnosi di malattia raccogliendo i sintomi forniti dall'utente grazie ai quali effettuerà la valutazione.

Il dataset a cui si è fatto riferimento per l'implementazione del sistema è MDD ($MedicalDiagnosisDialogue$). Il seguente dataset include 2.374 dialoghi, 12 tipi di malattia e 118 tipi di sintomi, inoltre è stato realizzato a partire da pazienti reali quindi molto vicino al concreto scenario di diagnosi clinica.

Capitolo 2

Stato dell'arte

Lo scopo principale di questo progetto è quello di sviluppare un chatbot che, mediante un dialogo con il paziente per conoscerne i sintomi, sia in grado di rappresentano l'intermediario ideale tra medico e paziente, fornendo un giudizio preliminare sulla possibile malattia di quest'ultimo. L'uso di un chatbot in ambito medico, infatti, offre molti vantaggi:

- aiuta a gestire contemporaneamente molte richieste, snellendo i processi amministrativi e smistando attività ripetitive;
- risponde alle esigenze di disponibilità e tempestività che i pazienti richiedono offrendo soluzioni immediate per problemi di salute meno gravi;
- offre un triage preventivo del paziente, raccogliendo i dati sulla sintomatologia per renderli successivamente disponibili alla valutazione del medico;
- permette di conoscere in anticipo il motivo della richiesta di contatto del paziente;
- riduce i tempi di consultazione e addirittura elimina le visite inutili.

Abbiamo quindi studiato dei lavori che ci aiutassero a capire come poter portare a termine questo compito nel migliore dei modi.

2.1 Serena

Serena è il chatbot di Paginemediche, che considerata la crescente necessità di snellire i processi e ridurre i tempi di attesa nel settore medico. Serena aiuta il medico ad effettuare il triage preventivo per l'accesso all'ambulatorio online e allocare la corretta priorità di appuntamento al paziente. In pratica, questo programma di intelligenza artificiale interroga i pazienti sui sintomi manifesti e in base alle risposte fornite è capace di orientare alle possibili patologie ad essi associate, oltre che offrire l'opportunità di prenotare anche una visita specialistica qualora necessario. Il chatbot, inoltre, è programmato anche per aiutare il medico a stabilire il rischio di contagio correlato all'infezione da Covid-19.
Con l'analisi di Serena si è capito quanto sia importante focalizzarsi sui sintomi del paziente, cercando di carpire il maggior numero di informazioni possibili per effettuare un'analisi accurata.

2.2 Buoy Health

Buoy Health, localizzata a Boston e fondata nel 2014, ha sviluppato un suo chatbot per il controllo dei sintomi sfruttando l'IA per fornire diagnosi personalizzate e più accurate. Il paziente fornisce i sintomi al chatbot che, tramite l'elaborazione del linguaggio naturale, li abbinerà a tutte le possibili condizioni e quindi chiederà chiarimenti per restringerli fino alla selezione migliore.

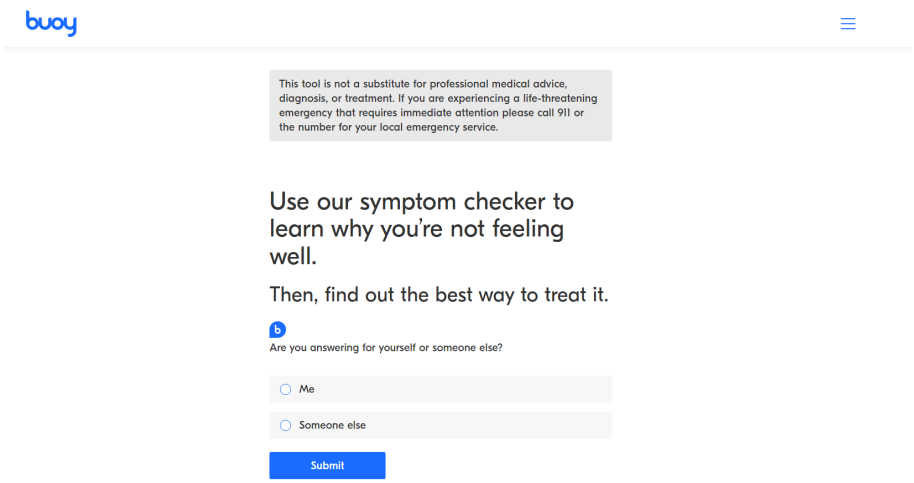


Figura 2.1: Interfaccia del sistema buoy health

La cosa che più ha colpito di Buoy Health, e che è stato tenuto presente per l'implementazione del nostro chatbot, è l'utilizzo di un sistema a risposte multiple, che mette il paziente nella miglior condizione possibile per poter rispondere in maniera chiara e precisa alle possibili domande sul suo stato di salute.

2.3 Molly

La californiana Sensely, fondata nel 2013, ha messo sul mercato un assistente medico virtuale dotato di IA per fare diagnosi basate sui sintomi descritti dal paziente tramite smartphone: Molly. La piattaforma usa algoritmi addestrati tramite numerosi dati clinici, come protocolli medici e informazioni sulle malattie croniche. In questo modo può interpretare i sintomi del paziente e raccomandare una diagnosi appropriata. I pazienti descrivono i loro sintomi all'assistente medico virtuale chiamato Molly, usando parole, testo, immagini e video. L'azienda mira a raggiungere un sentimento empatico e meno robotico sulla sua piattaforma e, oltre ai sintomi, considera anche l'umore del paziente.

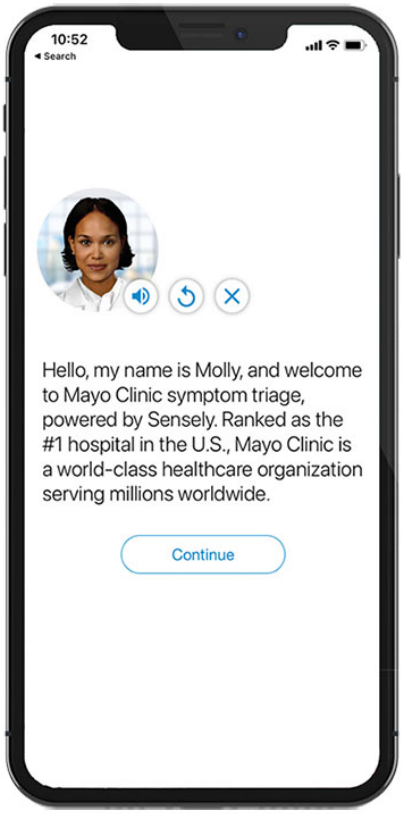


Figura 2.2: Interfaccia dell'assistente medico Molly

Tale progetto può essere tenuto in considerazione per futuri sviluppi del chatbot, in quando si passerebbe da una conversazione meccanica a una più naturale e colloquiale con il paziante.

Capitolo 3

Dataset

Come già detto in precedenze nell'introduzione, il dataset che è stato utilizzato per la realizzazione del Chatbot è *MDD* (Medical Diagnosis Dialogue). Questo dataset è stato realizzato convertendo cartelle cliniche di pazienti reali in dizionari usando il linguaggio di programmazione Python [1]. I dizionari riportati sono caratterizzati dai seguenti elementi:

- il tag della malattia;
- i sintomi che comporta la malattia.

3.1 Data profiling

In questa sezione verranno mostrati e descritti i passi che sono stati compiuti nella fase di data profiling.

3.1.1 Primi passi nel data profiling

I dati presenti all'interno del dataset *MDD* presentano un'estensione di tipo *.pk* (Peak File). I file con estensione .pk, sono file binari usati per memorizzare i dati di picco della forma d'onda per la rappresentazione visiva di una traccia audio.
Il primo passo che è stato compiuto per l'elaborazione del dataset, ha riguardato la conversione dei file con estensione .pk in file con estensione .csv. Una

volta portata a termine questa conversione il dataset MDD si presentava nella seguente forma:

	Nausea	Pain behind the breastbone	Chest tightness	...	Photophobia	disease_tag
0	-1	0	0	...	-1	Esophagitis
1	-1	0	0	...	-1	Esophagitis
2	-1	-1	-1	...	-1	Esophagitis
3	1	-1	-1	...	-1	Esophagitis
4	-1	-1	-1	...	-1	Esophagitis

Figura 3.1: Esempio tuple dataset

Nell'immagine sopra riportata sono presenti alcune tuple riguardanti la malattia "Esofagite", con alcuni dei sintomi che comporta, riscontrata in alcuni pazienti.
I valori che possono assumere i sintomi relativi ad una malattia sono:

- 0, nel caso in cui quel sintomo NON si sia verificato per quello specifico paziente;
- 1, nel caso in cui quel sintomo si sia verificato per quello specifico paziente;
- -1, quando in quel caso il sintomo non è stato preso in analisi.

3.1.2 Analisi e data cleaning

Una volta aver convertito il dataset con estensione .pk ad estensione .csv, è stato possibile proseguire con l'analisi dei dati contenuti in MDD.
In seguito allo studio di MDD è stato possibile fare alcune importanti osservazioni. Per alcuni sintomi presenti all'interno del dataset si è riscontrato, in relazione a tutte le malattie, solo valori -1 e 0 o, in alcuni casi, solo pari a -1.

Sono di seguito riportati alcuni esempi dei casi che sono stati appena descritti.

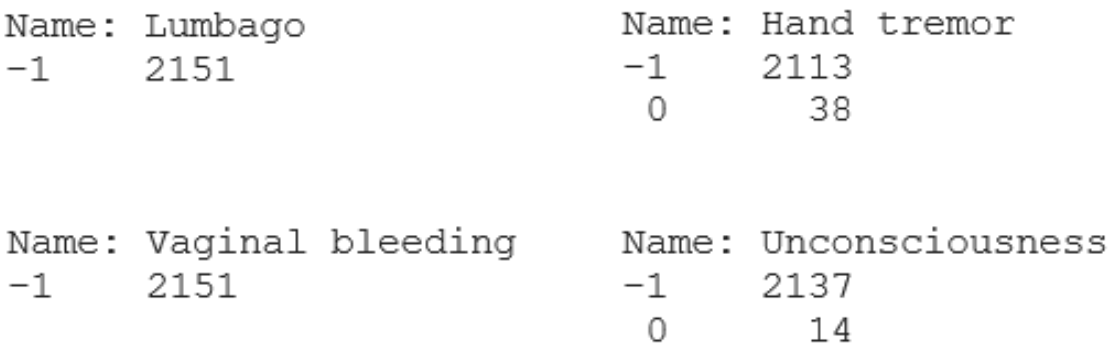


Figura 3.2: Esempio sintomi irrilevanti

Ricordiamo che il valore -1 e 0 associato al sintomo sta a significare rispettivamente che il sintomo non è stato preso in analisi e che il sintomo non si è verificato per quello specifico paziente. Dunque, analizzando gli esempi appena sopra riportati, possiamo concludere che i seguenti sintomi, caratterizzati solo da -1 o 0 sono irrilevanti per il nostro studio.
Una volta giunti a questa conclusione si è quindi intervenuti eliminando tutti i sintomi che presentavano la peculiarità appena analizzata. I 16 sintomi che sono stati eliminati dal dataset sono i seguenti:

- Runny nose;
- Hoarse;
- Night sweats;
- Hematuria;
- Painful urination;
- Defecate;
- Cyanosis;
- Mucus pus and blood in the stool;

- Nasal congestion;
- Headache and dizziness;
- Snore;
- Hearing loss;
- Hand tremor;
- Unconsciousness;
- Incontinence;
- Congestion of skin and mucous membranes.

Una volta terminato con le operazioni di analisi e pulizia del dataset MDD è ora possibile passare all'individuazione dei modelli di machine learning utili per continuare lo studio.

Capitolo 4

Modelli e risultati

In un problema di classificazione ($classification$) l'algoritmo assegna una classe (etichetta) a ogni esempio del dataset preso in considerazione, che nel nostro caso riguarda una serie di sintomi e diagnosi mediche. Nel capitolo precedente il dataset ha subito tutta una fase di preprocessing, volta a migliorare la qualità dei dati in nostro possesso.
A questo punto il passo successivo riguarda la selezione e l'addestramento di un modello di *machinelearning* volto ad eseguire la classifcazione, la cui bontà ed efficienza sarà poi valutata in base alla sua matrice di confusione (una rappresentazione dell'accuratezza di classificazione statistica) e a una serie di metriche di valutazioni:

- *accuracy* - rapporto tra l'osservazione correttamente prevista e le osservazioni totali;
- *precision* -rapporto tra le osservazioni positive previste correttamente e il totale delle osservazioni positive previste;
- *recall* - rapporto tra le osservazioni positive previste correttamente e tutte le osservazioni nella classe effettiva;
- $f1 - score$ - media ponderata di *precisione* e *recall*.

4.1 Modelli di classificazione

Prima di approfondire i vari classificatori adottati durante l'analisi e lo sviluppo di questo progetto, vanno definiti una serie di passi preliminari e comuni a ognuno di essi:

- dividere il dataset in due sezioni: quella relativa alle *features* e quella contenente le *label* indicanti la malattia;
- richiamare il metodo $train - test - split()$ della libreria Sklearn[2] così da generare le sezioni $X - train, X - test, y - train, y - test$.

Quest'ultimi saranno in parte destinati alla fase di training (70% dei dati), mediante l'applicazione del metodo $fit()$, e in parte alla fase di test (30% dei dati) mediante il metodo $predict()$.
Un'altro aspetto da sottolineare è che il nostro problema di classificazione non è binario, ma multiclasse. In tale casistica l'istanza non deve più essere classificata solo tra due classi, ma tra tre o più classi (9 nel caso specifico). Ciò ha limitato la scelta dei modelli adottabili, in quanto non tutti supportano la classificazione multiclasse.

4.1.1 Multi-Layer Perceptron

Il Multi-Layer Perceptron (MLP) è un modello di rete neurale artificiale che mappa insiemi di dati in ingresso in un insieme di dati in uscita appropriati. È fatta di strati multipli di nodi in un grafo diretto, con ogni strato completamente connesso al successivo. Per l'addestramento della rete, MLP utilizza la *backpropogation*, infatti viene definito un metodo di apprendimento profondo.

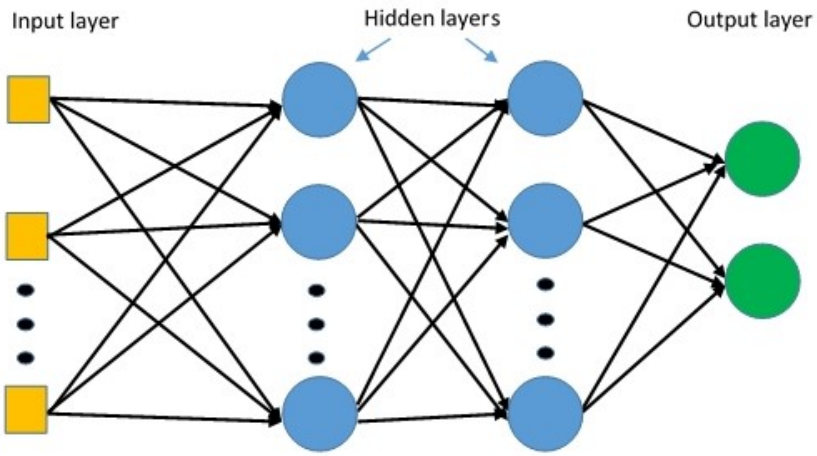


Figura 4.1: Architettura interna di un multilayer perceptron

Per poterlo utilizzare all'interno del nostro codice basta richiamare il metodo $MLPClassifier()$ della libreria Sklearn[3], che prende in input la dimensione degli strati nascosti e ottimizza la funzione $log - loss$ utilizzando LBFGS o la discesa stocastica del gradiente.

Una vola effettuate le fasi di training e test sui dati opportunamente divisi, si procede alla stampa della matrice di confusione

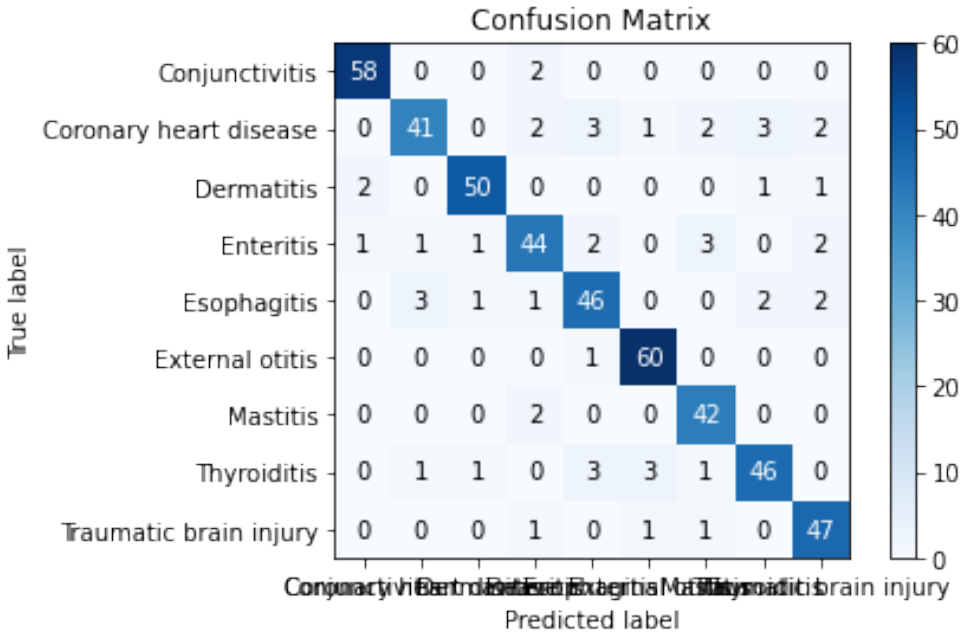


Figura 4.2: Matrice di confusione per il modello MLP

dalla cui rappresentazione è possibile avere un'idea dell'accuratezza del modello scelto. Dalla matrice, infatti, è ben visibile che malattie come la congiuntivite e l'otite vengono classificati quasi sempre nel modo corretto. Risulta più complicata, invece, la classificazione di malattie alle coronarie o alla tiroide. Analizzando tutti i risultati ottenuti (compresi quelli relativi alle metriche di valutazione) risulta che tale modello, avente un'*accuracy* dell'89% nel test contro una del 98% nel training, non è ancora abbastanza accurato per effettuare un'analisi medica.

4.1.2 Random Forest

Una Random Forest (RF) è un algoritmo di machine learning supervisionato ottenuto dall'aggregazione tramite *bagging* di alberi di decisione. È uno degli algoritmi più usati grazie alla sua accuratezza, semplicità e flessibilità. Il fatto che possa essere usato sia per compiti di classificazione che di regressione, unito alla sua natura non lineare, lo rende altamente adattabile a una gamma di dati e situazioni, come quelli relativi al problema in analisi.

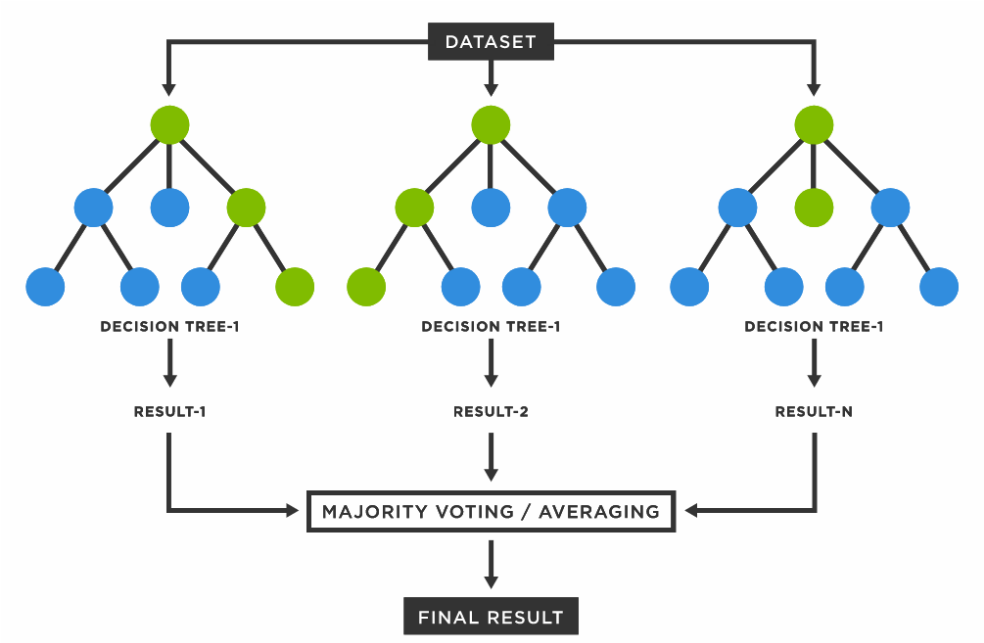


Figura 4.3: Diagramma semplificato di un random decision forest

Per adottarlo all'interno del codice si usa il metodo $RandomForestClassifier()$ della libreria Sklearn[4], il quale è un metastimatore che adatta una serie di classificatori ad albero decisionale su vari sottocampioni del set di dati e utilizza la media per migliorare l'accuratezza predittiva e controllare l'$overfitting$. La dimensione del sottocampione è controllata con il parametro $max-samples$ se $bootstrap = True$ (default), altrimenti viene utilizzato l'intero set di dati per costruire ogni albero.
Una volta effettuate le fasi di training e test, si procede alla stampa della matrice di confusione

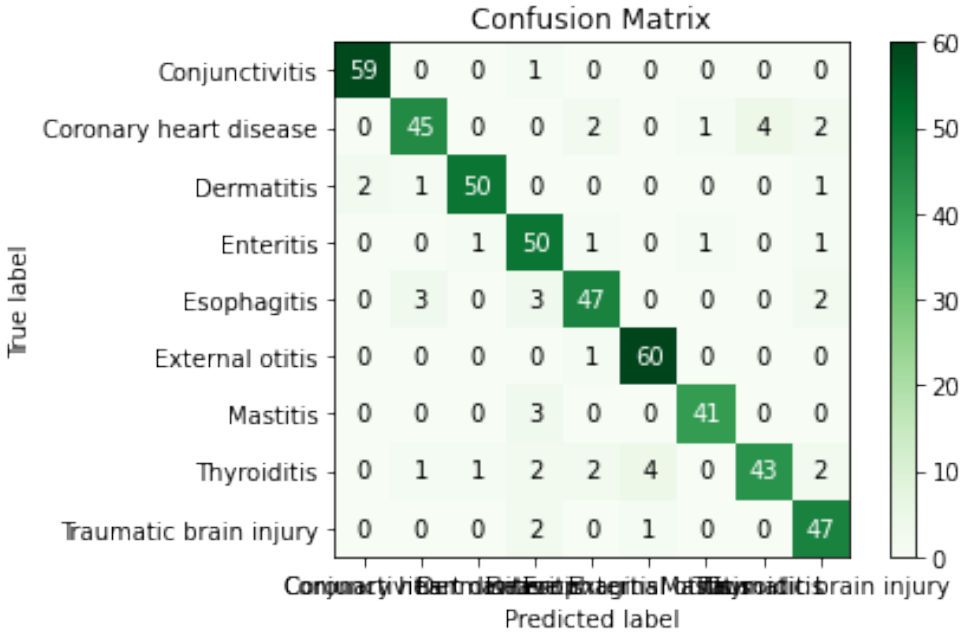


Figura 4.4: Matrice di confusione per il modello RF

da cui si evince una situazione piuttosto affine o peggiore di quella ottenuta mediante l'utilizzo del classificatore *multilayerperceptron*. Analizzando tali risultati e le relative metriche di valutazione risulta che tale modello, avente un'*accuracy* del 91% nel test contro una del 100% nel training, non è assolutamente adatto per effettuare una diagnosi medica.

4.1.3 Naive Bayes

Il Naive Bayes (NB) è un algoritmo per risolvere problemi di classificazione e apprendimento automatico che utilizza il teorema di Bayes. L'algoritmo Naive Bayes è un algoritmo probabilistico: calcola la probabilità di ogni etichetta per un determinato oggetto osservando le sue caratteristiche. Sceglie infine l'etichetta con la probabilità maggiore.

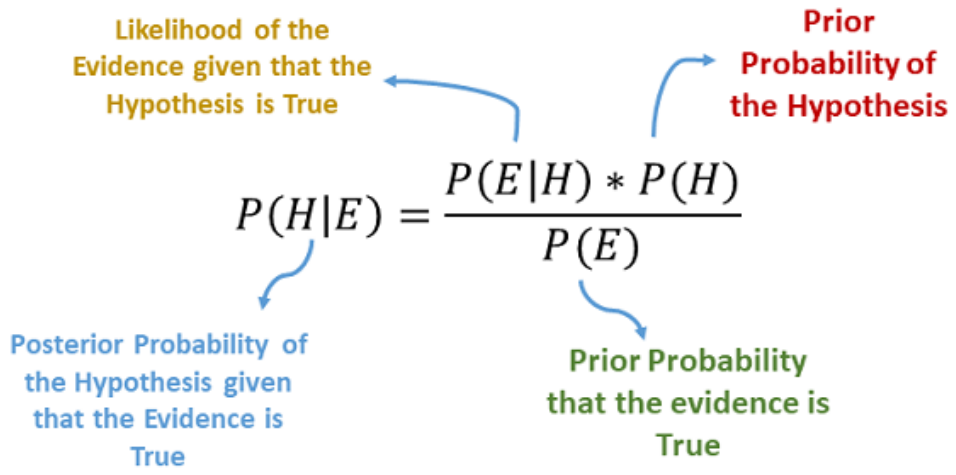


Figura 4.5: Algoritmo Naive Bayes

Per il suo utilizzo la libreria Sklearn mette a disposizione il metodo $GaussianNB()$[5]. Una volta effettuate le fasi di training e test, si procede alla stampa della matrice di confusione

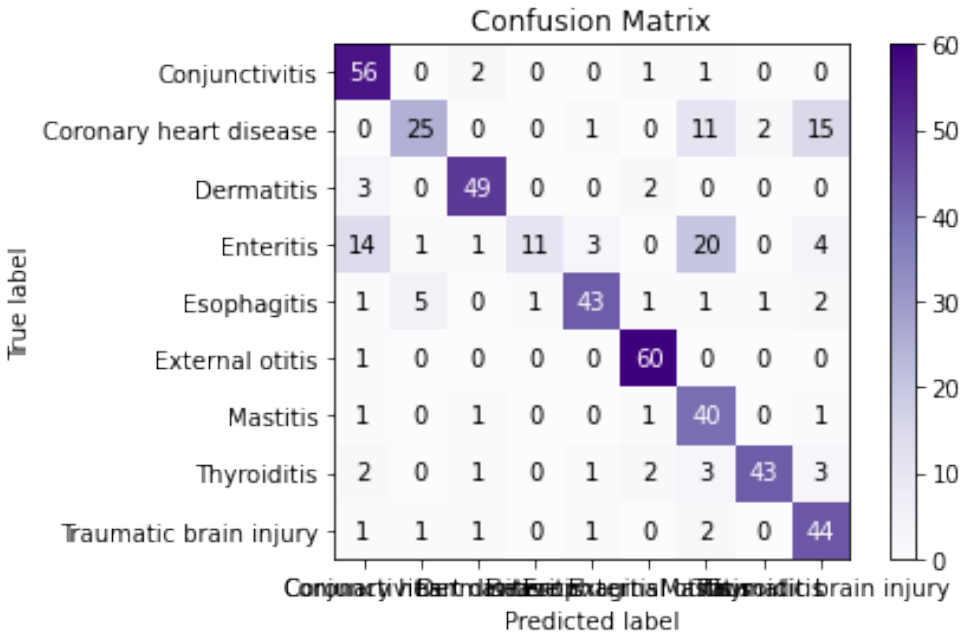


Figura 4.6: Matrice di confusione per il modello NB

la quale mostra la peggior situazione fin'ora ottenuta in quanto, pur classificando quasi sempre correttamente la congiuntivite e l'otite, risulta fortemente in dubbio sulle malattie relative alle coronarie o nel caso dell'enterite.
Analizzando le varie misure di valutazione ottenute e tali risultati risulta che in tale modello emerge un'*accuracy* del 76% nel test contro una del 79% nel training, la quale rappresenta un'accuracy troppo bassa per effettuare diagnosi mediche.

4.1.4 One-VS-All

Il metodo One Versus All (OVA), spesso chiamato One Versus The Rest (one-versus-the-rest), viene utilizzato per i problemi di classificazione multipla e prevede l'impiego di tanti classificatori binari quanti sono le classi da prevedere, ciascuno allenato a riconoscere una specifica classe. Oltre alla sua efficienza computazionale (sono necessari solo n-classi di classificatori), un vantaggio di questo approccio è la sua interpretabilità. Poiché ogni classe è rappresentata da uno e un solo classificatore, è possibile acquisire conoscenze sulla classe ispezionando il suo classificatore corrispondente.

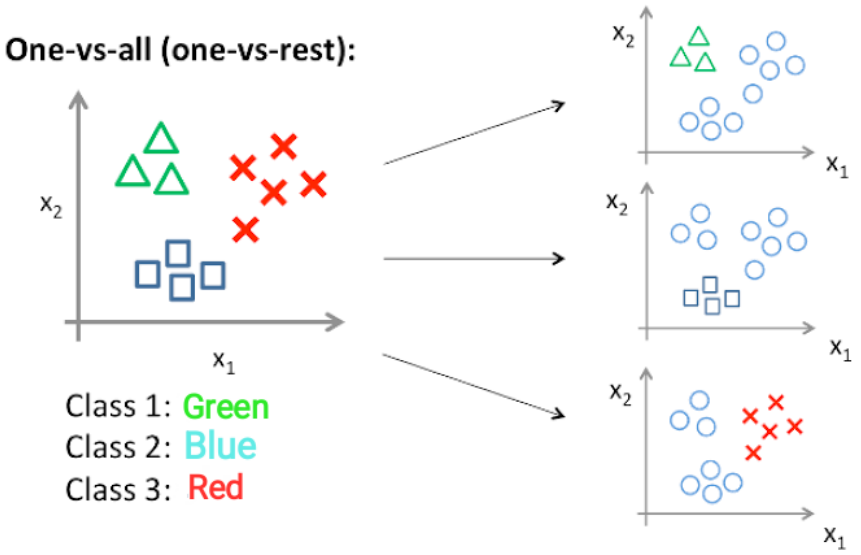


Figura 4.7: Struttura del modello one-vs-all

Per richiamarlo si usa il metodo $OneVsRestClassifier()$ della libreria Sklearn[6]. Per utilizzare questa funzione, è necessario fornire una matrice di indicatori per il target y quando si chiama $.fit$. In altre parole, le etichette target devono essere formattate come una matrice binaria 2D (0/1), dove [i, j] == 1 indica la presenza dell'etichetta j nel campione i. Questo stimatore utilizza il metodo della rilevanza binaria per eseguire la classificazione *multilabel*, che prevede l'addestramento di un classificatore binario indipendente per ogni etichetta.
Una volta effettuate le fasi di training e test, si procede alla stampa della matrice di confusione

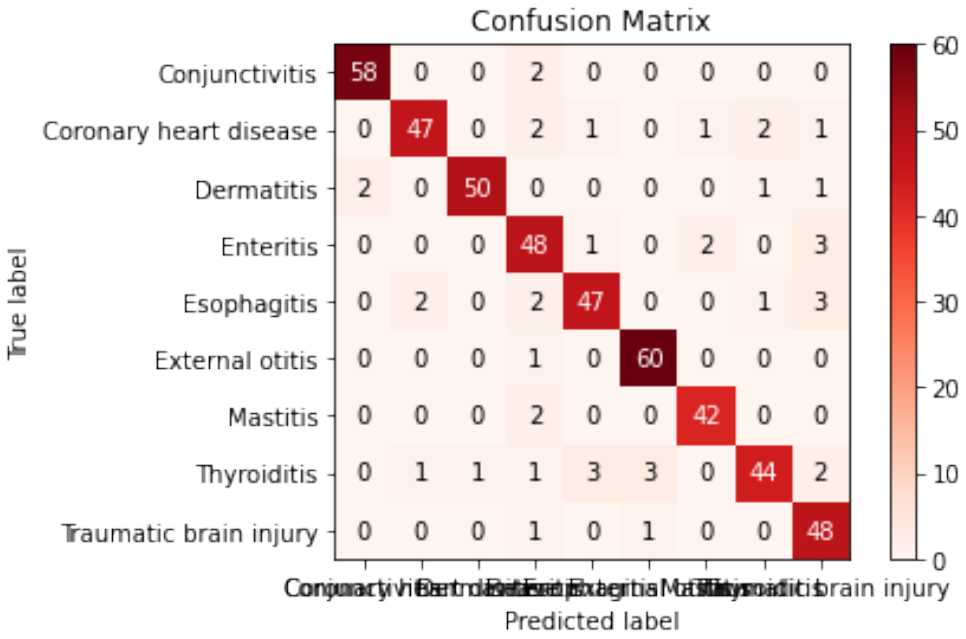


Figura 4.8: Matrice di confusione per il modello OVA

la quale mostra una situazione più bilanciata verso una classificazione corretta della maggior parte delle malattie (la più incerta sembra essere la diagnosi di mastite) rispetto a quelle mostrate dai classificatpri precedentemente utilizzati. Analizzando tutti i risultati ottenuti, si evince che tale modello, avente un'*accuracy* del 91% nel test contro una del 97% nel training, rappresenta una scelta efficiente per effettuare una diagnosi medica.

4.1.5 One-VS-One

Nella classificazione One-vs-One (OVO) la strategia utilizzata consiste nel montare un classificatore per ogni coppia di n classi e, al momento della previsione, viene selezionata la classe che ha ricevuto il maggior numero di voti. Poiché richiede l'adattamento di $n - classi * (n - classi - 1)/2$ classificatori, questo metodo è di solito più lento di $one - vs - all$, a causa della sua complessità

$$O(n - classi^2).$$

Tuttavia, questo metodo può essere vantaggioso per algoritmi come gli algoritmi kernel che non scalano bene con n-campioni. Questo perché ogni singolo

problema di apprendimento coinvolge solo un piccolo sottoinsieme dei dati, mentre con $one - vs - all$ l'intero set di dati viene utilizzato n-classi volte.

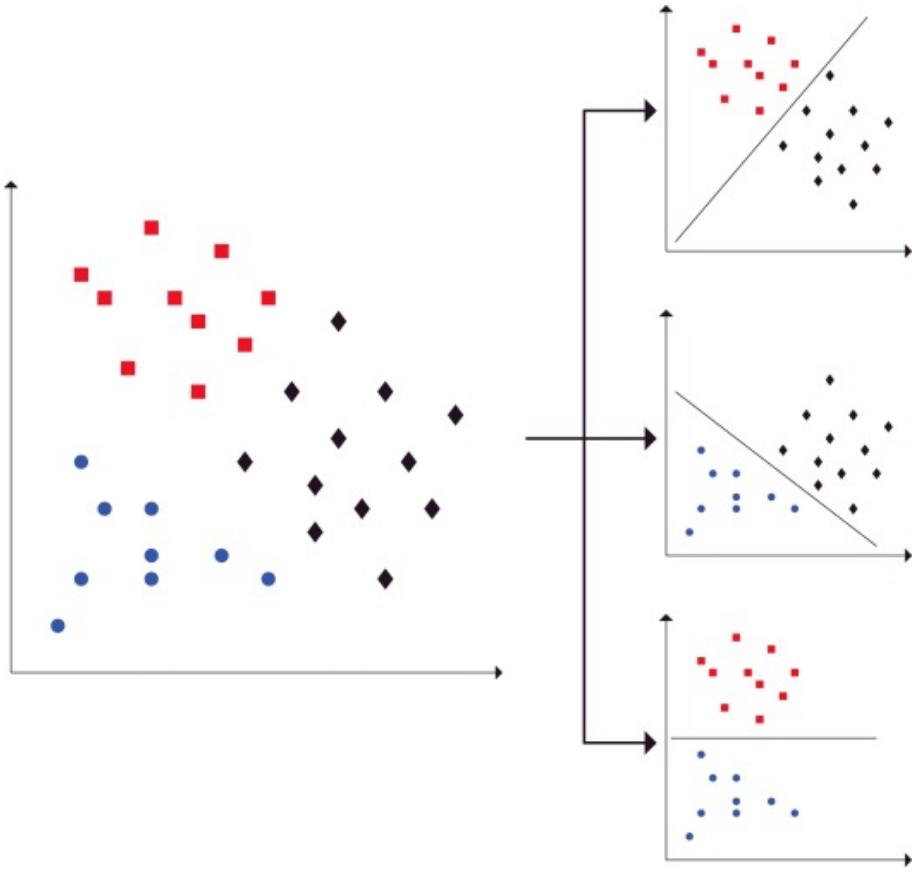


Figura 4.9: Decomposizione mediante one-vs-one

Per utilizzarlo si usa il metodo $OneVsOneClassifier()$ della libreria Sklearn[7]. Una volta effettuate le fasi di training e test, si procede alla stampa della matrice di confusione

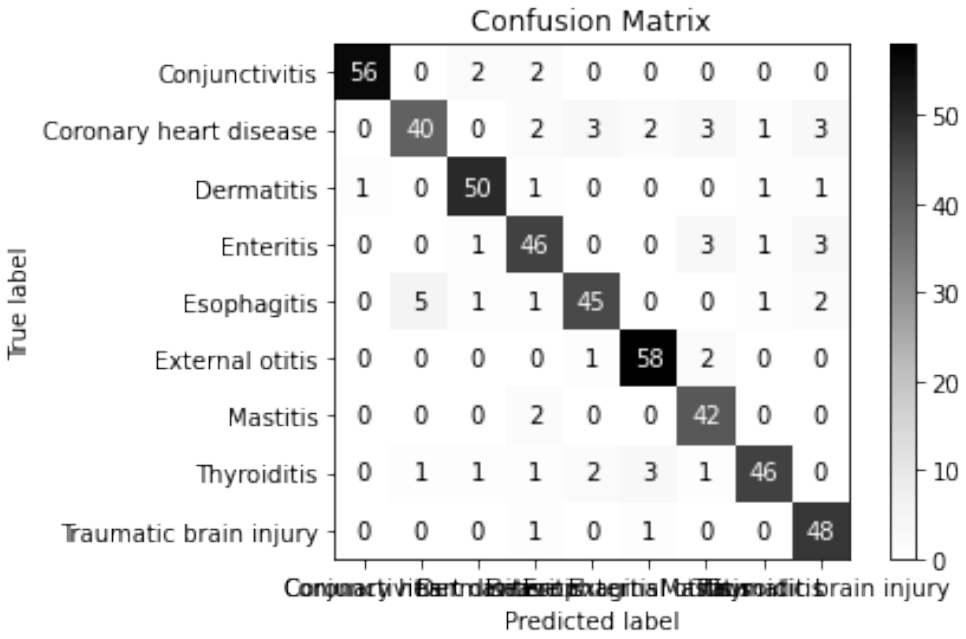


Figura 4.10: Matrice di confusione per il modello OVA

da cui si vede un risultato meno preciso rispetto a quello ottenuto mediante il classificatore one-vs-all, con difficoltà maggiori nel diagnosticare malattie delle coronarie, la tiroide o l'esofagite.
Analizzando tali risultati risulta che in tale modello emerge un'*accuracy* dell'89% nel test contro una del 97% nel training, la quale rappresenta un'accuracy abbastanza efficiente per effettuare una diagnosi medica.

4.2 Risultati e Curva ROC

In questa sezione si riassumono tutti i risultati ottenuti dalla valutazione dell'*accuracy* dei test dei modelli considerati:

- Multi-Layer Perceptron: 89%;
- Random Forest: 91%;
- Naive Bayes: 76%;
- One-vs-All: 91%;

- One-vs-One: 89%.

Dando una rapida occhiata la scelta potrebbe ricadere sia sull'utilizzo della Random Forest che del modello One-vs-All, ma visto che il nostro è un problema di classificazione multipla la scelta finale ricade su quest'ultimo.
Prima di accettare definitivamenta tale modello, però, c'è un'ultima cosa da tenere in considerazione: la curca ROC (Receiver Operating Characteristic). Quest'ultima non è altro che un grafico che mette in relazione la sensibilità e la specificità di un test diagnostico al variare del valore di $cut - off$ (valore soglia).
Nella rappresentaizione della curva ROC si hanno:

- sull'asse delle ordinate (asse y) i valori di sensibilità del test, o tasso dei veri positivi (TPR - True Positive rate);
- sull'asse delle ascisse (asse x) i valori 1 - specificità del test, o tasso dei falsi positivi (FPR - False Positive rate);
- un valore di cut-off;
- una curva con andamento "a scaletta";

e in generale un test risulta più accurato quanto più la sua curva ROC si avvicina all'angolo sinistro del grafico.
Detto ciò, grazie all'utilizzo della libreria *matplotlib.pyplot* [8] e stato possibile rappresentare la curva ROC relativa al modello di classificazione One-vs-All

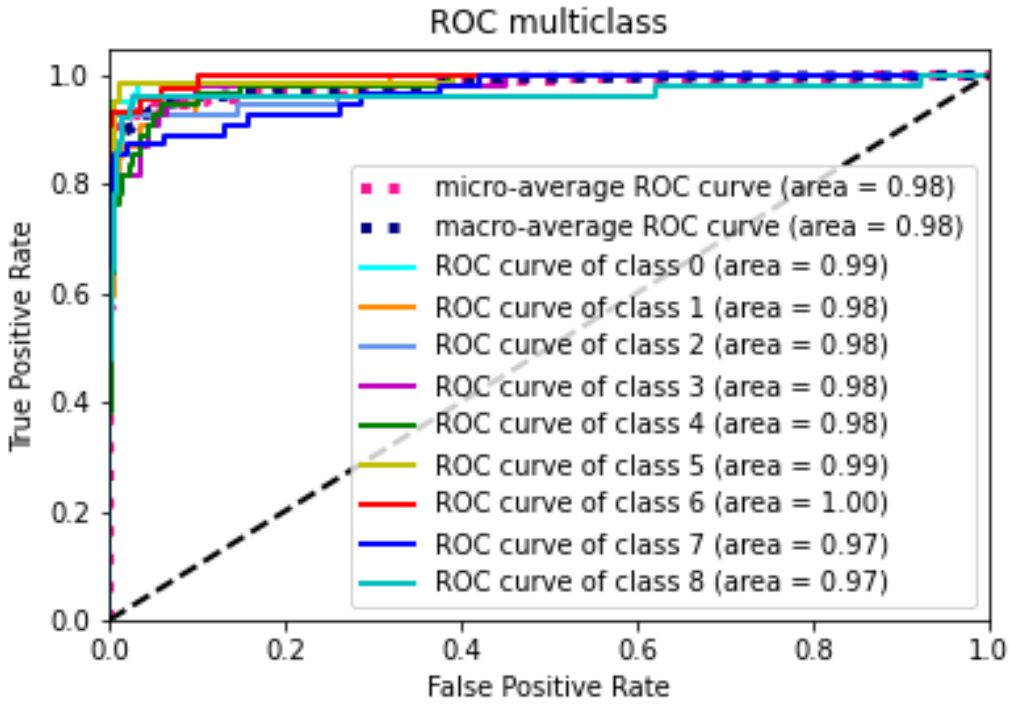


Figura 4.11: Curva ROC per il modello OVA

che rispetta il criterio base di accettazione per la sua curva ROC.

4.3 Salvataggio del modello

A questo punto si è giunti alla fase finale in cui è desiderabile salvare il modello ottenuto dal processo di selezione di un modello.

Per fare ciò si ulitizza la libreria *pickle* e il suo metodo *pickle.dump*(), che consente di salvare il modello addestrato su uno specifico file. In questo modo sarà possibili richiamare il modello nelle fasi successive dello sviluppo del ChatBot.

Capitolo 5

ChatBot

In questo capitolo verrà mostrato come è stato sviluppato il Chatbot Telegram per l'analisi diagnostica di alcune malattie, al quale è stato affidato il compito di facilitare il dialogo con il paziente mediante un sistema basato su "scelte multiple". Così facendo si eviterà che l'utente possa sbagliare nel digitare i suoi sintomi e l'esperienza con il sistema sarà più facile e intuitiva.

5.1 Implementazione

Per implementare il Chatbot si è utilizzato l'editor di codice $VisualStudioCode$[9] e si è continuato con l'utilizzo del linguaggio di programmazione Python [1], il quale presenta un'apposita libreria per lo sviluppo di interfacce di dialogo Python per la piattaforma Telegram definita $python - telegram - bot$[10].
Di seguito è riportata la struttura dei file (*classification.py* e *chatbot.py*) all'interno del progetto del Chatbot

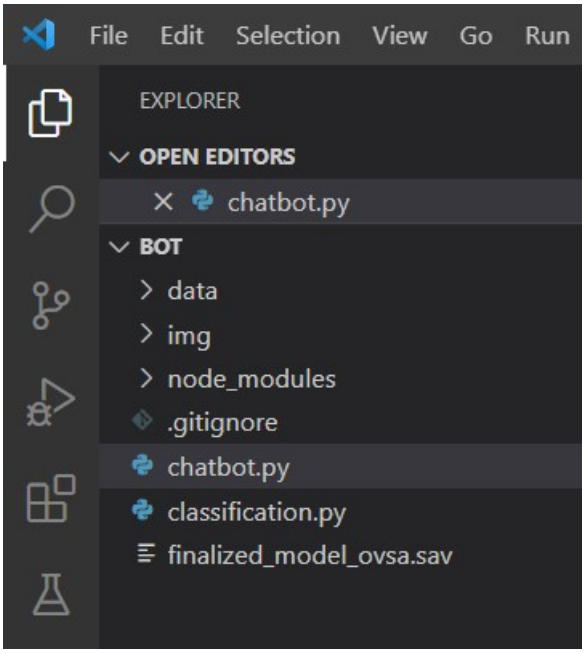


Figura 5.1: Struttura dei file del Chatbot

5.1.1 Struttura

La struttura implementativa del Chatbot prevede pricipalmente tre gestori (*handler*):

- il primo *handler* si occupa di avviare la conversazione catturando e gestendo il comando */start* immesso dall'utente al primo avvio del chatbot;
- il secondo *handler* si occupa di effettuare la conversaione vera e propria con l'utente/paziente;
- il terzo *handler* si occupa di terminare la conversaizione o di farne partire una nuova.

Il focus principale, però, ricade sul secondo *handler* in quanto rappresenta il fulcro del nostro Chatbot di analisi diagnostica. È qui che, mediante un sistema a "risposta multipla", l'utente può man mano indicare al sistema quali sono i suoi sintomi e, infine, poter ricevere una diagnosi.
Per la fase di diagnosi vera e propria è stato implementato un apposito metodo *diagnosis*(). Quest'ultimo prende in input i sintomi selezionati dal paziente e

poi, dopo averli inseriti all'interno del dataset, vi applica il metodo *predict*() del modello One-vs-All, il quale viene appositamente caricato dal file su cui era stato precedentemente salvato. Tale medoto restituisce in output la malattia che con probabilità maggiore può rispondere ai vari sintomi selezionati dal paziente.

5.1.2 Hosting

Una volta strutturato il Chatbot e averne verificato il funzionamento in locale, si è passati alla fase di *hosting*.
Come host si è optato per Heroku [11], una piattaforma cloud di programmazione progettata per aiutare a realizzare e distribuire applicazioni online.
La prima fase riguarda la creazione di un nuovo progetto.

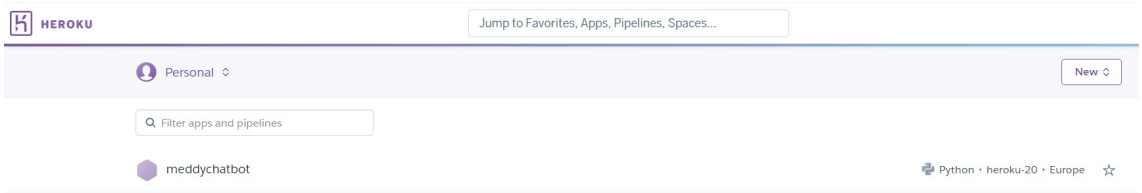


Figura 5.2: Schermata di Heroku

Si passa, quindi, alla fase di *deployment* che è stata effettuata tramite GitHub [12], su cui è stato caricato il codice sorgente del progetto più i file necessari per la sua integrazione con Heroku.

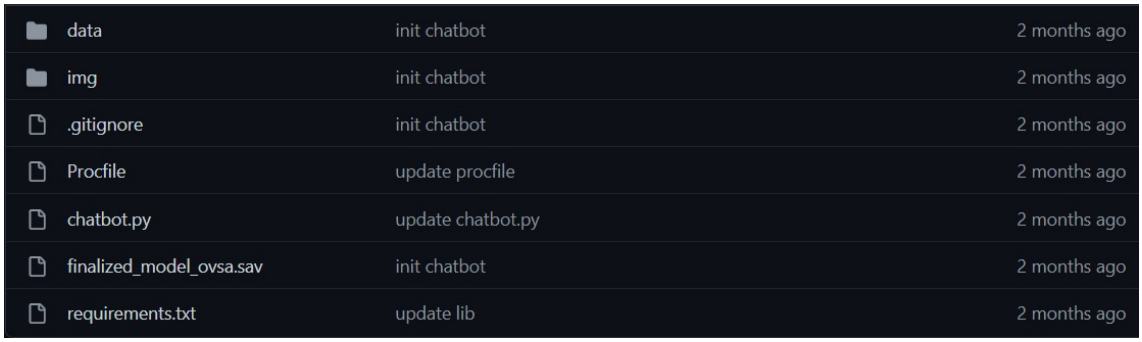


Figura 5.3: Schermata dei file GitHub

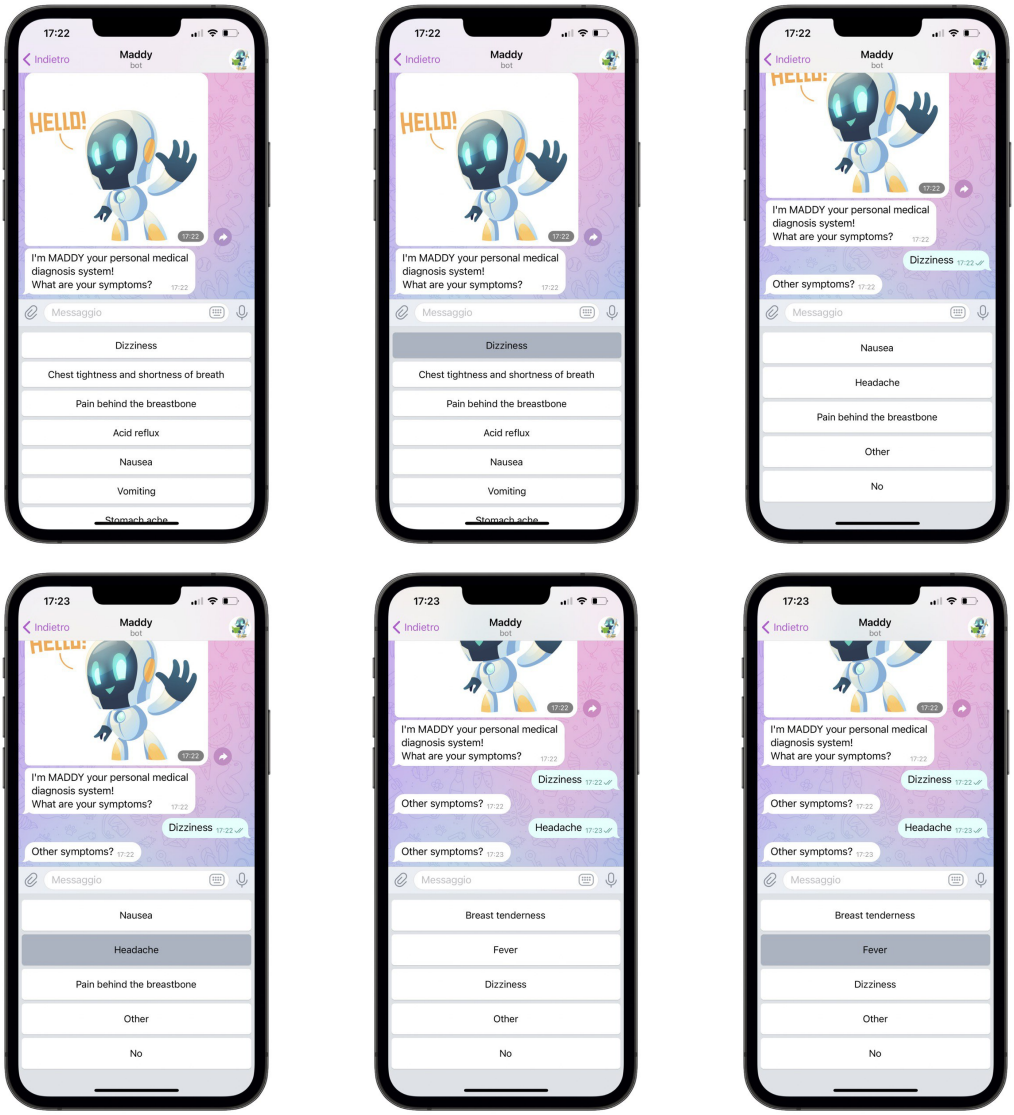
Completate tutte le fasi preliminari, tra cui fornire all'host la KEY del bot Telegram, il Chatbot è stato messo online e può ora fornire supporto a tutti i pazienti che lo richiederanno.

5.1.3 Interfaccia utente

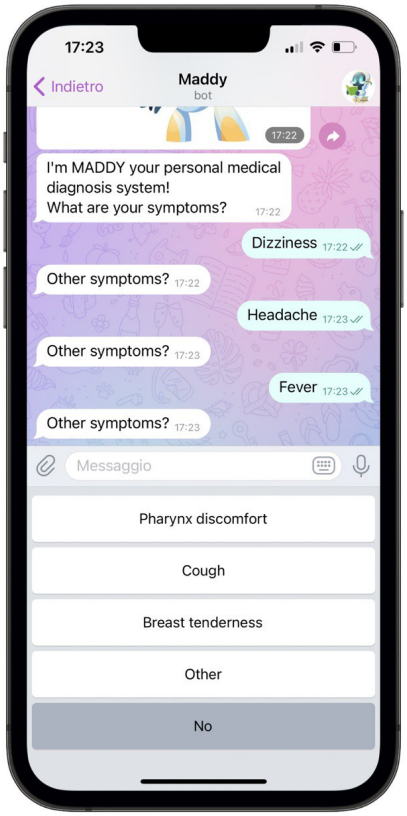
In questa sezione si va a mostrare il funzionamento finale del Chatbot di analisi diagnostica quando vi è un'interazione con il paziente.
Aperta la chat l'utente si troverà davanti una schermata di presentazione, in cui Maddy (così è stato chiamato il nostro assitente) va a spiegare il suo ruolo.

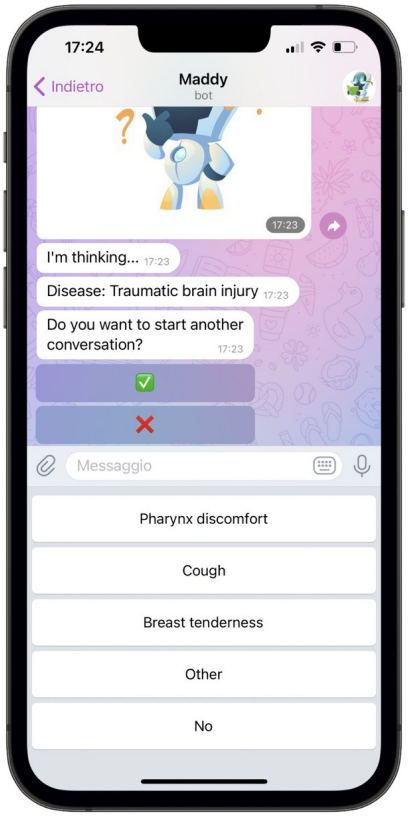
Figura 5.4: Schermata di avvio del Chatbot

Avviato il bot, si procedere con le schermate relative alla conversazione.



Una volta selezionati tutti i sintomi, il paziente può selezionare l'opzione "NO" in modo da poter ricevere una diagnosi dal sistema.





A questo punto l'utente può decidere se lasciare la conversazione o farne partire una nuova.

Capitolo 6

Conclusioni e sviluppi futuri

Il sistema di dialogo, avente lo scopo di effettuare analisi diagnostica in ambito medico e sviluppato durante questo progetto, risulta essere uno strumento utile per fornire un supporto medico agli utenti/pazienti che non riescono a rivolgersi prontamente al prorpio medico. Grazie al sistema di dialogo, infatti, è possibile avere a portata di smartphone un primo parere medico, che aiuterà il paziente a farsi una prima idea sulla propria malattia.
Il Chatbot, però, non è perfetto e possibili sviluppi futuri potrebbero riguardare:

- *aumentare il dataset:* ampliare il dataset aggiungendo altri sintomi è molto utile. In fase di analsi, infatti, è stata questa la fase più difficile da gestire in quanto non erano presenti molte informazioni sui sintomi relativi a specifiche malattie;

- *migliorare il sistema di dialogo:* sviluppare un sistema di dialogo basato su NLP (Natural Language Processing) sarebbe ottimale per poter gestire un vero e proprio dialogo con l'utente.

Non va comunque dimenticato che l'utilizzo di tale Chetbot non sostituisce minimamente il lavoro del medico, ma resta solo uno strumento per avere un parere preliminare prima di rivolgersi al proprio medico di fiducia.

Bibliografia

[1] *Python*. URL: https://www.python.org/.

[2] *scikit-learn Machine Learning in Python*. URL: https://scikit-learn.org/stable/.

[3] *sklearn.neural-network.MLPClassifier*. URL: https://scikit-learn.org/stable/modules/generated/sklearn.neural_network.MLPClassifier.html.

[4] *sklearn.ensemble.RandomForestClassifier*. URL: https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestClassifier.html.

[5] *sklearn.naive-bayes.GaussianNB*. URL: https://scikit-learn.org/stable/modules/generated/sklearn.naive_bayes.GaussianNB.html.

[6] *sklearn.multiclass.OneVsRestClassifier*. URL: https://scikit-learn.org/stable/modules/generated/sklearn.multiclass.OneVsRestClassifier.html.

[7] *sklearn.multiclass.OneVsOneClassifier*. URL: https://scikit-learn.org/stable/modules/generated/sklearn.multiclass.OneVsOneClassifier.html.

[8] *Matplotlib*. URL: https://matplotlib.org/.

[9] *Visual Studio Code*. URL: https://code.visualstudio.com/.

[10] *python-telegram-bot*. URL: https://pypi.org/project/python-telegram-bot/.

[11] *Heroku*. URL: https://id.heroku.com/login.

[12] *GitHub*. URL: https://github.com/.