**Introduzione**

L’OMS, l’organizzazione mondiale della sanità, ha stimato che ogni anno nel mondo si verificano 18,5 milioni di decessi a causa di malattie cardiache. La statistica risale a fine 2021, come confermato anche da un articolo sul sole 24 ore del 29 settembre scorso.

Nell’articolo in questione viene affermato che le malattie cardio-vascolari rappresentano la prima causa di morte anche in Italia, sia negli uomini (31.7 %), che nelle donne (37.7%). La prognosi precoce di malattie cardiovascolari, pertanto, può aiutare a prendere decisioni in merito al cambiamento dello stile di vita nei pazienti ad alto rischio e, a sua volta, a ridurre le complicanze.

Lo scopo di questa tesi è quello di costruire un modello di Machine Learning, in particolare un classificatore, che sia in grado di predire il manifestarsi di malattie cardiache. L’algoritmo verrà codificato utilizzando il linguaggio di programmazione Python, che rappresenta uno dei linguaggi attualmente più utilizzati in ambito Intelligenza Artificiale e, più specificatamente, nel campo del Machine Learning.

La scelta di Python come strumento di codifica è legata al fatto che tale linguaggio mette a disposizione dell’utente una serie di librerie molto potenti, grazie alle quali è possibile realizzare algoritmi di Machine Learning in modo relativamente semplice e con un numero ridotto di righe di codice.

Il termine ‘modello di Machine Learning’ si riferisce ai cosiddetti algoritmi di apprendimento automatico, ossia a quei programmi grazie ai quali i computer imparano da soli ad eseguire una determinata attività, senza essere esplicitamente programmati per l’esecuzione di quel task.

Allo stato attuale l’apprendimento automatico è una disciplina che sta avendo un grande successo, trovando ampia applicazione in diversi settori. Considerando il World Wide Web, gli algoritmi di Machine Learning sono alla base dei motori di ricerca, per consentire agli utenti di esplorare la rete in maniera più efficace, e dei Social Network, all’interno dei quali vengono utilizzati per implementare tantissime funzioni, come il riconoscimento automatico delle persone nelle immagini. Inoltre, l’apprendimento automatico trova largo utilizzo nell’analisi dei dati, generati in grande quantità dagli utenti di Internet attraverso diversi dispositivi digitali, e che sarebbe impossibile analizzare mediante sistemi tradizionali. Questa analisi permette a molte aziende di conoscere in maniera più approfondita i propri clienti, e quindi di andare a soddisfare con maggior puntualità i loro interessi ed esigenze.

Il Machine Learning, inoltre, viene utilizzato in diversi settori scientifici, come la biologia, la medicina e l’ingegneria. Le applicazioni principali riguardano la guida autonoma, il riconoscimento della scrittura a mano, l’elaborazione del linguaggio naturale, la Computer Vision, ed altro ancora. Non dimentichiamo, inoltre, che tale disciplina viene utilizzata anche nei cosiddetti programmi di raccomandazione, allo scopo di personalizzare i servizi di streaming mediante algoritmi che imparano da soli a comprendere quali siano le preferenze di ciascun utente.

La realizzazione di un modello di apprendimento automatico necessita la presenza di un dataset, ossia di una sorta di database contenente informazioni significative riguardanti l’argomento d’interesse, da utilizzare nella fase di addestramento. Tale fase rappresenta, appunto, l’intervallo temporale in cui l’algoritmo impara effettivamente ad eseguire in autonomia una specifica attività.

Più è ampio il dataset di addestramento, maggiore sarà la precisione con cui l’algoritmo imparerà ad eseguire il proprio lavoro.

Ogni training set contiene un insieme di attributi, detti features, che sono nient’altro che le caratteristiche dei dati stessi o, se vogliamo, le caratteristiche del problema d’interesse. Ad esempio, considerando un algoritmo di apprendimento automatico che vuole predire il prezzo di vendita di una casa, un ipotetico training set potrebbe contenere attributi quali la dimensione, il numero di stanze, il numero di piani, il numero di anni trascorsi da quello di costruzione, ecc.

Anche il numero di features utilizzate influisce sulla capacità dell’algoritmo di espletare bene il suo compito: più è alto il numero degli attributi utilizzati, più l’algoritmo imparerà a predire il valore di output con maggior precisone.

Sono proprio i dati di addestramento che determinano la prima grande classificazione all’interno del mondo del Machine Learning, dove vengono distinte essenzialmente due categorie di algoritmi: apprendimento supervisionato e non supervisionato.

Gli algoritmi appartenenti alla prima categoria utilizzano dati di addestramento etichettati, in cui ogni record del dataset rappresenta un singolo esempio che contiene l’indicazione della risposta corretta.

Nell’apprendimento supervisionato, pertanto, è come se ci fosse un insegnante in grado di guidare l’algoritmo verso la giusta direzione.

Gli algoritmi di apprendimento non supervisionato utilizzano, invece, dati di addestramento non etichettati. In questo caso, non essendo presente la risposta corretta, le macchine non hanno una guida nella ricerca della soluzione e, quindi, imparano effettivamente da sole ad eseguire determinate attività. Generalmente, gli algoritmi appartenenti a questa categoria riescono a individuare elementi simili attraverso l’analisi degli attributi presenti nel dataset.

In ogni caso, le tecniche di apprendimento supervisionato sono quelle più utilizzate, grazie alla loro maggiore efficacia dovuta ai dati etichettati.

Nelle applicazioni pratiche, non sempre i dati etichettati sono disponibili in maniera completa e soddisfacente. A volte, infatti, l’applicazione delle etichette potrebbe essere un’attività molto costosa e, quindi, non conveniente.

In questi casi, le tecniche supervisionate e quelle non supervisionate vengono utilizzate contemporaneamente. Questo connubio porta alla realizzazione di sistemi di apprendimento semi-supervisionato.

Un esempio classico è quello relativo ai filtri antispam, in cui sarebbe impossibile etichettare a mano ogni singolo messaggio, vista la grande mole.

Un’altra grande distinzione che viene fatta all’interno del Machine Learning riguarda il tipo di problema che un algoritmo deve imparare a risolvere. In generale, vengono identificate due grandi categorie di attività: regressione e classificazione.

Si parla di problema di regressione quando l’algoritmo deve imparare a predire valori continui, come può essere il prezzo di una casa, la temperatura ambientale o il prezzo di vendita nel mercato azionario.

Nei problemi di classificazione, invece, l’algoritmo deve imparare a categorizzare gli elementi di input, ossia a distinguerli in diverse classi. In questo caso, si dice anche che l’algoritmo deve imparare a predire valori discreti, ognuno dei quali corrisponde ad una specifica classe.

Un esempio di classificatore è un sistema che sia in grado di distinguere un cancro maligno da uno benigno.

**Costruzione di un modello di Machine Learning**

In questa sezione verranno descritte, dal punto di vista teorico, tutte le attività necessarie alla costruzione di un modello di Machine learning, per poi passare alla realizzazione pratica del nostro classificatore nelle sezioni successive. Ci tengo a precisare che l’obiettivo è quello di applicare tecniche supervisionate, vista la loro maggiore efficienza.

**Pre-elaborazione del dataset**

Il primo passo nella costruzione del modello è sicuramente quello di recuperare un dataset contenente informazioni relative alla problematica che si vuole affrontare. Per la realizzazione di modelli reali, penso sia chiaro che la qualità delle informazioni disponibili è di fondamentale importanza.

Generalmente, i dataset utilizzati nel Machine learning sono costituiti da un file, o da un insieme di file, strutturati o semi-strutturati, in cui le informazioni sono organizzate in un formato tabellare o similare. Le colonne di questa ipotetica tabella rappresentano gli attributi o caratteristiche del problema, in gergo denominate feature.

In ambito supervisionato, oltre alle feature troviamo anche la colonna dedicata alla variabile target, rappresentante quella grandezza sulla quale il modello finale effettuerà predizioni.

Ogni record del dataset rappresenta un esempio del problema, di cui possiamo leggere i valori dei singoli attributi e quello della variabile target.

Nella maggior parte dei casi, i dataset che vengono realizzati o che si recuperano dalle varie sorgenti informative, non sono idonei ad essere utilizzati direttamente nella fase di addestramento, per cui si rende necessario applicare alcune modifiche.

Queste attività di adattamento, dette operazioni di pre-processing o preelaborazione, hanno come scopo ultimo quello di rendere più efficiente la fase di apprendimento. Le operazioni di preelaborazioni dipendono, anche, dal modello che viene scelto.

In generale, le principali attività che vengono svolte in questa fase sono:

* vettorizzazione delle feature;
* gestione dei valori mancanti;
* scaling.

La vettorizzazione consiste essenzialmente nel trasformare i valori di una determinata feature da non-numerici a numerici. Questa trasformazione si rende necessaria perché, generalmente, gli algoritmi di apprendimento sono più efficienti quando le feature sono tutte in formato numerico.

La vettorizzazione porta normalmente ad aumentare il numero delle feature nel dataset.

Questo aspetto è più facilmente comprensibile con un esempio.

Immaginiamo di avere una feature di nome ‘sport’ che può assumere i seguenti valori: ‘soccer’, ‘basket’, ‘tennis’.

Applicando la vettorizzazione, questo attributo verrà sostituito da tre feature di tipo binario, ognuna corrispondente ad un valore della caratteristica originaria. Nel caso specifico, verranno create le feature ‘soccer’, ‘basket’ e ‘tennis’ e, all’interno del dataset, i valori di questi attributi saranno adattati in base alla valorizzazione della feature originaria. Ad esempio, se in un determinato record avevamo ‘sport’ = ‘soccer’, nel dataset modificato, in corrispondenza dello stesso esempio, avremo la feature ‘soccer’ valorizzata ad 1 e le altre due a 0.

Questa semplice tecnica, che suddivide una feature contenente n valori in n feature binarie, viene denominata ‘one hot encoding’ e viene applicata quando i valori di un attributo sono numericamente limitati. Qualora i possibili valori di una feature fossero diverse decine, non avrebbe senso applicare questa tecnica in quanto lo spazio delle feature aumenterebbe sensibilmente, ottenendo l’effetto contrario a quello desiderato.

Una tecnica simile a quella appena descritta viene applicata anche agli attributi che contengono del testo come valore. La forma più semplice di questa tecnica, denominata ‘word bag model’, ossia modello del sacchetto di parole, trasforma l’attributo contenente n possibili parole in altrettante feature, similmente a quanto visto nel ‘one hot encoding’. In questo caso, però, il valore memorizzato nelle feature derivate corrisponde al numero di volte in cui compare quella specifica parola. Anche in questo caso, un esempio potrà chiarire meglio il concetto.

Immaginiamo di avere un attributo del nostro dataset che contenga i seguenti testi:

Immagine che contiene testo

Descrizione generata automaticamente

Il processo di vettorizzazione sostituirà l’attributo in questione con quattro feature derivate, corrispondenti rispettivamente alle parole ‘amico’, ‘ciao’, ‘cane’ e ‘gatto’. Facendo riferimento al primo esempio, nel nuovo dataset post vettorizzazione le quattro feature saranno valorizzate nel modo seguente: ‘‘amico’ = 1, ‘ciao’ = 1, ‘cane’ = 0 e ‘gatto’ = 1. Nell’ultimo esempio, invece avremo: ‘‘amico’ = 0, ‘ciao’ = 2, ‘cane’ = 0 e ‘gatto’ = 0, poiché compare solo la parola ‘ciao’ ripetuta due volte.

Come nel caso precedente, questa tecnica diventa inefficiente se il numero di parole da memorizzare è notevolmente elevato.

Può succedere che il dataset in nostro possesso presenti alcuni esempi con delle feature non valorizzate.

Generalmente, se i record in questo stato sono una percentuale irrisoria, si può anche decidere di eliminarli definitivamente dal dataset. Altrimenti, esistono diverse tecniche per generare artificialmente i valori mancanti. Quelle più semplici prevedono l’utilizzo di un valore costante, stabilito a priori, oppure l’inserimento del valore che si ripete più frequentemente. Nel caso di feature numeriche, si può decidere anche di utilizzare il valore medio o la mediana. Le tecniche, comunque, sono molteplici e ne esistono anche diverse molto complicate.

La gestione dei valori mancanti è un aspetto delicato da valutare seriamente, altrimenti si corre il rischio di rendere il dataset poco realistico e quindi inaffidabile.

Lo scaling o normalizzazione è un’altra tecnica molto importante che serve a migliorare il processo di apprendimento, consentendo di realizzare modelli più efficienti.

Il significato dello scaling è fare in modo che le feature varino tutte più o meno nello stesso range. Questo è importante perché alcuni algoritmi di addestramento potrebbero concentrare il loro interesse su quelle feature che mostrano una variabilità più ampia, dando meno importanza alle altre. C’è da dire che non tutti gli algoritmi risentono di questo fenomeno, ma è buona norma applicare lo scaling per evitare comportamenti indesiderati. Inoltre, l’applicazione di questa tecnica rende il processo di apprendimento più veloce.

Esistono diverse tecniche di normalizzazione, quella più semplice applica l’operazione di divisione tra il valore dell’attributo e il suo massimo previsto, garantendo per quella feature l’intervallo [0,1] come dominio.

Un’altra tecnica, detta normalizzazione media, applica la seguente formula:

**Significato di apprendimento**

La selezione di un valido dataset rappresenta sicuramente il primo passo verso la realizzazione di un algoritmo di Machine Learning, ma siamo solo all’inizio del lavoro e ci sono ancora molti aspetti da prendere in considerazione.

In precedenza, abbiamo parlato di apprendimento automatico, sottolineando come il dataset rappresenti lo strumento necessario ad eseguire questo task. Ma cosa intendiamo effettivamente per apprendimento?

La risposta a questa domanda non è affatto banale e per ottenerla dobbiamo comprendere in maniera più approfondita il funzionamento di un algoritmo di Machine Learning, concentrandoci in particolare sul caso ‘supervisionato’, che rappresenta il modello di nostro interesse.

Un algoritmo di questo tipo, per poter funzionare in maniera efficiente e svolgere bene il proprio lavoro, deve conoscere una funzione che sia in grado di approssimare nel miglior modo possibile il valore della variabile target, in corrispondenza a specifici valori di input assegnati alle feature. Dal punto di vista matematico, stiamo parlando di una funzione in variabili, dove con vogliamo indicare il numero di attributi o feature utilizzati dall’algoritmo.

Nel gergo dell’apprendimento automatico, tale strumento viene denominato ‘funzione ipotesi’ ed è generalmente indicata con la lettera : . Inoltre, molto spesso, in ambito Machine Learning si preferisce utilizzare espressioni vettoriali, quindi, l’insieme delle feature viene rappresentato con un vettore ad dimensioni contrassegnato dalla lettera . A tal proposito, l’espressione della funzione ipotesi assume la seguente forma: .

Una volta introdotta la funzione ipotesi, siamo in grado di comprendere il significato di apprendimento.

Con questo termine, infatti, vogliamo indicare quella fase in cui, grazie all’applicazione di una serie di tecniche, l’algoritmo ‘studia’ le informazioni presenti nel dataset di apprendimento in modo tale da costruire una funzione ipotesi efficiente. Questo significa che tale funzione dovrà essere in grado di stimare il valore della variabile target minimizzando l’errore possibile.

All’atto pratico la fase di apprendimento si realizza tramite un algoritmo iterativo che, partendo da un’espressione iniziale della funzione ipotesi, esegue continue modifiche alla stessa durante la fase di studio del dataset, fino al raggiungimento di una sua versione finale. Quest’ultima rappresenterà il modello utilizzato per predire i valori della variabile di output, a fronte di uno specifico input.

Gli algoritmi disponibili in grado di eseguire questo compito sono molteplici, ma non è scopo della presente tesi affrontare l’argomento. Quello che possiamo affermare è che il funzionamento di ognuno di essi si basa sul concetto di funzione di costo.

Si tratta di uno strumento, definito a priori, che guida il processo di apprendimento. Generalmente, nei casi più semplici, la funzione di costo selezionata calcola l’errore che viene commesso nella stima della variabile target. Di conseguenza, il compito dell’algoritmo di apprendimento diventa quello di trovare il suo valore minimo, poiché l’obiettivo è creare un modello in grado svolgere il proprio compito con il minor tasso d’errore possibile. Questo spiega il motivo per cui la fase di apprendimento si compone di iterazioni successive, fino a minimizzare la funzione di costo. L’ipotesi calcolata in corrispondente di quel valore minimo rappresenterà il modello definitivo utilizzato dall’algoritmo di Machine Learning per effettuare le stime.

Il meccanismo sopra descritto ci fa capire in maniera più chiara il significato di apprendimento automatico. Si tratta di una disciplina che ribalta completamente il paradigma dell’informatica classica, secondo la quale è compito del programmatore scrivere la funzione necessaria ad elaborare determinati dati di input, allo scopo di ottenere l’output desiderato. Nel Machine Learning, invece, la funzione da utilizzare per il calcolo della variabile di output viene dedotta in maniera automatica dalla macchina. Se vogliamo, in questo caso il programmatore stabilisce soltanto il tipo di funzione ipotesi da utilizzare, definendone la forma generale, mentre il modello effettivo viene calcolato grazie al processo di apprendimento.

Quella del progettista, comunque, è una figura centrale nella realizzazione del modello, pur non partecipando attivamente alla costruzione della sua versione finale, poiché ha il compito di stabilirne la tipologia. Si tratta sicuramente di una scelta non banale e generalmente, per la risoluzione di uno stesso problema, vengono presi in considerazione più modelli, ai quali si applicano specifiche metriche per valutarne le prestazioni. Quest’ultimo passo aiuta il progettista nella scelta del modello più idoneo da utilizzare.

**Modelli di Machine learning**

Nel paragrafo precedente abbiamo affermato che è compito del progettista stabilire il tipo di modello da utilizzare, mentre quello effettivo sarà dedotta automaticamente tramite il processo di apprendimento.

Da un punto di vista matematico, scegliere il tipo di modello significa sostanzialmente stabilire a priori la forma generale che deve avere la funzione ipotesi .

Sicuramente quelli lineari rappresenta i modelli più semplici da costruire. Questa caratteristica li rende adatti in determinate situazioni, soprattutto quando le feature in gioco sono numericamente molto ridotte. Se ci limitiamo ai problemi di regressione, un modello si dice lineare quando la funzione ipotesi è rappresentabile, appunto, come combinazione lineare delle feature di input attraverso specifici coefficienti, denominati pesi del modello. Ad esempio, immaginando di avere a disposizione un dataset contenente solamente due feature, l’espressione generale della funzione ipotesi sarebbe la seguente: , dove i coefficienti rappresentano i pesi del modello.

Il processo di apprendimento avrà il compito di dedurre i valori effettivi dei coefficienti da applicare alla forma generale, in modo da costruire il modello desiderato.

Per capire il significato geometrico della funzione ipotesi di un modello lineare, supponiamo di avere un solo attributo, in modo da ricondurre lo studio ad un piano cartesiano. In questo caso, l’espressione della funzione ipotesi identifica una specifica retta i cui punti rappresentano tutte le possibili soluzioni del nostro problema.

Qualora le feature fossero due, siamo ancora in grado di immaginare il significato geometrico di questa funzione, che in questo caso identificherà un piano nello spazio tridimensionale.

Con un numero superiore di feature, si parlerà di iperpiano dello spazio n-dimensionale.

Una funzione ipotesi della forma vista in precedenza è idonea a risolvere un problema di regressione, poiché il valore di output è di tipo continuo, ma risulta essere non adeguata nei problemi di classificazione, in cui il valore atteso è di tipo discreto.

Se immaginiamo di voler costruire un classificatore binario, quindi in grado di distinguere elementi appartenenti a due categorie, l’ideale sarebbe che la funzione ipotesi generasse in uscita soltanto due valori: 0 o 1.

L’obiettivo viene raggiunto grazie all’utilizzo della funzione sigmoidale, la cui espressione generale è la seguente:

Come possiamo notare, infatti, qualunque sia il valore della variabile indipendente, la risposta della funzione sarà sempre un valore compreso tra 0 e 1.

Questo primo passo non risolve definitivamente il problema, poiché il valore in output è comunque di tipo continuo. L’ostacolo viene definitivamente superato interpretando l’output della funzione sigmoidale nel modo seguente:

* se , il modello emetterà in uscita il valore 1;
* se, invece, , il classificatore emetterà in uscita il valore 0.

Con questa interpretazione il valore emesso in output dalla funzione sigmoidale assume un significato ben preciso: nello specifico rappresenta la probabilità che l’elemento in input appartenga alla classe etichettata con il valore 1.

Infatti, qualora il valore calcolato dalla funzione sigmoidale fosse 0.80, significherebbe avere l’80% di probabilità di appartenere alla classe etichettata con il valore 1, e questo sarebbe coerente con il valore discreto emesso in output dal classificatore.

Analogo discorso può essere fatto qualora il valore stimato dalla funzione sigmoidale fosse, ad esempio, 0.35. Una situazione di questo tipo significherebbe avere una probabilità del 35% di appartenere alla classe etichettata con il valore 1, quindi una probabilità del 65% di appartenere all’altra classe, confermando la coerenza con quanto emesso in output dal classificatore.

Dopo aver compreso l’importanza della funzione sigmoidale, facciamo un paso successivo: cerchiamo di capire come costruire un modello di classificazione di tipo lineare.

La soluzione è all’interno della variabile z definita nella funzione sigmoidale. Se poniamo , raggiungiamo immediatamente il nostro scopo.

Un modello di classificazione con una funzione ipotesi che abbia le caratteristiche di linearità sopra menzionate, viene denominato modello di regressione logistica.

Analogamente a quanto visto nel modello di regressione lineare, l’equazione assume un significato geometrico importante anche nel modello di regressione logistica.

A tal proposito, consideriamo le seguenti osservazioni:

* ; 🡪 il modello emette in output il valore 1
* ; 🡪 il modello emette in output il valore 0

Per le caratteristiche appena descritte, l’equazione nel modello di regressione logistica prende il nome di ‘decision boundery’ o limite decisionale.

Ricapitolando, nel caso della regressione lineare la funzione rappresenta l’ipotesi del modello, che contiene tutte le possibili soluzioni al problema, mentre, nel caso della regressione logistica, assume il significato di limite decisionale.

I modelli lineari sono molto apprezzati per la loro semplicità e facilità di interpretazione, ma sicuramente non sono quelli più efficienti. In particolare, le loro performance tendono a degradare all’aumentare del numero di feature da elaborare.

Inoltre, nelle situazioni reali, raramente l’output varia linearmente rispetto all’input.

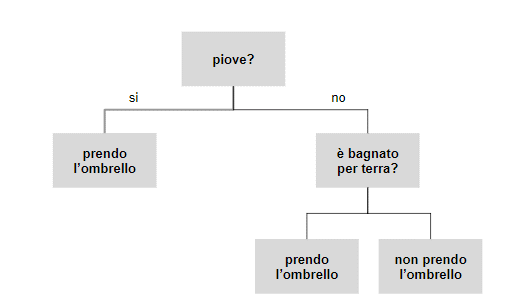
Pertanto, quando si devono affrontare problemi che richiedono l’elaborazione di un numero elevato di attributi, per raggiungere livelli di performance accettabili è necessario utilizzare dei modelli di complessità maggiore.

Rimanendo in ambito supervisionato, tra gli algoritmi più utilizzati ricadono sicuramente il ‘Decison Tree’, detto anche ‘Albero Decisionale’, e il ‘Random Forrest’. Entrambi sono utilizzati sia per problemi di regressione che di classificazione.

Il modello ‘Decison Tree’, come suggerisce il nome, presenta una struttura ad albero in cui è possibile distinguere tre tipi di nodi:

* nodo radice, che rappresenta il punto di partenza del processo decisionale e contiene tutti gli altri nodi dell’albero;
* nodi genitori, ognuno dei quali possiede almeno due nodi figli;
* nodi foglia, che contengono le variabili di output;

Ogni nodo verifica una condizione su una determinata variabile di input e ha due o più diramazioni verso il basso. Il processo consiste in una sequenza di test. Si comincia sempre dal nodo radice, il nodo genitore situato più in alto nella struttura, per poi procedere verso il basso. A seconda dei valori rilevati in ciascun nodo, il flusso prende una direzione oppure un'altra e procede progressivamente verso il basso. Man mano che il processo di selezione prosegue, lo spazio delle ipotesi si riduce perché gran parte dei rami decisionali dell'albero vengono eliminati. La decisione finale si trova nei nodi foglia terminali. L’immagine seguente mostra un esempio di albero decisionale:



Gli alberi decisionali hanno l'indiscusso vantaggio della semplicità. Inoltre, sono facili da capire e da eseguire. Un’altra importante caratteristica è l’espressività, ossia sono facilmente sviluppabili sotto forma di codice di programmazione, poiché possono essere rappresentati con qualsiasi linguaggio proposizionale.

Il modello ‘Random Forest’ non è altro che un insieme di Alberi Decisionali messi insieme allo scopo di raggiungere migliori performance. Infatti, valutando più ‘Decision Tree’ contemporaneamente, è possibile fare una media consentendoci di realizzare un modello più robusto e con migliori capacità di generalizzazione.

L’obiettivo di questo progetto è quello di realizzare un classificatore che sia in grado di predire la presenza di malattie cardiache. Allo scopo di mettere a confronto diversi modelli, nella realizzazione pratica del classificatore utilizzeremo tutti e tre gli algoritmi che abbiamo descritto in questo paragrafo, ossia il modello di regressione logistica, l’albero decisionale e il Random Forest.

**Validazione del modello**

Un altro aspetto da tenere in debita considerazione quando si realizza un modello di Machine Learning è quello della validazione.

Si tratta di un procedimento che si prefigge l’obiettivo di valutare le prestazioni del modello al termine dell’addestramento, per capirne il livello di efficienza raggiunta.

A tal proposito, si utilizza una tecnica che prevede la suddivisione del dataset in due porzioni distinte: la prima, quella più corposa, sarà utilizzata per l’effettivo addestramento del modello, mentre la seconda andrà a comporre il dataset di test e sarà utilizzata nella fase di validazione.

Generalmente il dataset di test viene costruito in maniera randomica, a partire dal dataset originale, considerando una porzione di quest’ultimo compresa tra il 10% e il 30%.

Il processo di validazione consiste nell’applicare delle metriche grazie alle quali vengono valutate le prestazioni del modello addestrato.

Le metriche che possono essere utilizzate sono diverse, dipendono essenzialmente dal tipo di modello realizzato.

Generalmente le prestazioni di un classificatore vengono misurate utilizzando l’accuratezza (accuracy).

Questa grandezza viene calcolata nel modo seguente:

Nel caso di un modello di regressione, invece, generalmente le prestazioni vengono valutate attraverso l’errore quadratico medio (MSE = Mean Squared Error):

dove:

* è il valore predetto dal modello;
* è il valore atteso;

Nel nostro caso, dovendo realizzare un classificatore, faremo riferimento alla prima metrica descritta, ossia all’accuratezza.

**Svolgimento del progetto**

Dopo aver descritto a livello teorico tutti gli aspetti salienti che riguardano la costruzione di un modello di Machine Learning, in questa sezione ci occuperemo della realizzazione pratica del progetto.

Come accennato nel paragrafo introduttivo, l’implementazione del modello verrà effettuata utilizzando il linguaggio Python e, in particolare, la libreria scikit-learn, che rappresenta lo strumento principale per realizzare modelli di Machine Learning. Verranno utilizzate anche altre librerie di supporto per implementare alcune funzioni specifiche.

Grazie alla potenza degli strumenti messi a disposizione, allo stato attuale Python è considerato uno dei linguaggi principali nel campo del Machine Learning.

Per ogni fase del progetto specificheremo il codice in Python utilizzato per la sua implementazione.

Come ambiente di sviluppo verrà utilizzato l’IDE Pycharm 2021.3.1 Community Edition e la versione 3.10 di Python.

**Setting dell’ambiente di sviluppo**

Dall’interfaccia principale dell’IDE, creiamo il nuovo progetto attraverso il pulsante ‘New Project’:

Immagine che contiene testo, screenshot, monitor

Descrizione generata automaticamente

La successiva GUI ci permette di assegnare il nome al progetto, che chiamiamo ‘HeartDiseaseClassifier’, definire la directory target relativa all’ambiente virtuale e selezionare l’interprete di Python da utilizzare, qualora ce ne fossero più versioni installate.

L’utilizzo dell’ambiente virtuale è molto comodo perché consente di installare tutti gli strumenti necessari all’interno del progetto stesso, senza intaccare i file di sistema.

Le informazioni sopra descritte sono visibili nell’immagine seguente:

Immagine che contiene testo

Descrizione generata automaticamente

Premendo il tasto ‘Create’ viene creato il nuovo progetto con le impostazioni definite:

Immagine che contiene testo, screenshot, monitor, portatile

Descrizione generata automaticamente

Il passo successivo è il setting dell’ambiente virtuale con l’installazione delle librerie necessarie alla sua realizzazione, scikit-learn in primis.

L’installazione di una qualunque libreria può essere effettuata attraverso la sezione ‘Terminal’ dell’IDE, visibile nell’immagine precedente, mediante l’utilizzo del comando ‘pip’:

Immagine che contiene testo, screenshot, monitor, interni

Descrizione generata automaticamente

Quando si installa scikit-learn, il sistema installa automaticamente anche lo strumento ‘numpy’, un’altra libreria di supporto molto utilizzata.

C’è da sottolineare il fatto che non è obbligatorio installare fin da subito tutte le librerie necessarie al progetto. Questa operazione può essere eseguita anche in un secondo momento, quando si palesa l’esigenza di uno specifico strumento non utilizzato fino a quel momento.

La sezione ‘Python Packages’ contiene la lista di tutti i pacchetti installati fino a quel momento nell’ambiente virtuale:

Immagine che contiene testo, screenshot, monitor, interni

Descrizione generata automaticamente

Successivamente alla sua installazione, la specifica libreria può essere utilizzata nel programma attraverso l’operazione di import.

**Il dataset**

Abbiamo detto nel paragrafo introduttivo che per costruire un qualunque algoritmo di apprendimento automatico è necessario munirsi di un dataset di addestramento. Pertanto, considerando le finalità della presente tesi, il primo passo compiuto è stato proprio quello di ricercare fonti di dati utili, riguardanti l’argomento d’interesse, ossia le malattie cardiache, che avessero un livello elevato di attendibilità e qualità delle informazioni. La ricerca, come facilmente immaginabile, si è sviluppata setacciando letteralmente la ‘grande rete’, cercando di sfruttare nel miglior modo possibile le potenzialità messe a diposizione da Internet.

Il sito <https://www.kaggle.com/> ha soddisfatto le mie aspettative, all’interno del quale ho trovato molto interessante il dataset denominata ‘Heart Failure Prediction Dataset’, disponibile al seguente link e scaricabile in formato csv: <https://www.kaggle.com/fedesoriano/heart-failure-prediction>.

Trattandosi di un dataset etichettato, ho pensato immediatamente che fosse l’ideale per realizzare modelli di Machine Learning di tipo supervisionato. Tale caratteristica è stata particolarmente apprezzata perché strettamente compatibile con l’obiettivo prefissato.

Prima di immergermi nella costruzione del modello, il mio interesse si è rivolto all’analisi del dataset selezionato, allo scopo di approfondire le informazioni in esso contenute. Il dataset in questione è costituito da 918 record, ognuno dei quali si riferisce ad uno specifico paziente esaminato. Le informazioni raccolte provengono da diversi luoghi, nello specifico:

* Cleveland Clinic Foundation, situata nella città di Cleveland (Stati Uniti);
* Istituto di cardiologia di Budapest (Ungheria);
* V.A. Medical Center, Long Beach California (Stati Uniti):
* Università di Zurigo (Svizzera).

Il database originale è composto da 76 attributi, ma soltanto un sottoinsieme ridotto di quest’ultimi è stato effettivamente utilizzato. Il dataset finale, infatti, è composto da solo 12 colonne: le prime 11 sono le feature considerate più significative, mentre l’ultima rappresenta l’elemento target, ossia la diagnosi della malattia cardiaca. Tale campo, denominato ‘HeartDisease’, contiene soltanto due valori possibili, 0 e 1, interpretabili nel modo seguente:

* 0: probabilità che la malattia cardiaca si manifesti inferiore al 50%;
* 1: probabilità che la malattia cardiaca si manifesti superiore al 50%.

La tipologia del campo target, che assume valori di tipo discreto, ci fa capire che il dataset può essere utilizzato per costruire un classificatore, risultando idoneo per l’obiettivo prefissato. Nello specifico, il modello potrà distinguere solamente due classi, una etichettata con il valore 0, l’altra con il valore 1. Generalmente gli algoritmi che risolvono problemi di questa valenza vengono denominati ‘classificatori binari’, proprio perché sono in grado di riconoscere soltanto due categorie: la classe positiva e quella negativa.

Normalmente la prima corrisponde a quella etichettata con il valore 1, ma non esiste una regola standard.

Viene riportata di seguito una breve descrizione delle undici feature presenti nel dataset (dedotta dal file testuale ‘heart-disease’):

1. Age: età del paziente espressa in anni;
2. Sex: indica il sesso del paziente (M = male, ossia maschio, F = famale, ossia femmina);
3. ChestPainType: esprime il tipo di dolore al petto riscontrato nel paziente. Per questo campo sono stati previsti i seguenti valori:
   1. ASY: acronimo di ‘asymptomatic’ (asintomatico, ossia nessun dolore al petto);
   2. ATA: acronimo di atypical angina’ (dolore atipico);
   3. NAP: acronimo di ‘non-anginal pain’ (dolore non anginoso);
   4. TA: acronimo di ‘typical angina’ (dolore tipico);
4. RestingBP: acronimo di ‘resting blood pressure’ (pressione sanguigna a riposso) espressa in mm Hg (millimetri di mercurio);
5. Cholesterol: livello di colesterolo espresso in mg/dl, ossia milligrammi di glucosio per decilitri di sangue;
6. FastingBS: acronimo di ‘fasting blood sugar’, ossia livello di glicemia a digiuno. Si tratta di un attributo di tipo booleano, quindi con solo due valori possibili. Il valore 1 è associato a quei pazienti il cui livello glicemico è superiore ai 120 mg/dl, altrimenti il campo assume valore 0;
7. RestingECG: acronimo di ‘resting electrocardiographic results’, esprime i risultati dell’esame elettrocardiografico a riposo. A questo campo sono associati i seguenti valori:
   1. Normal;
   2. ST: il paziente presenta inversioni dell’onda T e/o elevazione o depressione del segmento ST;
   3. LVH: acronimo di ‘left ventricular hypertroph’ (ipertrofia ventricolare sinistra).
8. MaxHR: acronimo di ‘maximum heart rate achieved’, indica la massima frequenza cardiaca raggiunta, misurata in battiti al minuto;
9. ExerciseAngina: acronimo di ‘exercise induced angina’, ossia dolore al petto indotto da esercizio fisico. Per tale campo sono previsti solo due valori: N (No) indica l’assenza di dolore, il valore Y (Yes), invece, ne indica la presenza;
10. Oldpeak: indica la depressione del segmento ST indotta dall'esercizio fisico rispetto al riposo, espressa in mm;
11. ST\_Slope: indica la pendenza del segmento ST di picco indotta dall’esercizio fisico. Per questo campo sono previsti i seguenti valori:
    1. upsloping (salita);
    2. flat (piatto);
    3. downsloping (discesa).

**Acquisizione del dataset**

La prima operazione da svolgere è l’acquisizione del dataset.

A tal proposito, la libreria ‘pandas’ mette a disposizione la funzione ‘read\_csv()’ che permette di memorizzare in una variabile il contenuto di un file csv. Come specificato precedentemente, tale libreria va installata prima di essere utilizzata nel programma.

La funzione di cui sopra, che restituisce un oggetto della classe ‘DataFrame’, accetta come parametro di input una stringa che specifica il percorso del file da acquisire. Inserendo il documento direttamente all’interno della directory del progetto, sarà sufficiente specificare il nome del file.

L’immagine seguente mostra il codice che implementa quanto appena descritto a cui è stata aggiunta, a scopo di test, un’istruzione ‘print()’ per visualizzare il contenuto della variabile ‘dataset’, in modo da confrontarlo con le informazioni presenti nel file originario.

Come è facilmente visibile, l’output del programma, oltre a mostrare alcuni record del dataset, mette in evidenza anche importanti informazioni, quali il numero di righe e di colonne che compongono l’oggetto di tipo ‘DataFrame’ incapsulato nella variabile ‘dataset’ (918 X 12):

Immagine che contiene testo, monitor, computer, screenshot

Descrizione generata automaticamente

**Preelaborazione del dataset**

Una volta acquisito il dataset, il passo successivo è implementare la fase di pre-processing.

Nella feature ‘Sex’ troviamo due valori, ‘M’ e ‘F’, che indicano rispettivamente il sesso maschile e quello femminile. Allo scopo di renderli di tipo numerico, andiamo ad associare il valore 0 a ‘F’ e il valore 1 a ‘M’. A tal proposito, utilizziamo il metodo ‘replace()’ della classe ‘DataFrame’ che definisce due parametri principali:

* ‘to\_replace’ per specificare il valore da sostituire;
* ‘value’ per indicare il nuovo valore.

Il tipo restituito dal metodo in questione è sempre un oggetto ‘DataFrame’.

Immagine che contiene testo, screenshot, monitor, portatile

Descrizione generata automaticamente

Le istruzioni 11 e 12, mostrate nell’immagine precedente, effettuano le modifiche descritte. Anche in questo caso sono state inserite delle istruzioni ‘print()’ a scopo di test, prima e dopo la modifica, per visualizzare il cambiamento. Notiamo come l’istruzione ‘dataset[‘Sex’]’ ci permetta di accedere ai valori della specifica feature. Inoltre, l’output del programma mette in evidenza come la modifica provochi anche il cambio del tipo di dato memorizzato. Originariamente, infatti, i valori contenuti nell’attributo ‘Sex’ erano interpretati da Python come di tipo ‘object’. In seguito alla modifica, invece, i valori di questa feature diventano di tipo intero.

Applichiamo una modifica analoga alla feature ‘ExerciseAngina’, dove i valori memorizzati sono ‘Y’ (Yes) e ‘N’ (No):

Immagine che contiene testo, screenshot, monitor, portatile

Descrizione generata automaticamente

Analizzando gli altri attributi del dataset, notiamo come su alcuni ci sia la necessità di applicare la vettorizzazione. Ci riferiamo in particolare alle feature con valori multipli ‘ChestPainType’, ‘RestingECG’ e ‘ST\_Slope’ sui quali si potrebbe applicare la funzione ‘One Hot Encoding’ descritta precedentemente.

L’applicazione di tale funzione è molto semplice in Python. La libreria scikit-learn, infatti, mette a disposizione la classe ‘OneHotEncoder’, appartenente al package ‘preprocessing’, che esegue proprio questo compito.

Più in generale, le trasformazioni sulle feature in Python vengono gestite attraverso oggetti ‘ColumnTransformer’, grazie ai quali è possibile applicare contemporaneamente trasformazioni diverse ai vari attributi del dataset.

Per istanziare un oggetto di questa classe, definita nel package ‘compose’ di scikit-learn, si deve invocare il suo costruttore specificando come parametro di input la lista dei trasformatori che si desidera applicare. Un singolo trasformatore si definisce attraverso una lista di tre elementi:

* una variabile stringa che funge da label;
* un oggetto che implementa la specifica trasformazione da applicare;
* una lista che indica le colonne del dataset su cui applicare la trasformazione.

Di seguito il codice che esegue quanto sopra descritto:

Immagine che contiene testo

Descrizione generata automaticamente

I passi salienti sono:

* le righe di codice 11-13 definiscono la lista dei trasformatori da applicare, che nel nostro caso è composta da un solo elemento. Tale elemento implementa la funzione ‘One Hot Encoding’ attraverso un oggetto della classe ‘OneHotEncoder’, a sua volta istanziato tramite l’invocazione del relativo costruttore. Come possiamo notare, alla riga 12 vengono specificate anche le colonne su cui applicare questa trasformazione che corrispondono ai tre attributi del dataset di cui abbiamo parlato in precedenza;
* l’istruzione alla riga 16 definisce un oggetto della classe ‘ColumnTransformer’ invocandone il suo costruttore. I parametri passati sono la lista dei trasformatori e l’attributo ‘remainder’ valorizzato a ‘passthrough’. Quest’ultimo permette di mantenere nel dataset anche le altre colonne le quali, altrimenti, sarebbero state eliminate;
* le istruzioni 19 e 21 eseguono materialmente le modifiche, in particolare il metodo ‘fit()’ addestra l’oggetto ‘ColumnTransformer’, mentre il metodo ‘transform()’ applica le trasformazioni alle colonne;

Con queste modifiche il dataset passa da 11 a 18 feature (19 colonne totali se consideriamo anche la variabile target), poiché l’attributo ‘ChestPainType’ viene splittato in 4 feature binarie, mentre gli altri due attributi coinvolti vengono suddivisi in tre feature ciascuno, per un totale di 7 feature aggiuntive.

Per terminare la fase di preelaborazione manca l’operazione di scaling.

Anche per questo aspetto Python mette a disposizione dello sviluppatore diversi strumenti che semplificano il lavoro. In particolare, le classi più utilizzate sono ‘StandarScaler’ e ‘MinMaxScaler’, appartenenti al package ‘preprocessing’ di scikit-learn.

La prima trasforma le feature numeriche del dataset in modo tale da ottenere un valor medio nullo e varianza = 1.

La seconda, invece, considera i valori minimi e massimi e poi li comprime nell’intervallo [0, 1].

Utilizziamo quest’ultima soluzione nel nostro programma:

Immagine che contiene testo

Descrizione generata automaticamente

Alla riga 30 viene istanziato un oggetto della classe ‘MinMaxScaler’ tramite il suo costruttore.

L’operazione di scaling si concretizza con le istruzioni alle righe 32 e 34. Il metodo ‘fit()’ addestra lo scaler in modo che possa calcolare i valori minimi e massimi di tutti gli attributi, mentre il metodo ‘transform()’ modifica i valori effettuando la normalizzazione. L’istruzione alla riga 37 è stata inserita solo a scopo di verifica.

Terminata questa fase, il dataset è pronto per essere utilizzato nell’addestramento del modello.

**Costruzione del modello**

In questo paragrafo ci occuperemo della realizzazione del modello.

In realtà, l’idea di base è quella di costruire più modelli di classificazione diversi, addestrarli utilizzando lo stesso dataset per poi mettere a confronto le loro performance.

Come capiremo a breve, la costruzione e l’addestramento di un modello di Machine Learning utilizzando la libreria scikit-learn è una pratica relativamente semplice.

Per ogni tipologia di modello, infatti, questa libreria definisce una specifica classe. Inoltre, tutte le classi che implementano modelli di Machine Learning mettono a disposizione due metodi fondamentali che agevolano notevolmente il compito dello sviluppatore: si tratta del metodo ‘fit()’, utilizzato per addestrare il modello, e il metodo ‘predict()’, che, invece, ci permette di effettuare le predizioni.

Il compito dello sviluppatore, pertanto, risulta essere molto semplificato poiché basterà implementare i seguenti passi:

1. costruire il modello di base desiderato istanziato un oggetto della classe corrispondente mediante l’invocazione del suo costruttore;
2. eseguire l’addestramento del modello attraverso il metodo ‘fit()’;
3. effettuare predizioni invocando il metodo ‘predict()’.

Come possiamo notare, questa struttura logica è simile a quella vista in precedenza per i ‘Transformer’ e per gli ‘Scaler’ utilizzati nella fase di preelaborazione del dataset.

L’invocazione del metodo ‘fit()’ richiede due parametri di input, tuttavia molto intuitivi se si ricorda il significato di apprendimento automatico:

* la lista degli esempi contenenti i valori delle feature;
* i corrispondenti valori della variabile target.

Questo comporta la necessità di separare le feature dalla variabile target nel dataset di addestramento.

Il metodo ‘predict()’, invece, richiede un solo parametro di input, ossia la lista degli esempi di cui si vuole stimare la variabile target. Tale stima sarà proprio il valore restituito dal metodo.

Abbiamo detto in precedenza che, in ottica validazione, l’addestramento del modello viene generalmente eseguito utilizzando solo una porzione del dataset di input, lasciando il resto per la fase di test. Questo ci permette successivamente di valutare le prestazioni del modello addestrato.

Quindi, prima di eseguire la fase di apprendimento, dovremo dividere il dataset in due porzioni, che chiameremo rispettivamente dataset di training e dataset di test.

Python mette a disposizione una funzione molto utile per effettuare questa operazione. Si tratta della funzione ‘train\_test\_split()’, definita nel package ‘model\_selection’ di scikit-learn, che esegue la separazione selezionando gli esempi in modo casuale.

Il codice seguente mostra le operazioni appena descritte:

Immagine che contiene testo, computer, screenshot, portatile

Descrizione generata automaticamente

Le righe di codice 39 e 40 estrapolano dal dataset rispettivamente le feature e i valori della variabile target, mentre l’istruzione alla riga 43 costruisce due dataset distinti a partire dall’originale, quello di training e quello di test.

Come possiamo notare, la funzione ‘train\_test\_split()’ restituisce quattro tuple il cui significato è intuitivo:

* X\_train e y\_train compongono il dataset di addestramento;
* X\_test e y\_test compongono, invece, il dataset di test.

Inoltre, nella chiamata a questa funzione è stato specificato anche il parametro opzionale ‘test\_size’, che definisce la porzione del dataset originale da utilizzare per il test. L’output del programma conferma la suddivisione stabilita.

Dopo aver fissato questi concetti principali, passiamo concretamente alla realizzazione dei vari modelli di classificazione. L’obiettivo è quello di implementare un modello di regressione logistica, un albero decisionale e un Random Forest.

Il modo più semplice per realizzare questi algoritmi con scikit-learn è quello di utilizzare le classi ‘LogisticRegressor’, ‘DecisionTreeClassifier’ e ‘RandomForestClassifier’, appartenenti rispettivamente ai package ‘linear\_model’, ‘tree’ e ‘enseble’. Invocando i rispettivi costruttori, possiamo creare questi tre modelli in modo molto semplice, per poi addestrarli attraverso il metodo ‘fit()’. Ecco il codice che realizza quanto appena descritto:

Immagine che contiene testo, screenshot, portatile, computer

Descrizione generata automaticamente

Nel codice sopra riportato, i tre modelli sono stati creati utilizzando la versione base dell’algoritmo corrispondente. Questo è deducibile dal fatto che i costruttori delle tre classi sono stati invocati senza specificare alcun parametro di input, e questo implica l’utilizzo dei valori di default.

Qualora si volessero costruire modelli più specifici, sarà necessario approfondire la documentazione della libreria scikit-learn per studiare il significato dei singoli parametri, valutare eventuali variazioni rispetto al default e implementare la versione desiderata del modello.

**Valutazione dei modelli**

Una volta costruiti e addestrati i modelli, la fase successiva è quella di valutarne le prestazioni.

L’obiettivo è quello di capire il livello di efficienza raggiunto.

Come già detto in precedenza, in genere la validazione di un classificatore viene effettuata attraverso la metrica dell’accuratezza (accuracy in inglese).

Questa grandezza viene calcolata in due step:

* si utilizza il modello da valutare per effettuare delle predizioni sui dati di test;
* si mettono a confronto i valori dell’array ‘y\_p’, contenente le predizioni, con quelli contenuti nell’array ‘y\_test’, che invece rappresentano le risposte desiderate.

Per ogni classe che rappresenta un modello di Machine Learning, Python definisce il metodo ‘predict()’ per effettuare delle predizioni su specifici dati di input. Nel nostro caso utilizzeremo quelli contenuti nella variabile ‘X\_test’.

Per il calcolo dell’accuratezza, Python mette a disposizione una funzione dedicata a questo scopo appartenente al package ‘metrics’ della libreria scikit-learn che si chiama, appunto, ‘accuracy\_score’. Come facilmente deducibile, tale funzione prende in input due variabili: la prima è l’array contenente le risposte desiderate, mentre la seconda è l’array dei valori predetti. Il risultato ottenuto rappresenta l’accuratezza del modello in esame.

Immagine che contiene testo, screenshot, monitor, portatile

Descrizione generata automaticamente

L’immagine precedente mostra i passi appena descritti. Le variabili ‘acc\_lrm’, ‘acc\_dtm’ e ‘acc\_rfm’ rappresentano l’accuratezza dei tre modelli. Per come è definita tale grandezza, più il suo valore si avvicina all’unità, maggiore è la precisione del modello realizzato.

Le istruzioni ‘print()’ alle righe 75-77 sono state aggiunte allo scopo di visualizzare a video il valore delle tre grandezze sopra menzionate.

Se eseguiamo il programma più volte, notiamo che ad ogni istanza otteniamo valori di accuratezza diversi. Questo comportamento dipende essenzialmente da due fattori:

* il dataset di training e quello di test vengono determinati ogni volta in modo casuale;
* le istanze delle classi utilizzate vengono inizializzate di nuovo ad ogni esecuzione.

Allo scopo di scegliere quello più efficiente, un’idea di base potrebbe essere quella di eseguire il programma un numero predefinito di volte per poi memorizzare il modello che ha riportato il valore di accuratezza più elevato. Quest’ultimo, in una fase successiva, potrebbe essere pubblicato inserendolo, ad esempio, all’interno di un sito web in modo da poter essere utilizzato da qualunque utente.

Uno dei modi più semplici per serializzare un modello di Machine Learning e salvarlo in un file è quello di utilizzare la libreria ‘pickle’, che mette a disposizione la funzione ‘dump()’ a tale scopo. La funzione ‘load()’, invece, esegue il compito inverso: permette di caricare in memoria il modello precedentemente salvato su disco rigido per poi utilizzarlo nelle predizioni.

Di seguito alcune immagini che mostrano le modifiche apportate al programma allo scopo di calcolare il modello più efficiente e salvarlo su disco rigido.

Immagine che contiene testo, screenshot, computer

Descrizione generata automaticamente

Alla riga 11 troviamo l’istruzione per importare la libreria ‘pickle’.

Immagine che contiene testo

Descrizione generata automaticamente

Alle righe 46-49 sono state introdotte due nuove variabili:

* accurancy conterrà l’accuratezza corrispondente al modello migliore;
* modelType, invece, è una variabile stringa che conterrà il tipo di modello scelto.

L’immagine mostra come il corpo principale del programma sia stato inserito all’interno di un ciclo che si ripete per n volte (nel caso specifico n = 1000).

Immagine che contiene testo, screenshot, elettronico, computer

Descrizione generata automaticamente

L’ultima immagine visualizza la parte terminale del ciclo (righe 83-98), in cui vengono confrontate le accuratezze dei singoli modelli per scegliere quello più efficiente.

Le righe di codice 106 e 107 riportano le istruzioni per definire il nome di un file su disco rigido e per salvare, all’interno di quest’ultimo, il modello di Machine Learning scelto.

Di seguito viene mostrato il risultato ottenuto al termine dell’esecuzione del programma:

Immagine che contiene testo, screenshot, elettronico, computer

Descrizione generata automaticamente

Il modello migliore ottenuto risulta essere quello di regressione logistica, con un’accuratezza dell’88.7% circa. Tale modello viene salvato all’interno del file ‘finalized\_model.sav’ e, in questo modo, diventa riutilizzabile in futuro per effettuare predizioni su dati relativi pazienti a nuovi.

Il risultato ottenuto, che ha privilegiato il modello di regressione logistica rispetto a quelli non lineari, dipende da diversi fattori:

* il dataset a disposizione ha una dimensione ridotta;
* anche il numero di features utilizzate non è molto elevato;
* sono stati utilizzati algoritmi con parametrizzazioni di default.

Per poter realizzare modelli più complessi e precisi, sarebbe necessario approfondire la documentazione della libreria sci-kit learn e studiare come ottimizzare gli algoritmi modificandone i parametri di configurazione.

Possiamo simulare l’utilizzo del modello ottenuto considerando il dataset a disposizione nella sua totalità, immaginando che le informazioni contenute appartengano a nuovi pazienti. L’immagine seguente mostra questa simulazione:

Immagine che contiene testo

Descrizione generata automaticamente

Come possiamo notare, le prestazioni del modello sull’intero dataset è più bassa rispetto a quella calcolata sul dataset di test.

Questo risultato è comprensibile, poiché l’addestramento ha utilizzato solo 296 esempi, mentre il dataset ne contiene complessivamente 916.

Un’accuratezza dell’83.4% rappresenta, comunque, un livello prestazionale molto realistico per un modello di regressione logistica.