**Sistema per la diagnostica di malattie cardiovascolari: DOCUMENTO TECNICO**

Introduzione:

Questo sistema per la diagnostica di malattie cardiovascolari è stato realizzato come progetto di Ingegneria della Conoscenza (ICon) e si divide in due parti principali:

- l’utilizzo di algoritmi di apprendimento supervisionato per predire la presenza di malattia/e cardiovascolare/i su uno stesso dataset;

- l’utilizzo di un sistema esperto per la diagnostica di sintomi su malattie cardiovascolari e la verifica di condizioni di salute ottimali, oltre che alla lettura di un’ontologia per la conoscenza dei vari sintomi proposti.

Le malattie cardiovascolari sono un gruppo di patologie cui fanno parte le malattie ischemiche del cuore e le malattie cerebrovascolari. Esse sono in gran parte prevenibili, in quanto riconoscono, accanto a fattori di rischio non modificabili (età, sesso e familiarità), anche fattori modificabili, legati a comportamenti e stili di vita (fumo, alcool, scorretta alimentazione, sedentarietà) spesso a loro volta causa di diabete, obesità, ipercolesterolemia, ipertensione arteriosa.

Algoritmi di Apprendimento Supervisionato:

L’apprendimento avviene nel file ***models\_main.py***, dove:

- si costruiscono le rappresentazioni dei modelli (descritti in seguito nel documento);

- si addestrano i modelli su un dataset comune suddiviso in 35000 esempi di training e 35000 esempi di test (contenuto nel file cardio\_train.csv);

- si utilizzano i modelli addestrati per la predizione;

- si realizzano le matrici di confusione e si calcolano le metriche precision (precisione), recall (richiamo), f1-score e accuracy per ciascun modello.

La libreria principale utilizzata per l’apprendimento supervisionato è scikit-learn, libreria open source di apprendimento automatico che contiene algoritmi di classificazione, regressione e clustering, progettato per operare con le librerie NumPy e SciPy.

**Dataset:**

Il dataset utilizzato viene gestito nel file ***data.py*** ed è contenuto nel file ***cardio\_train.csv***, reperibile al seguente link:

<https://www.kaggle.com/datasets/sulianova/cardiovascular-disease-dataset>

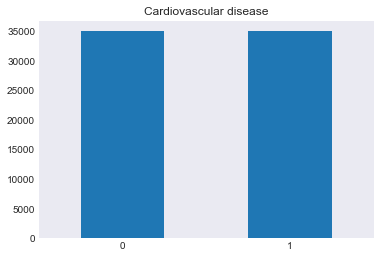
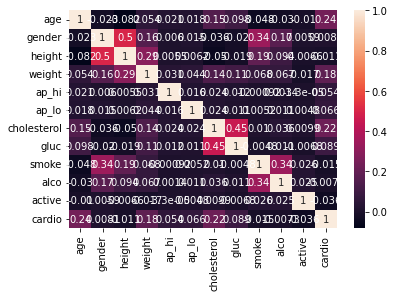
Nel dataset ci sono 3 tipi di feature di input:

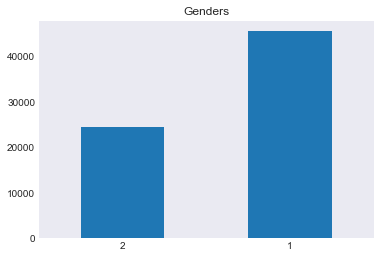
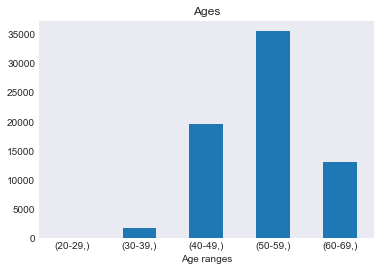
* *Oggettivo*: informazioni fattuali;
* *Esame*: risultati della visita medica;
* *Soggettivo*: informazioni fornite dal paziente.

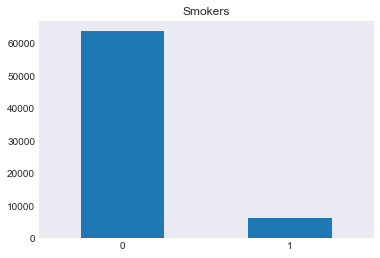
Feature presenti:

1. Età | Oggettivo | age | intero (giorni)
2. Altezza | Oggettivo | height | intero (cm)
3. Peso | Oggettivo | weight | reale (kg)
4. Genere | Oggettivo | gender | 1: uomo, 2: donna
5. Pressione sistolica | Esame | ap\_hi | intero
6. Pressione diastolica | Esame | ap\_lo | intero
7. Colesterolo | Esame | cholesterol | 1: normale, 2: sopra del normale, 3: ben al di sopra del normale
8. Glucosio | Esame | gluc | 1: normale, 2: sopra del normale, 3: ben al di sopra del normale
9. Fumatore | Soggettivo | smoke | 0: falso, 1: vero
10. Consumatore di alcool | Soggettivo | alco | 0: falso, 1: vero
11. Praticante di attività fisica | Soggettivo | active | 0: falso, 1: vero
12. Presenza di malattia cardiovascolare | Target | cardio | 0: falso, 1: vero

I seguenti grafici vengono realizzati per avere una maggiore visione del dataset:







Da questi grafici si capisce che il dataset è abbastanza equilibrato: circa una metà di persone ha una malattia cardiovascolare, mentre la parte restante no; inoltre l’età della maggior parte delle persone di questo dataset è compresa tra 50 e 59 anni.

**Modelli:**

La rappresentazione dei classificatori, così come il calcolo delle metriche, la creazione delle matrici di confusione e la comparazione dei modelli vengono gestiti nel file ***models.py*** attraverso la classe **CvDModel** e le sottoclassi **CvDDecisionTree** (per l’albero di decisione), **CvDLogisticRegression** (per la regressione logistica), **CvDSupportVectorMachine** (per il SVM), **CvDRandomForest** (per la random forest), **CvDAdaBoost** (per l’adaboost), **CvDKNearestNeighbors** (per il k-nearest neighbors), e le funzioni **ROC\_space** e **precision\_recall\_space**.

Di seguito la descrizione dei vari modelli:

**Albero di decisione (o di classificazione):**

Un albero di decisione è un albero di classificatori (Decision Stump) dove ogni nodo interno è associato ad una particolare “domanda” su una feature (caratteristica o attributo). Da questo nodo dipartono tanti archi quanti sono i possibili valori che la feature può assumere, fino a raggiungere le foglie che indicano la categoria associata alla decisione.

**Regressione logistica:**

In statistica, il modello logistico (binario) è un modello statistico che modella la probabilità che un evento (su due alternative) si verifichi facendo in modo che le log-odds (il logaritmo delle probabilità) per l'evento siano una combinazione lineare di una o più variabili indipendenti ("predittori"). Nell'analisi di regressione, la regressione logistica stima i parametri di un modello logistico (i coefficienti nella combinazione lineare). Formalmente, nella regressione logistica binaria esiste una singola variabile binaria dipendente, codificata da una variabile indicatore, dove i due valori sono etichettati "0" e "1", mentre le variabili indipendenti possono essere ciascuna una variabile binaria (due classi, codificate da una variabile indicatore) o una variabile continua (qualsiasi valore reale). La probabilità corrispondente del valore etichettato "1" può variare tra 0 (certamente il valore "0") e 1 (certamente il valore "1"), da cui l'etichettatura; la funzione che converte le log-odd in probabilità è la funzione logistica, da cui il nome, della forma:

sigmoid(x) = 1 / (1 + e^-x)f ( x ) = L 1 + e – k ( x – x 0 ) , {\displaystyle f(x)={\frac {L}{1+e^{-k(x-x\_{0})}}},}

**Support Vector Machine (o SVM):**

SVM permette di ottenere un classificatore lineare basato su una funzione discriminante nella forma:

|  |  |
| --- | --- |
| \begin{displaymath} f(\mathbf{x}) = \mathbf{w} \cdot \mathbf{x} + b \end{displaymath} |  |

con w weight vector e b bias.

L'iperpiano ottimo in $\mathbb{R}^{n}$viene generato in maniera tale da separare “fisicamente” (decision boundary) gli elementi del problema di classificazione binario, ovvero si pone come obiettivo quello di massimizzare il margine di separazione tra le classi. Questo ragionamento premia molto per quanto riguarda la generalizzazione del classificatore.

**Random Forest:**

Le random forest sono un metodo di ensemble learning per classificazione e regressione che opera costruendo una moltitudine di alberi di decisione durante l'addestramento. Per le attività di classificazione, l'output della random forest è la classe selezionata dalla maggior parte degli alberi. Per le attività di regressione, viene restituita la predizione media o media dei singoli alberi. Le random forest generalmente superano gli alberi di decisione, tuttavia, le caratteristiche dei dati possono influire sulle loro prestazioni.

**ADAptive BOOSTing (o AdaBoost):**

L'idea base di AdaBoost è quella di costruire una lista di classificatori assegnando, in maniera iterativa, un peso ad ogni nuovo classificatore considerando la sua capacità di riconoscere campioni non correttamente identificati dagli altri classificatori già coinvolti nell'addestramento. Tutti questi classificatori coinvolti voteranno con il peso loro assegnato e la scelta finale avverà per maggioranza.

Le tecniche di Boosting permettono di generare un classificatore nella forma di modello additivo:

|  |  |
| --- | --- |
| \begin{displaymath} F_T(\mathbf{x}) = f_1(\mathbf{x}) + f_2(\mathbf{x}) + \ldots + f_T(\mathbf{x}) = \sum_{t=1}^{T} f_t(\mathbf{x}) \end{displaymath} |  |

con f1, f2, ..., fT singoli classificatori.

**K-nearest neighbors:**

L'algoritmo k-nearest neighbors (k-NN) è un metodo di apprendimento supervisionato non parametrico utilizzato per classificazione e regressione. In entrambi i casi, l'input è costituito dai k esempi di addestramento più vicini in un set di dati.

Nella classificazione l'output è un'appartenenza a una classe. Un oggetto è classificato in base a un voto di pluralità dei suoi vicini, con l'oggetto assegnato alla classe più comune tra i suoi k vicini più prossimi (k intero positivo).

Nella regressione l'output è il valore della proprietà per l'oggetto. Questo valore è la media dei valori di k vicini più vicini.

k-NN è un tipo di classificazione in cui la funzione viene approssimata solo localmente e tutto il calcolo viene differito fino alla valutazione della funzione. Poiché questo algoritmo si basa sulla distanza per la classificazione, se le caratteristiche rappresentano unità fisiche diverse o sono disponibili in scale molto diverse, la normalizzazione dei dati di allenamento può migliorarne notevolmente l'accuratezza.

**Misure per i modelli predittivi:**

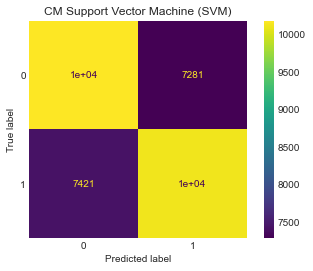
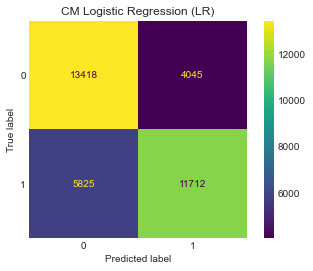
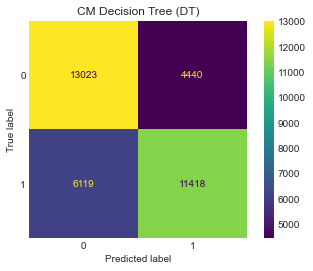
Dopo l’addestramento dei modelli e il calcolo delle predizioni, avviene la creazione e la stampa delle varie Confusion Matrix (o matrici di confusione) e il calcolo delle varie metriche per poter valutare e comparare i modelli.

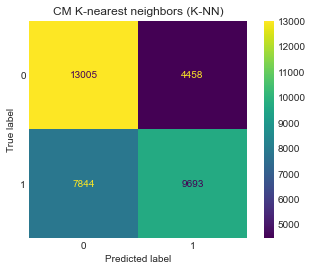
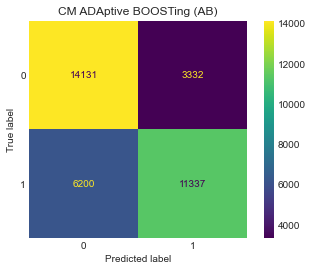
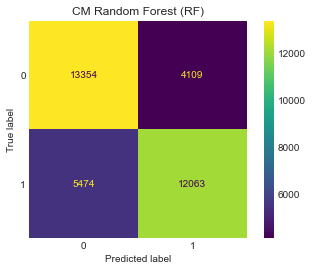
**Confusion Matrix:**

Nell'ambito dell’intelligenza artificiale, la **matrice di confusione**, detta anche **tabella di errata classificazione**, restituisce una rappresentazione dell'accuratezza di classificazione statistica.

Ogni colonna della matrice rappresenta i valori predetti, mentre ogni riga rappresenta i valori reali. L'elemento sulla riga i e sulla colonna j è il numero di casi in cui il classificatore ha classificato la classe "vera" i come classe j. Attraverso questa matrice è osservabile se vi è "confusione" nella classificazione di diverse classi.

Di seguito sono riportate le varie matrici di confusione per i modelli utilizzati:





Dalle matrici di confusione si può capire che i modelli migliori sono il random forest e l’adaboost, mentre il modello peggiore è il SVM.

**Metriche:**

Dopo la generazione delle matrici di confusione avviene il calcolo delle metriche accuracy, precision, recall e f1.

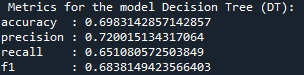
L’accuracy rappresenta la proporzione di predizioni corrette sul numero totale di casi esaminati: (tp + tn) / (tp + fp + tn + fn).

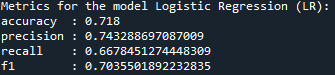
La precision (precisione) rappresenta la proporzione di esempi veramente positivi (tp) sugli esempi predetti come positivi (pp): tp / (tp + fp).

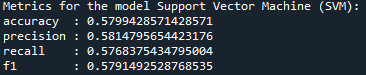
Il recall (richiamo o true-positive rate, TPR) rappresenta la proporzione di veri positivi (tp) su tutti i positivi effettivi (ap): tp / (tp + fn).

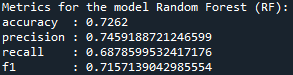
La f1 rappresenta la media armonica tra precision e recall: 2 / ((1 / R) + (1 / P)).

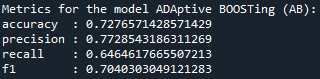
Le metriche calcolate per ogni modello sono mostrate di seguito:

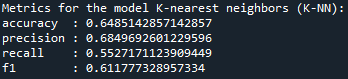










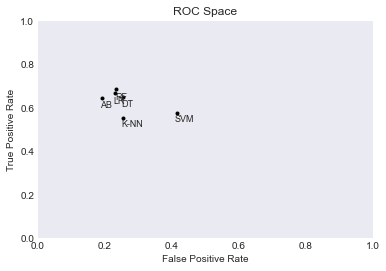


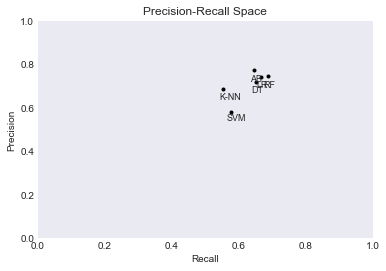
**Confronto Modelli:**

Il confronto dei modelli in maniera indipendente dai costi avviene mediante la costruzione dello spazio ROC (receiver operating characteristic) e dello spazio precision-recall.

Per lo spazio ROC viene realizzato un grafico in cui lungo l’asse delle x vi saranno i valori di false-positive rate (FPR, cioè fp / (fp + tn)), mentre lungo l’asse delle y vi saranno i valori di true-positive rate (TPR) dei modelli; si ha che un modello più in alto e più a sinistra è migliore di uno più in basso e più a destra.

Per lo spazio precision-recall viene realizzato un grafico in cui lungo l’asse delle x vi saranno i valori di recall (TPR, o richiamo), mentre lungo l’asse delle y vi saranno i valori di precision (precisione) dei modelli.





Si possono fare delle osservazioni analoghe a quelle fatte per le matrici di confusione.

Sistema esperto:

Un sistema esperto è un esempio di sistema basato sulla conoscenza. In generale, un sistema esperto include i seguenti componenti: una base di conoscenza (**knowledge base**), un motore di inferenza (**inference engine**), una struttura esplicativa (**explanation facility**), una struttura di acquisizione della conoscenza (**knowledge acquisition facility**) e un'interfaccia utente (**user interface**).

La base di conoscenza rappresenta fatti sul mondo.

Il motore di inferenza è un sistema di ragionamento automatizzato che valuta lo stato attuale della base di conoscenza, applica le regole pertinenti e quindi afferma nuova conoscenza nella base di conoscenza. Il motore di inferenza può includere anche capacità di spiegazione, in modo da poter spiegare a un utente la catena di ragionamento utilizzata per arrivare a una conclusione particolare ripercorrendo l'attivazione delle regole che hanno portato all'asserzione.

Esistono principalmente due modalità per un motore di inferenza: *forward chaining* e *backward chaining*. I diversi approcci sono dettati dal fatto che il motore di inferenza sia guidato dall'antecedente (lato sinistro) o dal conseguente (lato destro) della regola. Nel *forward chaining* un antecedente afferma il conseguente.

Il sistema esperto di questo progetto viene realizzato nel file ***expert\_system.py***, utilizzando la libreria Experta, e utilizzato nel file ***expert\_main.py***. Di seguito la descrizione di ciò che è stato utilizzato da Experta:

**Experta:**

**Facts (Fatti):**

In Experta i fatti sono l'unità di informazione di base e sono utilizzati dal sistema per ragionare sul problema.

Per la dichiarazione di fatti in Experta è necessario istanziare oggetti della classe Fact, sottoclasse di dict. Inoltre differentemente da dict, è possibile istanziare oggetti della classe Fact senza chiavi (cioè solo con valori); in questa maniera verranno assegnati indici numerici a tali valori.

**>>> fact = Fact(a = 1, b = 2) >>> fact = Fact(‘x’, ‘y’, ‘z’)**

**>>> fact[‘a’] >>> fact[0]**

**1 ‘x’**

**Rules (Regole):**

In Experta una regola è un callable, decorato con Rule. Le regole hanno due componenti, LHS (lato sinistro) e RHS (lato destro); LHS descrive (utilizzando schemi) le condizioni in base alle quali la regola dovrebbe essere eseguita, mentre RHS rappresenta l'insieme di azioni da eseguire quando la regola viene attivata. Inoltre le regole devono essere metodi di una sottoclasse di KnowledgeEngine.

**@Rule(AND(OR(Fact(fever = ‘yes’), Fact(syncope = ‘yes’)), Fact(fatigue = ‘yes’)))**

**def medical\_problems(self):**

**‘’’actions’’’**

**…**

con AND e OR operatori logici per esprimere condizioni LHS complesse.

**DefFacts:**

I sistemi esperti necessitano la maggior parte delle volte che una serie di fatti sia presente affinché il sistema funzioni: ciò è lo scopo del decoratore DefFacts. I vari DefFacts all’interno di un KnowledgeEngine verranno chiamati ogni volta che viene chiamato il metodo reset.

**@DefFacts()**

**def first\_action(self):**

**yield Fact(start = 'yes')**

**KnowledgeEngine:**

Un KnowledgeEngine fa parte di un decision-support system che combina i dati con modelli di dati e regole di inferenza per fornire un'interfaccia per le persone che vogliono prendere decisioni o scoprire dati correlati.

In Experta per l’esecuzione di un KnowledgeEngine è necessario:

- creare una sottoclasse di esso, con la definizione delle regole (con @Rule) per decorare i suoi metodi;

- istanziare tale sottoclasse;

- invocare il metodo reset, dichiarando il fatto speciale InitialFact() e dichiarando tutti i fatti prodotti dai metodi decorati con @DefFacts;

- invocare il metodo run, avviando così il ciclo di esecuzione.

**from experta import \***

**class Greetings(KnowledgeEngine):**

**@DefFacts()**

**def first\_action(self):**

**yield Fact(start = ‘yes’)**

**@Rule(Fact(start = ‘yes’))**

**def ask\_name(self):**

**self.declare(Fact(name = input(‘What's your name? ‘)))**

**engine = Greetings()**

**engine.reset()**

**engine.run()**

Per aggiungere fatti alla factlist si può utilizzare il metodo declare:

**>>> engine = KnowledgeEngine()**

**>>> engine.reset()**

**>>> engine.declare(Fact(score = 5))**

**<f-1>**

**>>> engine.facts**

**<f-0> InitialFact()**

**<f-1> Fact(score=5)**

con facts lista di fatti conosciuti dall’engine.

Con Experta, il flusso del programma non ha bisogno di essere definito in modo esplicito, come succede invece in uno stile di programmazione convenzionale. La conoscenza (Regole) e i dati (Fatti) sono separati e il KnowledgeEngine viene utilizzato per applicare la conoscenza ai dati.

**Avvio del sistema esperto:**

Il file ***expert\_main.py*** gestisce il sistema esperto e l’ontologia. Eseguendo il file appena citato verrà posta una domanda all’utente chiedendo di inserire un valore (1 - 2).

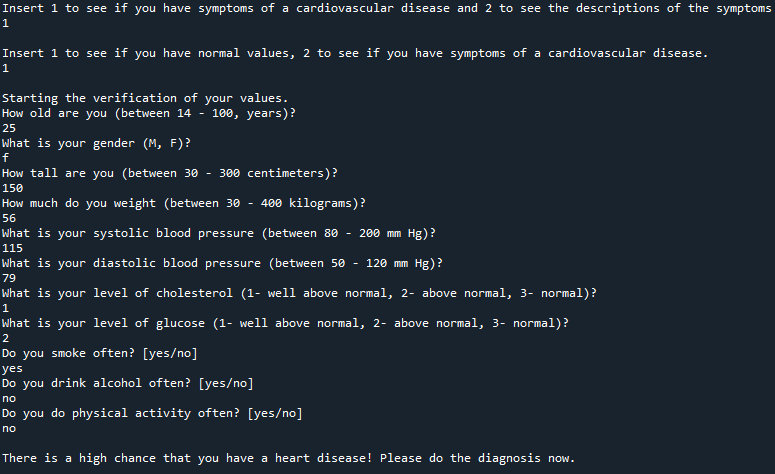
Inserendo il valore 1 il sistema esperto verrà avviato.

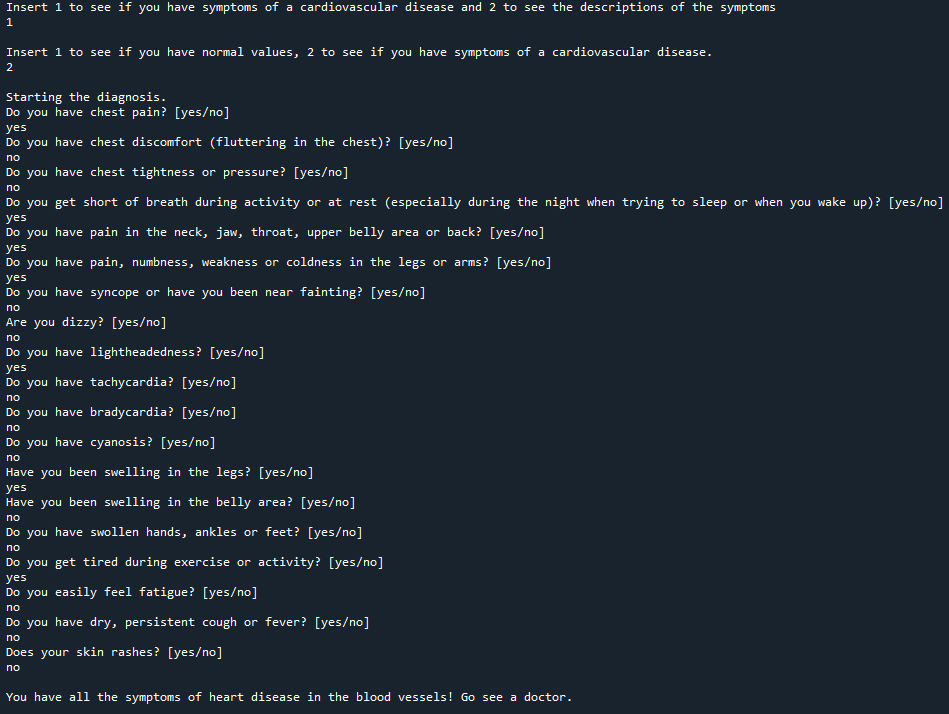
Verrà quindi posta una domanda all’utente per cui si hanno due scelte:

- inserendo il valore 1 verrà dichiarato Fact(start\_verification = 'yes'). Il sistema quindi inizierà a porre una sequenza di domande all’utente (completamente in inglese) relative allo stato di salute dell’utente, come ad esempio inserire la pressione sistolica o inserire il livello di colesterolo, sul quale il sistema ragionerà in seguito. Se si avranno dei valori anomali di qualsiasi tipo (livello di glucosio alto, BMI troppo alto, ecc...) verrà dichiarato Fact(high\_chance\_disease = 'yes') e verrà comunicato all’utente che è a rischio. Altrimenti verrà dichiarato Fact(low\_chance\_disease = 'yes') e verrà comunicato all’utente che non è a rischio;

- inserendo il valore 2 verrà dichiarato Fact(start\_diagnosis = 'yes'). Il sistema inizierà a porre una sequenza di domande all’utente, in maniera analoga alla scelta precedente, relative alla presenza di sintomi di una malattia cardiovascolare. Se si avranno alcuni o tutti i sintomi di una particolare malattia cardiovascolare verrà dichiarato il fatto associato alla presenza della malattia e verrà comunicato all’utente che presenta i sintomi di quella malattia specifica. Altrimenti se non si presenteranno sintomi di nessuna malattia cardiovascolare verrà dichiarato Fact(absence\_of\_symptoms = 'yes') e verrà comunicato all’utente che non presenta alcun sintomo. Le malattie cardiovascolari che possono essere diagnosticate dal sistema sono: *Vasculite* (*blood vessels disease*), *Aritmia Cardiaca* (*heart arrhythmias*), *presenza di Difetti Cardiaci Congeniti* (*congenital heart defects*), *Cardiomiopatia* (*cardiomyopathy*), *malattia delle Valvole Cardiache* (*valvular heart disease*).

Di seguito vengono proposti due esempi di utilizzo del sistema esperto:





**Ontologia:**

Inserendo il valore 2, invece, si avrà accesso all’ontologia del sistema esperto, cioè l’ontologia che specifica il significato (in inglese) dei vari sintomi per le malattie cardiovascolari proposte.

In informatica una ontologia è una rappresentazione formale, condivisa ed esplicita di una concettualizzazione di un dominio di interesse, cioè si tratta di una teoria assiomatica di primo ordine esprimibile in una logica descrittiva. Il termine ontologia formale è entrato in uso nel campo dell'intelligenza artificiale e della rappresentazione della conoscenza, per descrivere il modo in cui diversi schemi vengono combinati in una struttura dati contenente tutte le entità rilevanti e le loro relazioni in un dominio.

L’ontologia è stata creata e letta utilizzando la libreria Owlready2, modulo per l’ontology-oriented programming in Python che consente un accesso trasparente alle ontologie OWL (contrariamente alle normali API basate su Java), nel file ***web\_ontology.py***.

Di seguito è proposto un esempio di lettura dell’ontologia:

