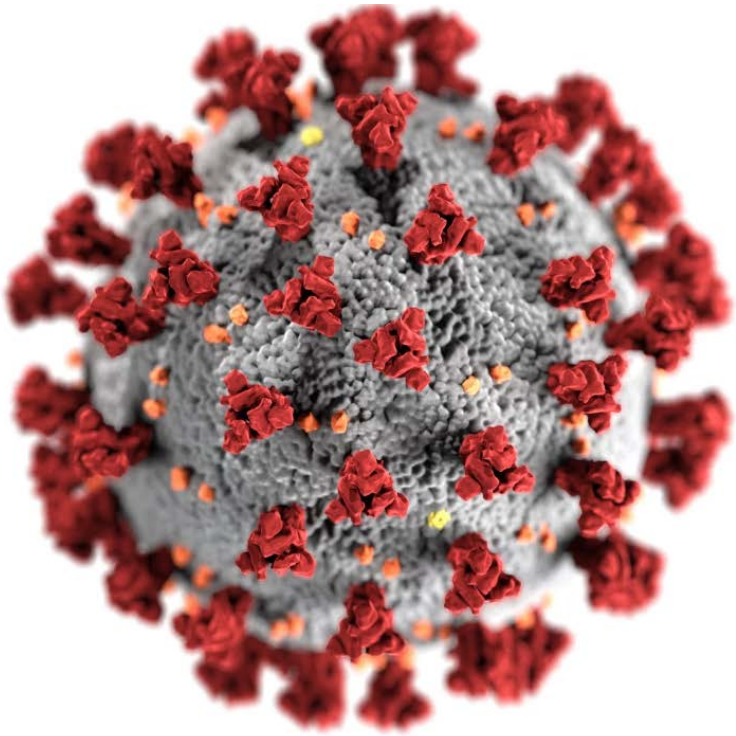
RELAZIONE PROGETTO DI INGEGNERIA DELLA CONOSCENZA
Anno Accademico 2022 – 2023

Sistema di Diagnostica sul Covid-19



Studenti:

-Angelo Bisceglia	718939	a.bisceglia11@studenti.uniba.it
-Matteo Brento	718753	m.brento@studenti.uniba.it
-Michele Fiore	718849	m.fiore91@studenti.uniba.it

INDICE:

INTRODUZIONE

La seguente documentazione rappresenta la descrizione tecnica del progetto "**Covid_Prediction**" realizzato dagli studenti Angelo Bisceglia, Matteo Brento, Michele Fiore, per il corso di Ingegneria della Conoscenza (A.A. 2022/2023).

Panoramica del progetto

Il progetto realizzato rappresenta un sistema di verifica e predizione del covid su una serie di pazienti presi in esame, considerando ciascun sintomo un elemento fulcro per la predizione. Sostanzialmente il nostro progetto si divide in tre macroaree:

1)-Sistema esperto con gestione di ontologia e CSP
2)-Apprendimento non supervisionato
3)-Apprendimento supervisionato

Obiettivi

-Utilizzare un'ontologia adeguata al sistema
-Popolarla attraverso i sintomi e le rispettive descrizioni
-Gestire le turnazioni delle prenotazioni attraverso il CSP(Constraint Satisfaction Problem)
-Utilizzare un sistema esperto per la diagnostica
-Analisi e studio di modelli di classificazione di apprendimento supervisionato e valutazione della qualità tramite delle metriche
-Analisi e studio del modello di clustering K-Means e considerazione degli indici di valutazione, nell'apprendimento non supervisionato

SISTEMA ESPERTO

Per la diagnostica del covid abbiamo deciso di utilizzare un sistema esperto, per fare ciò dopo varie ricerche abbiamo constatato che la libreria python “experta” facesse al caso nostro.

Funzionamento experta

Fondamentalmente experta si basa su fatti e regole.
-Per fatti intendiamo l’oggetto sul quale verranno verificate le regole.
-Una regola è un callable formato da due componenti
-LHS (sinistro): descrive le condizioni base alle quali la regola dovrebbe essere eseguita.
-RHS (destro) è l’insieme delle azioni da eseguire quando la regola viene eseguita.
Ogni regola affinchè possa essere utilizzata deve essere un metodo di una sottoclasse KnowledgeEngine.

Gestione dei fatti

Un fatto può essere:

-Dichiarato: Aggiunta di un nuovo fatto all’elenco dei fatti.
-Ritrattato: Rimozione di un fatto esistente dall’elenco dei fatti.
-Modificato: Ritiro di un fatto e aggiunta del medesimo fatto modificato.
-Duplicato: Aggiunta di un nuovo fatto all’elenco utilizzando un fatto esistente come modello inserendo alcune modifiche.

Funzionamento sistema esperto di diagnostica-predizione sul covid

L’utente è sottoposto ad una serie di domande attraverso le quali il sistema dichiara dei facts, nonché proposizioni sulle cui verrà effettuato un ragionamento considerando le varie Rule utile a predirre la diagnosi sul covid.

PREPROCESSING DEL DATASET

Per poter lavorare al meglio secondo la nostra idea di predizione della positività dei pazienti probabilmente affetti da covid-19 abbiamo preso in esame il seguente dataset
https://www.kaggle.com/datasets/iamhungundji/covid19-symptoms-checker
Il dataset contiene diverse colonne, solo alcune di esse si sono rivelate utili ai fini del nostro lavoro, infatti, per questo motivo è stato necessario effettuare un cleaning del dataset per rimuovere le colonne inutili e/o ridondanti; tra cui tutte quelle colonne che da un punto di vista diagnostico non risultano essere concretamente utili ai fini della predizione, per esempio quelle sull'età o sul genere del paziente. In particolare, abbiamo rimosso le colonne dal dataset originale e abbiamo riscritto solamente le colonne che ci servivano in un nuovo dataset che poi è stato usato per tutto il resto del progetto.

ONTOLOGIA

Definizione Ontologia

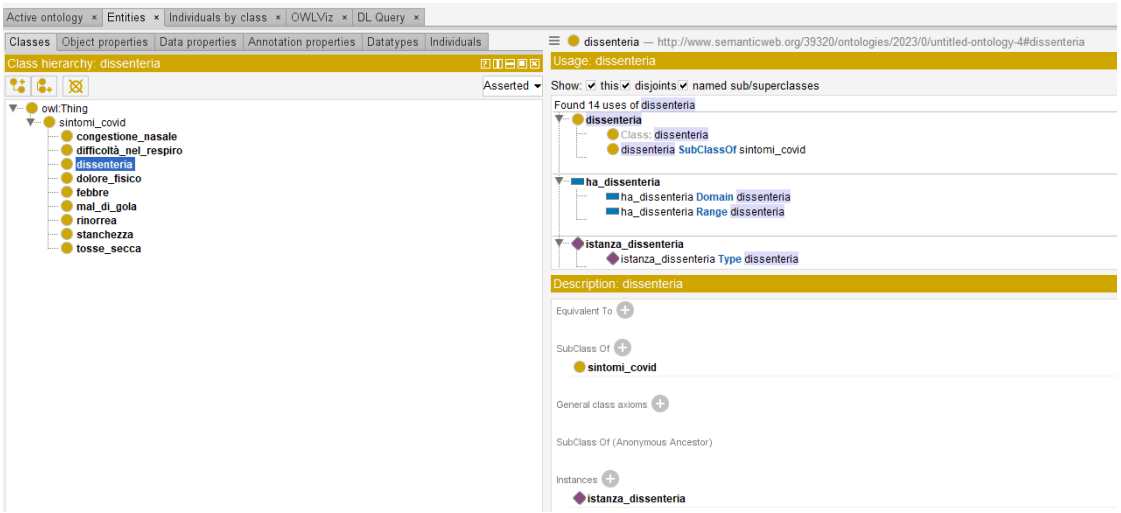
In informatica, un'ontologia `e una rappresentazione formale, condivisa ed esplicita di una concettualizzazione di un dominio di interesse. Il termine ontologia formale è entrato in uso nel campo dell'intelligenza artificiale e della rappresentazione della conoscenza, per descrivere il modo in cui diversi schemi vengono combinati in una struttura dati contenente tutte le entità rilevanti e le loro relazioni in un dominio.
Le ontologie distribuite sulla rete vengono descritte attraverso il linguaggio OWL (che sta per Web Ontology Language);
esso permette di descrivere il mondo in termini di:

- Individui: entità del mondo
- Classi: insieme di individui, accomunati da delle caratteristiche comuni
- Proprietà: relazioni che associano agli elementi del proprio dominio (individui), un elemento del codominio (valori che può assumere).

In particolare, si dividono in:
- Datatype property : se il codominio contiene solo tipi primitivi (interi, stringhe, …)
- Object property: quando il codominio contiene altre classi

Descrizione dell' ontologia

La strutturazione dell'ontologia è avvenuta tramite il software applicativo protège il quale ne permette la realizzazione in maniera intuitiva e completa, con l'aggiunta di plug-in molto utili al fine progettuale.
Con protège è stato possibile creare la nostra ontologia sul covid creando le varie entità corrispondenti a ciascun sintomo con le rispettive proprietà.



Successivamente abbiamo creato il file “covid.owl” contenente l’ontologia per poterla gestire tramite la libreria Owlready2 in python. Innanzitutto, abbiamo creato la classe covid_ontology.py in cui prendendo in input il file “covid.owl” è possibile istanziare l’ontologia.

```python
class covid_ontology:
    def __init__(self):
        self.ontology = get_ontology(os.path.basename("covid.owl")).load()
        self.dict_symptoms = {}

    def get_symptoms_descriptions(self):
        dict_symptoms_onto = {}

        for i in self.ontology.individuals():
            dict_symptoms_onto[str(i)] = i.descrizione_sintomo
        for k in dict_symptoms_onto.keys():
            k1 = k
            k1 = k1.replace("covid.istanza_", "")
            self.dict_symptoms[k1] = dict_symptoms_onto[k]

    def print_symptoms(self):

        i = 1
        dict_nums_symptoms = {}
        dict_nums_keys = {}

        for k in self.dict_symptoms.keys():
            print("Sintomo [%d]: Nome: %s" % (i, k))
            dict_nums_symptoms[i] = self.dict_symptoms[k]
            dict_nums_keys[i] = k
            i = i + 1

        return dict_nums_symptoms, dict_nums_keys
```

Utilizzando la regola ask_sympton=”si” istanziamo il rispettivo sintomo ogni qualvolta l’utente in input risponde di sì alla domanda appartenete all’ontologia.

Invece, in seguito, vi è il caso in cui l’utente abbia tutti i sintomi del covid, la regola richiama a sua volta la richiesta del tampone, appartenente al modulo del CSP.

```python
@Rule(Fact(ask_sympton="si"))
def sympotns(self):
    s1 = self.prototype_ask_sympton("Hai la febbre? [si/no]", Fact(febbre="si"))
    s2 = self.prototype_ask_sympton("Hai dei sintomi di stanchezza? [si/no]", Fact(stanchezza="si"))
    s3 = self.prototype_ask_sympton("Hai la tosse secca? [si/no]", Fact(tosse_secca="si"))
    s4 = self.prototype_ask_sympton("Hai difficolta nel respirare? [si/no]", Fact(difficolta_respiro="si"))
    s5 = self.prototype_ask_sympton("Ti senti irritato in gola? [si/no]", Fact(gola="si"))
    s6 = self.prototype_ask_sympton("Accussi dolori fisici senza un'apparente motivazione? [si/no]",
                                    Fact(dolore="si"))
    s7 = self.prototype_ask_sympton("Percepisci una senzazione di ostruzione nel naso? [si/no]",
                                    Fact(congestione_nasale="si"))
    s8 = self.prototype_ask_sympton("Hai il naso che ti cola? [si/no]", Fact(rinorrea="si"))
    s9 = self.prototype_ask_sympton("Le tue feci sono di una consistenza liquida? [si/no]", Fact(diarrea="si"))

    if s1 == "no" and s2 == "no" and s3 == "no" and s4 == "no" and s5 == "no" and s6 == "no" and s7 == "no"\
            and s8 == "no" and s9 == "no":
        self.no_sympton = 1

@Rule(AND(Fact(febbre="si"), Fact(stanchezza="si"), Fact(tosse_secca="si"), Fact(difficolta_respiro="si"), Fact(gola="si"),
          Fact(dolore="si"), Fact(congestione_nasale="si"), Fact(rinorrea="si"), Fact(diarrea="si")))
def all_symptons(self):
    print("\nPossiedi tutti i sintomi del covid!\n")
    self.declare(Fact(tutti_sintomi="si"))
    self.declare(Fact(chiedi_tampone="si"))
```

Nell’immagine seguente vediamo altre due regole: la prima il caso in cui l’utente non abbia alcun sintomo riconducibile al covid, mentre la seconda mostra un caso incerto, quello in cui l’utente abbia solo alcuni sintomi. È bene specificare che la prima regola è stata ideata a partire dalla negazione della regola della precedente immagine.

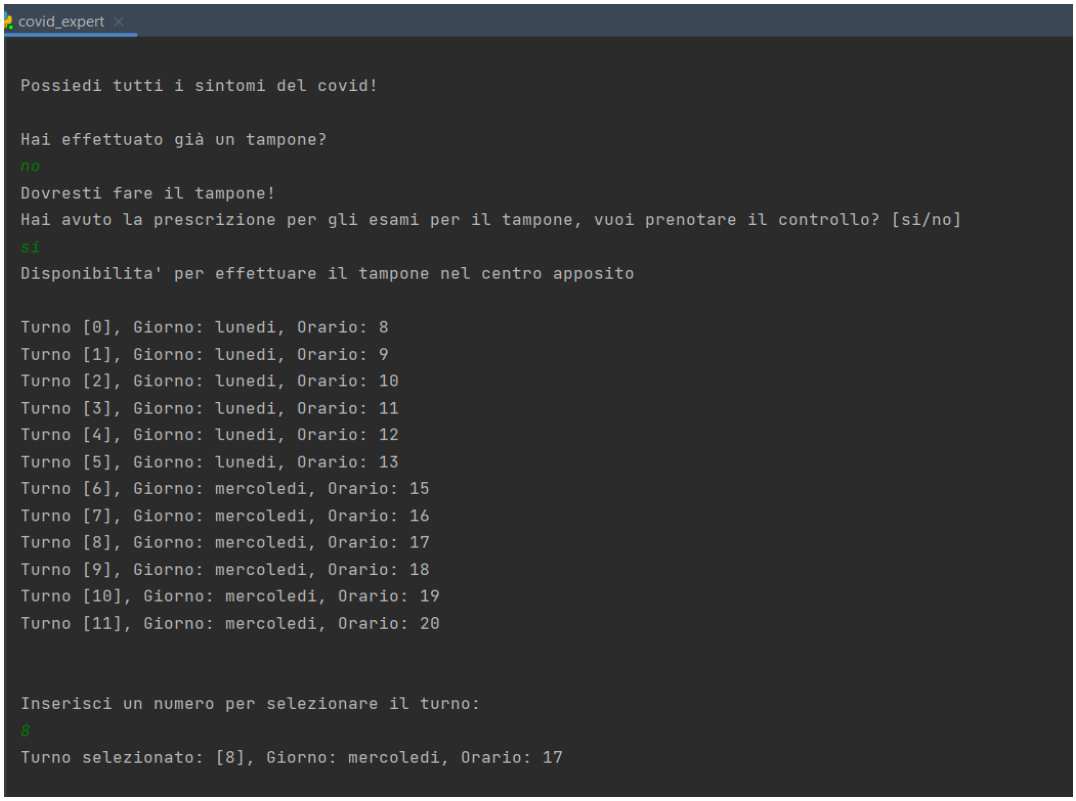
```
@Rule(NOT(AND(Fact(febbre="si"), Fact(stanchezza="si"), Fact(tosse_secca="si"), Fact(difficolta_respiro="si"),
           Fact(gola="si"), Fact(dolore="si"), Fact(comgestione_nasale="si"), Fact(rinorrea="si"), Fact(diarrea="si"))))
def no_symptons(self):
    if self.number_prints == 0 and self.no_sympton == 1:
        print("\nNon hai nessun sintomo riconducibile al covid!\n")
        self.declare(Fact(no_sintomi="si"))
        self.number_prints = self.number_prints+1

@Rule(AND(Fact(febbre="si"), Fact(stanchezza="si"), Fact(tosse_secca="si"), NOT(Fact(difficolta_respiro="si")),
          Fact(gola="si"), NOT(Fact(dolore="si")), NOT(Fact(congestione_nasale="si")), NOT(Fact(rinorrea="si")),
          NOT(Fact(diarrea="si"))))
def some_symptons(self):
    print("\nPossiedi solo alcuni sintomi riconducibili al covid, si consiglia comunque di fare il tampone!")
    self.declare(Fact(alcuni_sintomi="si"))
    self.declare(Fact(chiedi_tampone="si"))
```

CSP (Constraint Satisfaction Problem)

Molti problemi nell'ambito dell'Intelligenza Artificiale sono classificabili come Problemi di Soddisfacimento di Vincoli (Constraint Satisfaction Problem o CSP); Formalmente, un CSP può essere definito su un insieme finito di variabili (X1,X2, . . . ,Xn) i cui valori appartengono a domini finiti di definizione(D1,D2, . . . ,Dn) e su un insieme di vincoli(C1,C2, . . . ,Cn). Un vincolo su un insieme di variabili è una restrizione dei valori che le variabili possono assumere simultaneamente. Concettualmente, un vincolo può essere visto come un insieme che contiene tutti i valori che le variabili possono assumere contemporaneamente: un vincolo tra k variabili C(Xi1 ,Xi2 , . . . ,Xik), è un sottoinsieme del prodotto cartesiano dei domini delle variabili coinvolte Di1 ,Di2 , . . . ,Dik che specifica quali valori delle variabili sono compatibili con le altre. Questo insieme può essere rappresentato in molti modi, come ad esempio, matrici, equazioni, disuguaglianze o relazioni.
Per mettere in pratica questo sistema ci siamo avvalsi della libreria di nome constraint, la quale ci ha permesso di realizzare un semplice CSP in grado di mostrare la disponibilità dei centri per effettuare i tamponi, nel caso in cui il sistema lo preveda.
Il funzionamento del nostro CSP è il seguente:

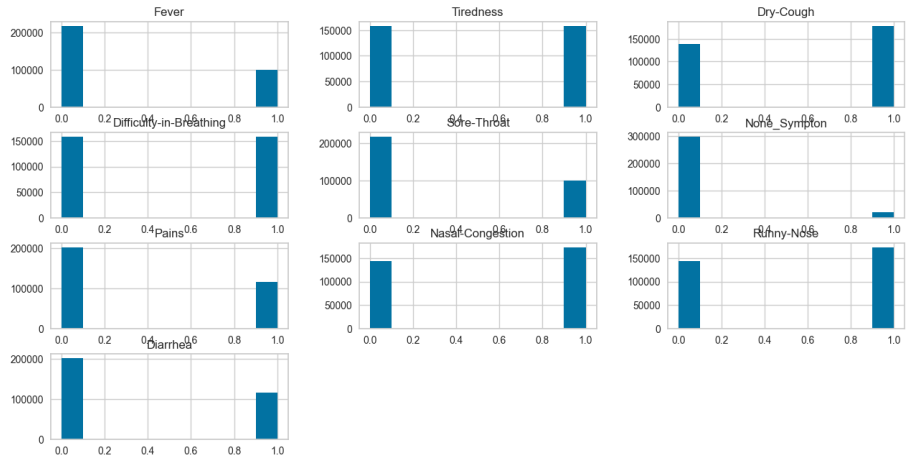
- Alla base vi è una sottoclasse di Problem (classe già definita in constraint, che modella un CSP).
- Vengono aggiunte variabili con proprio dominio associato in maniera esplicita.
- In base alle risposte dell'utente, il sistema decide se mostrare la possibilità di prescrivere una visita.
- Se il sistema decide di prescrivere il controllo, allora qui entra in gioco il CSP.
- Il CSP è relativo agli orari di apertura del centro adibito all'effettuare i tamponi per il covid-19.
- Ad esempio: il sistema indica l'ora e il giorno in cui è possibile effettuare la prenotazione per il tampone.
- In generale, il sistema, indica i possibili orari a cui l'utente può presentarsi per sottoporsi ai controlli per verificare la positività o meno al virus del covid-19.

Un esempio:



APPRENDIMENTO NON SUPERVISIONATO

Gli algoritmi di apprendimento non supervisionato lavorano esclusivamente con le variabili indipendenti (x) fornite al modello.
Il compito del modello è quello di trovare caratteristiche comuni in modo tale da ricavare le variabili dipendenti (y).
Il Clustering è una delle tecniche algoritmiche più importanti e popolari per l'apprendimento non supervisionato.
In questo metodo è possibile elaborare e identificare i gruppi (Cluster) a partire da questi dati.
Prima di visualizzare nello specifico il Clustering effettuato, tramite l'importazione di librerie come Matplotlib, Seaborn e Plotly abbiamo reso possibile la visualizzazione di dati e la creazione di grafici.
Nello specifico abbiamo creato un Istogramma iniziale che vada a dare un'idea di come è impostato il dataset e della correlazione che c'è tra i dati.



Come si può notare, il nostro è un dataset basato esclusivamente su valori binari, 0 e 1. Pertanto, il precedente Istogramma mostra la capacità effettiva di 0 e di 1 relativa ad ogni sintomo che una persona può riscontrare.
Si associa a 0 un riscontro di negatività al sintomo e ad 1 un riscontro di positività.

Il seguente Istogramma distribuisce sui due livelli, 0 ed 1, il numero preciso totale di persone che hanno riscontrato un sintomo (1) oppure no (0), mostrando dunque, una netta differenza di viralità tra i diversi sintomi.

K-MEANS

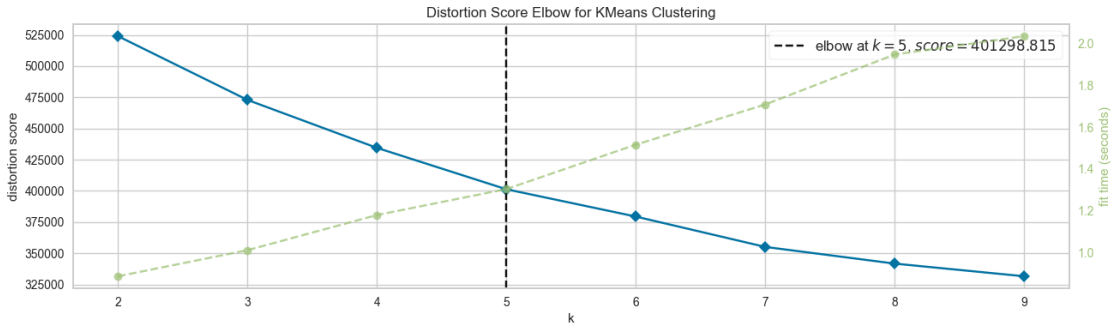
Il clustering K-Means è un algoritmo di apprendimento non supervisionato basato sulla distanza in cui i punti dati vicini tra loro sono raggruppati in un determinato numero di 'Cluster' identificabili come gruppi. Gli step che questo algoritmo esegue sono i seguenti:

- Inizializza 'K', cioè il numero di cluster da creare;
- Assegna casualmente K punti centroidi;
- Assegna ogni punto dati al centroide più vicino per creare K cluster;

All'interno del nostro progetto utilizziamo il Metodo Elbow per la rappresentazione grafica della ricerca della 'K' ottimale in un Clustering K-Means.
Funziona trovando WCSS (Within-Cluster Sum of Square), cioè la somma della distanza quadrata tra i punti in un cluster e il centroide del cluster.
Esso mostra i valori WCSS (y) corrispondenti ai diversi valori di K (x). Quando vediamo una forma a gomito nel grafico, scegliamo il valore K in cui viene creato il gomito.
Per l'implementazione del nostro codice in python, abbiamo utilizzato le seguenti librerie:

- Scikit Learn, Sklearn.Cluster per l'importazione di KMeans e Birch e delle metriche;
- Standard Scaler per la trasformazione del dataset;
- Matplotlib.pyplot per la visualizzazione dati;
- Plotly e Seaborn per i grafici;
- Numpy e Pandas per la gestione dati;
- Yellowbrick per il KElbow Visualizer.

Il grafico ottenuto dall'esecuzione del Metodo Elbow per la ricerca della K ottimale è il seguente:



Esso seleziona il punto a gomito più preciso in K=5 e lo considera quello più ottimale. Pertanto, abbiamo quindi stabilito all'interno della nostra implementazione di codice del Clustering che l'n_clusters ottimale è pari a 5.

Nel caso in cui il metodo Elbow non riesca a stabilire un valore definito di K si può utilizzare la seguente metrica: Silhouette Score.

Il punteggio Silhouette è un metodo utile per trovare K quando il metodo Elbow non mostra un punto esatto. Il valore del punteggio Silhouette è compreso in un range tra -1 e 1 dove:

- 1: I punti sono perfettamente assegnati in un cluster e i cluster sono facilmente distinguibili;
- 0: I cluster sono sovrapposti;
- -1: I punti sono assegnati erroneamente in un cluster.

Nella nostra situazione, nonostante l'ottenimento di un K esatto tramite il metodo Elbow, abbiamo provato a testare la seguente metrica per esaminare il valore ritornato:

```
score_silhouette_train = silhouette_score (X_train, pred_train)
score_silhouette_test = silhouette_score (X_test, pred_test)
print("--> TRAIN : For n_clusters = 5, silhouette score is {}".format(score_silhouette_train))
print("--> TEST : For n_clusters = 5, silhouette score is {}".format(score_silhouette_test))
```

Il seguente codice ha poi prodotto il seguente risultato:

```
--> TRAIN : For n_clusters = 5, silhouette score is 0.16628191323120114
--> TEST : For n_clusters = 5, silhouette score is 0.1664385739890668
```

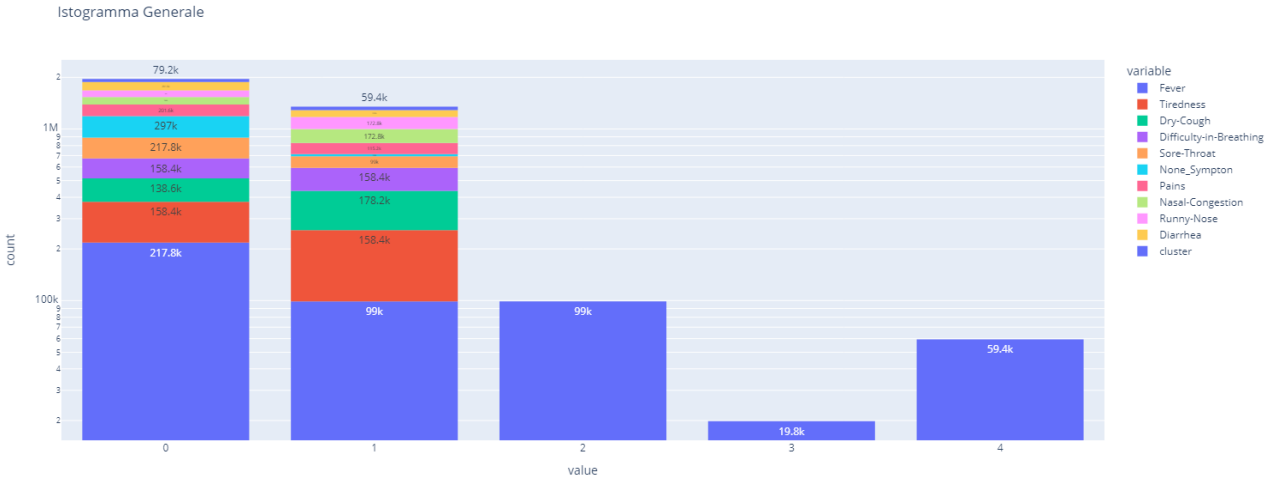
Come possiamo dedurre, il valore della silhouette score relativo al nostro modello di clustering è di poco superiore allo 0, pertanto considerando i parametri di valutazione precedentemente elencati, I cluster tendono a sovrapporsi.
Ciò è dovuto alla lavorazione di un dataset molto ristretto, composto di soli 0 e 1 relativi ai diversi sintomi.
Abbiamo utilizzato oltre alla Silhouette Score, un'ulteriore indice chiamato: Davies Bouldin Score.
Il punteggio è definito come la media di somiglianza di ciascun cluster con il suo cluster più simile, dove la somiglianza è il rapporto tra le distanze all'interno del cluster e le distanze tra i cluster.
Il seguente codice ne mostra l'implementazione:

```
score_davies_train = metrics.davies_bouldin_score(X_train, pred_train)
score_davies_test = metrics.davies_bouldin_score(X_test, pred_test)
print("--> TRAIN : For n_clusters = 5, davies_bouldin score is {}".format(score_davies_train))
print("--> TEST : For n_clusters = 5, davies_bouldin score is {}".format(score_davies_test))
```

E il seguente, è il risultato ottenuto:

```
--> TRAIN : For n_clusters = 5, davies_bouldin score is 2.003950410068355
--> TEST : For n_clusters = 5, davies_bouldin score is 1.9983005992217815
```

Considerati gli n_clusters=5 definiti all'interno dell'implementazione del nostro codice, l'indice lavorerà su un range da 0 a 4.
Essendo un indice per la valutazione della qualità, restituisce il numero ottimale di cluster.
Nel nostro caso, il risultato dell'indice sia in fase di Test che di Train è pari a 2 approssimativamente, rileva dunque un numero maggiore di Cluster pari a 2, considerandolo il valore ottimale.
Il seguente Istogramma mostra quanto appena detto:



Come si può evincere dal seguente grafico, la suddivisione in cluster effettuata è la seguente:

- 0: 79.2k Totali;
- 1: 59.4k Totali;
- 2: 99k Totali (Valore maggiore);
- 3: 19.8k Totali;
- 4: 59.4k Totali.

Il valore maggiore è 2.

APPRENDIMENTO SUPERVISIONATO
ALGORITMI DI CLASSIFICAZIONE

In questa sezione di progetto sono stati implementati diversi algoritmi di Apprendimento Supervisionato appartenenti, in modo specifico, alla categoria degli Algoritmi di Classificazione. Nel nostro specifico caso l'output restituito dal modello è una variabile qualitativa di tipo binario e gli algoritmi utilizzati sono:

- **K-Nearest Neighbours**;
- **Decision Tree Classifier**;
- **Random Forest Classifier**;
- **XG Boost Classifier**.

Le librerie open source utilizzate sono:

- **Pandas**: Si occupa del caricamento e della manipolazione dei dati all'interno del nostro ambiente di lavoro;
- **Numpy**: Contiene le funzioni matematiche che permettono di operare sul nostro dataset;
- **Scikit Learn**: Contiene al suo interno i modelli di classificazione che utilizzeremo e le diverse metriche che sfrutteremo per valutare le prestazioni del modello;
- **Seaborn** e **Matplotlib**: Si occupano della creazione e della visualizzazione di grafici statistici.

Non è stato utilizzato il nostro dataset in forma 'grezza', ma quello modificato durante la fase di pre-processing dei dati.

KNN CLASSIFIER

Il nostro obiettivo è quello di riuscire a prevedere la viralità di un sintomo specifico, in questo caso 'Fever' (Febbre) in relazione al numero di persone che nel nostro dataset presentano o meno il sintomo in questione.
Dopo aver importato e stampato a video in Python il nostro dataset sfruttando la libreria Pandas, come variabili d'uscita, nonché quelle da prevedere, assegniamo 'Fever', ossia il sintomo 'Febbre' che corrisponde tra l'altro al primo in ordine di disposizione all'interno del nostro dataset. Le restanti features rappresentano le nostre variabili d'ingresso.
Suddividiamo le variabili di input e output in test e training. Nel nostro caso il 30% del dataset sarà casualmente assegnato per poter comparare le predizioni con i risultati attesi.
Qui di seguito è proposto il codice formulato:

```
df = pd.read_csv('C:/Users/matte/OneDrive/Desktop/Cleaned-Data2.csv')
print(df.head(5))

X = df.drop(['Fever'], axis=1)
y = df['Fever']

from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3)
```

Successivamente importiamo dalla libreria Scikit Learn (Sklearn) l'Algoritmo del KNN e una metrica che andrà a misurare l'accuratezza delle previsioni del nostro modello. Da sklearn.metrics importiamo 'Classification Report' per calcolare tutti i parametri di Accuracy, Precision, F1, Recall. Come parametro di tuning scegliamo la distanza euclidea come metrica per il calcolo della distanza fra i vari punti del dataset.
In seguito, andiamo ad allenare il nostro classificatore sul dataset di training.
La figura qui di seguito riporta l'implementazione del codice dell'algoritmo.

```
from sklearn.metrics import classification_report
from sklearn.neighbors import KNeighborsClassifier
classifier = KNeighborsClassifier(n_neighbors=1,metric='euclidean')
classifier.fit(X_train,y_train)
y_pred = classifier.predict(X_test)
print("Accuratezza:",metrics.accuracy_score(y_test, y_pred))

pred_y_df=pd.DataFrame({'Valore corrente':y_test, 'Valore predetto':y_pred})
pred_y_df[0:20]

print(classification_report(y_test,y_pred))
```

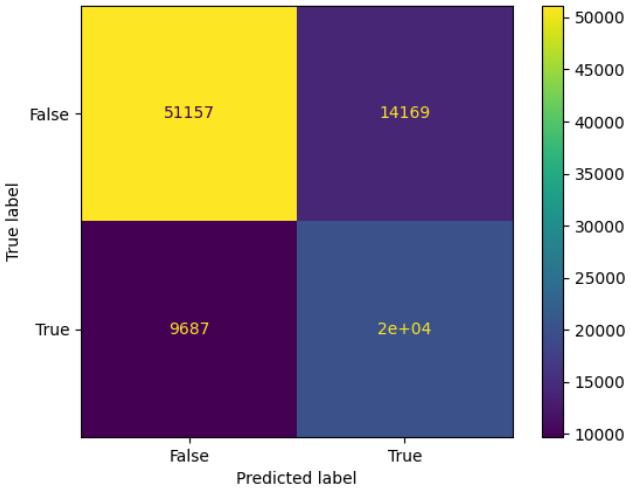
Qui di seguito vengono stampati a video i risultati delle metriche calcolate. L'accuratezza calcolata tramite metrics.accuracy_score(y_test, y_pred) e il Classification Report.

$$\text{Accuracy} = \frac{\text{veri positivi+veri negativi}}{\text{veri positivi+veri negativi+falsi positivi+falsi negativi}}$$

```
Accuratezza: 0.748989898989899
              precision    recall  f1-score   support

           0       0.84      0.78      0.81     65326
           1       0.59      0.67      0.63     29714

    accuracy                           0.75     95040
   macro avg       0.71      0.73      0.72     95040
weighted avg       0.76      0.75      0.75     95040
```

Utilizziamo una Matrice di Confusione per analizzare gli errori compiuti da un modello di machine learning. Più nello specifico è utile per valutare la qualità delle previsioni del modello di classificazione. In particolar modo mette in evidenza, dove sbaglia il modello, in quali istanze risponde meglio e in quali peggio. Possiamo distinguere quattro casi:

- True Positive (TP);
- True Negative (TN);
- False Positive (FP);
- False Negative (FN).



Dalla seguente Matrice di Confusione possiamo riscontrare che oltre agli errori di previsione sono presenti anche un numero (seppur ridotto) di False Positive e False Negative che contribuiscono ad aumentare l'imprecisione del nostro dataset.
Sulla base dei dati forniti dalla Matrice di Confusione possiamo calcolare le differenti metriche messe a disposizione da Scikit Learn quali: Accuracy, Precision, Recall, F1.

Accuracy: Misura la percentuale delle previsioni esatte sul totale delle istanze. Varia da 0 (Peggiore) ad 1 (Migliore). Nel nostro specifico caso abbiamo un Accuratezza del KNN pari a 0.748989898989899, pertanto un valore molto più vicino ad 1 (Migliore) che a 0 (Peggiore). Ciò dimostra una non perfetta, ma buona accuratezza del modello.

$$ACC = \frac{TP + TN}{TP + TN + FP + FN} = 1 - ERR$$

Precision: Percentuale delle previsioni positive corrette (TP) sul totale delle previsioni positive del modello (TP+FP). Nel nostro caso c'è una precisione relativa agli 0 di 0.84 e agli 1 di 0.59. PR = 0.59

$$PR = \frac{TP}{TP + FP}$$

Recall: Percentuale delle previsioni positive corrette (TP) sul totale delle istanze positive (TP+FN), varia da 0 (Peggiore) ad 1 (Migliore). Nel nostro caso c'è un Recall relativo agli 0 di 0.78 e agli 1 di 0.67. Recall = 0.67.

$$Recall = \frac{TP}{TP + FN}$$

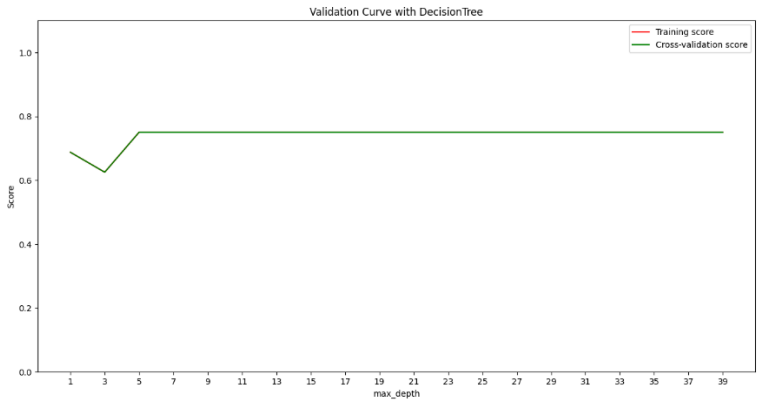
F1: Media armonica delle metriche Precision e Recall. Varia da 0 (Peggiore) ad 1 (Migliore). Nel nostro caso c'è un F1 relativa agli 0 di 0.81 e agli 1 di 0.63. F1 = 0.63.

$$FS = \frac{2 \cdot Recall \cdot Precision}{Recall + Precision}$$

Il nostro obiettivo ora, dopo aver studiato il comportamento e preso visione dei risultati del KNN, è quello di verificare, se utilizzando modelli diversi dal KNN, riusciamo ad ottenere risultati migliori che contengano meno False Positive (FP) e False Negative (FN).

DECISION TREE CLASSIFIER

Andiamo ad analizzare ora il Decision Tree Classifier, lavorando ovviamente sempre sullo stesso Dataset. Da 'sklearn.tree' importiamo il nostro DecisionTreeClassifier. Andiamo a definire un importante parametro di tuning: 'max_depth', che stabilisce il numero massimo di nodi che l'albero può raggiungere e ho testato il modello con max_depth impostato a 40.



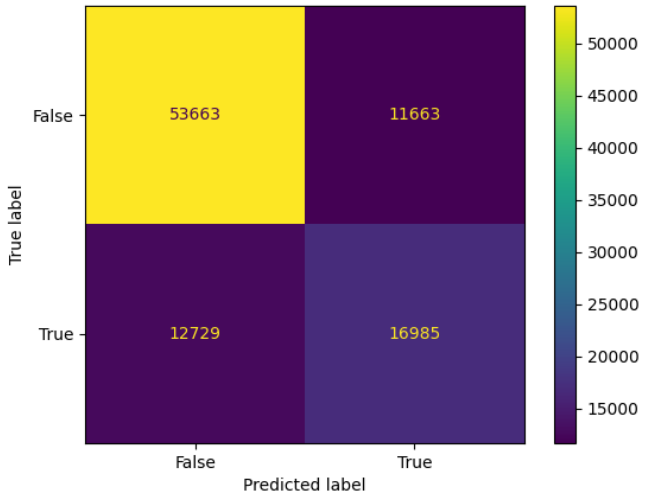
Qui di seguito è proposto il codice relativo all'implementazione del Decision Tree:

```
from sklearn.tree import DecisionTreeClassifier
dtc = DecisionTreeClassifier(max_depth=40)
dtc.fit(X_train,y_train)
y_dtc_pred = dtc.predict(X_test)
print("Accuratezza:",metrics.accuracy_score(y_test, y_dtc_pred))

pred_y_df=pd.DataFrame({'Valore corrente':y_test, 'Valore predetto':y_dtc_pred})
pred_y_df[0:20]

print(classification_report(y_test,y_dtc_pred))
```

Il risultato ottenuto dall'esecuzione del codice relativo al calcolo delle metriche è il seguente:

```
Accuratezza: 0.7431502525252526
              precision    recall  f1-score   support

           0       0.82      0.80      0.81     65326
           1       0.58      0.63      0.60     29714

    accuracy                           0.74     95040
   macro avg       0.70      0.71      0.71     95040
weighted avg       0.75      0.74      0.75     95040
```

Qui di seguito è mostrata la Matrice di Confusione del Decision Tree:



Come si può evincere dalla visualizzazione del Classification Report del Decision Tree Classifier, il valore dell'Accuratezza è di pochissimo inferiore rispetto a quello del KNN.
Il valore di Precision è di 0.59 nel KNN mentre di 0.58 nel Decision Tree, anche in questo caso di poco inferiore e dunque poco meno preciso.
Il valore di Recall è di 0.67 nel KNN mentre 0.63 nel Decision Tree.
Il valore di F1 è di 0.63 nel KNN mentre 0.60 nel Decision Tree.
La somma di Falsi Positivi e Falsi Negativi è superiore rispetto a quella del KNN.
In linea di massima possiamo quindi concludere che il KNN è più preciso del Decision Tree.

RANDOM FOREST CLASSIFIER

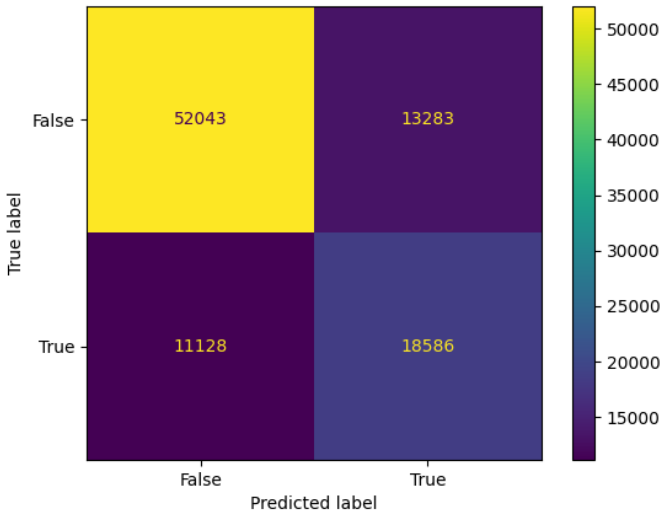
Analizziamo ora un ulteriore modello da confrontare: Random Forest Classifier.
Oltre alla variabile 'max_depth' dichiarata precedentemente anche per il Decision Tree, in questo caso troviamo una ulteriore variabile 'n_estimators' che serve per stabilire il numero di alberi decisionali che si occuperanno di ritrovare un risultato al nostro problema.
Dalla seguente 'Validation Curve', importata tramite libreria 'sklearn.model_selection' possiamo notare che la nostra Validation Score non presenta picchi nel grafico ma è lineare pertanto, il valore degli n_estimators è impostato a 120, ma cambia poco.
Qui di seguito è mostrato il codice relativo all'implementazione del Random Forest Classifier:

```
from sklearn.ensemble import RandomForestClassifier
rfc = RandomForestClassifier(n_estimators=120, max_depth=40)
rfc.fit(X_train,y_train)
y_rfc_pred = rfc.predict(X_test)
print("Accuratezza:",metrics.accuracy_score(y_test, y_rfc_pred))

pred_y_df=pd.DataFrame({'Valore corrente':y_test, 'Valore predetto':y_rfc_pred})
pred_y_df[0:20]

print(classification_report(y_test,y_rfc_pred))
```

Il risultato ottenuto dall'esecuzione del codice relativo al calcolo delle metriche è il seguente:

```
Accuratezza: 0.7433501683501683
              precision    recall  f1-score   support

           0       0.81      0.82      0.81     65326
           1       0.59      0.57      0.58     29714

    accuracy                           0.74     95040
   macro avg       0.70      0.70      0.70     95040
weighted avg       0.74      0.74      0.74     95040
```

Qui di seguito è mostrata la Matrice di Confusione del Random Forest:



Come si può evincere dalla visualizzazione del Classification Report del Random Forest Classifier, il valore dell'Accuratezza è leggermente superiore rispetto a quello ritornato dal Decision Tree Classifier, ma inferiore a quello del KNN.
Il valore Precision è uguale a quello del Decision Tree e ancora inferiore al KNN mentre Recall ed F1 risultano inferiori rispetto ai valori ritornati dai due precedenti modelli.
Valutando la qualità del nostro modello tramite il grafico della Matrice di Confusione, il numero di Falsi Positivi e Falsi Negativi è superiore rispetto al KNN.
In definitiva, il KNN risulta ancora il modello più ottimale tra i tre analizzati.

XG BOOST CLASSIFIER

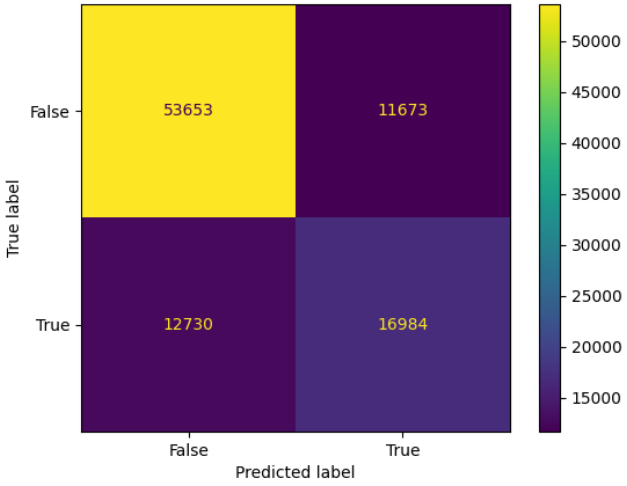
Introduciamo ora la libreria XG Boost, o meglio, Extreme Gradient Boosting Classifier. Questo modello si differenzia dai due precedenti per la capacità di apprendere dagli errori della prima fase di apprendimento e correggerli per creare un successivo modello più preciso.
Un parametro che impostiamo per migliorare la precisione del nostro modello è 'learning_rate', il quale imposta la misura con cui andare a migliorare di volta in volta la precisione e ridurre gli errori residui presenti ad ogni passaggio. Un valore pari a 0.01 ci permette di fare 100 passaggi intermedi in più rispetto a valore 1, all'interno dei quali sono creati di volta in volta alberi decisionali che vanno a ridurre sempre di più l'errore.
Ne mostriamo l'implementazione del codice qui di seguito:

```
from sklearn import ensemble
gb_clf = ensemble.GradientBoostingClassifier(learning_rate=0.05, max_depth=20)
gb_clf.fit(X_train, y_train)
y_gb_pred=gb_clf.predict(X_test)
print("Accuratezza:",metrics.accuracy_score(y_test, y_gb_pred))

pred_y_df=pd.DataFrame({'Valore corrente':y_test, 'Valore predetto':y_gb_pred})
pred_y_df[0:20]

print(classification_report(y_test,y_gb_pred))
```

Il risultato ottenuto dall'esecuzione del codice relativo al calcolo delle metriche è il seguente:

```
Accuratezza: 0.7432344276094276
              precision    recall  f1-score   support

           0       0.81      0.82      0.81     65326
           1       0.59      0.57      0.58     29714

    accuracy                           0.74     95040
   macro avg       0.70      0.70      0.70     95040
weighted avg       0.74      0.74      0.74     95040
```

Qui di seguito è mostrata la Matrice di Confusione dell'XG Boost Classifier:



Come si può evincere dalla visualizzazione del Classification Report, il valore dell'accuratezza è inferiore rispetto a quello ritornato dal KNN ed approssimativamente uguale a quello del Decision Tree e Random Forest.
Qui di seguito sono riportati e messi a confronto i valori prodotti dalle matrici di confusione durante la prima esecuzione dei quattro modelli studiati.

	True-Positive	True-Negative	False-Positive	False-Negative
KNN	20.000	51.157	14.169	9.687
Decision Tree	16.985	53.663	11.663	12.729
Random Forest	18.586	52.043	13.283	11.128
XG Boost	16.894	53.653	11.673	12.730

Sulla base di essi è stata prodotta la seguente griglia di metriche relativa ad ogni modello analizzato:

	Accuracy	Precision	Recall	F1Score
KNN	0,7489	0,59	0,67	0,63
Decision Tree	0,7431	0,58	0,63	0,60
Random Forest	0,7433	0,59	0,57	0,58
XG Boost	0,7432	0,59	0,57	0,58

Per ottenere un'analisi più accurata abbiamo effettuato più test del codice (esecuzioni) e qui di seguito ne riportiamo i valori ottenuti:

TEST N.2

	True-Positive	True-Negative	False-Positive	False-Negative
KNN	17.308	53.879	11.356	12.497
Random Forest	18.193	52.614	12.621	11.612
Decision Tree	17.664	53.146	12.089	12.141
XG Boost	15.504	55.321	9.914	14.301

	Accuracy	Precision	Recall	F1Score
KNN	0,749	0,6	0,58	0,59
Random Forest	0,745	0,59	0,59	0,59
Decision Tree	0,745	0,59	0,61	0,6
XG Boost	0,7452	0,61	0,52	0,56

TEST N.3

	True-Positive	True-Negative	False-Positive	False-Negative
KNN	18.277	53.075	12.395	11.293
Random Forest	20.000	51.099	14.371	10.000
Decision Tree	18.981	51.645	13.825	10.589
XG Boost	18.441	52.189	13.281	11.129

	Accuracy	Precision	Recall	F1Score
KNN	0,75	0,6	0,62	0,61
Random Forest	0,743	0,58	0,64	0,61
Decision Tree	0,743	0,58	0,64	0,61
XG Boost	0,743	0,58	0,62	0,6

VALUTAZIONE RISULTATI DI TEST

Considerando i valori dei Veri Positivi, Veri Negativi, Falsi Positivi e Falsi Negativi ritornati dai tre Test effettuati, possiamo effettuare le seguenti valutazioni:

1. Il numero di Veri Positivi più variabile, relativo ai 4 modelli di Classificazione usati è nel primo test effettuato.
2. Il valore dei Veri Negativi varia da modello a modello in relazione alla quantità di Veri Positivi ritornati.
3. Nei primi due test il valore dei Falsi Positivi e dei Falsi Negativi sono equamente bilanciati e per ogni Test effettuato, 2 Modelli ritornano un numero maggiore di Falsi Positivi e 2 un numero maggiore di Falsi Negativi.
4. Nel terzo test il valore di Falsi Positivi è sempre maggiore rispetto al valore dei Falsi Negativi.
5. Il terzo test è quello che ritorna risultati più simili in termini di Veri Positivi, ciò significa che i 4 modelli hanno effettuato una misurazione approssimativamente simile.

Considerando i valori delle metriche di Accuracy, Precision, Recall e F1 calcolabili sulla base dei VP,VN,FP e FN ritornati dalle matrici di confusione, possiamo effettuare le seguenti valutazioni:

1. Il valore approssimativo dell'Accuratezza, considerando i tre test effettuati, è sempre attorno a 0.75 ed il Modello di Classificazione più preciso tra i 4 utilizzati nel calcolo di quest'ultima è il KNN, che ne restituisce il valore più ottimale.
2. Il valore di Precision più ottimale, basato sul confronto dei risultati ottenuti dai 4 modelli nei tre Test effettuati, è pari a 0.61 nell'XG Boost della seconda misurazione. In generale il valore medio di Precision ritornato considerando tutti i Modelli nell'arco delle tre fasi di Test oscilla tra 0.59 e 0.6.
3. Il valore di Recall più ottimale, basato sul confronto dei risultati ottenuti dai 4 modelli nei tre Test effettuati, è pari a 0.67 nel KNN della prima misurazione. In generale il valore medio di Recall ritornato considerando tutti i Modelli nell'arco delle tre fasi di Test oscilla tra 0.62 e 0.64.
4. Il valore di F1 più ottimale, basato sul confronto dei risultati ottenuti dai 4 modelli nei tre Test effettuati, è pari a 0.63 nel KNN della prima misurazione. In generale il valore medio di F1 ritornato considerando tutti i Modelli nell'arco delle tre fasi di Test è approssimativamente attorno a 0.61.

5. Vedendo come si eguagliano tra loro i risultati delle metriche qualitative applicate, possiamo determinare che tra i tre Test effettuati, il terzo è quello che ha ritornato dei risultati più simili, quasi uguali, tra loro. Infatti KNN, in termini di Accuratezza restituisce 0.75 (valore più alto) e gli altri tre modelli 0.743. In termini di Precision ritornano un valore tra 0.58 o 0.59. In termini di Recall c'è l'unica grande differenza di misurazione che oscilla da un minimo di 0.57 ad un massimo 0.67 (KNN). In termini di F1 è ritornata una media approssimativa attorno a 0.61.

Dunque, fatta eccezione per la metrica di Precision per la quale prevale la misurazione del Modello XG Boost, per le restanti tre metriche qualitative il Modello KNN è quello che restituisce in media, valori sempre più ottimali e accurati rispetto agli altri tre modelli considerati.

CROSS VALIDATION

Quando si regolano i modelli, si cerca di aumentare le prestazioni complessive del modello su dati invisibili. L'ottimizzazione degli iperparametri può portare a prestazioni migliori sui set di test. L'ottimizzazione dei parametri di test però può portare alla perdita di informazioni, nel momento in cui vengano effettuate più esecuzioni sullo stesso set di dati in quanto nel calcolo delle varie metriche possono verificarsi lievi errori di previsione che tendono però a variarne i risultati.
Per correggere ciò usiamo la **Cross Validation**.

Tramite il KFold, i dati di addestramento vengono suddivisi in un numero k di insiemi più piccoli. Il modello viene addestrato su k-1 pieghe del set di addestramento.
Qui di seguito è proposto il codice di esecuzione della Cross Validation tramite KFold:

```
clf = DecisionTreeClassifier(random_state=42)
k_folds = KFold(n_splits = 5)
scores = cross_val_score(clf, X, y, cv = k_folds)
print("Decision Tree Cross Validation Scores: ", scores)
print("Average CV Score: ", scores.mean())
print("Number of CV Scores used in Average: ", len(scores))
```

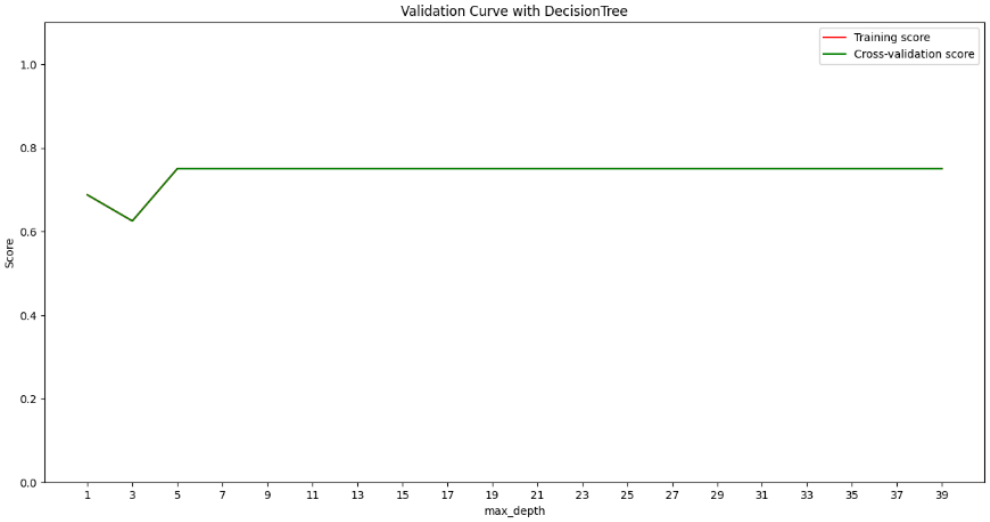
IL risultato ritornato è il seguente:

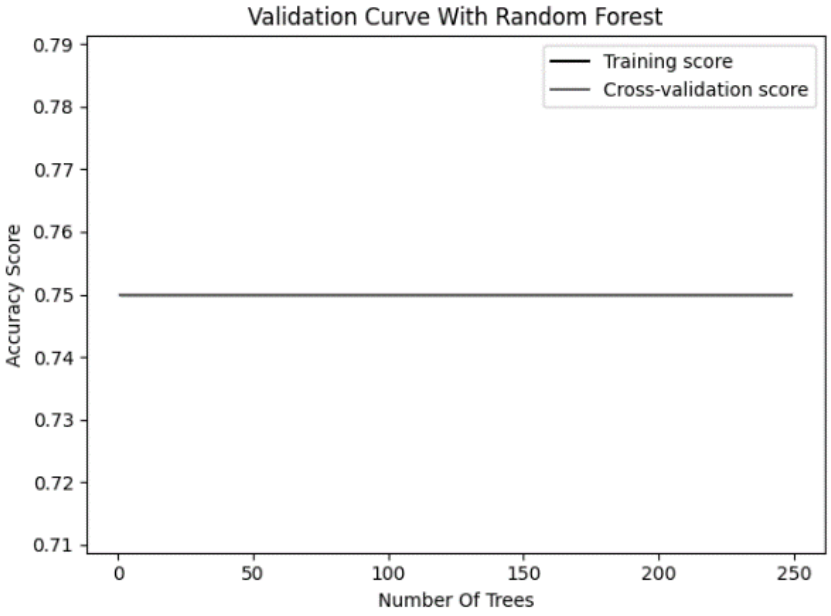
```
Decision Tree Cross Validation Scores:  [0.75 0.75 0.75 0.75 0.75]
Average CV Score:  0.75
Number of CV Scores used in Average:  5
```

Lo abbiamo successivamente confrontato, eseguendo lo stesso procedimento con un altro modello utilizzato come il Random Forest ed il risultato ottenuto è esattamente uguale:

```
Random Forest Cross Validation Scores:  [0.75 0.75 0.75 0.75 0.75]
Average CV Score:  0.75
Number of CV Scores used in Average:  5
```

Per poter accertare l'effettiva validità di questi risultati abbiamo creato i due seguenti grafici relativi alla Cross Validation dei due modelli:





In conclusione, come possiamo vedere dai due grafici qui sopra, la Cross Validation Score si sovrappone con Training Score in entrambi i modelli ed il valore specifico ritornato è pari a 0.75, esattamente come mostrato in output precedentemente dall'esecuzione del codice della Cross Validation dei due modelli comparati (Decision Tree Classifier & Random Forest Classifier).

CONSIDERAZIONI FINALI

In conclusione, il sistema realizzato soddisfa gli obiettivi da noi stabiliti inizialmente. Per il sistema esperto con la relativa gestione dell'ontologia, abbiamo riscontrato una funzionalità soddisfacente sia nell'implementazione del CSP con cui gestiamo le turnazioni dei tamponi e sia con la diagnostica dei sintomi dei pazienti relativa all'ontologia.
Inoltre, abbiamo riscontato dei buoni risultati in linea con il nostro dataset riguardo i modelli e indici utilizzati per le due fasi di apprendimento, supervisionato e non supervisionato.