Immagine che contiene testo

Descrizione generata automaticamente

DOCUMENTAZIONE DI PROGETTO – CORSO DI INGEGNERIA

DELLA CONOSCENZA A.A. 2021-2022

Obesity Prediction:

Analisi e predizione di condizioni di obesità in base alle abitudini alimentari e alle condizioni fisiche: base di conoscenza, ontologia, classificazione e previsione

ANTONIO SEQUENZA (728325) [a.sequenza@studenti.uniba.it](mailto:a.sequenza@studenti.uniba.it)

ZAGARIA RUGGIERO (727707) [r.zagaria15@studenti.uniba.it](mailto:r.zagaria15@studenti.uniba.it)

SOMMARIO

[INTRODUZIONE 2](#_Toc113994819)

[OBIETTIVO 2](#_Toc113994820)

[STRUMENTI 3](#_Toc113994821)

[DATASET 4](#_Toc113994822)

[DOMINIO 4](#_Toc113994823)

[KNOWLEDGE BASE 5](#_Toc113994824)

[GESTORE BASE DI CONOSCENZA 6](#_Toc113994825)

[ONTOLOGIA 7](#_Toc113994826)

[GESTORE ONTOLOGIA 11](#_Toc113994827)

[MACHINE LEARNING 13](#_Toc113994828)

[DATASET 13](#_Toc113994829)

[PRE-PROCESSING 15](#_Toc113994830)

[APPRENDIMENTO SUPERVISIONATO 16](#_Toc113994831)

[CLASSIFICAZIONE 18](#_Toc113994832)

[RANDOM FOREST (1° task) 18](#_Toc113994833)

[RANDOM FOREST (2° task) 20](#_Toc113994834)

[K-NEAREST NEIGHBORS (1° task) 22](#_Toc113994835)

[K-NEAREST NEIGHBORS (2° task) 24](#_Toc113994836)

[SUPPORT VECTOR MACHINE (1° task) 26](#_Toc113994837)

[SUPPORT VECTOR MACHINE (2° task) 28](#_Toc113994838)

[RISULTATI FINALI 30](#_Toc113994839)

[INTERFACCIA 31](#_Toc113994840)

[CONCLUSIONI 32](#_Toc113994841)

# INTRODUZIONE

## OBIETTIVO

Il progetto nasce con l’obiettivo di predire una condizione di obesità nelle persone in base al loro stile di vita, ed in particolare in base alle loro abitudini alimentari e alle loro condizioni fisiche. Per prima cosa, il dominio di interesse è stato rappresentato tramite knowledge base e ontologia, le quali sono state successivamente interrogate per estrapolare le informazioni. Inoltre, sono state utilizzate tecniche di apprendimento supervisionato, nello specifico la classificazione binaria, e la valutazione di quest’ultima. I classificatori utilizzati sono i seguenti: Random Forest, K-Nearest Neighbors, ed SVM. Infine, dopo aver salvato uno dei modelli, è stata creata un’interfaccia che prevede l’obesità dell’utente in base ai suoi dati, sfruttando l’algoritmo salvato.

## STRUMENTI

Gli strumenti utilizzati per la realizzazione di questo progetto sono i seguenti:

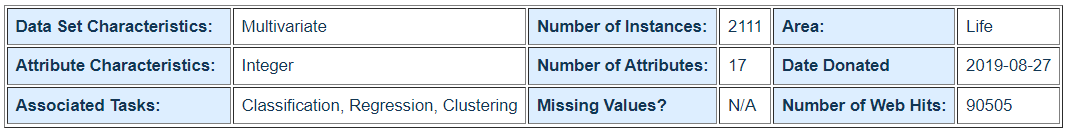
* **PyCharm**:IDE nel quale è stata implementata la maggior parte del progetto, con linguaggio di programmazione Python (<https://www.jetbrains.com/pycharm/>)
* **Protégé**: software per la creazione e modellazione delle ontologie (<https://protege.stanford.edu/>)
* **SWI-Prolog**: strumento utilizzato per la creazione della base di conoscenza in Prolog (<https://www.swi-prolog.org/>)

Le librerie utilizzate in Python sono le seguenti:

* **PySWIP:** libreria che consente di integrare SWI-Prolog all’interno di Python (<https://pypi.org/project/pyswip/0.2.2/>).
* **OwlReady2**: libreria usata per gestire ontologie all’interno di Python(<https://pypi.org/project/Owlready2/>)
* **Pandas**: libreria utilizzata per la manipolazione e l’analisi dei Dataset (<https://pandas.pydata.org/>)
* **Matplotlib**: libreria utilizzata per la creazione e visualizzazione dei grafici (<https://matplotlib.org/>)
* **Seaborn**: libreria utilizzata per la creazione dei e la visualizzazione dei dati (<https://seaborn.pydata.org/>)
* **Sklearn**: libreria utilizzata per il pre-processing e apprendimento supervisionato (<https://scikit-learn.org/stable/>)
* **Pickle5 :** libreria usata per serializzare e deserializzare una struttura di oggetti (<https://pypi.org/project/pickle5/>)

## DATASET

Inizialmente il Dataset utilizzato, reperibile attraverso il link <https://archive.ics.uci.edu/ml/datasets/Estimation+of+obesity+levels+based+on+eating+habits+and+physical+condition>+, ha una struttura composta da 2111 esempi e 17 features:



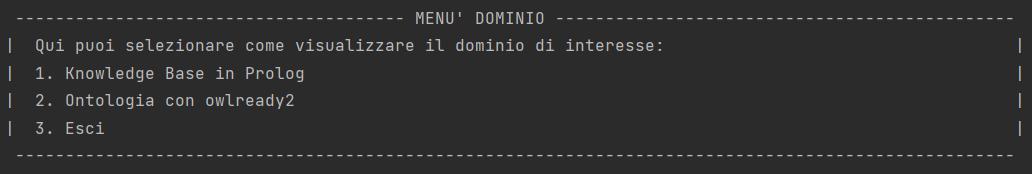
Tali attributi vengono etichettati con la variabile di classe ‘NObesity’ (Obesity Level) , che permette di classificare i dati utilizzando i seguenti valori:

* Insufficient\_Weight;
* Normal\_weight;
* Overweight\_Level\_I;
* Overweight\_Level\_II;
* Obesity\_Type\_I;
* Obesity\_Type\_II;
* Obesity\_Type\_III;

# DOMINIO

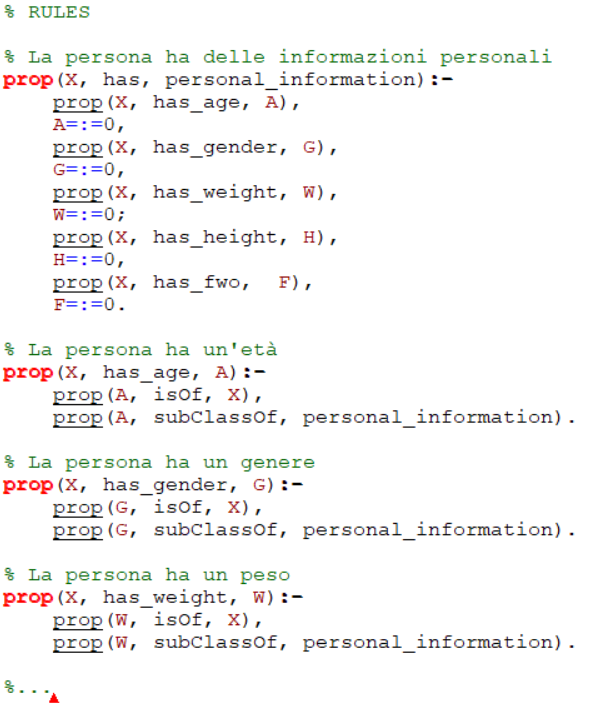
Sono state create due rappresentazioni diverse del dominio di interesse. La prima attraverso la base di conoscenza sviluppata in SWI-Prolog e successivamente importata in PyCharm tramite la libreria PySWIP, mentre la seconda tramite l’ontologia sviluppata in Protégé e importata in PyCharm tramite la libreria OWLReady2.

Per poter visualizzare entrambe le rappresentazioni del dominio basta avviare lo script denominato **domain.py**, e, attraverso un menù è possibile aprire i menù relativi alla KB e all’Ontologia:



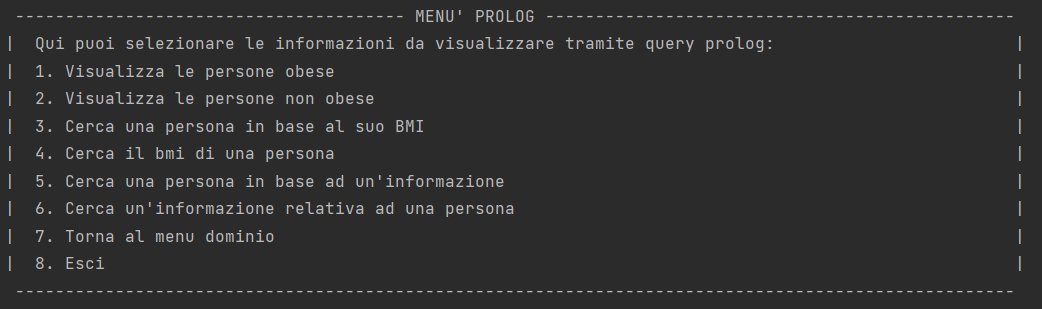
## KNOWLEDGE BASE

Una base di conoscenza è un insieme di regole e fatti validi per il dominio di interesse. A tal proposito, è stata costruita una base di conoscenza in Prolog, composta da 17 regole, e poi importata in Python mediante la libreria PySWIP, permettendo la l’interrogazione e la verifica sia di proprietà di informazioni personali come l’altezza, il peso, il genere, e sia proprietà più complesse come le abitudini alimentari e le condizioni fisiche della persona:

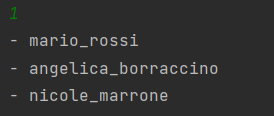


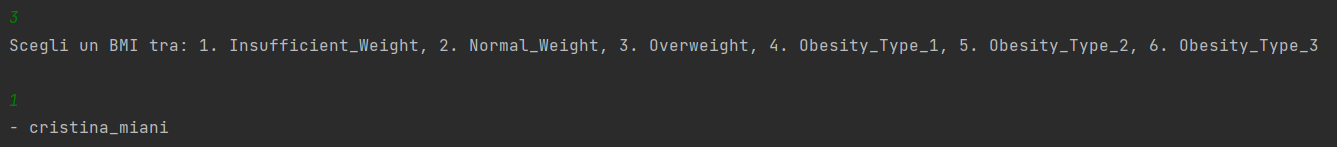
### GESTORE BASE DI CONOSCENZA

Selezionando dunque dal menù dominio l’opzione 1, si apre un ulteriore menù che permette l’interrogazione della base di conoscenza tramite apposite query in PySWIP sviluppate all’interno del file denominato **KBManager.py**:

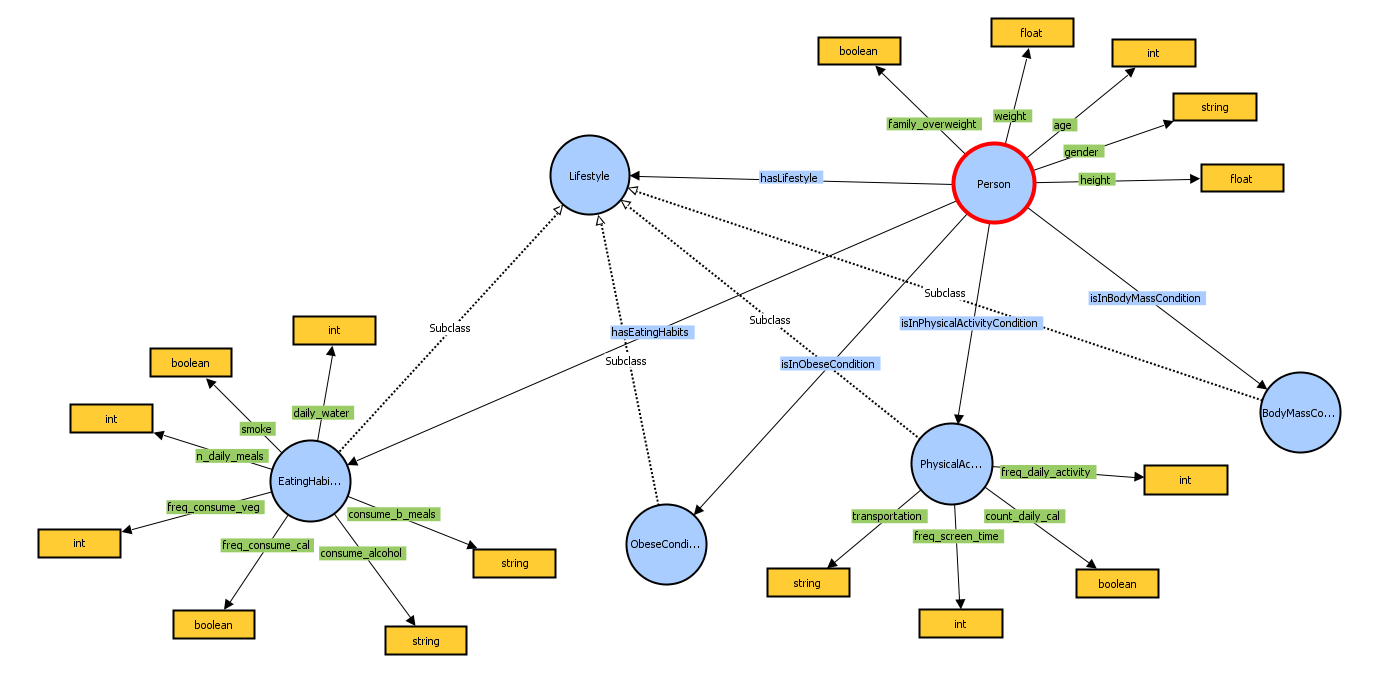


Attraverso il precedente menù si possono essere effettuate ad esempio delle query che permettono la visualizzazione delle persone obese:

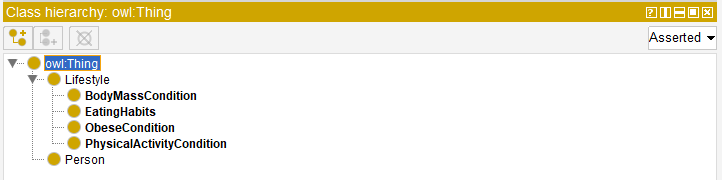


Oppure si possono trovare le persone appartenente una determinata categoria di peso:

## ONTOLOGIA

Un’ontologia è una specifica del significato dei simboli usato in un sistema d'informazione. Per poter crearla è stato utilizzato il programma Protégé. Sono state create quindi varie classi e sottoclassi, e sono state aggiunte proprietà di oggetto e proprietà di dati con un dominio e un relativo range. Inoltre è stato possibile interrogare l’ontologia effettuando delle query usando l’estensione *DL Query*. Di seguito viene mostrata l’ontologia per intero visualizzata tramite il plugin di Protégé chiamato VOWL:

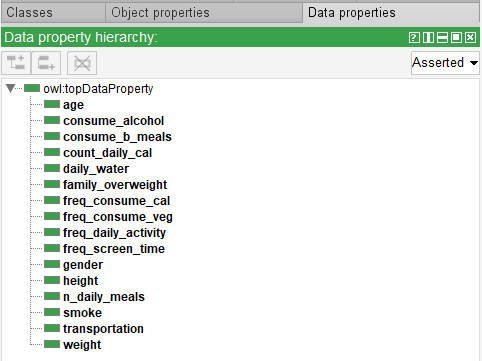
Dunque l’ontologia è composta essenzialmente dalle seguenti classi, che rappresentano i concetti essenziali:



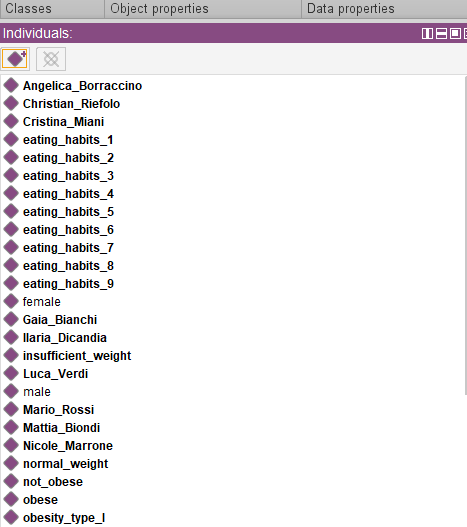
Si riportano inoltre le proprietà di oggetto, ovvero delle proprietà in grado di mettere in relazione due individui di stessa classe oppure diversa:

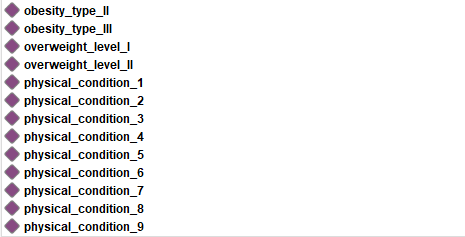


Oltre a quest’ultime, sono state anche definite le proprietà sui dati che mettono in relazione un individuo con il suo valore primitivo (int, string, etc), come mostrato nella seguente figura:



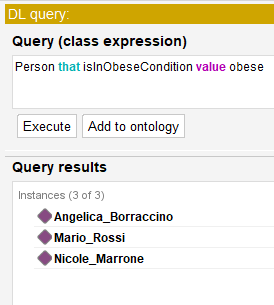
Successivamente è stata creata una serie di istanze per ogni entità. Ad esempio abbiamo le istanze relative alle persone, alle condizioni fisiche, alle abitudini alimentari ecc:





Infine, dopo aver creato gli individui, si è passati a formulare delle query tramite il plug-in *Query DL* per interrogare l’ontologia e visualizzare la correttezza degli output:

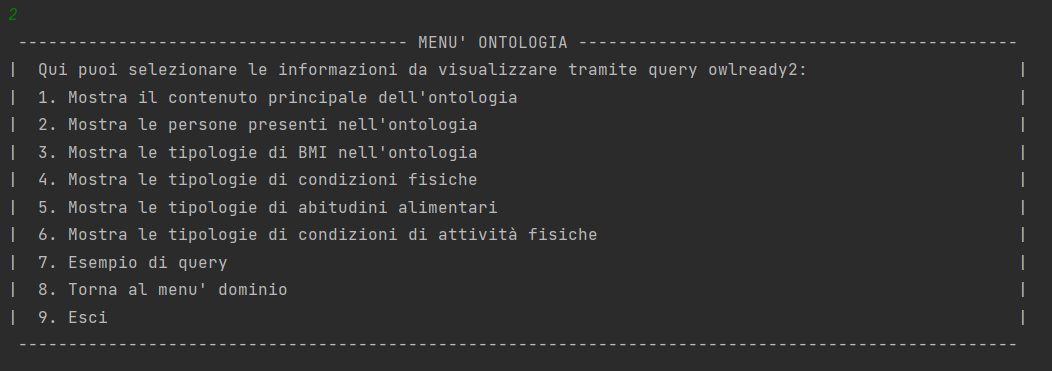
Immagine che contiene testo

Descrizione generata automaticamente

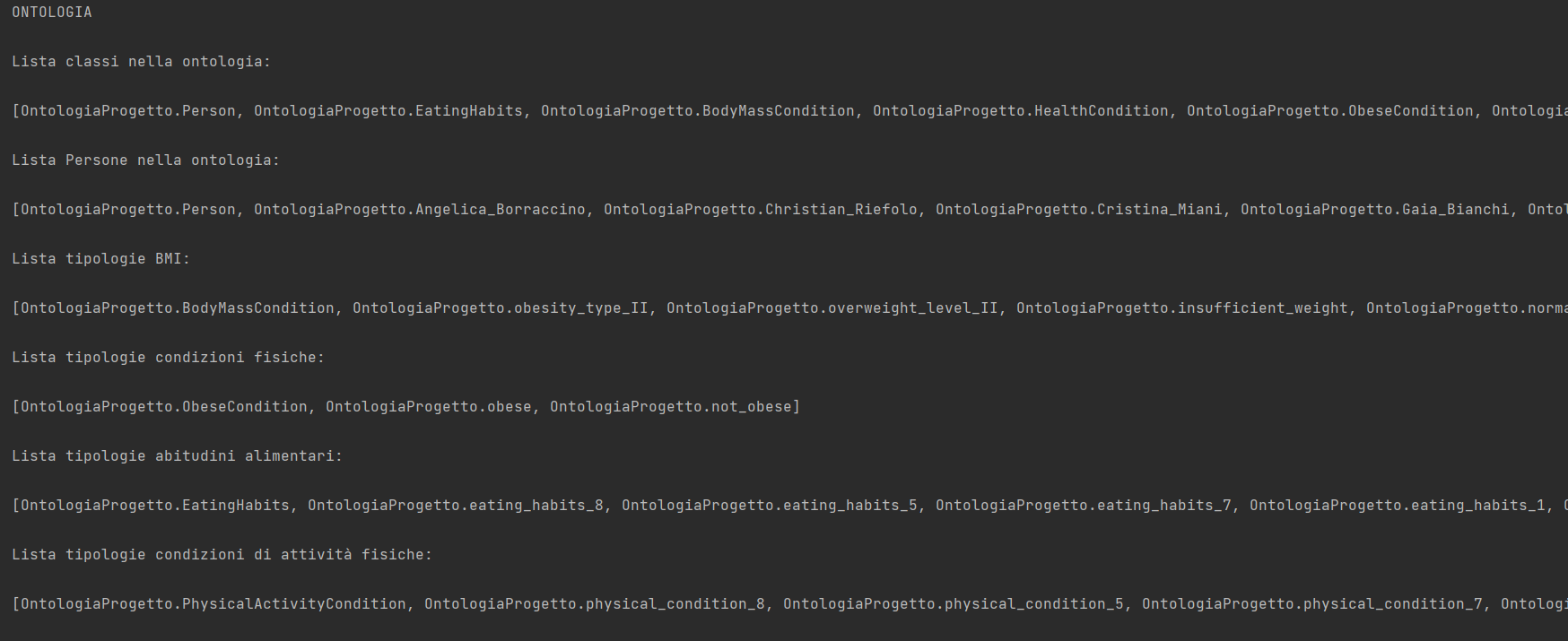
* Figura 1 *La prima query restituisce le persone che rientrano nella categoria di tipo ‘Obese’, mentre la seconda restituisce le persone che hanno delle abitudini alimentari di tipo 4 oppure le persone che si trovano in una condizione fisica di tipo 1*

### GESTORE ONTOLOGIA

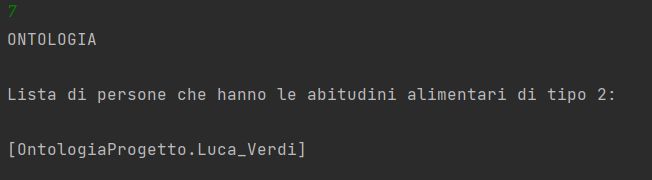
Nell’ambito del dominio studiato, l’ontologia è stata sviluppata in Protégé, e successivamente importata in Python tramite la libreria OWLReady2. Selezionando dal menù dominio l’opzione 2 si può accedere al menù relativo all’ontologia, che permette *un’ulteriore* interrogazione di quest’ultima tramite query sviluppate all’interno del file denominato **ontologyManager.py**:



Attraverso il precedente menù possono essere effettuate ad esempio delle query che permettono la visualizzazione delle informazioni contenute all’interno dell’ontologia.

Il conteuto dell’ontologia è il seguente:

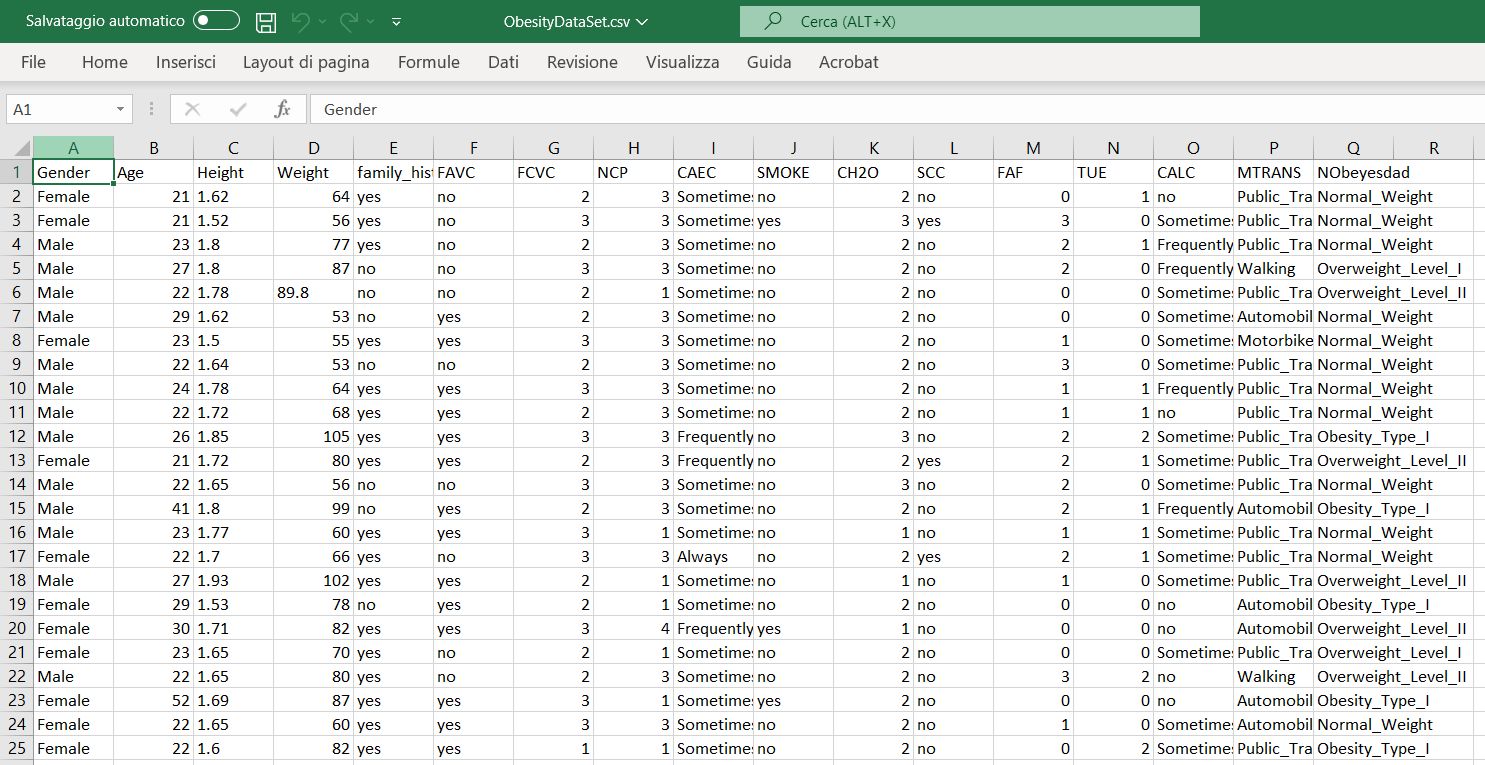
Un altro esempio di query è il seguente, con la quale è possibile stampare la lista di persone che hanno abitudini alimentari di tipo 2;



# MACHINE LEARNING

## DATASET

Il dataset inizialmente si presenta nel seguente modo in Excel:



Le Features sono dunque le seguenti:

Features relative alle informazioni personali:

* **Gender**: identifica il sesso [Male/ Female];
* **Age**: identifica l’età [14 – 61];
* **Height**: identifica l’altezza [1.45 - 1.98];
* **Weight**: identifica il peso [39 – 173];
* **Family\_history**: storico utilizzato per indicare familiari che soffrono o hanno sofferto per sovrappeso [yes /no];

Features relative alle abitudini alimentari:

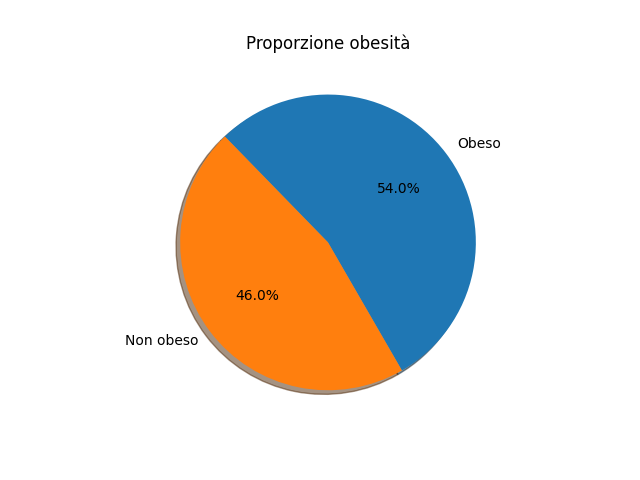
* **FAVC**: indica il consumo frequente di cibi ipercalorici [yes/ no];
* **FCVC**: indica la frequenza di consumo di verdure [Never/ Sometimes/ Always];
* **NCP**: indica il numero di pasti principali effettuati di solito [Between 1 or 2/ Three/ More than three];
* **CAEC**: indica, in termini di frequenza, il consumo di ulteriore cibo tra un pasto e l’altro [Never/ Sometimes/ Frequently/ Always];
* **SMOKE**: indica se il paziente fuma o meno [yes/ no];
* **CH20**: indica la frequenza del consumo di acqua giornaliero [Less than a liter/Between 1 and 2 L/ More than 2 L];
* **CALC**: indica la frequenza del consumo di alcohol [no/ Sometimes/ Frequently/ Always];

Features relative alla condizione fisica:

* **SCC**: indica se viene effettuato un monitoraggio del consumo calorico [yes/ no];
* **FAF**: indica la frequenza di attività fisica svolta [Never/ 1 or 2 days/ 2 or 4 days/ 4 or 5 days];
* **TUE**: indica la frequenza dell’uso di dispositivi elettronici [0–2 hours/ 3–5 hours/ More than 5 hours];
* **MTRANS**: indica i mezzi di trasporto utilizzati [Public transportation, Motorbike, Bike, Walking];

## PRE-PROCESSING

Il Dataset è composto inizialmente da 17 features e 2111 istanze. Dopo aver verificato l’eventuale presenza di valori nulli, sono state rinominate alcune colonne per migliorare la leggibilità, e successivamente sono stati convertiti i valori delle features in valori numerici per poter effettuare la classificazione. Inoltre, è stato riportato il pie chart per visualizzare la distribuzione della classe target:



Come si evince dal pie chart, la distribuzione delle classi target è abbastanza bilanciata, dunque non c’è bisogno di effettuare quest’ultima operazione.

## APPRENDIMENTO SUPERVISIONATO

Con l’Apprendimento Supervisionato si cerca di costruire un modello partendo da dei dati di addestramento etichettati, con i quali si cerca di fare previsioni su dati non disponibili o futuri. Infatti, con il termine ‘Supervisione’ si intende che nell’insieme dei campioni, i segnali di output desiderati sono già noti poiché precedentemente etichettati. Nel contesto di interesse, l’obiettivo sarà quello di addestrare i classificatori per poter predire la categoria di obesità appartenente alle persone. Infatti, le classi iniziali erano le seguenti:

* Insufficient\_Weight;
* Normal\_Weight;
* Overweight\_Level\_I;
* Overweight\_Level\_II;
* Obesity\_Type\_I;
* Obesity\_Type\_II;
* Obesity\_Type\_III;

Le suddette classi sono state riunite in due grandi classi, sulle quali poi è stata applicata la *classificazione binaria*:

* **Not\_obese** (Insufficient\_Weight, Normal\_Weight, Overweight\_Level\_I, Overweight\_Level\_II)
* **Obese** (Obesity\_Type\_I, Obesity\_Type\_II, Obesity\_Type\_III)

Sono stati implementati i seguenti classificatori addestrati sul Dataset:

* **Random Forest**
* **K-Nearest Neighbors**
* **SVM**

In particolare, una volta modificato il dataset, sono stati effettuati due task differenti utilizzando i suddetti classificatori per poter confrontare i risultati ottenuti:

* Split in train e test set pari a 75%-25%
* Split in train e test set pari a 60%-40%

Inoltre, per tutti i classificatori è stato effettuato il tuning degli Iperparametri tramite GridSearchCV con una stratified Kfold cross-validation (default) per poter sfruttare al massimo le capacità dei classificatori in questione. Infine, per ogni algoritmo di apprendimento supervisionato sono state prodotte le seguenti misure e i seguenti grafici:

* **Accuratezza training e test**
* **Report di classificazione**
* **Matrice di confusione**
* **Curva Precision-Recall**

## CLASSIFICAZIONE

### RANDOM FOREST (1° task)

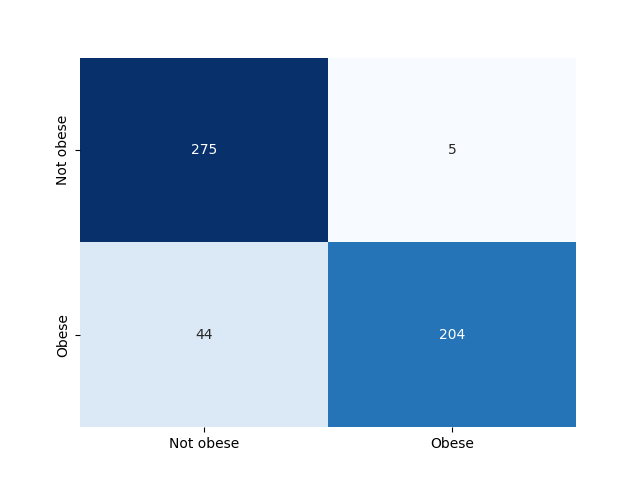
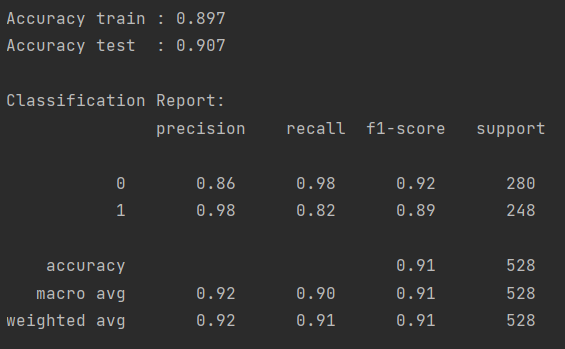
Il Random Forest è un modello costituito da molti alberi di decisione, ognuno dei quali fornisce una predizione. Esse vengono poi, combinate allo scopo di ottenere una previsione complessiva della foresta per un dato esempio.

Sono stati selezionati i seguenti parametri per il classificatore Random Forest:

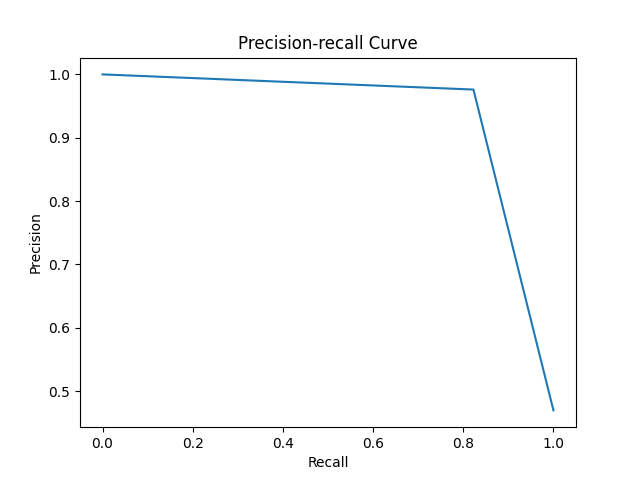
* *n\_estimator=5*
* *max\_depth=2*
* *random\_state=0*

Il numero di stimatori (alberi) è dunque pari a 5, e si è deciso di selezionare una profondità iniziale bassa pari a 2.

Le misure di valutazione e i grafici per il Random Forest sono invece i seguenti:



Come si evince dal report l’accuratezza è del 91%, dunque un risultato abbastanza buono per questo primo task, soprattutto per quanto riguarda la precision per la classe 1 che riguarda gli obesi. Anche la matrice di confusione evidenzia dei buoni risultati ma con un numero un po’ alto di falsi negativi (44), quindi li considera come non obesi ma in realtà sono effettivamente obesi.

Infine, la curva precision-recall evidenzia un buon rapporto tra le due metriche. Dunque è evidente che nella classificazione non sono riscontrati particolari problemi e i risultati sono abbastanza buoni.

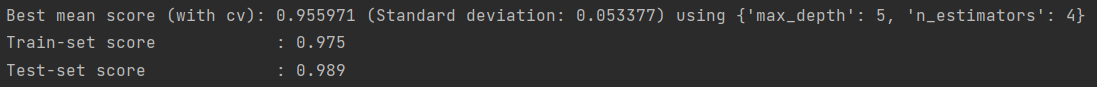
Successivamente sono stati selezionati i seguenti Iperparametri ed è stata applicato la GridSearchCV con stratified Kfold cross-validation con split di valore 10:

*parameters = {'n\_estimators': (2, 3, 4, 5, 10, 100),*

*'max\_depth': (1, 2, 3, 4, 5)*

*}*

Di seguito vengono riportati i risultati del tuning degli Iperparametri assieme ai migliori Iperparametri:

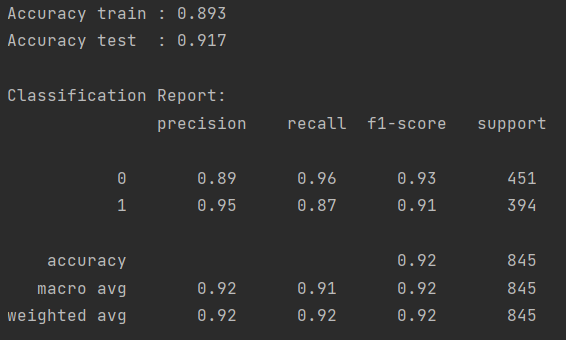
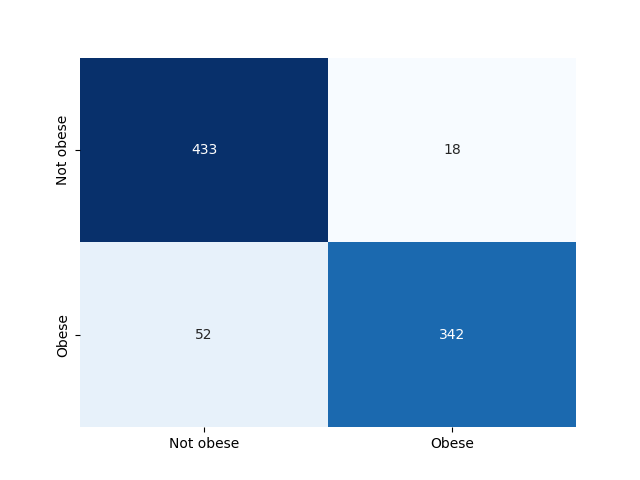


Come si può notare dai risultati della GridSearchCV, nonostante sia stata scelta una profondità massima ottimale pari a 5 rispetto alla precedente (2) e un numero di stimatori ottimali ridotto di 1, c’è stato un buon incremento dell’accuratezza (media), così come anche gli score riguardanti il train-set e test-set.

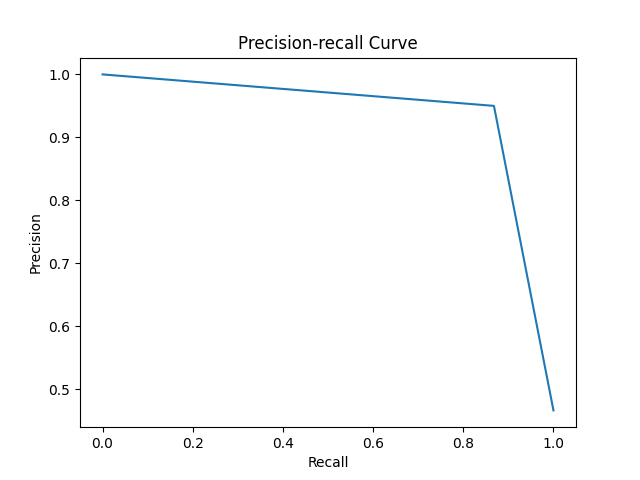
### RANDOM FOREST (2° task)

Dopo aver effettuato il secondo split (60%-40%) è stato applicato il classificatore Random Forest per una seconda valutazione. Le metriche utilizzate sono identiche alle precedenti:

* *n\_estimator=5*
* *max\_depth=2*
* *random\_state=0*

Le misure di valutazione e i grafici per il Random Forest sono invece i seguenti:

Come si evince da questa seconda valutazione, l’accuratezza del classificatore è aumentata dell’1% rispetto alla prima. Anche riguardo la matrice di confusione non sono stati rilevati grandi cambiamenti, ma il numero dei falsi positivi è comunque alto come la precedente.



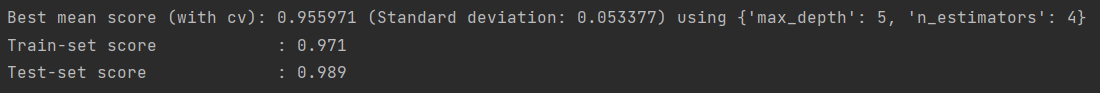
Riguardo infine la curva precision-recall, considerando l’andamento delle precedenti metriche, è logico aspettarsi una curva simile alla precedente.

Successivamente sono stati selezionati i seguenti Iperparametri ed è stata applicato la GridSearchCV con stratified Kfold cross-validation con split di valore 10:

*parameters = {'n\_estimators': (2, 3, 4, 5, 10, 100),*

*'max\_depth': (1, 2, 3, 4, 5)*

*}*

Dopo aver effettuato anche qui il tuning degli Iperparametri i risultati sono stati i seguenti: 

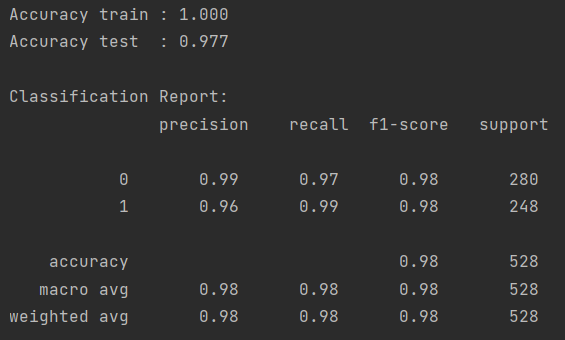
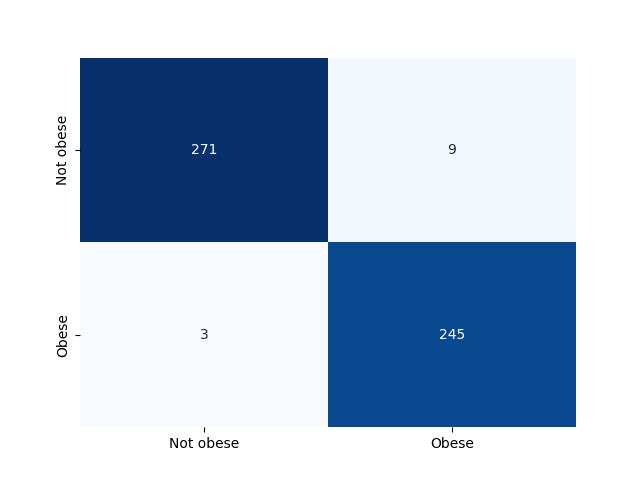
Come si nota lo score è sicuramente più alto del pre-tuning (2° task) e gli Iperparametri sono anche uguali a quelli 1° task. Dunque da questo 2° task effettuato su uno split differente si evince che il classificatore ha ottenuto dei risultati leggermente migliori ma comunque simili.

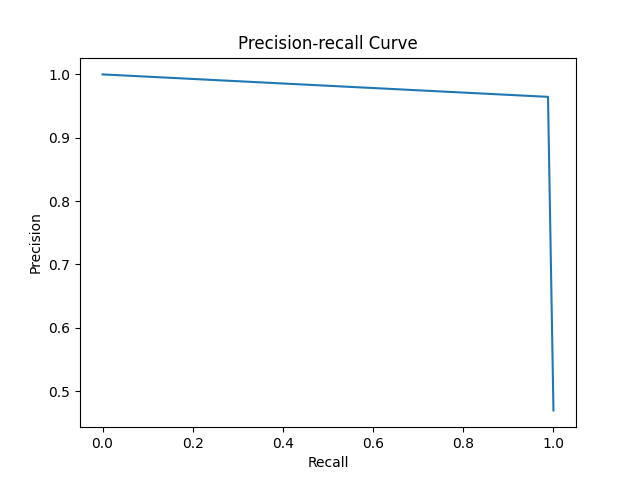
### K-NEAREST NEIGHBORS (1° task)

Il K-Nearest Neighbors è un algoritmo di apprendimento supervisionato che consiste nell’individuare i k esempi più vicini a quello che si intende classificare, a quest’ultimo viene quindi attribuita la categoria “più ricorrente” tra i k esempi più vicini. Sono stati scelti i seguenti parametri per il classificatore KNN:

* *n\_neighbors=5,*
* *weights=’distance’*
* *metric=’minkowski’*

E’ stato selezionato dunque un k iniziale pari a 5, e come peso una distanza, che è quella di mikowski, per non penalizzare i dati, non essendo il dataset normalizzato. Le misure di valutazione e i grafici per il KNN sono invece i seguenti:



Come si evince dal report e dalla matrice di confusione, i risultati sono davvero ottimi (98% di accuratezza). Anche precision e recall sono ottimi per entrambe le classi. L’unico problema potrebbe essere la presenza di overfitting, infatti come si nota l’accuratezza sui dati di training è maggiore rispetto a quella sui dati di test, e questo indica che il modello, essendo anche molto forte, si è adattato alla perfezione sui dati di training.

Successivamente sono stati selezionati i seguenti Iperparametri ed è stato applicato la GridSearchCV con stratified Kfold cross-validation con split di valore 10:

*parameters = {'n\_neighbors': (2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17,*

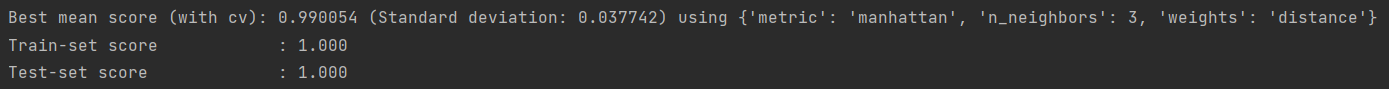
*18, 19, 20),*

*'metric': ['euclidean', 'manhattan', 'minkowski'],*

*'weights': ['uniform', 'distance']*

*}*

Di seguito vengono riportati i risultati del tuning degli Iperparametri assieme ai migliori Iperparametri:



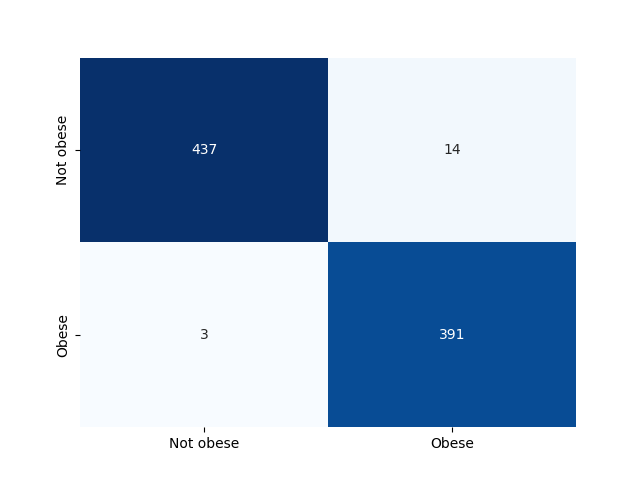
Si evince dunque che il problema dell’overfitting è stato risolto alla perfezione. Lo score migliore è del 99% e l’accuratezza sui due set di esempi è identica. Inoltre, la metrica scelta è questa volta quella di manhattan (che generalizza quella euclidea e di minkowski) anziché quella di minkowski iniziale, e con un k pari a 3, rispetto ai 5 iniziali.

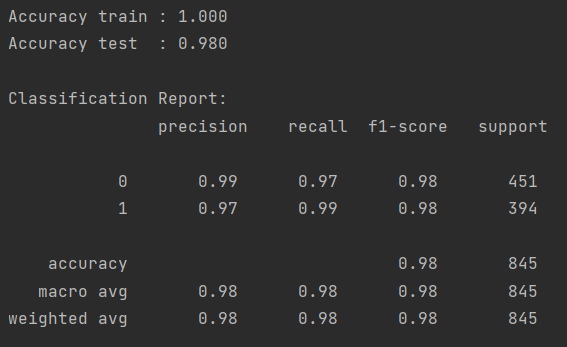
### K-NEAREST NEIGHBORS (2° task)

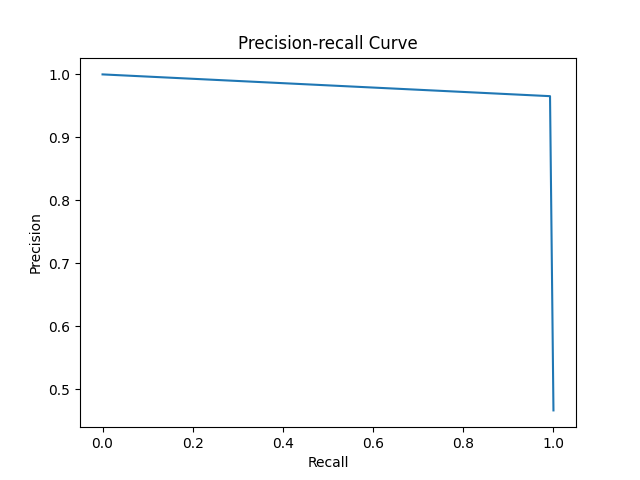
Dopo aver effettuato il secondo split (60%-40%) è stato applicato il classificatore KNN per una seconda valutazione. Le metriche utilizzate sono identiche alle precedenti:

* *n\_neighbors=5,*
* *weights=’distance’*
* *metric=’minkowski’*

Le misure di valutazione e i grafici per il KNN sono invece i seguenti:







Si evince dunque che l’accuratezza è molto simile a quella iniziale, dunque i risultati sono ottimi per tutte le metriche, ma soprattutto che il problema dell’overfitting potrebbe ripresentarsi anche in questo caso.

Anche la curva di precision-recall si presenta simile alla precedente, dunque con risultati ottimi.

Successivamente sono stati selezionati i seguenti Iperparametri ed è stato applicato la GridSearchCV con stratified Kfold cross-validation con split di valore 10:

*parameters = {'n\_neighbors': (2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17,*

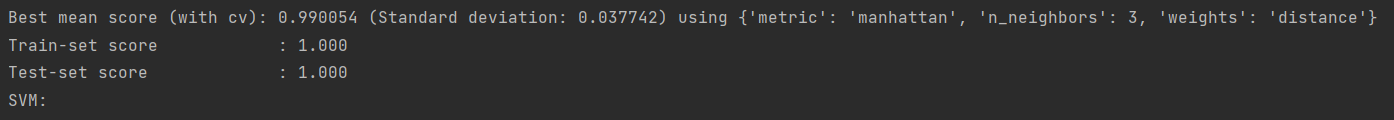
*18, 19, 20),*

*'metric': ['euclidean', 'manhattan', 'minkowski'],*

*'weights': ['uniform', 'distance']*

*}*

Dopo aver effettuato anche qui il tuning degli Iperparametri i risultati sono stati i seguenti:



Come si può notare, con GridSearchCV non ci sono stati cambiamenti, ed è stato risolto ancora una volta l’overfitting. Non c’è stata neanche una variazione del k. Dunque, similmente al 1° task di KNN, i risultati sono ottimi anche con uno split differente.

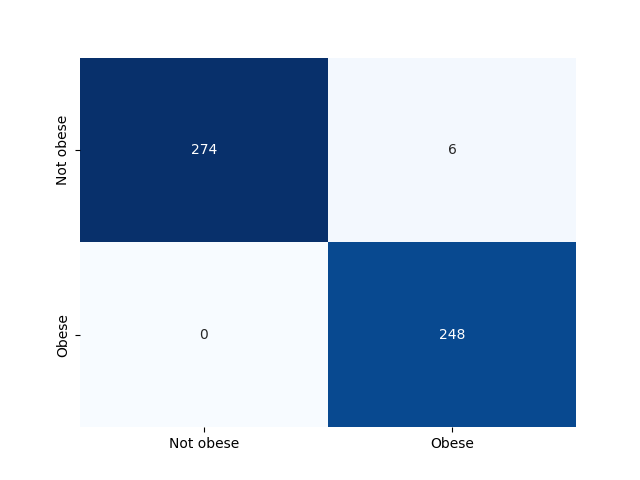
### SUPPORT VECTOR MACHINE (1° task)

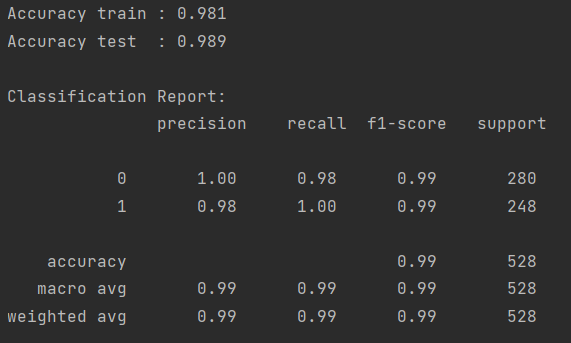
Un modello SVM è una rappresentazione degli esempi come punti nello spazio, mappati in modo tale che gli esempi appartenenti alle due diverse categorie siano chiaramente separati da uno spazio il più possibile ampio. I nuovi esempi sono quindi mappati nello stesso spazio e la predizione della categoria alla quale appartengono viene fatta sulla base del lato nel quale ricade.

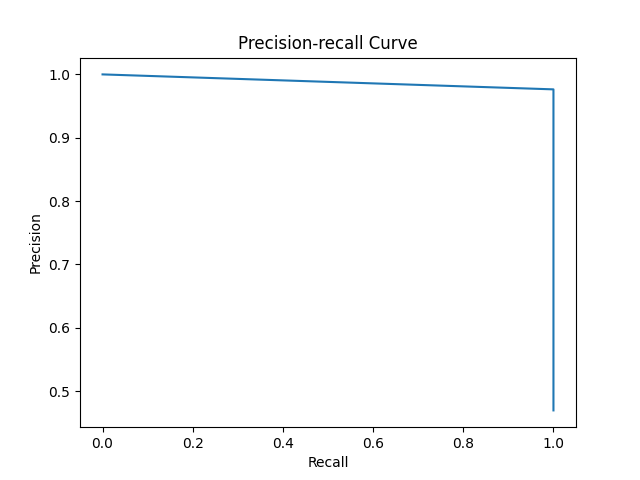
Sono stati selezionati i seguenti parametri per il classificatore SVM:

* *kernel='linear'*
* *C=1.0 (default)*

E’ stato scelto inizialmente un kernel semplice e lineare, con un C (parametro di regolarizzazione) di default.

Le misure di valutazione e i grafici per il SVM sono dunque i seguenti:





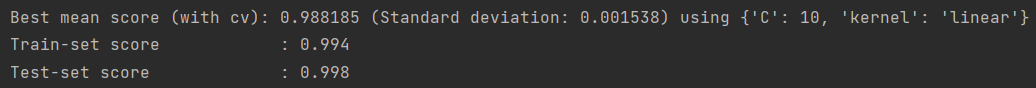
Come si evince dal report e dalle matrici di confusione, è evidente che nella classificazione non sono riscontrati problemi e i risultati sono ottimi. Si ha infatti un’accuratezza del 99%, così come per precision e recall i sono stati ottenuti risultati ottimi. Guardando la matrice di confusione infatti si può notare un numero vicinissimo allo 0 sia per i falsi positivi che per i falsi negativi. Infine, anche la curva di precision-recall evidenzia ottimi risultati.

Successivamente sono stati selezionati i seguenti Iperparametri ed è stato applicato la GridSearchCV con stratified Kfold cross-validation con split di valore 10:

*parameters = {'kernel': ['linear', 'poly', 'rbf', 'sigmoid'],*

*'C': [50, 10, 1.0, 0.1, 0.01]}*

Di seguito vengono riportati i risultati del tuning degli Iperparametri assieme ai migliori Iperparametri:

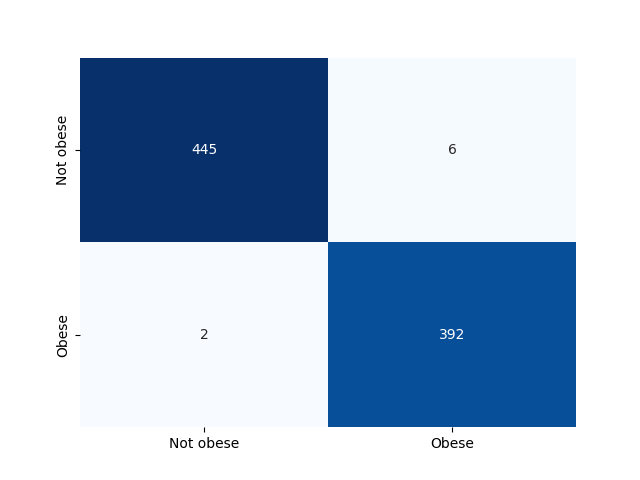


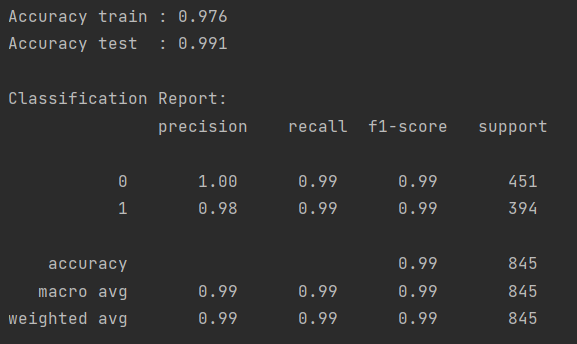
Come si può notare, i risultati sono simili, cambia solo il fattore di regolarizzazione (10) che applica una maggiore penalizzazione al modello. Si sono ottenuti dunque ottimi risultati anche riguardo agli score sui due set di dati.

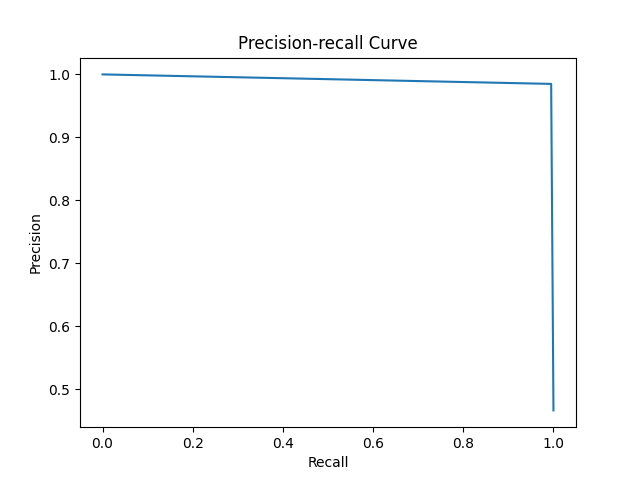
### SUPPORT VECTOR MACHINE (2° task)

Dopo aver effettuato il secondo split (60%-40%) è stato applicato il classificatore SVM per una seconda valutazione. Le metriche utilizzate sono identiche alle precedenti:

* *kernel='linear'*
* *C=1.0 (default)*

Le misure di valutazione e i grafici per l’SVM sono invece i seguenti:



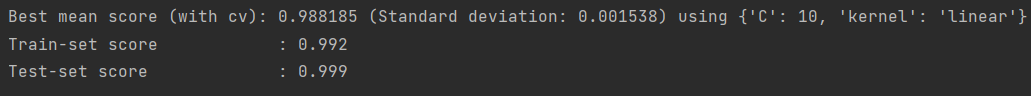
Rispetto alla precedente valutazione non c’è stata alcuna variazione significativa, e i risultati sono anche qui ottimi, con un’accuratezza pari al 99%. Naturalmente anche per quanto riguarda la matrice di confusione e la curva precision-recall i risultati sono molto simili a quelli del 1° task, e dunque quasi perfetti.

Successivamente sono stati selezionati i seguenti Iperparametri ed è stato applicato la GridSearchCV con stratified Kfold cross-validation con split di valore 10:

*parameters = {'kernel': ['linear', 'poly', 'rbf', 'sigmoid'],*

*'C': [50, 10, 1.0, 0.1, 0.01]}*

Di seguito vengono riportati i risultati del tuning degli Iperparametri assieme ai migliori Iperparametri:



Come per il primo task, sono stati selezionati come Iperparametri un C pari a 10 e un kernel di tipo lineare. Infatti, il modello ha lavorato quasi perfettamente su entrambi i set di dati e in entrambi i task, con risultati pressoché identici.

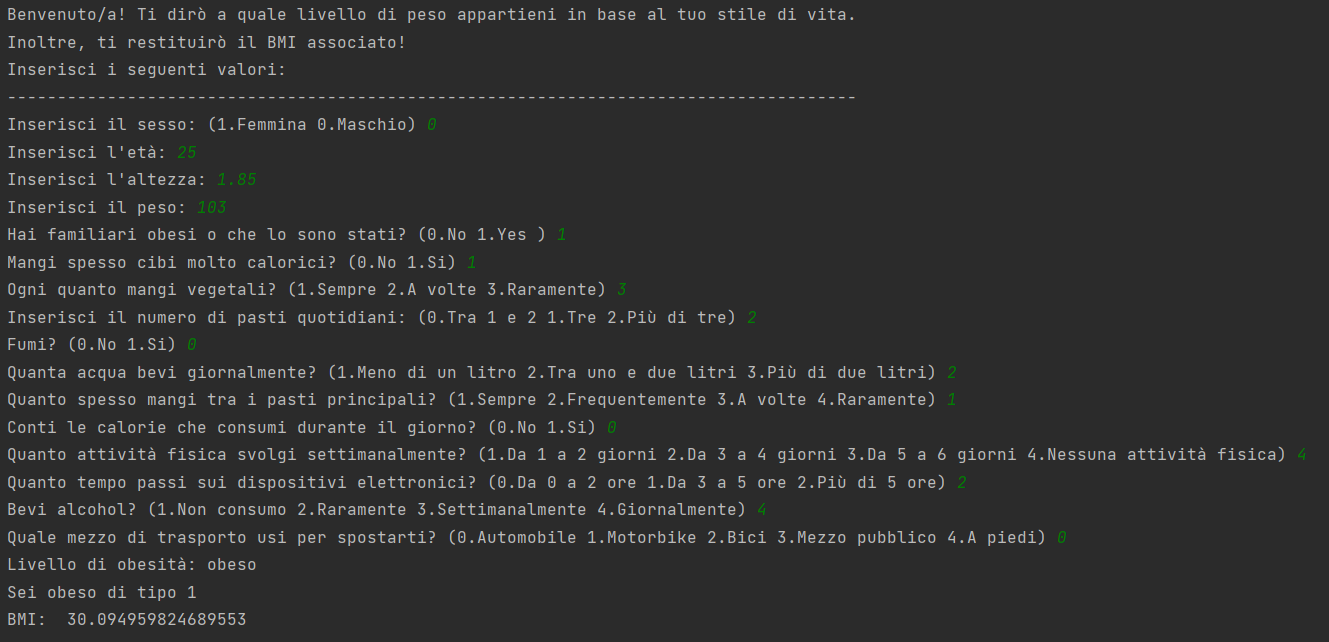
### RISULTATI FINALI

I risultati finali che vengono fuori dai due differenti task di classificazione binaria sono i seguenti:

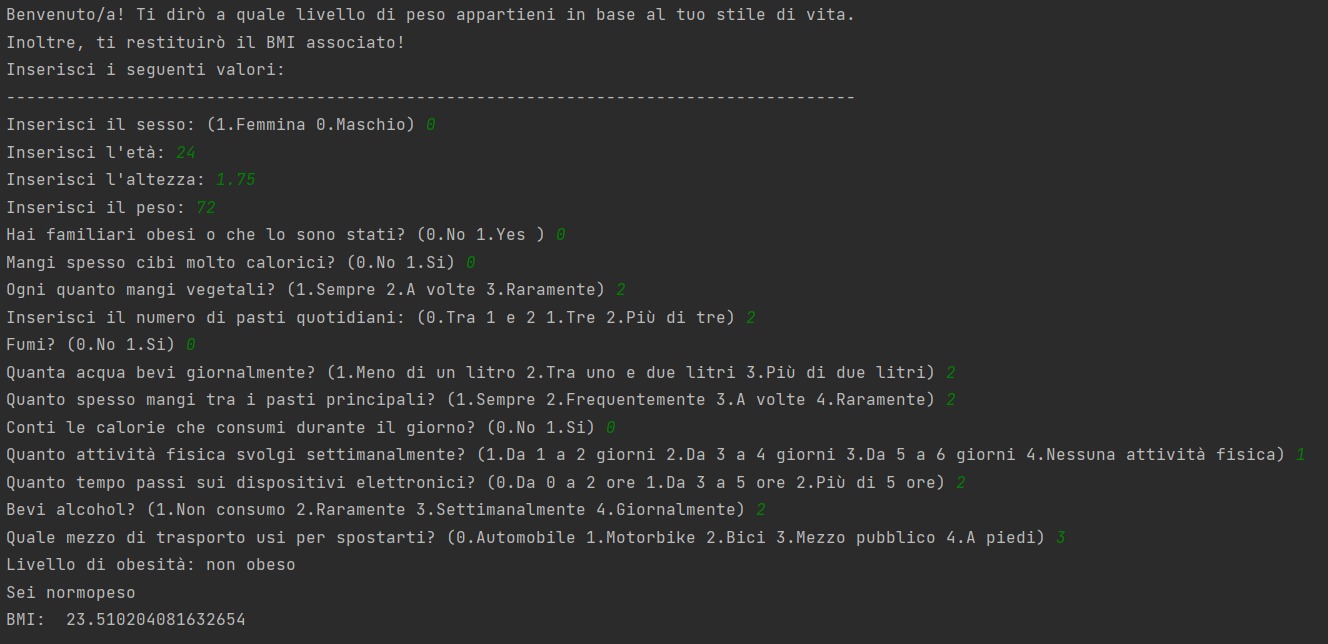
|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| CLASSIFICATORI | PRIMO TASK  75%-25% | | SECONDO TASK  60%-40% | |
|  | **ACCURATEZZA (PRE-TUNING)** | **ACCURATEZZA (POST-TUNING)** | **ACCURATEZZA (PRE-TUNING)** | **ACCURATEZZA (POST-TUNING)** |
| RANDOM FOREST | 91% | 95% | 92% | 95% |
| KNN | 98% | 99% | 98% | 99% |
| SVM | 99% | 99% | 99% | 99% |

# INTERFACCIA

Dopo aver effettuato l’addestrato dei vari classificatori, è stato esportato il modello SVM (che ha l’accuratezza maggiore), e re-implementato all’interno dell’interfaccia, il cui scopo è predire una condizione di obesità in basi ai dati inseriti dall’utente. Non avendo effettuato la classificazione multi-classe, è stata implementata anche una funzione per il calcolo delle fasce del BMI. Di seguito sono riportati due esempi:



* *Predizione per un individuo obeso*

**

* *Predizione per un individuo non obeso*

# CONCLUSIONI

Il progetto è stato un banco di prova importante, nel quale si sono presentate numerose difficoltà riguardanti principalmente l’utilizzo di nuovi strumenti come Python, Protégé e Prolog. Nonostante ciò, sono stati raggiunti gli obiettivi principali, e si è tratto il meglio da questa esperienza. Inoltre, questo progetto nasce principalmente come task di classificazione ma potrebbe essere anche esteso come task di regressione. E’ possibile poi estendere il progetto effettuando la classificazione multi-classe per predire le singole classi relative al BMI.