INGEGNERIA DELLA CONOSCENZA

CHRONIC KIDNEY DISEASE

Autori	Matricole
Massafra Francesco	*722081*
Ricchiuti Alberto	*724583*
Scavo Beatrice	*716480*

Repository GitHub: https://github.com/kekko4000/CHRONIC-KIDNEY-DISEASE-project

Introduzione

L'obiettivo principale del progetto è fornire supporto medico sia nella diagnosi preventiva di insufficienze renali, aiutando lo specialista anche a distinguere se si tratti di un disturbo cronico o meno, sia nell'individuare quelle che sono le caratteristiche comuni tra i pazienti affetti da tali disturbi. Così facendo è possibile capire quanto gli individui presenti siano simili tra loro e trovare, quindi, eventuali gruppi distinti di persone all'interno dello stesso campione di dati che possano, peer esempio, reagire nello stesso modo alla somministrazione di determinati farmaci.

Sono state utilizzati sia algoritmi di apprendimento supervisionato che di apprendimento non supervisionato.

Informazioni tecniche

Il progetto è stato realizzato in python.

Ambiente di sviluppo: PyCharm

Librerie principali utilizzate:

- sklearn -- per gli algoritmi di apprendimento e la loro valutazione;
- pandas e numpy -- per la manipolazione dei dati;
- matplot e seaborn -- per la rappresentazione grafica dei dati;
- pgmpy -- per lavorare con modelli grafici.

ORGANIZZAZIONE DEL DATASET

Link https://archive.ics.uci.edu/ml/datasets/Chronic_Kidney_Disease

Il dataset impiegato raccoglie, di 400 pazienti, le seguenti informazioni:

age – age	-	età
bp – blood_pressure	-	pressione sanguigna
sg – specific_gravity	-	gravità specifica
al – albumin	-	albumina
su – sugar	-	glicemia
rbc – red_blood_cells	-	globuli rossi
pc – pus_cell	-	cellule di pus
pcc – pus_cell_clumps	-	grumi di cellule di pus
ba – bacteria	-	batteri
bgr - blood_glucose_random	-	glucosio nel sangue
bu – blood_urea	-	sangue nelle urine
sc – serum_creatinine	-	creatinina sierica
sod – sodium	-	sodio
pot – potassium	-	potassio
hemo – hemoglobin	-	emoglobina
pcv – packed_cell_volume	-	ematoctito
wc - white_blood_cell_count	-	numero globuli bianchi
rc - red blood_cell_count	-	numero di globuli rossi
htn – hypertension	-	ipertensione
dm – diabetes_mellitus	-	diabete mellito
cad -coronary_artery_disease	-	problemi all’arteria coronaria
appet – appetite	-	appetito
pe – pedal_edema	-	edema
ane – anemia	-	anemia
class – class	-	problema cronico ai reni

ANALISI E PRE ELABORAZIONE DEL DATASET

Per prima cosa è stata fatta un'analisi generale sulle caratteristiche del dataset.

Poiché le colonne avevano nomi poco esplicativi, si è deciso di sostituirli nomi di più facile comprensione. Successivamente è stata eliminata la colonna *id* perché poco informativa:

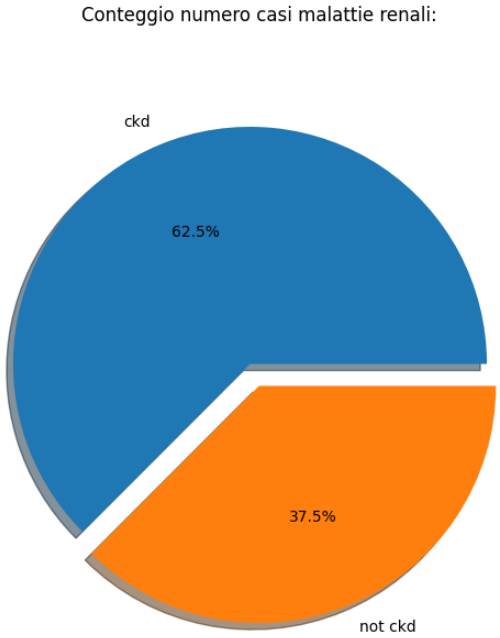
```
dataset.columns = ['age', 'blood_pressure', 'specific_gravity', 'albumin', 'sugar', 'red_blood_cells', 'pus_cell',
                   'pus_cell_clumps', 'bacteria', 'blood_glucose_random', 'blood_urea', 'serum_creatinine', 'sodium',
                   'potassium', 'haemoglobin', 'packed_cell_volume', 'white_blood_cell_count', 'red_blood_cell_count',
                   'hypertension', 'diabetes_mellitus', 'coronary_artery_disease', 'appetite', 'pedal_edema',
                   'anemia', 'class']
```

OSSERVAZIONI

Analizzando il dataset è risultato che il numero di pazienti con un problema cronico ai reni è di 250 contro ai 150 di quelli che non lo hanno. Inoltre, la base di dati presenta 11 colonne con valori numerici e 14 con valori categorici, tra i quali molti sono mancanti:

- Valori nulli per colonna

```
red_blood_cells             152
red_blood_cell_count        131
white_blood_cell_count      106
potassium                    88
sodium                       87
packed_cell_volume           71
pus_cell                     65
haemoglobin                  52
sugar                        49
specific_gravity             47
albumin                      46
blood_glucose_random         44
blood_urea                   19
serum_creatinine             17
blood_pressure               12
age                           9
bacteria                      4
pus_cell_clumps               4
hypertension                  2
diabetes_mellitus             2
coronary_artery_disease       2
appetite                      1
pedal_edema                   1
anemia                        1
class                         0
```

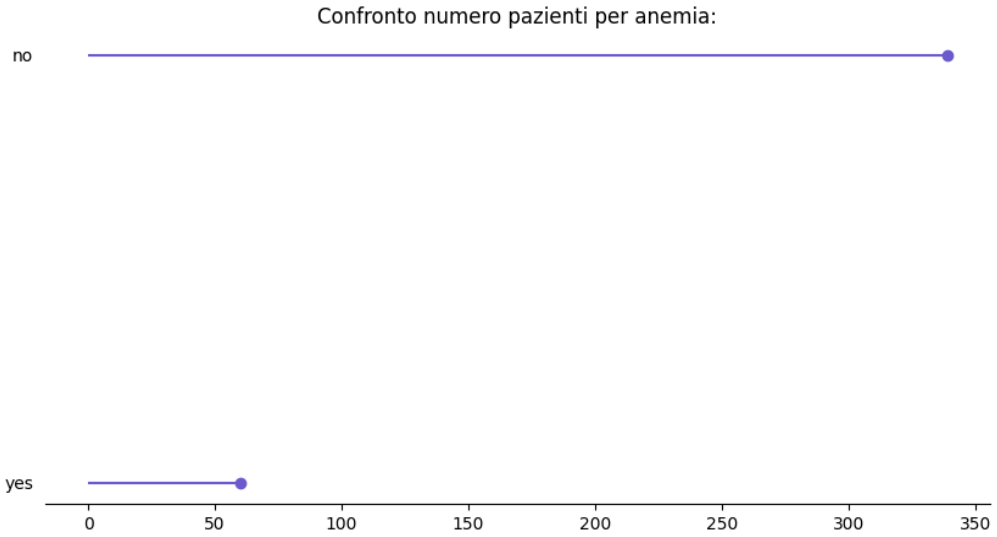


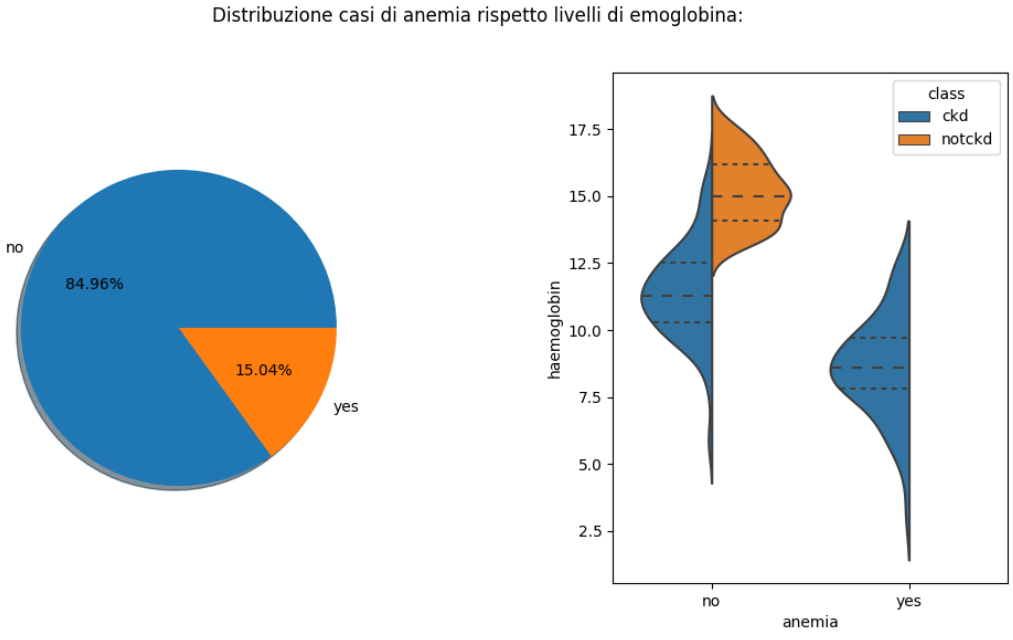
Poiché i valori di alcune colonne pur rappresentando dei dati numerici erano indicati come stringhe, si è optato per convertirli in *float*, e altri che invece presentavano uno stesso valore ma scritto diversamente (esempio 'yes'-' yes'), sono stati uniformati ad un'unica raffigurazione.

Successivamente è stata fatta un'approfondita esplorazione grafica dei dati.

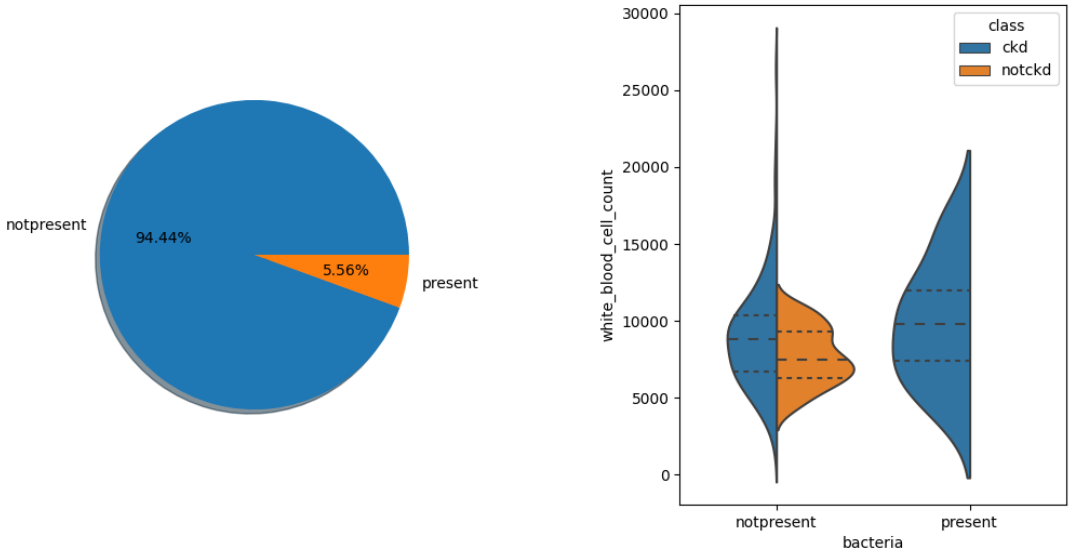
Osservazioni grafiche dei dati

Abbiamo iniziato con il visualizzare alcuni dati per comprenderli meglio, controllando la numerosità di alcune categorie nel dataset e cercando una possibile correlazione tra i dati che più sembravano connessi tra loro.



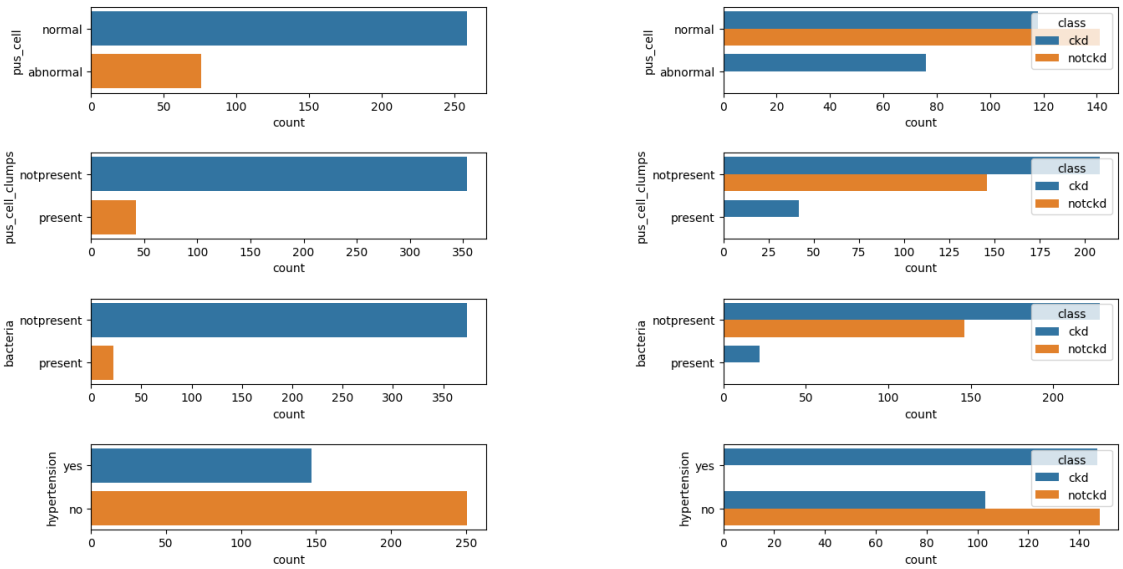


Distribuzione presenza di batteri rispetto numero di globuli bianchi:

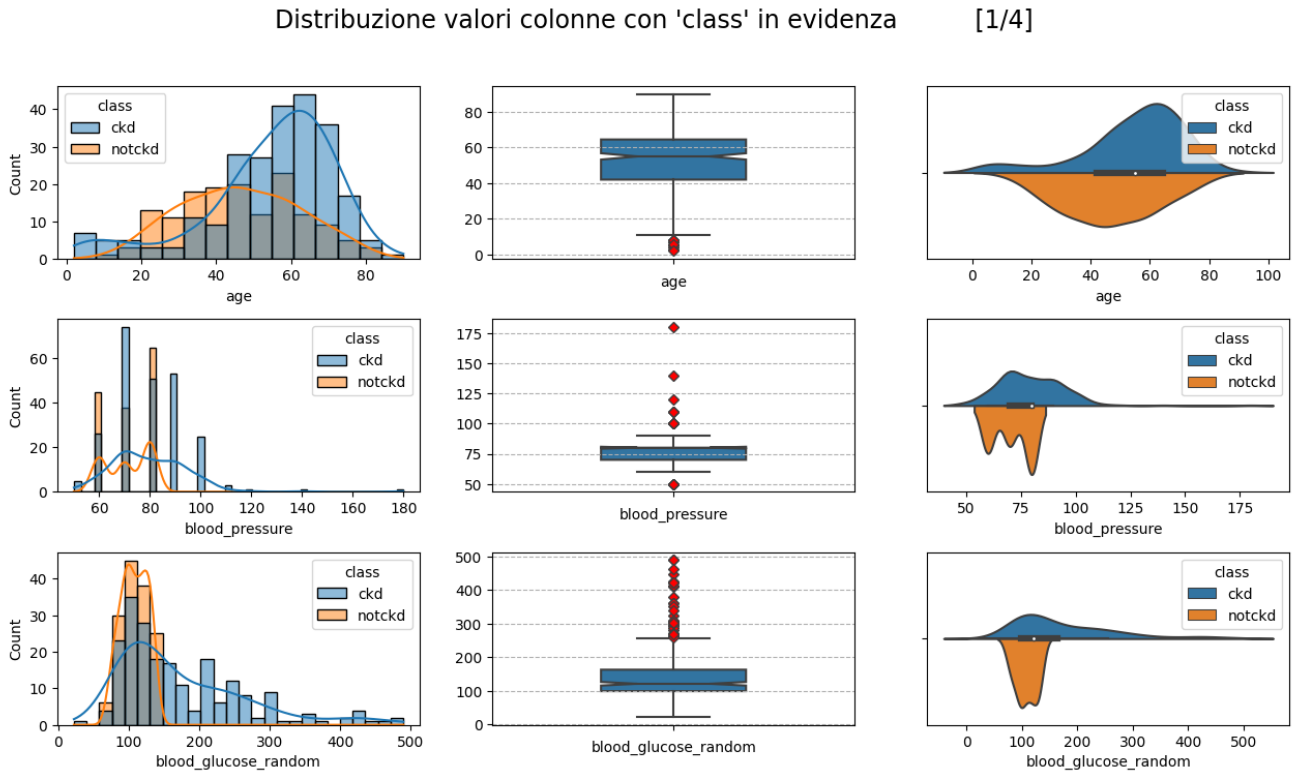


Sono stati creati dei grafici per visualizzare il conteggio di tutte le variabili di tipo categorico rispetto alle classi *ckd* e *notckd*.

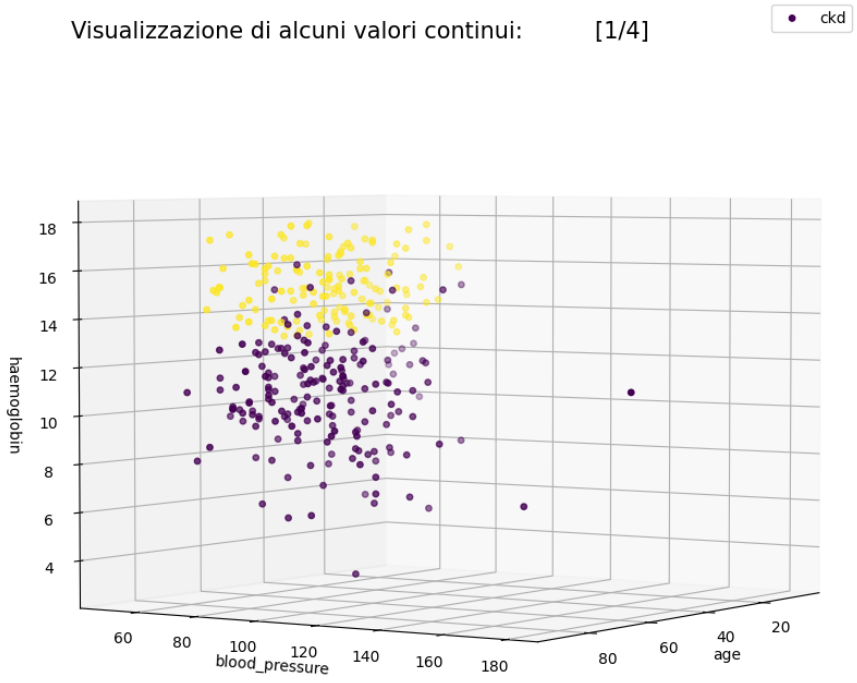
Conteggio valori colonne rispetto 'class' [2/4]

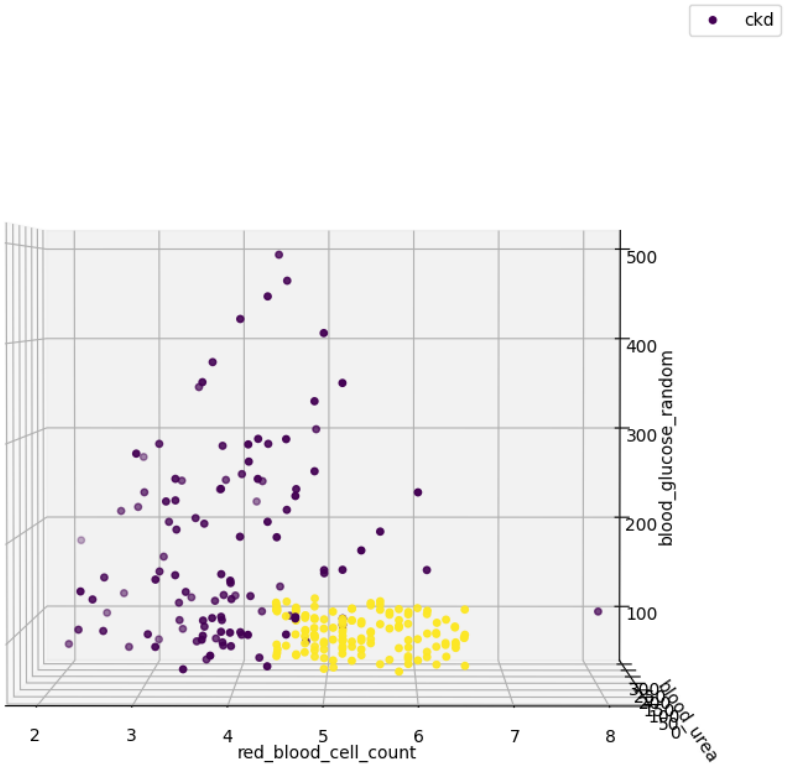


Sono state rappresentate anche le distribuzioni dei valori delle colonne di tipologia continua.



Inseguito si è cercato di analizzare in maniera più approfondita le correlazioni tra le variabili.





Con questo si è conclusa l’esplorazione grafica.

Ritornando al dataset, prima di utilizzare i dati, per i motivi precedentemente indicati, è stato necessario apportare delle modifiche, di cui le principali sono:

- Codifica del dataset: dato che le variabili categoriche che necessitavano una codifica esprimevano valori binari (es. *hypertension :* 'yes' / 'no'), per semplicità è stato utilizzato *LabelEncoder().*

- Riempimento dei valori mancanti: sono stati calcolati attraverso *KNNImputer* – sono stati trovati i dati dei pazienti più simili a quelli del soggetto a cui serviva assegnare valori perché non presenti. E' stato scelto, in base al miglior risultato ottenuto nei test condotti su vari modelli, un numero di vicini k pari a 2.

```
Controllo presenza valori nulli per colonna:

age                       0
blood_pressure            0
specific_gravity          0
albumin                   0
sugar                     0
red_blood_cells           0
pus_cell                  0
pus_cell_clumps           0
bacteria                  0
blood_glucose_random      0
blood_urea                0
serum_creatinine          0
sodium                    0
potassium                 0
haemoglobin               0
packed_cell_volume        0
white_blood_cell_count    0
red_blood_cell_count      0
hypertension              0
diabetes_mellitus         0
coronary_artery_disease   0
appetite                  0
pedal_edema               0
anemia                    0
class                     0
dtype: int64

Controllo numero tuple ridondanti:

0
```

Per quanto riguarda lo squilibrio delle due classi (disturbo cronico/non cronico), dopo una serie di test sulle prestazioni degli algoritmi fatti sia con altre istanze generate (mediante la tecnica *smote*) che senza, è stato deciso di non includere esempi fittizi in modo tale da non compromettere ulteriormente la veridicità dei risultati (dato che erano pressocché identici).

Successivamente si sono create due versioni del dataset: una standardizzata e una normalizzata. I dati con valori continui sono stati:

- standardizzati con *StandardScaler ( )* - il nuovo valore standard z di un campione x è calcolato come:

  $$z = (x - u) / s$$

  dove u è la media dei campioni di addestramento ed s è la deviazione standard dei campioni di addestramento.

- normalizzati con *MinMaxScaler( )* - i dati vengono trasposti nell' intervallo [0-1] attraverso la seguente formula:

$$x_{scaled} = \frac{x - x_{min}}{x_{max} - x_{min}}$$

Ulteriori elaborazioni dei dati richieste dagli algoritmi saranno specificate successivamente.

TECNICHE UTILIZZATE

APPRENDIMENTO NON SUPERVSIONATO

Il clustering, anche definito come classificazione non supervisionata, consiste in un insieme di metodi di analisi multivariata dei dati, volti a raggruppare elementi appartenenti a tali dati in classi omogenee: come per la classificazione, lo scopo è segmentare i dati ma senza assegnare etichette di classe e senza avere classi predefinite. Queste classi, dette anche cluster, sono insiemi di oggetti che presentano tra loro delle similarità, ma che, contemporaneamente, hanno caratteristiche diverse con oggetti in altri cluster. Lo scopo è quello di vedere come si distribuiscono le persone all'interno di questo dataset.

PRE-PROCESSAMENTO DEI DATI

Per prima cosa si è applicata la **PCA** sui dati standardizzati:

La **Principal Component Analysis**, o analisi delle componenti principali, è una tecnica utilizzata per semplificare i dati. Date n variabili qualunque è possibile calcolare altrettante componenti principali, operando una combinazione lineare tramite i coefficienti ottenuti dalla matrice degli n auto-vettori della matrice di correlazione delle variabili originali. Le componenti principali così calcolate non sono correlate, hanno varianza pari agli autovalori della matrice di correlazione e conservano tutta la variabilità iniziale dei dati. Inoltre (cosa più importante) la prima PC (principal component) estrae il massimo possibile della variabilità iniziale dei dati, la seconda PC estrae il massimo della variabilità rimanente e così via. In questo modo è possibile ridurre la dimensionalità dei dati prendendo le prime m PC (con m < n), che sono in grado di conservare una elevata percentuale dell'informazione (variabilità) inizialmente contenuta nei dati sperimentali originali.

Si è optato per questa tecnica visto l'elevato numero di feature (26), la loro poca informatività, e per ridurre il più possibile il rumore nei dati.

È stato scelto di usare il dataset standardizzato perché conducendo dei test, dalle metriche, risultava che fossero i dati che supportavano maggiormente l'algoritmo di clustering usato.

In questo caso specifico la funzione *PCA( )* per ridurre la dimensionalità lineare dei dati impiega la *Singular Value Decomposition*, che a differenza di altri metodi, può essere molto più efficiente ed è in grado di gestire matrici sparse.

Per scegliere il numero delle componenti principali sono state applicate due tecniche:

- *Regola di Kaiser*:

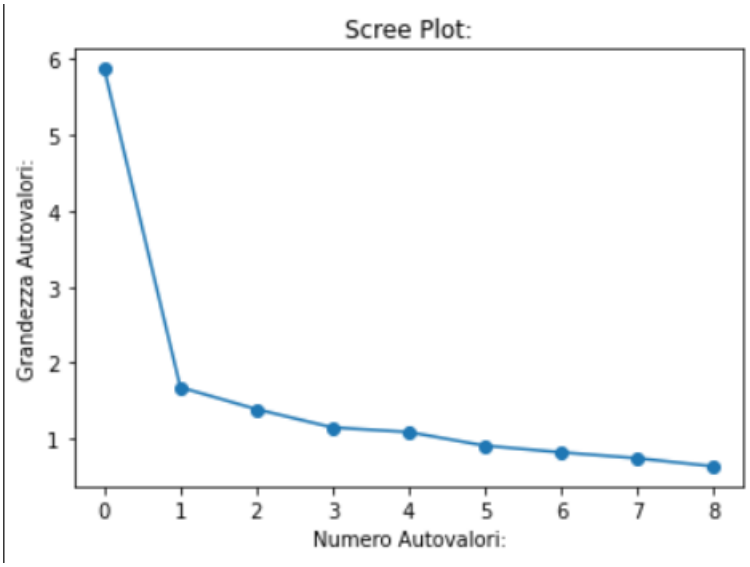
  secondo questa regola si dovrebbero includere nel modello finale tutte le componenti a cui corrisponde un autovalore uguale o maggiore di 1;

```
Autovalori:
[5.86950253 1.67833476 1.38983919 1.14935129 1.08921458 0.91024962
 0.82270421 0.74501032 0.63855227]
```

- *Scree Plot*:

  si basa su un grafico in cui sull'asse verticale sono riportati i valori degli autovalori e sull'asse orizzontale tutte le possibili componenti da estrarre (che saranno quindi in numero pari alle

variabili di partenza). Unendo i punti si otterrà una linea spezzata che in alcune parti avrà una forma concava ed in altri convessa. Seguendo questo criterio, il numero di componenti da estrarre è quello che coincide con il cambio di pendenza.



Analizzando i risultati ottenuti dalla regola di Kaiser e dallo Scree Plot, si è optato per prendere come componenti principali le prime 5.

ALGORITMI IMPIEGATI

KMEDOIDS

Funzione: KMedoids()

Il metodo basato su medoidi è una tecnica alternativa al metodo basato su centroidi per gli algoritmi di clustering partizionale: se il K-means cerca di minimizzare l'errore quadrato totale, il k-medoids minimizza la somma delle differenze tra i punti etichettati come essere in un cluster e un punto designato come il centro di quel cluster. Inoltre, il k-medoids sceglie i data points come centri. Con questo algoritmo è possibile analizzare anche dati categorici, cosa che con il k-means è controindicata.

E' stato usato come metodo del clustering k-medoid il Partitioning Around Medoids (PAM), perché dà risultati leggermente migliori rispetto al predefinito.

L'algoritmo PAM è articolato nel seguente modo:

1. Inizializzazione: seleziona casualmente k dei n punti dati come medoidi;

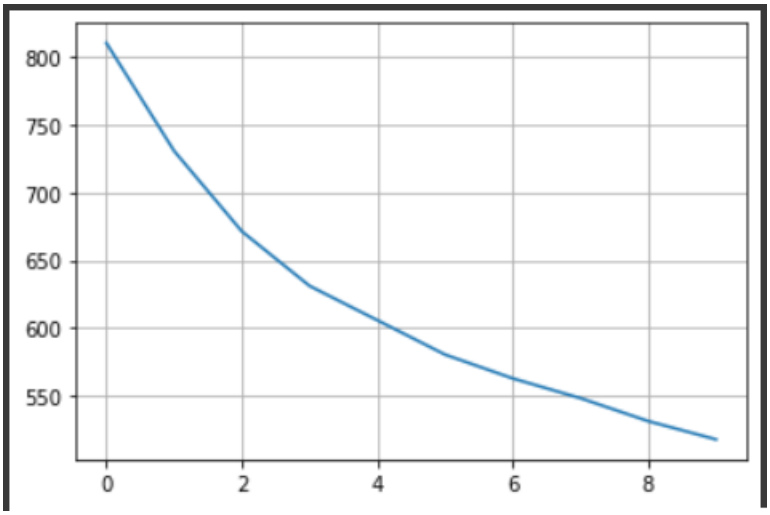
2. Fase di assegnazione: associa ciascun punto dati al medoide più vicino;

3. Passo di aggiornamento: per ogni medoide m e ogni data point x associato a m scambia m e x e calcola il costo totale della configurazione (cioè la differenza media di x rispetto a tutti i punti dati associati a m). Seleziona il medoide x con il costo più basso della configurazione.

Ripete i passaggi 2 e 3 fino a quando non vi è alcun cambiamento nelle assegnazioni.

Un passo importante per questo algoritmo è assegnare il numero appropriato di cluster (k). Per fare ciò sono stati usati i seguenti metodi:

- *curva a gomito*: cerca il punto per cui un certo score, dopo un certo valore di k, non ha una variazione significativa. Con:
  - *inertia_* si va a calcolare, per ogni assegnazione, la somma delle distanze quadratiche dei dati rispetto al loro cluster più vicino.



- *coefficiente di silhouette*:

  Il suo valore varia da -1 a 1:

  - Un punteggio di 1 indica che i cluster sono ben distanti l'uno dall'altro e chiaramente distinti.
  - Un punteggio di 0 indica che i cluster sono indifferenti, ovvero che la distanza tra i cluster non è significativa.
  - Un punteggio di -1 indica che i cluster sono assegnati in modo errato.

$$\text{Silhouette score} = \frac{p-q}{\max(p,q)}$$

p è la distanza media dai punti nel cluster più vicino (di cui il punto non fa parte)

q è la distanza media intra-cluster da tutti i punti nel proprio cluster.

```
Con n_clusters=2, il valore di silhouette 0.40556393245720956
Con n_clusters=3, il valore di silhouette 0.41121996054507565
Con n_clusters=4, il valore di silhouette 0.3725818615785702
Con n_clusters=5, il valore di silhouette 0.38379777995081205
Con n_clusters=6, il valore di silhouette 0.3954886979573057
Con n_clusters=7, il valore di silhouette 0.24144834576068047
Con n_clusters=8, il valore di silhouette 0.24251719152739867
Con n_clusters=9, il valore di silhouette 0.21307264034128878
Con n_clusters=10, il valore di silhouette 0.22051985824741074
Con n_clusters=11, il valore di silhouette 0.21141564548904035
```

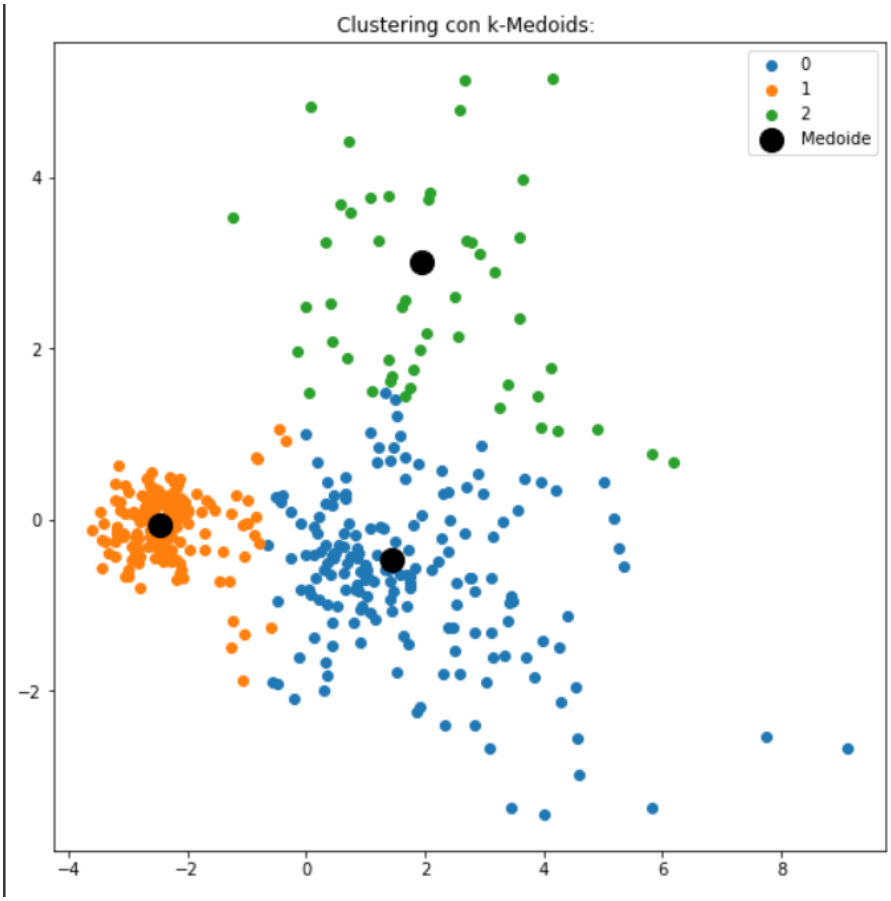
Poiché sia il Silhouette score che la regola del gomito danno come risultato ottimale 3, è stato deciso di addestrare l'algoritmo partizionando il dataset in 3 cluster.

Questo vuol dire che l'algoritmo di clustering ha trovato un ulteriore modo per raggruppare i dati rispetto a quello fornito nel dataset.

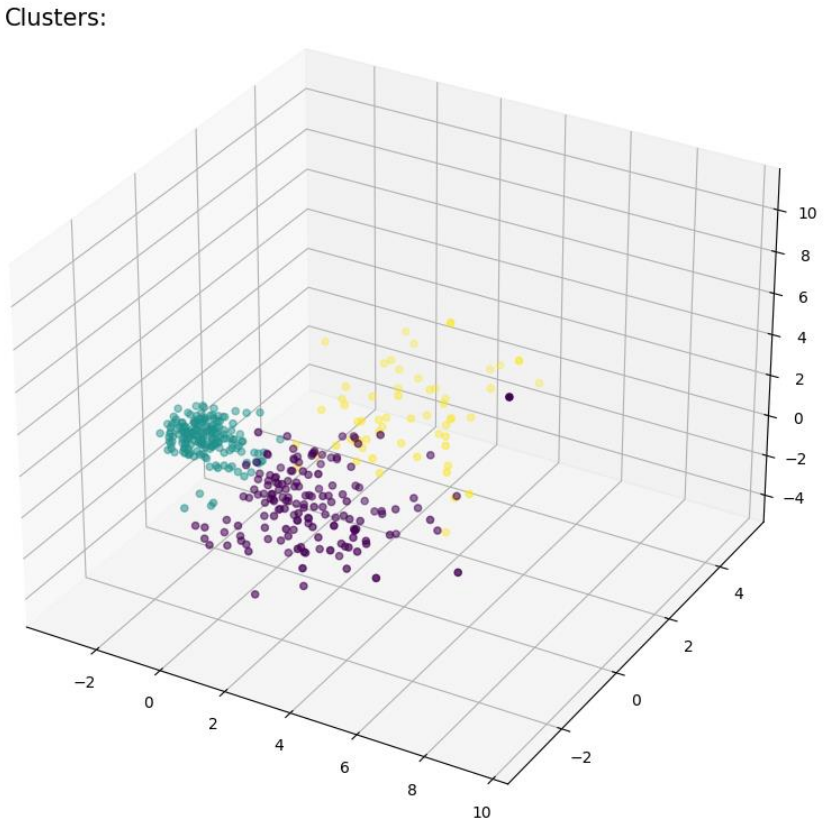
I medoidi trovati per i cluster sono:

Cluster	Age	Blood Pressure	Albumin	Pus Cell	Blood Glucose Random	Blood Urea	Sodium	Potassium	Haemoglobin	Packed Cell Volume	White Blood Cell Count	Red Blood Cell Count	Hypertension	Coronary Artery Disease	Pedal Edema	Class
0	55	90	2	0.50	143	88	132	4.85	11	34	10000	4.75	1	0	1	0
1	38.5	80	0	1	100	49	140	5	16.3	53	8500	4.9	0	0	0	1
2	60	90	1.5	0.5	269	51	138	3.7	11.5	35	10500	4.2	1	1	1	0

Rappresentazione del risultato rispetto le prime due PC:



Rappresentazione del risultato rispetto alle prime tre PC:



Metriche di valutazione del clustering:

```
Valutazione:

Omogeneità  :  0.7362042205711663
Completezza :  0.49533777158339803
V_measure   :  0.5922165226538961
```

Omogeneità : homogeneity_score verifica se i cluster contengono solo campioni appartenenti a una singola classe, nel nostro caso *class*. È definito come:

$$h_{score} = 1 - \frac{H(Y_{true}|Y_{pred})}{H(Y_{true})}$$

è delimitato tra 0 e 1. Più un valore tende a zero, meno il cluster è omogeneo.

Completezza : completeness_score verifica se tutti i data points che sono membri di una determinata classe sono elementi dello stesso cluster.

$$c_{score} = 1 - \frac{H(Y_{pred}|Y_{true})}{H(Y_{pred})}$$

V_misure : v_measure_score è la media armonica tra *homogeneity_score e completeness_score.*

$$v\_measure_{score} = \frac{(1+\beta)*h_{score}*c_{score}}{\beta*h_{score}+c_{score}}$$

Con β =1.

APPRENDIMENTO SUPERVISIONATO

La classificazione fa parte delle tecniche di apprendimento supervisionato. Gli algoritmi che se ne occupano lavorano su dataset etichettati e restituiscono come output una variabile qualitativa di tipo binario, nominale o ordinale.

Per quanto riguarda la parte di testing, sarà approfondita al termine della descrizione dei vari algoritmi.

ALGORITMI IMPIEGATI

- **KNEIGHBORS CLASSIFIER**

  Funzione: *KNeighborsClassifier()*

  Il k-nearest neighbors parte dal presupposto che punti simili (e quindi oggetti) possono essere trovati l'uno vicino all'altro. 'k' indica appunto il numero di punti più vicini a quello da classificare, ed è un intero positivo.

  In questo algoritmo, lo spazio viene partizionato in regioni in base alle posizioni e alle caratteristiche degli oggetti di apprendimento. Per calcolare la distanza tra essi, si è usata la distanza euclidea (=default metrica di minkowski con p=2). Infine, l'oggetto è assegnato alla classe che è la più frequente fra i suoi k vicini.

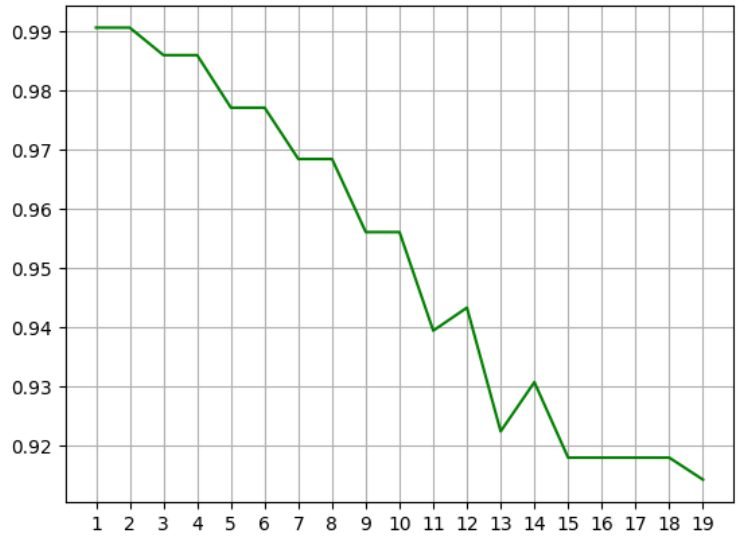
  È stato utilizzato il dataset normalizzato, poiché più indicato per gli algoritmi distance-based e inoltre, per i test condotti, è quello che porta l'algoritmo a risultati migliori.

  Per prima cosa bisogna scegliere quanto vale k. Per decidere ciò, l'algoritmo è stato addestrato a partire da un vicino, fino ad arrivare a venti. Ad ogni ciclo, è stata applicata la Cross Validation.

  - CROSS VALIDATION

    Consiste nel sezionare il set di esempi più volte, con porzioni di train e test sempre diverse, calcolandone per ognuna lo score. Nel nostro caso viene utilizzata come metrica la F1-measure. Con la cross-validation si evita l'overfitting e si ha una visione più completa del funzionamento dell'algoritmo. Il risultato finale è dato dalla media dei punteggi di tutte le combinazioni.

    Per scegliere il numero di vicini ideale è stata scelta una 5-fold cross validation e applicata al variare di k. Quando l'algoritmo ha raggiunto la F1 maggiore, è stato preso il corrispondente numero di vicini.



Stabilito k=2 il modello è stato addestrato.

- **DECISION TREE CLASSIFIER**

  Funzione: *DecisionTreeClassifier()*

  L'obiettivo di questo algoritmo è dividere la popolazione iniziale per il valore di una variabile che permette di creare due gruppi che sono il più omogenei possibile internamente e il più disomogenei possibile tra loro. Può lavorare su dati numerici e categorici contemporaneamente.

  Durante l'esecuzione l'algoritmo divide continuamente i dati di input in base a determinate metriche (Default gini - L'impurità Gini rappresenta la probabilità di classificare in modo errato un punto dati casuale nel set). Lo scopo è quello di trovare la migliore divisione.

  Nell'albero, i nodi sono i luoghi in cui i dati vengono *splittati* e le foglie i risultati intermedi o finali.

  È stato utilizzato il dataset standardizzato, poiché non vincolato da misure di distanza, per la migliore gestione degli outliers e inoltre è risultato, dai test eseguiti, quello in grado di portare l'algoritmo ad una performance migliore.

  Per questo e per i successivi algoritmi impiegati, al fine di selezionare i migliori iper-parametri, si è optato di usare *RandomizedSearchCV*. Quest'ultima è una funzione che dato un insieme di parametri, ne sceglie un sottoinsieme e testa il modello preso in considerazione al variare di questi, mediante una 5-fold cross validation e lo valuta con la F1-measure.

  Così facendo si possono adottare i parametri che portano allo score migliore.

Tra:

parametri = {'criterion': ['gini', 'entropy', 'log_loss']}

'entropy' è il criterio che ha portato a risultati migliori.

- **RANDOM FOREST CLASSIFIER**

  Funzione: *RandomForestClassifier()*

  Il metodo che usa questo algoritmo fa parte delle tecniche che utilizzano un gruppo di modelli imprecisi, come gli alberi di decisione, al fine di crearne uno più preciso.
  RandomForestClassifier() quindi, costruisce un gran numero di alberi decisionali individuali, su diversi sottoinsiemi del dataset originale, che operano come un insieme. Ogni singolo albero nel *Random Forest* fa una previsione di classe e la classe che è stata predetta maggiormente diventa il risultato che dà il modello.

  È stato utilizzato il dataset standardizzato, poiché non vincolato da misure di distanza, per la migliore gestione degli outliers e inoltre è risultato, dai test eseguiti, quello in grado di portare l'algoritmo ad una performance migliore.

  Tramite *RandomizedSearchCV* la scelta tra i parametri = {'n_estimators': [25, 50, 75, 100, 150, 200, 250]}, è risultata 25.

- **SVM**

  Funzione: *SVC()*

  Traccia ogni dato come un punto nello spazio n-dimensionale (dove n è un numero di caratteristiche disponibili) con il valore di ciascuna caratteristica che è il valore di una particolare coordinata. Successivamente esegue la classificazione trovando l'iperpiano, ovvero la superficie di decisione ottimale che divide positivi da negativi in uno spazio di embedding, utilizzando vettori di supporto.
  Rispetto agli algoritmi presenta due vantaggi principali: maggiore velocità e migliori prestazioni con un numero limitato di campioni (nell'ordine delle migliaia).

  SVM può essere di due tipi:
  - SVM lineare - quello appena descritto. Viene utilizzato per dati separabili linearmente, il che significa che se un set di dati può essere classificato in due classi utilizzando una singola linea retta. Tali dati vengono definiti come dati separabili linearmente e il classificatore che viene utilizzato è chiamato *classificatore SVM lineare*.

  - SVM non lineare - viene utilizzato per dati separabili non linearmente, ovvero il set di dati non può essere classificato utilizzando una linea retta. Tali dati vengono definiti come dati non lineari e il classificatore utilizzato viene chiamato *Classificatore SVM non lineare*.

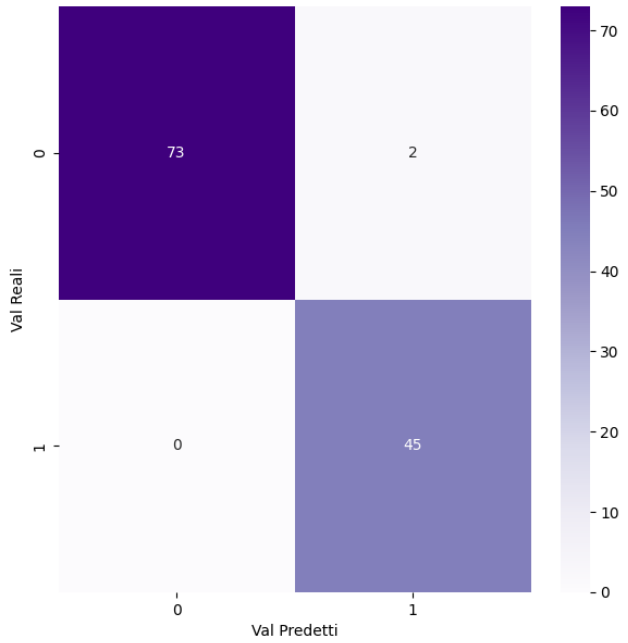
È stato utilizzato il dataset normalizzato, poiché più indicato per gli algoritmi distance-based e inoltre, per i test condotti, è quello che porta l'algoritmo a risultati migliori.

Tramite *RandomizedSearchCV* la scelta tra i parametri = parametri = {'C': [0.001, 0.01, 0.1, 1, 10, 100], 'gamma': [0.001, 0.01, 0.1, 1, 10, 100]}, è risultata C=100, gamma=0.1.

RISULTATI

Dopo aver addestrato i vari modelli descritti in precedenza si è fatta una prima valutazione su un test set ricavato dalla funzione *train_test_split* per avere una prima idea delle loro prestazioni, misurandole attraverso le metriche di accuracy, precision, recall e F1.

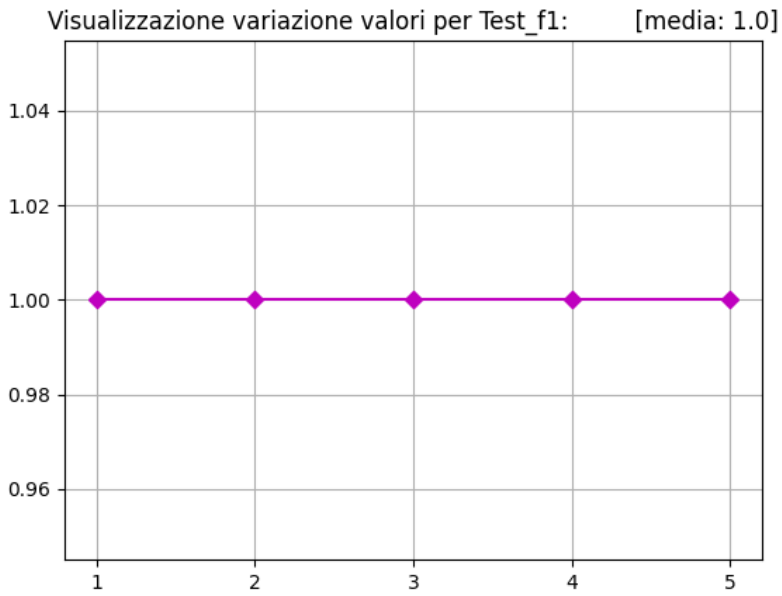
Esempio valutazione KNN:



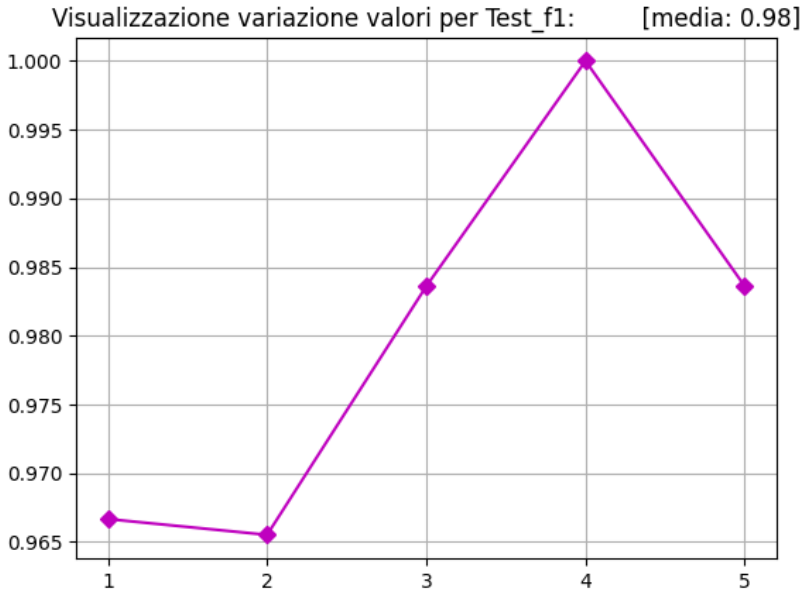
Successivamente per effettuare una valutazione più approfondita dei modelli è stata applicata una stratified k-fold cross validation per tre volte, ognuna con diversi valori di k.

- **STRATIFIED K-FOLD CROSS VALIDATION**
  Tecnica molto simile alla k-fold cross validation, ma con in più un proporzionamento dei vari esempi per classe target nei fold che si generano evitando di creare sbilanciamenti sulle istanze prese in esame.
  Nel nostro caso è stata eseguita una stratified k-fold cross validation per tre volte, in maiera tale da provare diverse suddivisioni del dataset variando anche k, infatti la prima validazione è stata fatta con k=5, la seconda con k=10, la terza con k=15.
  Per ognuna di esse sono state calcolate le metriche nominate in precedenza e si è osservato mediante grafici il variare delle prestazioni per ogni test.

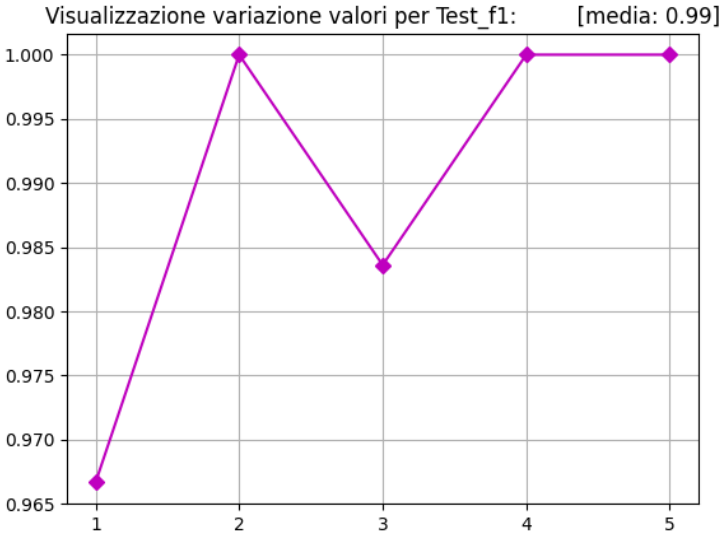
Di seguito sono riportati alcuni esempi:



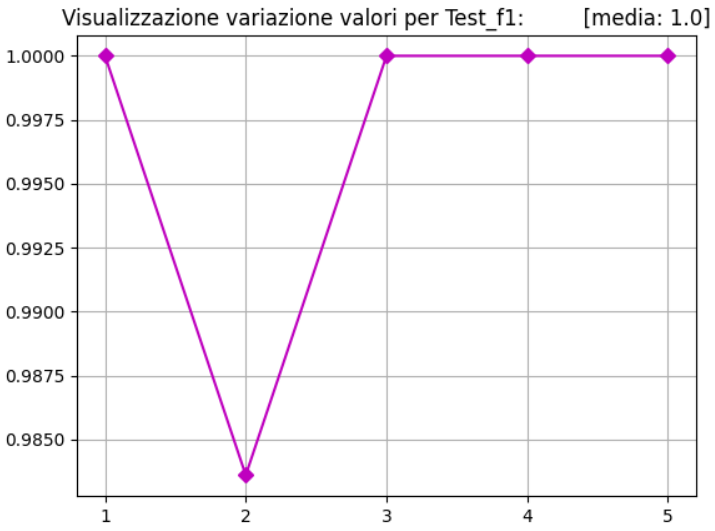
Random Forest con k=5.



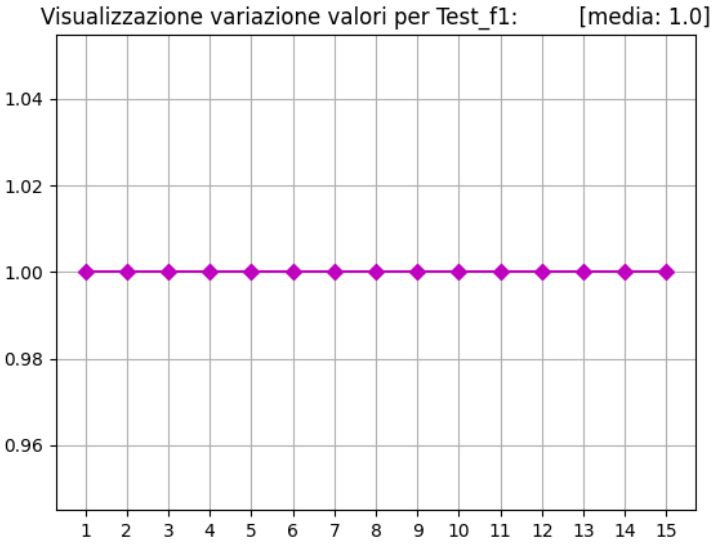
KNN.



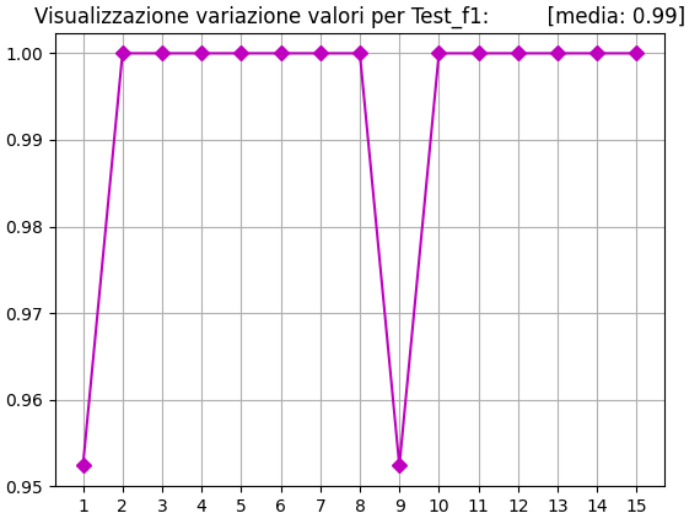
SVM.



Decision Tree.



Random Forest con k=15.



SVM.

In conclusione il valore medio della F1 sulle tre validazioni fatti sui vari modelli è:

```
--  F1 Media delle CV eseguite  --

K-NEARESTNEIGHBORS CLF          [1] :  0.979726092995408

C-SUPPORTVECTORMACHINE CLF      [2] :  0.987094535798925

DECISIONTREE CLF                [3] :  0.9978489027669356

RANDOMFOREST CLF                [4] :  1.0
```

BAYESIAN NETWORK

È stata implementata una rete bayesiana per capire, date tutte le caratteristiche (di quelle specificate nel dataset) di un soggetto, quanto questo abbia la probabilità di avere o meno un disturbo cronico.

Una rete bayesiana è un grafico aciclico diretto, dove:

- i nodi rappresentano le variabili,
- gli archi rappresentano le relazioni di dipendenza statistica tra le variabili e le distribuzioni locali di probabilità dei nodi figlio rispetto ai valori dei nodi genitori.

PRE-PROCESSAMENTO DEI DATI

In questo caso è stato necessario trasformare tutti i valori continui nel dataframe in valori discreti, per evitare di incorrere in eventuali problemi: infatti l'algoritmo utilizzato supporta pienamente solo questo tipo di valori. Ciò è stato fatto, per semplicità, convertendoli in interi.

ALGORITMO IMPIEGATO

BAYESIAN NETWORK

Funzione: *BayesianNetwork()*

Predisposizione della struttura della rete:

attraverso la *local hill climb search* si è stimata la struttura DAG che ha un punteggio ottimale, secondo il metodo di soring fornito. Inizia con il modello *start_dag* e procede con le modifiche della rete passo dopo passo fino al raggiungimento di un massimo locale. In questo caso specifico si è calcolato il punteggio, che misura quanto una data variabile è "influenzata" da una data lista di potenziali genitori, attraverso *K2Score()*, metodo che utilizza la distribuzione di Dirichlet con iper parametri impostati ad 1.

Nodi: *albumin, pus_cell, pedal_edema, white_blood_cell_count, bacteria, pus_cell_clumps, red_blood_cells, sugar, blood_glucose_random, appetite, serum_creatinine, blood_urea, haemoglobin, class, anemia, red_blood_cell_count, hypertension, diabetes_mellitus, coronary_artery_disease, packed_cell_volume, specific_gravity, sodium, blood_pressure, age*.

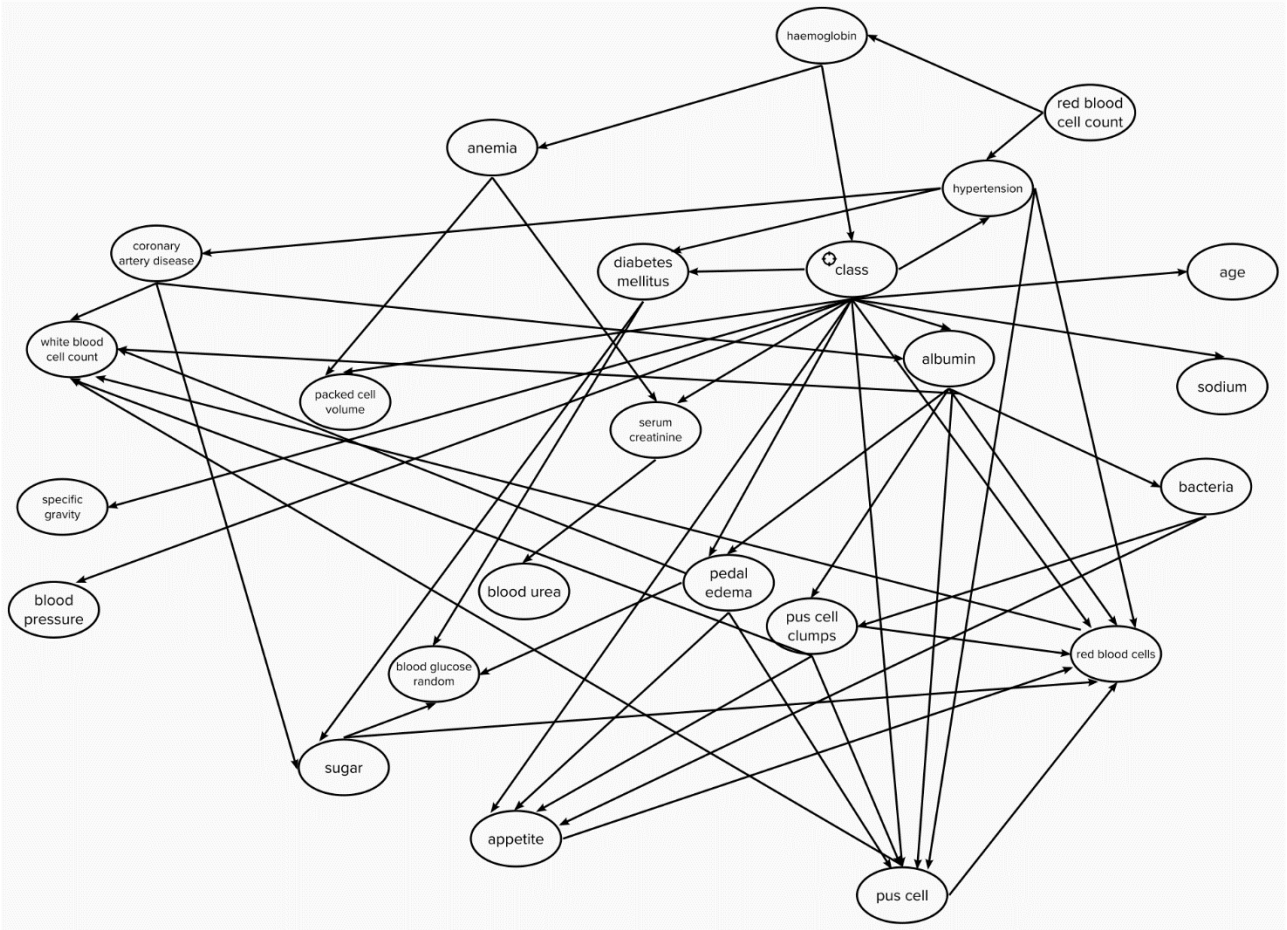
Archi: *(albumin, pus_cell), (albumin, pedal_edema), (albumin, white_blood_cell_count), (albumin, bacteria), (albumin, pus_cell_clumps), (albumin red_blood_cells), (pus_cell, white_blood_cell_count), (pus_cell, red_blood_cells), (pedal_edema, appetite), (pedal_edema, pus_cell), (pedal_edema, white_blood_cell_count), (pedal_edema, blood_glucose_random), (bacteria, pus_cell_clumps), (bacteria, appetite), (pus_cell_clumps, pus_cell), (pus_cell_clumps, white_blood_cell_count), (pus_cell_clumps, red_blood_cells), (pus_cell_clumps, appetite), (red_blood_cells, white_blood_cell_count), (sugar, blood_glucose_random), (sugar, red_blood_cells), (appetite, red_blood_cells), (serum_creatinine, blood_urea), (haemoglobin, class), (haemoglobin, albumin), (class, specific_gravity), (class, albumin), (class, packed_cell_volume), (class, hypertension), (class, red_blood_cells), (class, diabetes_mellitus), (class, serum_creatinine), (class, sodium), (class, appetite), (class, blood_pressure), (class, age), (class, pedal_edema), (class, pus_cell), (anemia, packed_cell_volume), (anemia, serum_creatinine), (red_blood_cell_count,*

haemoglobin), (red_blood_cell_count, hypertension), (hypertension, diabetes_mellitus), (hypertension, coronary_artery_disease), (hypertension, red_blood_cells), (hypertension, pus_cell), (diabetes_mellitus, sugar), (diabetes_mellitus, blood_glucose_random), (coronary_artery_disease, sugar), (coronary_artery_disease, albumin), (coronary_artery_disease, white_blood_cell_count.

Creazione della rete:

la rete è stata prima creata con ***BayesianNetwork()***, dando come argomento gli archi del modello stimato precedentemente e poi addestrata. Lo stimatore utilizzato è quello di default, ovvero *MaximumLikelihoodEstimator*. Quest'ultimo massimizza la funzione di verosimiglianza in modo che, secondo il modello statistico assunto, i dati osservati siano più probabili.

Rappresentazione grafica della rete:



Si è calcolata, infine, la probabilità che una certa persona, con determinati valori, abbia un disturbo cronico o meno:

```
Probabilità per un individuo di non avere un disturbo cronico ai reni:
+----------+--------------+
| class    |   phi(class) |
+==========+==============+
| class(0) |       1.0000 |
+----------+--------------+
| class(1) |       0.0000 |
+----------+--------------+
```

```
Probabilità per un individuo di avere un disturbo cronico ai reni:
+----------+--------------+
| class    |   phi(class) |
+==========+==============+
| class(0) |       0.0007 |
+----------+--------------+
| class(1) |       0.9993 |
+----------+--------------+
```