Dipartimento di informatica
CdS in Informatica
Ingegneria della conoscenza

Diabetics Classifier

Documentazione Progetto

AA 2022-2023

Progetto di

Aldo Mangione, 742287, a.mangione12@studenti.uniba.it

Repository GitHub:

https://github.com/Alman1236/Icon-22-23-Diabetes-Prediction

INDICE

Introduzione

Ricavare informazioni sulle possibili patologie di un paziente, a partire dalla sua cartella clinica è molto utile, specie se consideriamo che con il machine learning questo è reso particolarmente veloce e poco dispendioso. Un insieme di modelli in grado di identificare diverse patologie è dunque uno strumento di fondamentale importanza per la medicina. Con questo progetto lavorerò su modelli che si occuperanno nello specifico di rilevare il diabete, a partire dallo storico del paziente. Il progetto è stato anche utilizzato per studiare quale tipo di modello si adatta meglio al problema, verranno infatti raccolti dati sul testing e verranno confrontate le metriche che descrivono la performance dei classificatori. In aggiunta si proverà ad utilizzare come approccio anche il clustering, attraverso l'algoritmo K-Means.

Sommario

Il compito di valutare la possibile presenza del diabete è chiaramente un compito di classificazione su target feature binaria. Per questo scopo sono stati utilizzati quattro modelli diversi: Gaussian Naive Bayes, Knn, Random Forest Classifier e un Neural network, che sono stati confrontati in base a diverse metriche (precisione, F1 score, accuratezza). Per poter addestrare questi modelli sono state necessarie alcune operazioni di preprocessing di cui parlerò in seguito. Come già detto si è provato anche un approccio alternativo attraverso un algoritmo di clustering, chiamato K-Means. La qualità del clustering verrà valutata attraverso altre metriche, anche queste spiegate più avanti.

Elenco argomenti di interesse

- Apprendimento Supervisionato: K Nearest neighbors e Random Forest Classifier
- Neural network: Multilayer Perceptron Classifier
- Apprendimento Non supervisionato: K Means (Hard clustering)
- Apprendimento Probabilistico: Gaussian Naive Bayes

Risorse strumentali

Il progetto è stato realizzato con il linguaggio Python in Visual Studio Code. Le librerie utilizzate sono state le seguenti:

- sklearn: algoritmi di apprendimento, metriche, strumenti per il preprocessing;
- time: calcolo del tempo;
- pandas, numpy: manipolazione dei dati;
- matplot: rappresentazione grafica dei dati;

Informazioni dataset e Preprocessing

```
o-------------------------------------------o
| Progetto ICON 22-23: Diabetics classifier  |
o-------------------------------------------o


Prime 5 righe del dataset:
    gender   age  hypertension  heart_disease smoking_history    bmi  HbA1c_level  blood_glucose_level  diabetes
0  Female  80.0             0              1           never  25.19          6.6                  140         0
1  Female  54.0             0              0         No Info  27.32          6.6                   80         0
2    Male  28.0             0              0           never  27.32          5.7                  158         0
3  Female  36.0             0              0         current  23.45          5.0                  155         0
4    Male  76.0             1              1         current  20.14          4.8                  155         0


Dimensioni del dataset:
 (100000, 9)


Colonne del dataset:
 Index(['gender', 'age', 'hypertension', 'heart_disease', 'smoking_history',
       'bmi', 'HbA1c_level', 'blood_glucose_level', 'diabetes'],
      dtype='object')
```

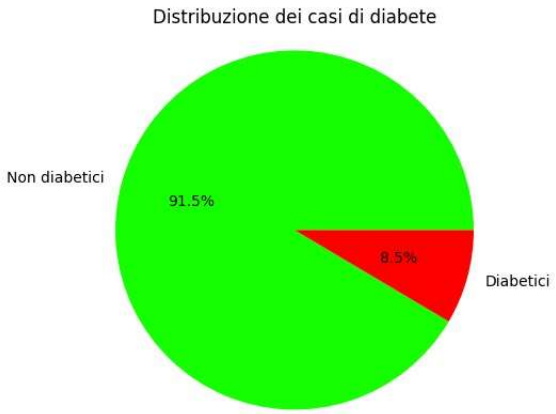
Il dataset prevede 100'000 record descritti dai seguenti campi: sesso, età, ipertensione, storia da fumatore, bmi, problemi al cuore, livello HbA1c, livello glucosio nel sangue, e diabete. Innanzitutto, son partito assicurandomi che non vi fossero campi nulli, e fortunatamente non ce ne sono.

```
Statistiche dataset:
        gender            age  hypertension  heart_disease smoking_history            bmi    HbA1c_level  blood_glucose_level       diabetes
count   100000  100000.000000  100000.00000  100000.000000          100000  100000.000000  100000.000000        100000.000000  100000.000000
unique       3            NaN           NaN            NaN               6            NaN           NaN                  NaN            NaN
top     Female            NaN           NaN            NaN        No Info            NaN           NaN                  NaN            NaN
freq     58552            NaN           NaN            NaN           35816            NaN           NaN                  NaN            NaN
mean       NaN      41.885856       0.07485       0.039420             NaN      27.320767      5.527507           138.058060       0.085000
std        NaN      22.516840       0.26315       0.194593             NaN       6.636783      1.070672            40.708136       0.278883
min        NaN       0.080000       0.00000       0.000000             NaN      10.010000      3.500000            80.000000       0.000000
25%        NaN      24.000000       0.00000       0.000000             NaN      23.630000      4.800000           100.000000       0.000000
50%        NaN      43.000000       0.00000       0.000000             NaN      27.320000      5.800000           140.000000       0.000000
75%        NaN      60.000000       0.00000       0.000000             NaN      29.580000      6.200000           159.000000       0.000000
max        NaN      80.000000       1.00000       1.000000             NaN      95.690000      9.000000           300.000000       1.000000
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 100000 entries, 0 to 99999
Data columns (total 9 columns):
 #   Column               Non-Null Count   Dtype
---  ------               --------------   -----
 0   gender               100000 non-null  object
 1   age                  100000 non-null  float64
 2   hypertension         100000 non-null  int64
 3   heart_disease        100000 non-null  int64
 4   smoking_history      100000 non-null  object
 5   bmi                  100000 non-null  float64
 6   HbA1c_level          100000 non-null  float64
 7   blood_glucose_level  100000 non-null  int64
 8   diabetes             100000 non-null  int64
dtypes: float64(3), int64(4), object(2)
memory usage: 6.9+ MB


Altro:
 None
```

Ho poi dovuto standardizzare alcuni campi, infatti 'smoking_history' è descritta da cinque possibili testi diversi ('never', 'ever', 'former', 'No info', 'current'), e gender è compilato in forma testuale. Detto questo, ho applicato le seguenti modifiche: ho aggiunto due campi al dataset, 'smoker_bool', che ci dice se l'individuo ha mai fumato, e 'male_bool' che ci dice in forma numerica se l'individuo è maschio o no. 'smoker_bool' è stato settato a 1 per tutti quegli individui che presentavano 'former' o 'current' in 'smoking_history'.

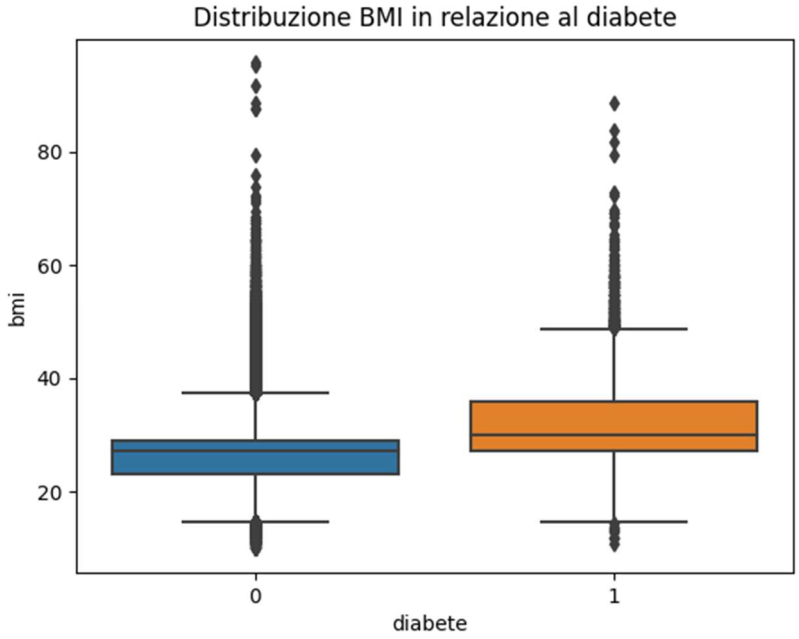
Analisi dei dati

Ho effettuato delle osservazioni sui dati e le statistiche che mi permettessero di valutare la correlazione dei dati e quindi effettuare delle scelte più appropriate per l'addestramento dei modelli.

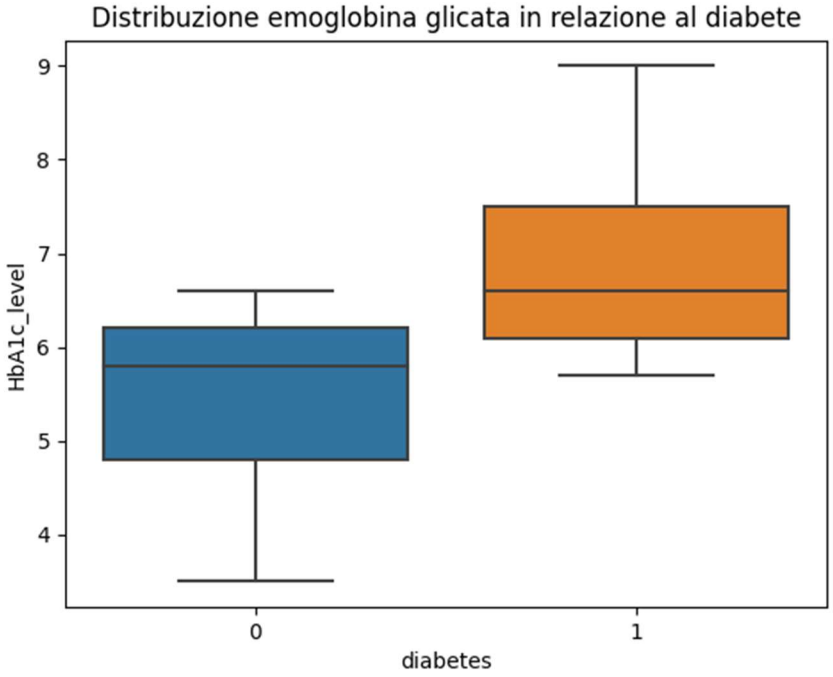


La prima cosa che ho notato è che il dataset è sproporzionato dal punto di vista del conteggio di diabetici. A causa di questo, è stato di fondamentale importanza assicurarsi di effettuare una giusta distribuzione dei casi diabetici nei casi per il training, e i casi per il test. Ciò è stato possibile attraverso il

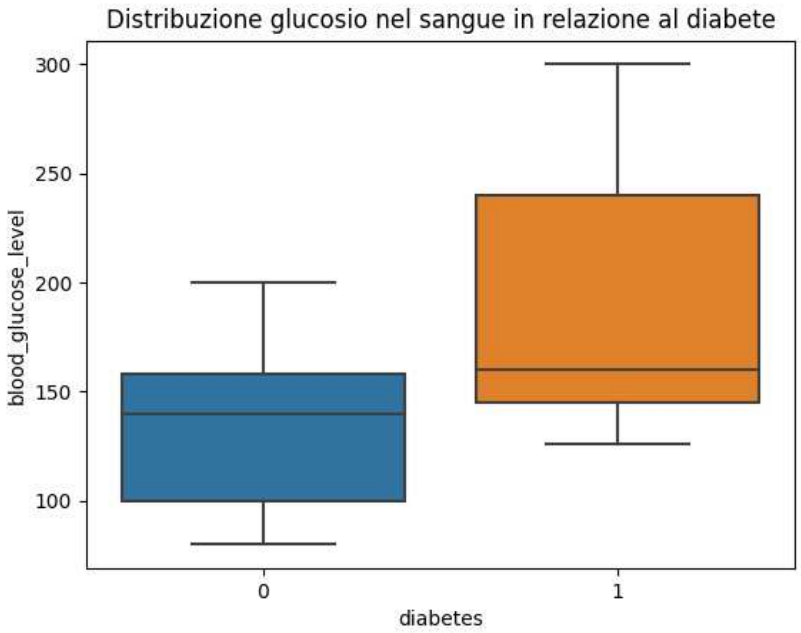
parametro 'stratify' in train_test_split(), che mi permette di dividere i casi positivi in modo proporzionato tra training e test set.



C'è una chiara correlazione tra bmi alto e presenza della patologia. In particolare, la maggior parte dei diabetici ha un bmi vicino o superiore al 30, che è la soglia che indica l'obesità.



Anche la presenza di emoglobina glicata è fortemente correlata ai casi di diabete. Si nota facilmente infatti che dopo la quantità superiore alle sei unità inizia ad essere presente molto più frequentemente il diabete.



Infine, un altro indicatore molto forte del diabete è il glucosio nel sangue. Notiamo infatti che avvicinandoci a 150 iniziano ad esserci la maggior parte dei casi di diabete.

Apprendimento non supervisionato

L'apprendimento non supervisionato è una modalità di addestramento di modelli di machine learning in cui non forniamo etichette o istruzioni specifiche sui dati. Al contrario, lasciamo che l'algoritmo scopra da solo modelli, strutture o raggruppamenti nei dati.

K Means (Hard clustering)

Il K-Means è un algoritmo di clustering, che è una tecnica utilizzata nell'apprendimento non supervisionato per organizzare un insieme di dati in gruppi omogenei chiamati "cluster". L'obiettivo è quello di raggruppare gli elementi simili insieme. Dopo aver deciso il numero di cluster, (uno dei metodi migliori per farlo è il metodo del gomito, che vedremo dopo) l'algoritmo procede secondo questi passi:

1) L'algoritmo seleziona casualmente K punti dai dati come "centroidi iniziali". Questi centroidi rappresentano il centro di ciascun cluster che stiamo cercando di creare.
2) Ora, per ogni punto nei dati, l'algoritmo calcola la distanza tra quel punto e i centroidi iniziali. Il punto viene assegnato al cluster del centroide più vicino secondo la metrica scelta e il tipo di descrizione dei dati.
3) Una volta che tutti i punti sono stati assegnati a un cluster, l'algoritmo calcola nuovi centroidi per ciascun cluster, che sono calcolati come la media di tutti i punti nel cluster.
4) I passi 2 e 3 vengono ripetuti più volte fino a quando i centroidi smettono di cambiare in modo significativo o fino a quando viene raggiunto un numero massimo di iterazioni.

L'obiettivo del K-Means è quello di minimizzare la somma delle distanze tra i punti e i centroidi dei loro cluster rispettivi. In altre parole, stiamo cercando di creare cluster in cui i punti all'interno di ciascun cluster siano quanto più vicini possibile al centroide del loro cluster.

Strumenti utilizzati e decisioni di progetto

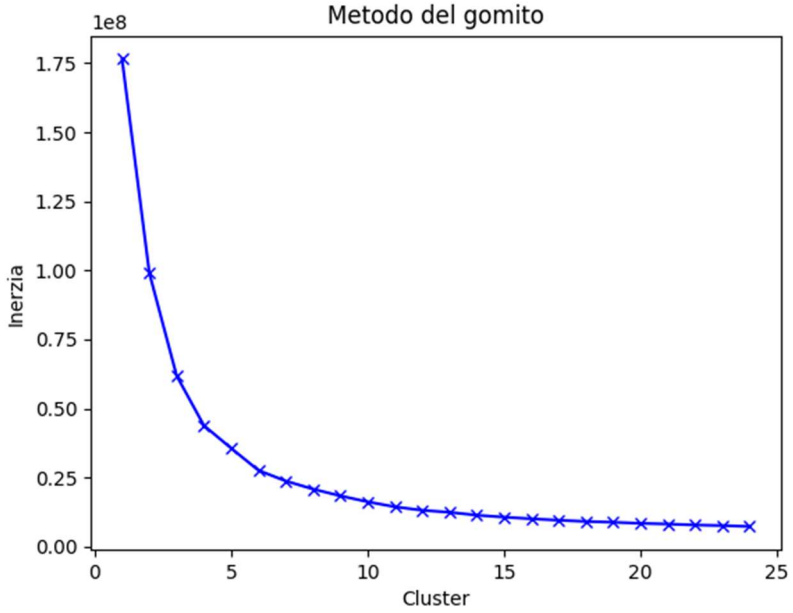
Per l'implementazione del modello K-Means è stata utilizzata la libreria SKLearn usando la classe KMeans. Per questo algoritmo è stato necessario normalizzare e standardizzare i dati (per avere media 0 e varianza 1). Per fare ciò è stato

utilizzato uno strumento dalla libreria SKLearn.preprocessing: lo StandardScaler.

Per la valutazione ho utilizzato delle metriche specifiche per questo tipo di modelli:

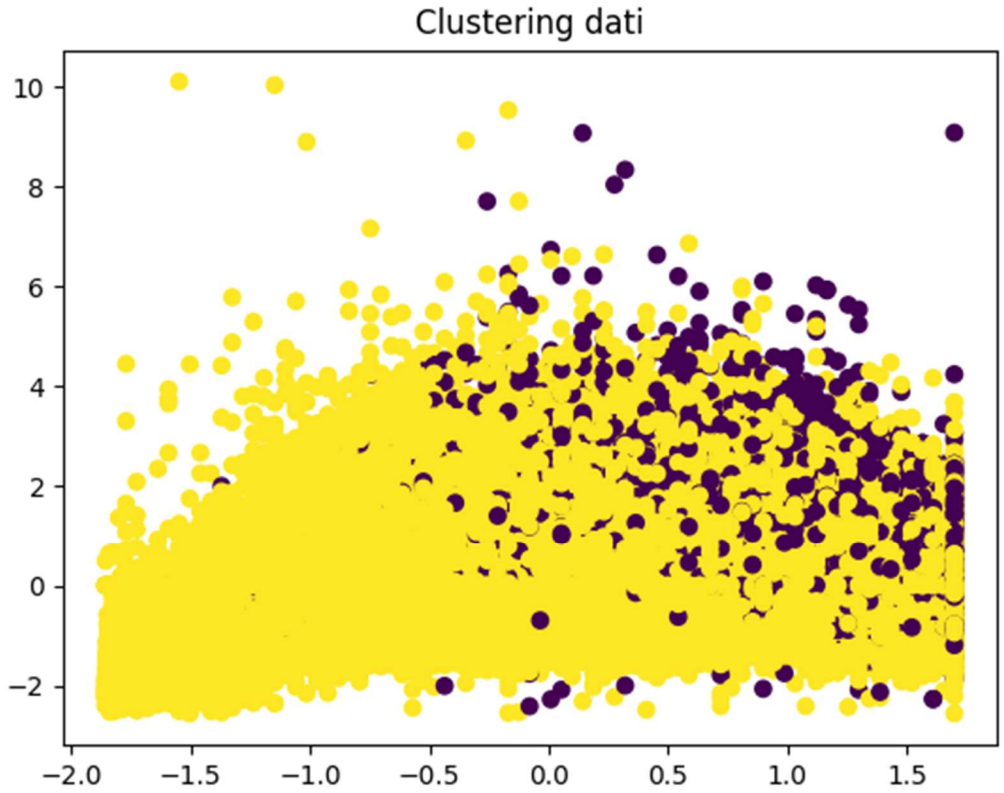
- Completezza: la completezza misura quanto bene tutti i membri di una stessa classe sono raggruppati insieme. La completezza può variare da 0 a 1, dove un valore più alto indica una migliore completezza.
- Omogeneità: L'omogeneità misura quanto bene tutti i membri di un cluster appartengono alla stessa classe. In pratica, un algoritmo di clustering è omogeneo quando ogni cluster contiene solo punti di dati appartenenti a una singola classe.
- Misura-v: La misura-v è una metrica che cerca di bilanciare completezza e omogeneità in un'unica misura. È la media armonica tra la completezza e l'omogeneità normalizzata.

Per iniziare, ho dovuto decidere il numero di cluster. Come ho già scritto uno dei metodi più utili per capire qual è un numero appropriato di cluster è il metodo del gomito, che ci dice quando l'aggiunta di ulteriori cluster non migliora significativamente la varianza all'interno dei cluster. Ciò si fa associando ad ogni numero di cluster possibile il valore di inerzia, come mostrato qui sotto.



L'inerzia è una misura della somma delle distanze quadrate tra i punti dati e i centroidi dei rispettivi cluster, quindi il nostro obiettivo è minimizzarla. Attraverso questo grafico individuiamo il "gomito" in corrispondenza del due.

Scelto il numero di cluster, ho sostituito l'inizializzazione randomica con l'approccio "k-means++", che seleziona i centroidi iniziali del cluster utilizzando il campionamento basato su una distribuzione di probabilità empirica del contributo dei punti all'inerzia complessiva, ottenendo i seguenti risultati:




```
Omogeneità (k-means++) :  0.982269167683695
Completezza (k-means++):  0.9749856026865589
Misura V  (k-means++) :  0.9786138329588479
```

Apprendimento supervisionato

L'apprendimento supervisionato è un tipo di approccio nell'ambito del machine learning in cui un algoritmo impara a fare previsioni basandosi su dati per l'addestramento che sono coppie di input e output corrispondenti. L'obiettivo dell'apprendimento supervisionato è quello di costruire un modello che possa generalizzare da questi esempi di addestramento, in modo da essere in grado di fare previsioni accurate su nuovi dati che non sono stati visti durante la fase di addestramento.

K Nearest Neighbors

L'algoritmo "K nearest neighbors" (abbreviato in k-nn) è un algoritmo di machine learning che si basa sull'apprendimento supervisionato. L'idea alla base di questo algoritmo è che entità simili sono vicine tra loro nello spazio delle descrizioni. Quando bisogna classificare un nuovo dato, si utilizzano i k "vicini meno distanti" e le loro categorie di appartenenza per stabilire come debba essere classificato il nuovo dato.

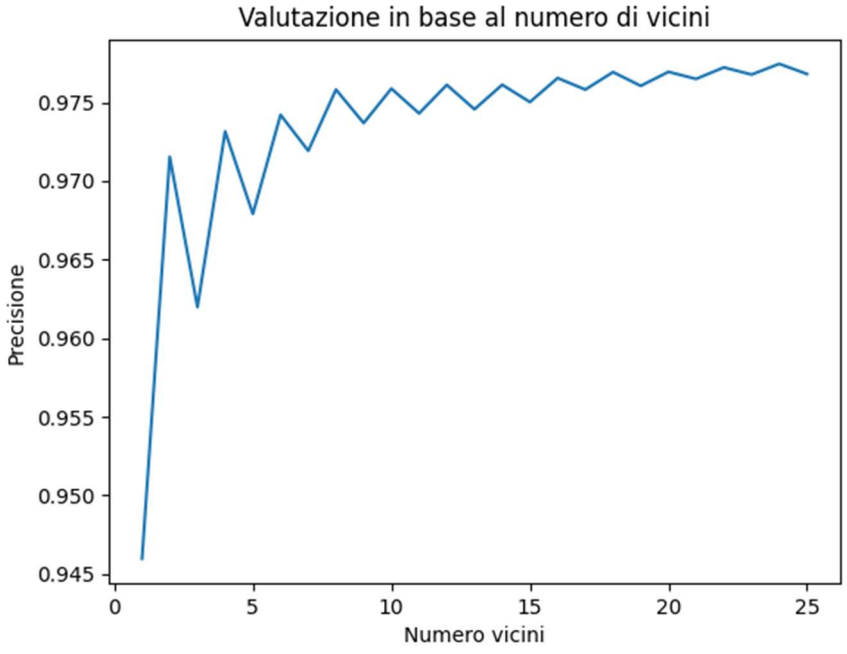
Strumenti utilizzati

Per l'implementazione del modello K Nearest Neighbors è stata utilizzata la libreria ScikitLearn usando la classe KNeighborsClassifier.

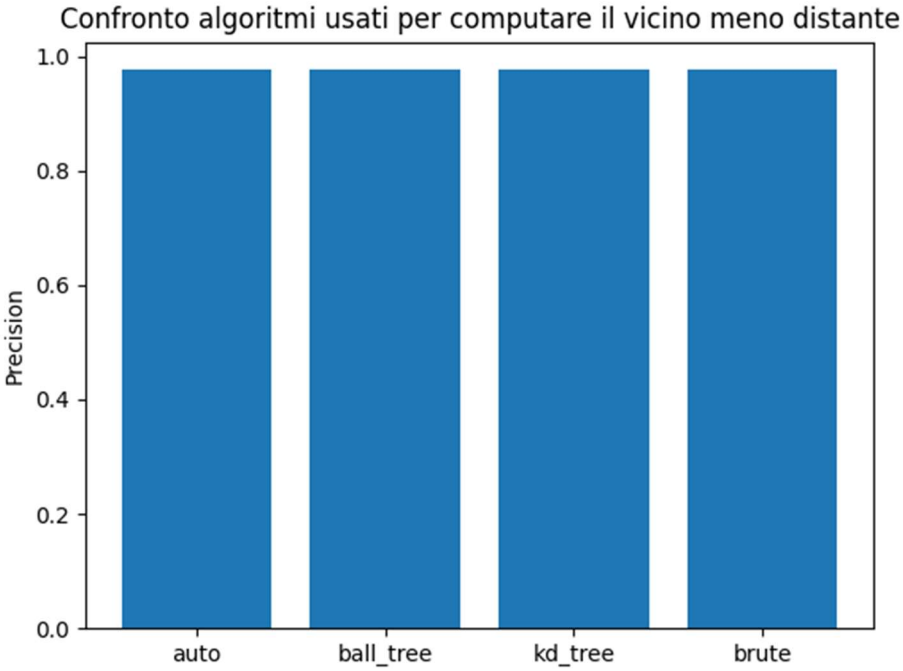
Decisioni di Progetto

Per costruire il modello di KNN migliore si è deciso di sperimentare con

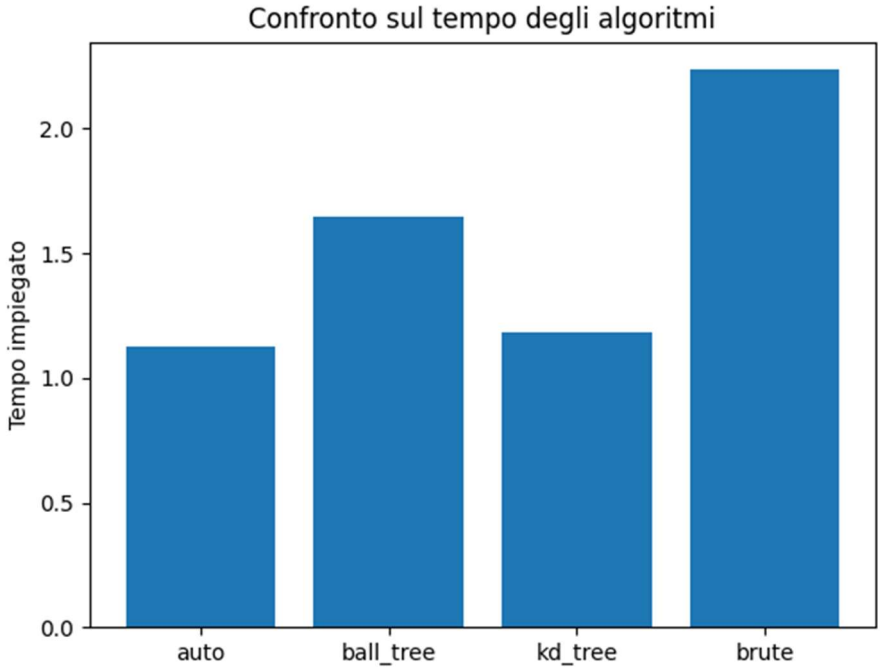
1. Il numero di vicini
2. Algoritmi usati per computare il vicino più simile



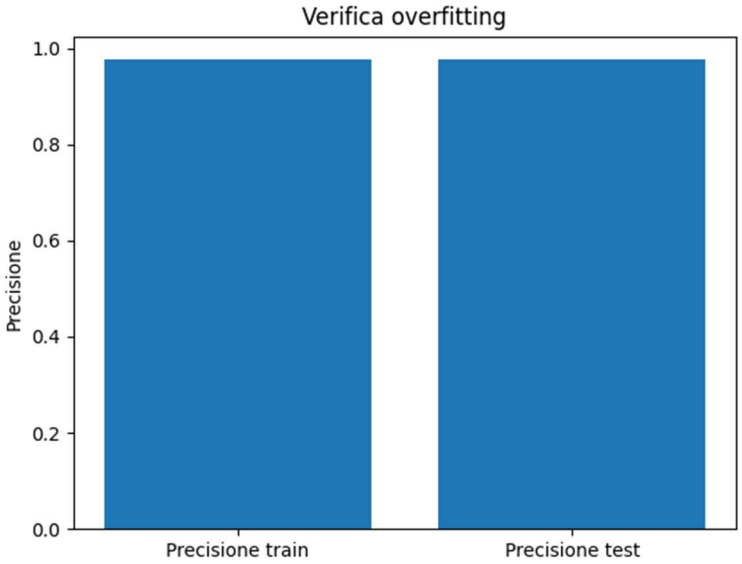
Ho deciso di proseguire con k = 10, in quanto da questo numero in poi si ha un miglioramento della precisione scarso a fronte di un calo delle performance discreto.



Gli algoritmi hanno performato tutti allo stesso modo, dunque ho deciso di confrontarli anche in base al tempo, ottenendo i seguenti risultati:



Essendo il tempo per l'addestramento e il test così breve, non c'è modo di stabilire realmente quale algoritmo è più efficiente per questo dataset, e ho proceduto quindi con la selezione automatica dell'algoritmo migliore: auto, che ha selezionato a sua volta kd_tree. Un KD-Tree (K-Dimensional Tree) è una struttura dati ad albero utilizzata per organizzare un insieme di punti nello spazio multidimensionale. La struttura del KD-Tree è simile a quella di un albero binario. Ogni nodo dell'albero rappresenta un punto nello spazio e ha associato un "piano" di divisione parallelo agli assi. In uno spazio bidimensionale come il mio, un nodo dividerà il piano in base a una coordinata x o y. I punti con valori più bassi rispetto a quella coordinata vengono collocati nel sottoalbero sinistro, mentre quelli con valori più alti vengono collocati nel sottoalbero destro. L'idea principale del KD-Tree è quella di dividere lo spazio multidimensionale in modo ricorsivo fino a quando ogni nodo contiene solo un singolo punto. In questo modo, la struttura dell'albero permette di organizzare i punti in modo da poter eseguire ricerche efficienti.



Sui classificatori ho verificato che la differenza di precisione sul training set e il test set non fosse troppo diversa, in modo da sapere se ci fosse il problema dell'overfitting.

Neural Network: Multi-layer Perceptron

Il Multi-layer Perceptron (abbreviato in MLP) è un tipo di modello utilizzato nell'ambito del machine learning, specificamente nel campo delle reti neurali. Un MLP è composto da una serie di strati:

- strati nascosti;
- strati di input;
- strati di output.

Ogni strato nascosto è composto da un insieme di neuroni, ognuno dei quali è connesso a tutti i neuroni nello strato precedente e successivo tramite pesi. Ogni connessione ha un peso associato che viene modificato durante il processo di addestramento del modello. Ogni neurone degli strati nascosti esegue una trasformazione lineare dei suoi input pesati seguita da una funzione di attivazione non-lineare. Questo processo permette all'MLP di apprendere relazioni nei dati.

Strumenti utilizzati e decisioni di progetto

Per l'implementazione del Neural Network è stata utilizzata la libreria Scikit learn usando la classe MLPClassifier. Inoltre, sapendo che il MLP è sensibile alla dimensione delle feature, ho deciso di standardizzare il vettore di input in modo da avere media 0 e varianza 1. Per ottenere risultati sensati è stato necessario effettuare la stessa operazione sul test set. Per fare ciò è stato utilizzato lo StandardScaler di Scikit-learn. Ecco i risultati sia senza, sia con standardizzazione.

Senza standardizzazione:

```
Neural network precision:  0.9626075848601736
Neural network accuracy:  0.95785
Neural network F1 Score:  0.9596975270280366
Seconds needed for train and test:  16.815346717834473
```

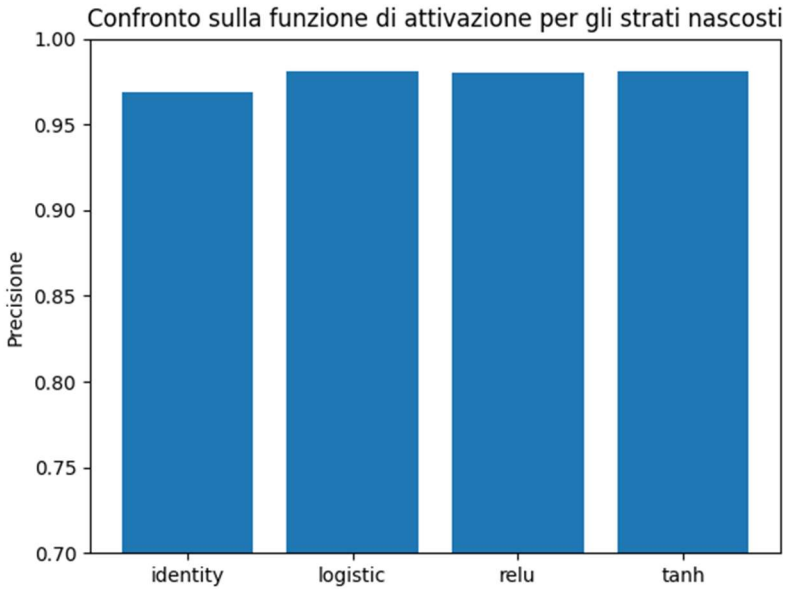
Con standardizzazione:

```
Neural network precision:  0.9682679191578271
Neural network accuracy:  0.96055
Neural network F1 Score:  0.9632406547215915
Seconds needed for train and test:  2.584480047225952
```

È facile notare il miglioramento, oltre che nelle metriche, specialmente nella quantità di tempo richiesta per effettuare addestramento e test.

Effettuato questo test, ho proceduto con il test degli iperparametri. Il costruttore del MLP durante ogni test è stato invocato impostando il parametro del numero massimo di iterazioni a 500, numero che permette di ottenere un aumento discreto dei risultati senza penalizzare troppo la performance. Per costruire il modello migliore si è deciso di sperimentare con i seguenti parametri:

1. Funzione di attivazione per gli strati nascosti
2. Dimensione della rete
3. Algoritmi per l'ottimizzazione dei pesi



Tra gli algoritmi per l'attivazione degli strati nascosti, 'identity' è stato il più rapido (vedere immagine sotto) ma anche quello che ha performato peggio (tuttavia con una differenza di precisione molto esigua).

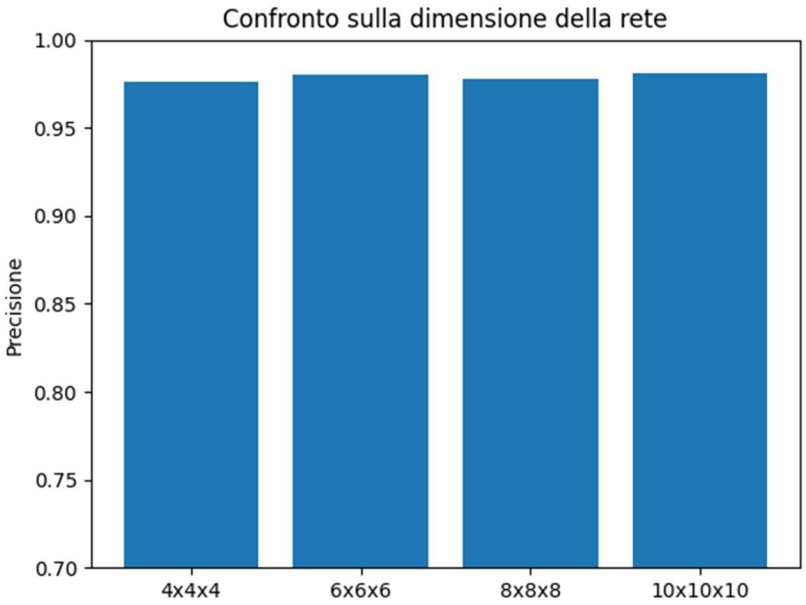
```
o--------------------------------------------------------------------------------o


Neural network ( identity  activation) precision:  0.9687409771777562
Neural network ( identity  activation) accuracy:  0.9602
Neural network ( identity  activation) F1 Score:  0.9631398437233866
Seconds needed for train and test:  2.090131998062134


Neural network ( logistic  activation) precision:  0.980946056412729
Neural network ( logistic  activation) accuracy:  0.97235
Neural network ( logistic  activation) F1 Score:  0.9747431837892511
Seconds needed for train and test:  47.070526123046875


Neural network ( relu  activation) precision:  0.9803461316297011
Neural network ( relu  activation) accuracy:  0.97205
Neural network ( relu  activation) F1 Score:  0.9743935834936498
Seconds needed for train and test:  10.729134798049927


Neural network ( tanh  activation) precision:  0.9808558000642881
Neural network ( tanh  activation) accuracy:  0.9721
Neural network ( tanh  activation) F1 Score:  0.9745419539535802
Seconds needed for train and test:  21.723828315734863

o--------------------------------------------------------------------------------o
```

A fronte di questi dati, ho deciso di proseguire con l'algoritmo 'relu', che a parità di punteggi è stato quello più rapido (rispetto a 'tanh' e 'logistic'). L'obiettivo principale di una funzione di attivazione è introdurre non linearità nella rete neurale, permettendo al modello di apprendere relazioni complesse nei dati.

La funzione ReLU (Rectified Linear Activation) è definita come:

$$ReLU(x) = Max(0, x);$$

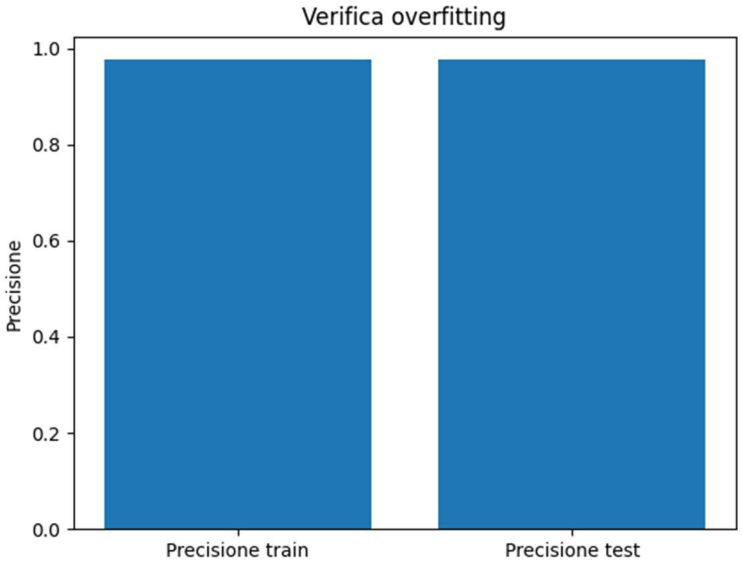
Questo significa che la funzione ReLU lascia passare tutti i valori positivi e imposta a zero i valori negativi. Quando un neurone riceve un input, applica la funzione ReLU all'input sommato con i pesi associati ai collegamenti entranti e, quindi, passa l'output al neurone successivo o all'output finale della rete.



Per quanto riguarda la dimensione della rete invece, già con 6x6x6 abbiamo raggiunto la precisione più alta, e dato che è sempre meglio optare per soluzioni più semplici (rasoio di Occam), ho proseguito con questa dimensione.

```
Neural network ( lbfgs  solver) precision:  0.9802473019929283
Neural network ( lbfgs  solver) accuracy:  0.97165
Neural network ( lbfgs  solver) F1 Score:  0.9740817893750094
Seconds needed for train and test:  32.91123557090759


Neural network ( sgd  solver) precision:  0.9783760353584056
Neural network ( sgd  solver) accuracy:  0.9709
Neural network ( sgd  solver) F1 Score:  0.9731142686314114
Seconds needed for train and test:  38.12906837463379
```

Infine, per l'ottimizzazione dei pesi, anche con 500 iterazioni l'algoritmo 'adam' non è riuscito a convergere, e quindi è stato scartato. Tra 'lbfgs' e 'sgd' invece, il primo ha performato leggermente meglio. L-BFGS (Limited-memory Broyden-Fletcher-Goldfarb-Shanno) è un ottimizzatore che opera nel modo seguente: calcola il gradiente della funzione di costo rispetto alle caratteristiche della rete neurale (il gradiente indica la direzione in cui la funzione di costo sta aumentando più rapidamente); l'obiettivo principale è minimizzare la funzione di costo, quindi l'algoritmo aggiorna i parametri del modello utilizzando il gradiente calcolato. La caratteristica chiave dell'algoritmo è la sua abilità di gestire le informazioni limitate sulla matrice Hessiana.



Come per il k-nn, ho verificato che non ci fosse un problema di overfitting.

Fatti questi esperimenti quindi ho definito il MLP con i parametri più adatti al mio problema e al mio dataset, secondo le osservazioni espresse precedentemente.

Random Forest

L'ensemble learning è una tecnica nell'ambito del machine learning in cui diversi modelli semplici (in questo caso degli alberi di decisione) vengono combinati insieme per migliorare le prestazioni generali e la capacità di generalizzazione rispetto a un singolo modello. L'idea fondamentale è che combinando le previsioni di più modelli diversi, si possono ottenere risultati migliori rispetto a quelli che ciascun modello potrebbe ottenere da solo. La "Random forest" è un metodo specifico di ensemble learning che combina l'approccio di bagging (costruzione di modelli indipendenti le cui previsioni vengono combinate tramite votazione) con l'utilizzo di alberi decisionali, che vengono addestrati su un subset casuale dei dati di addestramento (per garantire l'indipendenza).

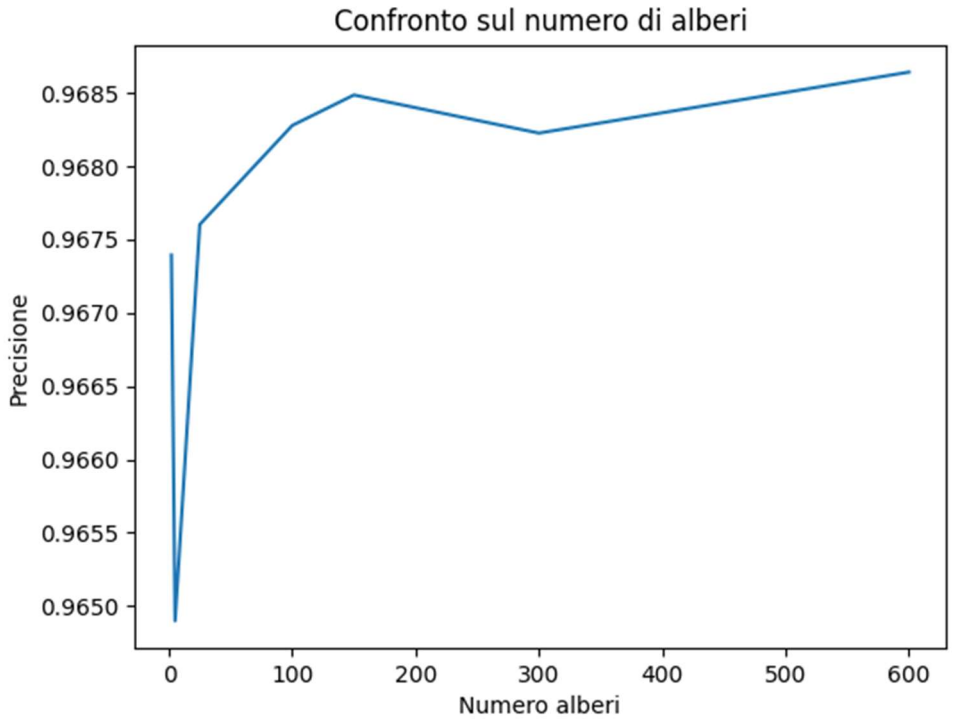
Strumenti utilizzati

Per l'implementazione del modello Random forest è stata utilizzata la libreria Scikit learn usando la classe RandomForestClassifier

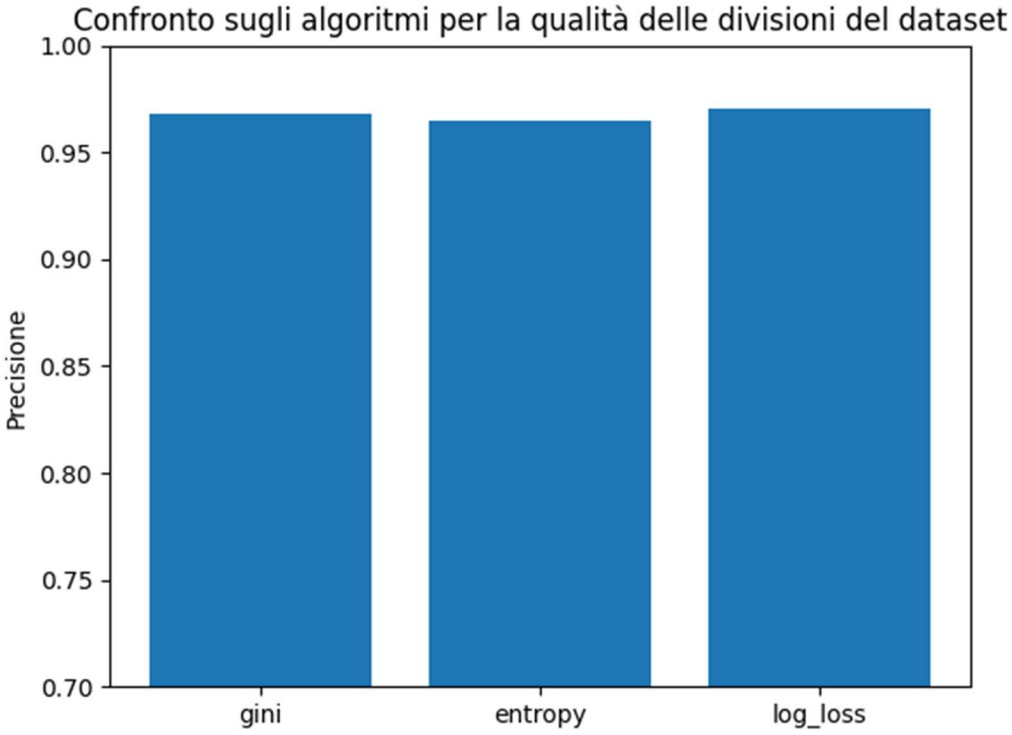
Decisioni di Progetto

Per costruire il modello di Random forest si è deciso di sperimentare con i seguenti parametri:

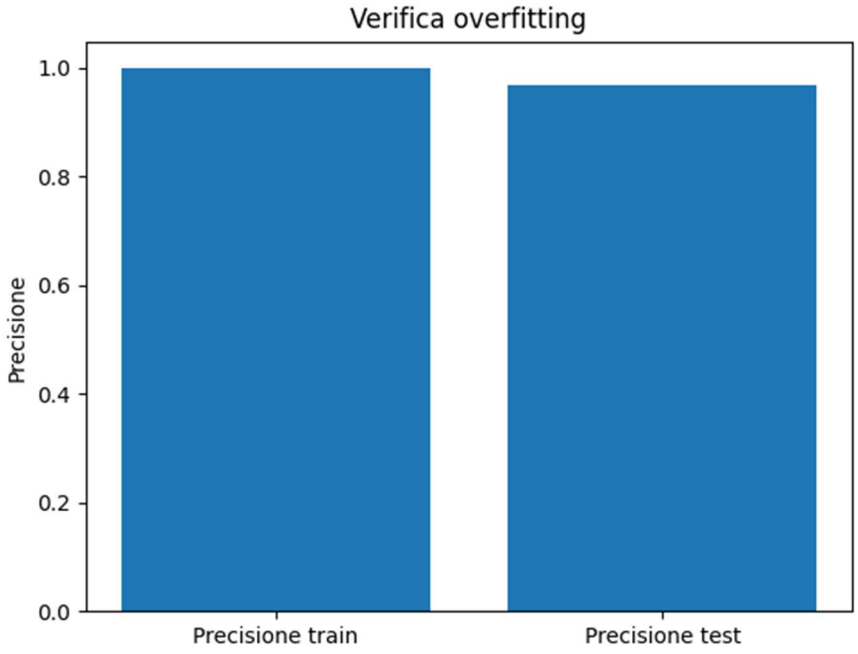
1. Il numero di alberi
2. Stimatori qualità dell'albero differenti



Il tempo richiesto per l'addestramento e il test è cresciuto linearmente con numero dii alberi, tuttavia la miglior precisione (seppur di pochi millesimi) si ha già a centocinquanta alberi.



Per quanto riguarda l'algoritmo per la qualità delle divisioni dei dati sugli alberi, log_loss si è dimostrato leggermente superiore agli altri. L'algoritmo di perdita logaritmica (log loss) non è specifico della Random Forest, ma è una metrica di valutazione comunemente utilizzata per misurare l'adattamento di modelli di classificazione probabilistica, compresi quelli utilizzati nelle Random Forest. La perdita logaritmica, anche nota come cross-entropy loss, è una misura della discrepanza tra le probabilità previste dal modello e le probabilità effettive delle classi di appartenenza dei campioni di addestramento.



Come per altri modelli anche qui ho eseguito un test per verificare che non ci fosse un problema di overfitting.

Gaussian Naive Bayes

Il Gaussian Naive Bayes è un algoritmo di classificazione basato sul teorema di Bayes, che utilizza l'assunzione di indipendenza condizionale tra le caratteristiche dei dati. È particolarmente utile quando si tratta di dati continui e si suppone che ogni caratteristica sia distribuita secondo una distribuzione gaussiana (normale). Sebbene qui l'assunzione di indipendenza non rispecchi la realtà, il modello ha comunque ottenuto risultati ottimi, come mostrato sotto.

Strumenti utilizzati

Per l'implementazione del Gaussian Naive Bayes è stata utilizzata la libreria Scikit learn usando la classe GaussianNB.

Decisioni di Progetto

Il costruttore del GaussianNB è stato invocato senza parametri, poiché non conosco la probabilità a priori che un paziente sia affetto da diabete.

```
Naive bayes precision:  0.8920492776805753
Naive bayes accuracy:  0.902028081123245
Naive bayes F1 Score:  0.894782146264842
```

Conclusioni

Il primo step per migliorare la qualità del programma, a parer mio, sarebbe quello di aggiungere moduli per il rilevamento di altre patologie. A quel punto collegando il programma ad un database di pazienti, o sotto richiesta di un utente, si potrebbe ricevere una lista dei rischi o delle patologie a cui è esposto il paziente in questione, e si potrebbe procedere con l'opinione di un medico ed eventualmente dei test medici che confermino o smentiscano i risultati ricevuti.