Università degli Studi di Bari Aldo Moro
Dipartimento di Informatica

Ingegneria della Conoscenza

Diagnosi del Diabete

Blanco Lorenzo

Matr. 784005

a.a 2024/2025

Link Repository GitHub : `https://github.com/blanco003/ICON`

Indice

Capitolo 1

Introduzione

Secondo la World Health Organization (WHO), un istituto specializzato dall'ONU per la salute, ci sono circa 830 milioni di persone in tutto il mondo che soffrono di diabete, e si stima che tale numero arriverà a 1.3 miliardi nel giro dei prossimi 30 anni. Il diabete è una malattia cronica caratterizzata dalla presenza di elevati livelli di glucosio nel sangue (iperglicemia) a causa di una alterata sintesi o funzione dell'insulina prodotta dal pancreas, un ormone che permette l'ingresso del glucosio nelle cellule. La diagnosi precoce del diabete è un fattore cruciale per un tempestivo ed adeguato trattamento, poichè la malattia danneggia i vasi sanguigni, causando il loro restringimento e riducendone l'afflusso di sangue, potrebbe portare a numerose complicazioni.

Il caso di studio si propone, attreverso l'applicazione di varie tecniche di Machine Learning, di sviluppare strumenti utili alla diagnosi del diabete, con l'obiettivo di migliorare la capacità di previsione e identificazione precoce della malattia basandosi sui dati clinici dei pazienti, in modo da fornire una tempestiva diagnosi e trattamento immediato per evitare ulteriori complicazioni sanitarie.

1.1 Strumenti utilizzati

Come linguaggio di programmazione si è scelto di usare **python**, data la facilità di scrittura e l'enorme vastità di librerie a disposizione. Per l'esecuzione dei vari esperimenti affrontati, date le enormi risorse computazionali richieste, è stato usato Google Colab, mentre per quanto riguarda le librerie utilizzate, saranno descritte nel seguito mentre verranno utilizzate.

Capitolo 2

Data Preprocessing

2.1 Informazioni sul Dataset

Il dataset preso in esame nel caso di studio, `diabetes_prediction_dataset.csv`, è stato trovato su Kaggle, e dispone di 100 000 campioni rappresentanti pazienti reali con 9 variabili cliniche.

In particolare, per ogni paziente, sono riportate le seguenti caratteristiche:

- **gender**: Indica il sesso del paziente `("Male", "Female", "Other")`
- **age**: Età del paziente espressa in anni.
- **bmi**: Indice di Massa Corporea misurato in base al peso e all'altezza. Valori BMI più elevati sono collegati a un rischio più elevato di diabete.
- **HbA1c_level**: Il livello di HbA1c (emoglobina A1c) è una misura del livello medio di zucchero nel sangue negli ultimi 2-3 mesi, espresso in percentuale. Livelli più elevati indicano un rischio maggiore di sviluppare il diabete
- **blood_glucose_level**: Livello di glucosio nel sangue misurato al momento dell'esame, espresso in mg/dL. Livelli elevati di glucosio nel sangue sono un indicatore chiave del diabete.
- **hypertension**: Indica se il paziente soffre di ipertensione, una condizione medica in cui la pressione sanguigna nelle arterie è costantemente elevata (`1` se presente, `0` se assente)
- **heart_disease**: Indica se il paziente ha malattie cardiache. Le malattie cardiache sono una condizione medica associata ad un aumentato rischio di sviluppare il diabete. (`1` se presente, `0` se assente)

- **smoking_history**: Abitudine al fumo del paziente (`"never", "not current", "current", "former", "ever" , "No Info"`). Anche il fumo è un fattore di rischio per il diabete e può peggiorare le sue complicanze.
- **diabetes**: Indica se il paziente è affetto da diabete, una condizione cronica in cui il corpo fatica a regolare lo zucchero nel sangue. (`1` se affetto, `0` se non affetto)

2.2 Pre Processing del dataset

Il dataset è stato inizialmente processato come segue :

1. **Controllo valori nulli** : il dataset non presenta valori nulli.
2. **Controllo valori duplicati** : il dataset contiene 3854 righe duplicate. Per evitare di introdurre bias nell'apprendimento e sbilanciamento nei dati sono state eliminate.
3. **Mapping** : riguardo la feature `smoking_history`, per evitare sparsità nelle categorie e un aumento della complessità, è stata mappata in 3 categorie : {`"never", "No info" : "non-smoker"` ; `"current" : "current"` ; `"ever", "former", "not current": "past-smokder"` }
4. **Encoding** : per rendere possibile l'uso delle features categoriche (`gender, smoking_history`) con i modelli di apprendimento, è stato effettuato l'encoding, in particolare il **One Hot Encoding**, con il quale ogni valore assumibile da una feature viene rappresentato come colonna binaria. Questa tecnica evita che il modello di apprendimento interpreti erroneamente la relazione tra i numeri.
5. **Scaling** : lo scaling delle features numeriche sarà eseguito in seguito, in base al particolare task che si sta svolgendo.

Capitolo 3

Apprendimento Supervisionato

L'apprendimento supervisionato è una branca Machine Learning, in cui un modello viene addestrato su un insieme di esempi, descritti da features, partizionate in features di input e target, con l'obiettivo di predire i valori delle features target per nuovi esempi date le sole features di input. Questo significa che per ogni esempio è già presente la corrispondente feature target corretta, il cosidetto ground truth (che nei task di classificazione è un etichetta, mentre nei task di regressione è un valore reale), che il modello utilizza per apprendere le relazioni sottostanti tra i dati disponibili. L'obiettivo è quello di fare previsioni accurate su nuovi dati mai visti prima.

Nel caso di studio l'apprendimento supervisionato è stato utilizzato con scopo di classificazione, in particolare nel classificare se un paziente è affetto da diabete.

3.1 Metodologia

Modelli di Apprendimento Supervisionato

Per svolgere il task di apprendimento supervisionato, si è scelto di prendere in considerazione due modelli base, il Decision Tree e la Logistic Regression, e due modelli di ensambling, di cui uno di bagging, il Random Forest, ed uno di boosting, il Gradient Boosting Classifier, in particolare :

- **Decision Tree** : Classificatore ad albero in cui le foglie rappresentano le classi di appartenenza, mentre la radice ed i nodi interni rappresentano delle condizioni sulle features di input. A seconda se tali condizioni sono rispettate o meno, verrà seguito un percorso piuttosto che un altro e alla fine si arriverà alla foglia, che darà la classificazione.

- **Random Forest** : Classificatore che si ottiene creando numerosi Decision Tree indipendenti, da cui la classificazione finale sarà data dalla maggioranza delle classi predette dai vari Decision Tree.

- **Logistic Regression** : Modello statistico con il quale la probabilità di appartenenza ad una classe viene modellata attraverso la funzione sigmoide, che trasforma una combinazione lineare delle features in un valore compreso tra 0 e 1. Il valore risultante viene quindi sogliato per assegnare l'osservazione ad una delle due classi.

- **Gradient Boosting Classifier** : Modello ensemble che combina diversi Decision Tree in modo sequenziale, dove in particolare ogni nuovo albero cerca di correggere gli errori commessi dai precedenti, pesando maggiormente gli esempi difficili più da classificare. Il processo di apprendimento si basa sulla minimizzazione della funzione di perdita tramite il gradiente, migliorando progressivamente la performance del modello.

Per l'implementazione, si sono sfruttati i modelli offerti dalla libreria `sklearn`.

Ricerca degli iper-parametri ottimali

Per la scelta degli iper-parametri si è utilizzata la tecnica `K-Fold Cross Validation`, con la quale il dataset viene diviso in k fold (insiemi disgiunti) e il modello viene addestrato k volte, dove per ogni iterazione vengono usati $k - 1$ fold per il training ed il restante fold per il testing. In questo modo è possibile testare e addestrare il modello su dati diversi per comprendere la bontà del modello. In combinazione, si è utilizzata la tecnica `GridSearch` per ricercare i migliori iper parametri dei modelli tra le varie configurazioni, in particolare consiste nel definire una griglia di tutte le possibili combinazioni tra i valori possibili di ogni iper-parametro, ed in seguito ogni possibile combinazione viene valuta, una per volta, tramite la CV. Infine sarà selezionata la combinazione che ha ottenuto la migliore valutazione per la costruzione dei modelli di apprendimento.

Iper-parametri

Gli iper-parametri sono parametri di un modello di apprendimento supervisionato, che devono essere scelti e fissati necessariamente prima della fase di addestramento del modello, essendo che non vengono. Tale scelta consiste in una fase cruciale dell'apprendimento, in quanto essi andranno ad incidere sulla complessità e sull'accuratezza delle predizioni del modello.

Gli iper-parametri considerati nella ricerca sono i seguenti :

- **Decision Tree** :

  - **criterion** : misura della qualità dello spit effettuato sui nodi : `["gini", "entropy"]`, in particolare `gini` misura quanto spesso un elemento viene classificato in modo sbagliato, mentre `entropy` misura la quantità di disordine nei dati

  - **max_depth** : altezza massima dell'albero : `[7, 10, 12]`
  - **min_samples_split** : numero minimo di esempi necessari affinché possa essere inserito un criterio di split, altrimenti se il numero è minore viene innestata una foglia : `[8, 10, 15]`
  - **min_samples_leaf** : numero minimo di esempi per poter creare una foglia : `[5,7, 10]`
  - **splitter** : strategia da utilizzare per il criterio di split, in particolare si è scelto e fissato `best`, che indica il miglior criterio di split possibile.

- **Random Forest** :

  - **n_estimators** : il numero di Decision Tree nella foresta : `[50,100,200]`
  - **criterion** : misura della qualità dello spit effettuato sui nodi : `["gini", entropy"]`, in particolare `gini` misura quanto spesso un elemento viene classificato in modo sbagliato, ed `entropy` misura la quantità di disordine nei dati
  - **max_depth** : altezza massima degli alberi di decisione : `[5, 10,15]`
  - **min_samples_split** : numero minimo di esempi necessari affinché possa essere inserito un criterio di split, altrimenti se il numero è minore viene innestata una foglia : `[5, 10, 15]`
  - **min_samples_leaf** : numero minimo di esempi per poter creare una foglia : `[3, 5, 10]`

- **Logistic Regression** :

  - **C** : Specifica quanto incide la regolarizzazione, in particolare più il valore è basso e più forte è la regolarizzazione : `[0.001, 0.01, 0.1, 1, 10, 100]`
  - **penalty** : tipo di penalizzazione usata, in particolare si è scelto e fissato l_2, la norma 2, ovvero la somma dei quadrati dei pesi, che favorisce pesi più bilanciati, e quindi risulta utile per prevenire l'overfitting.
  - **solver** : l'algoritmo utilizzato per l'apprendimento dei pesi : `["lbfgs", "liblinear"]`, in particolare `lbfgs` mantiene solo alcune informazioni delle derivate parziali risparmiando quindi molta memoria, mentre `liblinear` si basa sulla programmazione lineare
  - **max_iter** : il numero massimo di iterazioni : `[100000,150000]`

- **Gradient Boosting** :

  - **n_estimators** : il numero di Decision Tree usati nella sequenza, in particolare un numero maggiore di alberi può migliorare le prestazioni, ma aumenta il rischio di overfitting se non combinato con un adeguato `learning_rate` : `[50, 100, 150]`

- **learning_rate** : determina la velocità con cui il modello apprende, in particolare valori più bassi riducono il rischio di overfitting ma richiedono più alberi per ottenere buone prestazioni, al contrario valori alti possono rendere l'addestramento più veloce, ma aumenta il rischio di overfitting : `[0.01, 0.05, 0.1]`
- **max_depth** : altezza massima degli alberi di decisione : `[5, 7, 10]`
- **min_samples_split** : numero minimo di esempi necessari affinchè possa essere inserito un criterio di split, altrimenti se il numero è minore viene innestata una foglia : `[5, 8, 12]`
- **min_samples_leaf** : numero minimo di esempi per poter creare una foglia : `[3, 5, 7, 10]`

Fase di training e testing

Trovata la configurazione migliore degli iper-parametri, i modelli sono stati addestrati utilizzando la tecnica `Repeated K-Fold Cross Validation` con `n_splits` ed `n_repeats` pari a 5, dove `n_splits` indica il numero di fold in cui viene diviso il dataset (dataset diviso in 5 fold, e vengono effettuate 5 iterazioni dove 4 fold sono usati il training ed il restante come test fold a rotazione), ed `n_repeats` indica quante volte viene ripetuto l'intero processo.

Valutazione

Per la valutazione delle performance dei modelli addestrati si è deciso di utilizzare le seguenti metriche :

- **Accuracy** : misura generale della correttezza del modello, che rappresenta il rapporto tra il numero previsioni corrette ed il numero totale delle previsioni.
- **Precision Macro** : media aritmetica delle precisioni calcolate per ogni classe, dove la precisione per ogni classe c_i è definita come il rapporto tra il numero di istanze di classe c_i classificate correttamente con c_i, ed il numero totale di istanze classificate come c_i
- **Recall Macro** : media aritmetica delle recall calcolate per ogni classe, dove la recall per ogni classe c_i è definita come il rapporto tra il numero di istanze appartenenti alla classe c_i che sono state correttamente classificate con c_i ed il numero totale di istanze appartenenti a c_i
- **F1 Macro** : media armonica delle F1 calcolate per ogni classe, dove la f1 per ogni classe è definita come il rapporto tra 2 moltiplicato per la precision e per il recall, e la somma tra precision e recall.

Tuttavia tale metriche potrebbero, in alcuni casi speciali, essere ingannevoli, pertanto se considerassimo solo esse potremmo giungere a conclusioni errate. A tale scopo si è preso anche in considerazione :

- **Varianza e deviazione standard** : misurano la dispersione dei dati, in particolare, la varianza indica quanto le predizioni del modello si discostano dalla media delle predizioni stesse, mentre la deviazione standard è la radice quadrata della varianza e fornisce una misura della dispersione delle predizioni rispetto alla media.

- **Curve di apprendimento** : raffigurano l'andamento degli errorri commessi in fase di training e di test, e aiutano a identificare situazioni di underfitting, overfitting, o un buon apprendimento

.

Un modello di apprendimento con varianza e deviazione standard più basse è preferibile, ciò indica che le predizioni del modello sono più coerenti e meno sparse. In altre parole, il modello è più stabile e le sue prestazioni sono meno soggette a variazioni casuali. Questo significa che le sue predizioni sono affidabili e che il modello è meno incline a overfitting, ovvero a catturare rumore nei dati di addestramento piuttosto che le caratteristiche generali dei dati, e quindi più capace a generalizzare meglio su nuovi dati non visti.

3.2 Preparazione del dataset

Per quanto riguarda l'apprendimento dei modelli Decision Tree, Random Forest e Gradient Boosting, il dataset non necessità di modifiche ed è già pronto per l'apprendimento, in quanto tali modelli effettuano gli split direttamente sui valori delle features e non sono sensibili a diverse scale di valori.

Al contrario, essendo che il modello Logistic Regression è basato sull'assegnamento di diversi pesi alle feature, è pertanto sensibile alle scale dei valori usate per le featurese, in particolare se le feature hanno scale molto diverse, può succedere che alcune domino il processo di apprendimento, influenzando il modello e rendendo difficile far apprendere al modello le relazioni corrette. Per evitar ciò, è stata effettuata la **Normalizzazione** dei valori delle features. Nello specifico, la normalizzazione si occupa di portare tutti i valori nell'intervallo $[0, 1]$, effettuando la seguente trasformazione ;

$$x' = \frac{x - \min(x)}{\max(x) - \min(x)}$$

Per effettuare la normalizzazione è stato usato, grazie alla libreria sklearn. preprocessing, l'oggetto **MinMaxScaler**.

3.3 Primo esperimento

Iperparametri trovati con GridSearch :

Modello	Parametro	Valore
Decision Tree	criterion	entropy
	max_depth	10
	min_samples_split	8
	min_samples_leaf	10
Random Forest	n_estimators	100
	max_depth	15
	min_samples_split	10
	min_samples_leaf	10
	criterion	gini
Logistic Regression	C	0.1
	penalty	l2
	solver	liblinear
	max_iter	100000
Gradient Boosting Classifier	n_estimators	100
	learning_rate	0.01
	max_depth	10
	min_samples_split	12
	min_samples_leaf	5

Table 3.1: *Tabella degli iper-parametri trovati con Grid Search*

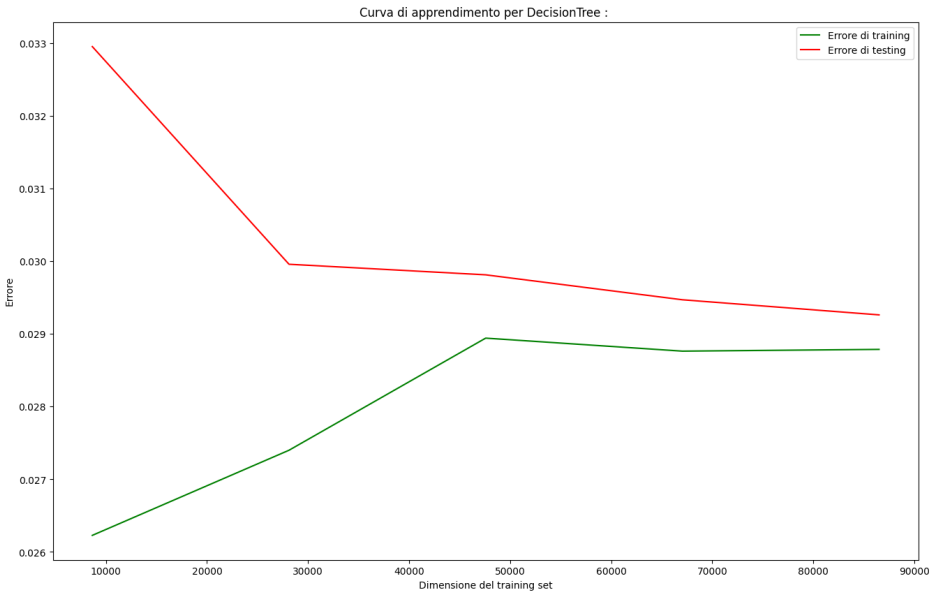


Figure 3.1: *Curva di Apprendimento Decision Tree*

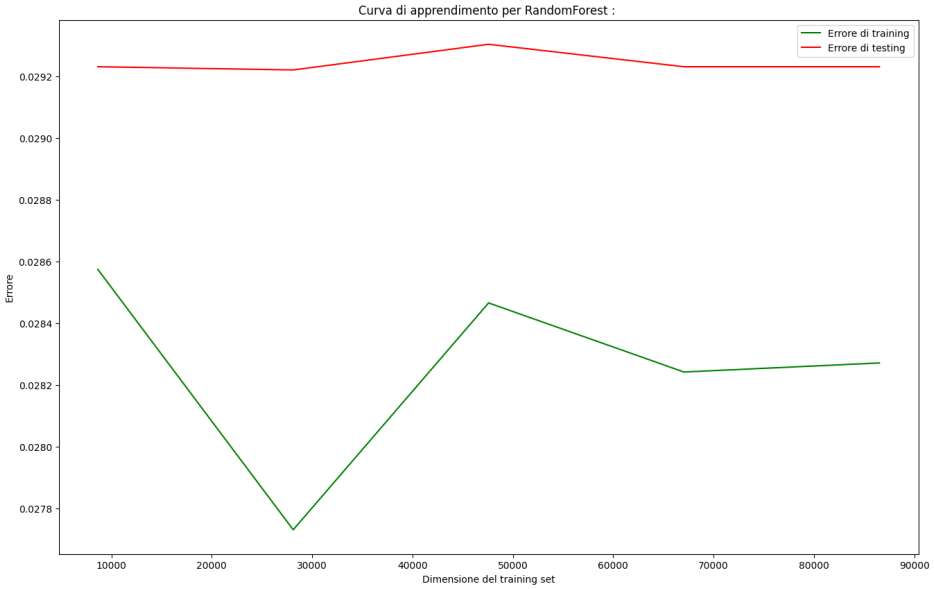


Figure 3.2: *Curva di Apprendimento Random Forest*

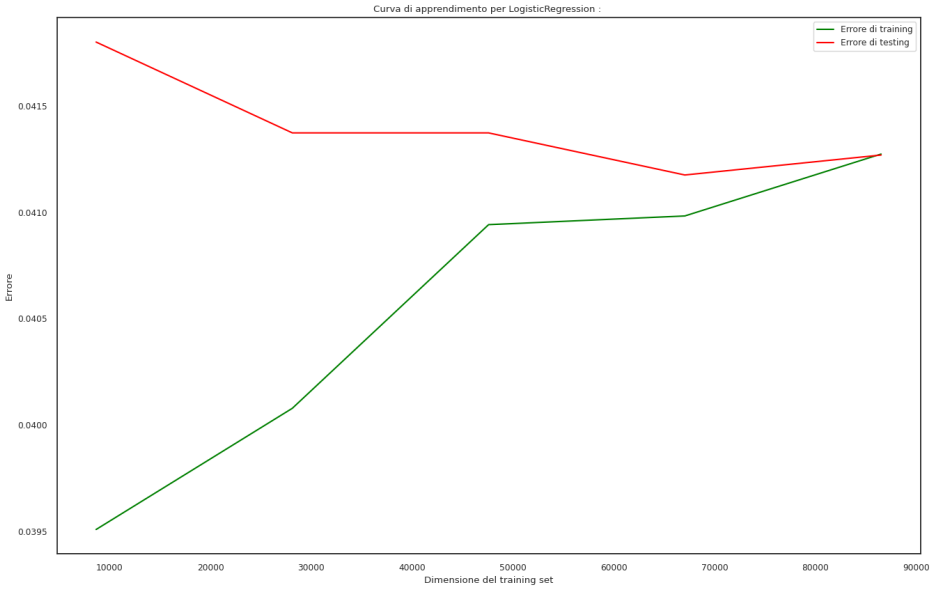


Figure 3.3: *Curva di Apprendimento Logistic Regression*

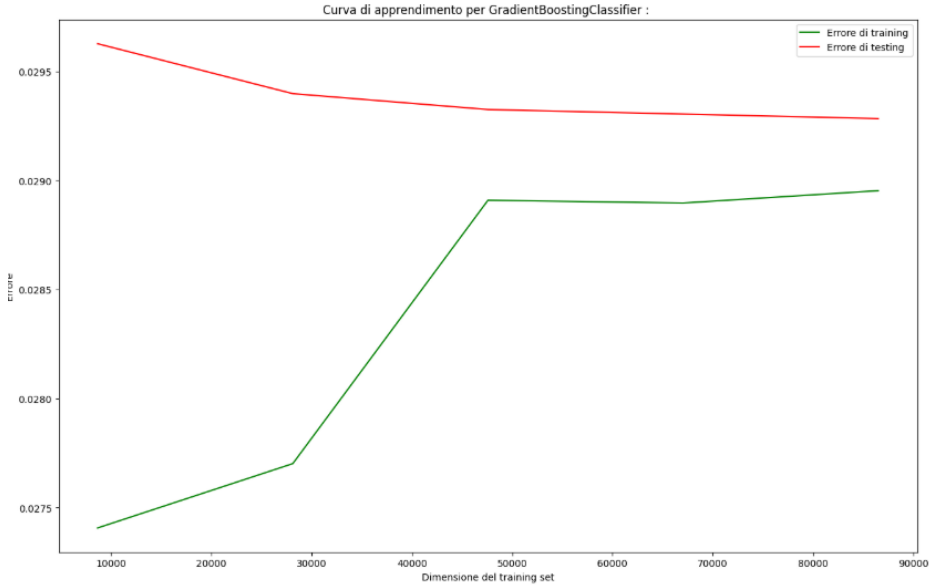
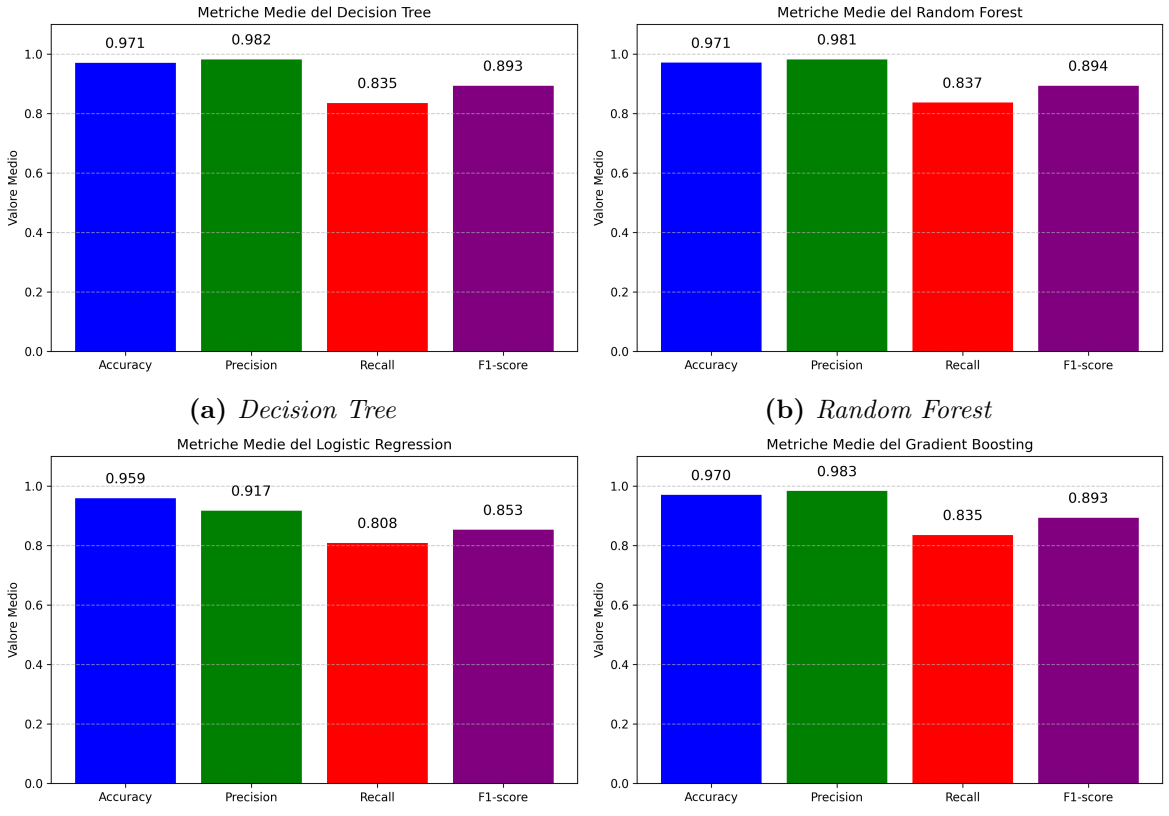


Figure 3.4: *Curva di Apprendimento Gradient Boosting Classifier*



(a) *Decision Tree* **(b)** *Random Forest*

(c) *Logistic Regression* **(d)** *Gradient Boosting*

Figure 3.5: *Metriche di valutazione*

Modello	Train Error Std	Test Error Std
Decision Tree	0.0001799	0.0013069
Random Forest	0.0001560	0.0013166
Logistic Regression	0.0001739	0.0012556
Gradient Boosting	0.0001815	0.0013508

Table 3.2: *Confronto Deviazione Standard dell'Errore per i Modelli*

Modello	Train Error Var	Test Error Var
Decision Tree	3.2390 e-08	1.7080 e-06
Random Forest	2.4354 e-08	1.7336 e-06
Logistic Regression	3.0249 e-08	1.5765 e-06
Gradient Boosting	3.2958 e-08	1.8249 e-06

Table 3.3: *Confronto Varianza degli Errori per i Modelli*

3.3.1 Considerazioni

N.B: si osservi che nei grafici rappresentanti le curve di apprendimento, le scale dell'errore (asse y) sono diverse.

Decision Tree :

- l'Accuracy e la Precision sono molto alte, tuttavia il Recall è più basso, ciò indica che il modello ha difficoltà nel riconoscere la classe minoritaria.
- la curva di training inizia con una salita abbastanza rapida ed infine tende a stabilizzarsi, mentre la curva di testing dopo una breve fase di diminuizione tende a stabilizzarsi e scendere molto lentamente. Le due curve si avvicinano, quindi sembra che il modello cominci a generalizzare bene.

Random Forest :

- l'Accuracy e la Precision sono molto alte, tuttavia il Recall, seppur si ha un leggero miglioramento rispetto al Decision Tree, è ancora relativamente basso per la classe minoritaria, ciò indica che il modello ha difficoltà nel riconoscere la classe minoritaria.
- l'errore di training dopo una prima fase di discesa e risalta tende a stabilizzarsi, mentre l'errore di testing è più alto, ma non in una fase di chiara diminuizione. La distanza tra le due curve è abbastanza ampia, ma esse sono in direzione di leggera convergenza, ciò suggerisce che il modello potrebbe beneficiare con l'aggiunta di nuovi dati.

Logistic Regression :

- l'Accuracy seppur alta è inferiore al Decision Tree e Random Forest, ed anche la Precision e Recall sono più basse rispetto agli altri modelli, di conseguenza anche la F1. La scarsa capacità predittiva sulla classe minoritaria è ancora più significativa in questo modello.

- l'errore di training è leggermente più alto rispetto agli altri modelli, segnale che la Logistic Regression ha meno capacità di adattarsi ai dati, e di conseguenza a generalizzare.

- dopo una fase iniziale, di salita della curva di training e discesa della curva di testing, restano abbastanza stabili, fino a salire entrambe leggermente, per l'errore di training ciò può essere dovuto ad una leggera difficoltà ad adattarsi ai dati di training, mentre per quello di testing, seppur meno evidente, potrebbe essere dovuto ad una leggera difficoltà nel classificare dati mai visti prima.

Gradient Boosting :

- l'accuracy e precision sono molto alte, e leggermente migliori rispetto agli altri modelli, mentre il recall rimane ugualmente più basso, segnale di una difficoltà nel riconoscere la classe minoritaria

- la deviazione standard dell'errore di training è leggermente più alta rispetto agli altri modelli, essendo un modello più sensibile ai dati di training perché apprende in modo sequenziale, correggendo continuamente gli errori, mentre quella di testing è relativamente più bassa.

- la curva di training dopo una salita iniziale resta stabile, mentre la curva di testing scende in maniera molto graduale, la distanza e l'andamento delle curve, sono un indice di buona generalizzazione, che potrebbe ulteriormente migliorare con nuovi dati disponibili.

Nessuno dei modelli presenta un evidente underfitting o overfitting.

A prima vista le metriche osservate potrebbero risultare sorprendenti, tuttavia è necessario effettuare un'analisi più accurata. Infatti, osservando con più attenzione il dataset, si è notato uno sbilanciamento abbastanza evidente nella distribuzione della feature target `diabetes`, come evince il seguente grafico :

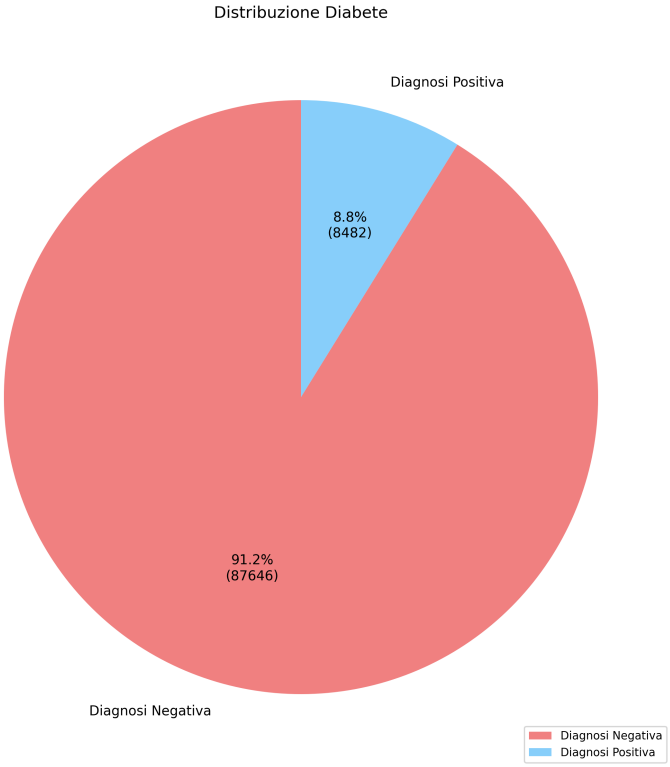


Figure 3.6: *Distribuzione Target*

Ciò potrebbe portare a diversi problemi :

- i modelli di apprendimento supervisionato potrebbero tendere a predire sempre la classe maggioritaria, ottenendo un'alta accuratezza senza sforzarsi ad apprendere la classe minoritaria, anche detti modelli "dumb"
- le metriche di valutazione dei modelli appressi potrebbero essere ingannevoli, facendo sembrare i modelli più accurati di quanto lo siano veramente
- i modelli di apprendimento supervisionato, non ricevendo abbastanza esempi della classe minoritaria, potrebbero non essere in grado di apprendere a fondo le caratteristiche di tale classe
- i modelli di apprendimento supervisionato, potrebbero andare in overfitting con scarse capacità di generalizzazione, fallendo su nuovi dati che presentano la classe minoritaria, pertanto si avrebbe una scarsa performance su test set

A tal scopo si è effettuato il bilanciamento degli esempi rispetto alla feature target, generando un numero sintetici di esempi, aventi valore della feature target `diabetes` della classe con minoritaria (Diagnosi Positiva), pari al numero di esempi necessari per bilanciare perfettamente le due classi ed avere una distribuzione degli esempi con

50% aventi diagnosi positiva e 50% aventi diagnosi negativa, grazie alla tecnica di oversampling **Smote**, della libreria `imblearn.over_sampling`

A seguito dell'OverSampling della classe minoritaria (pazienti con diagnosi positiva), gli esempi sono così distribuiti rispetto al target :

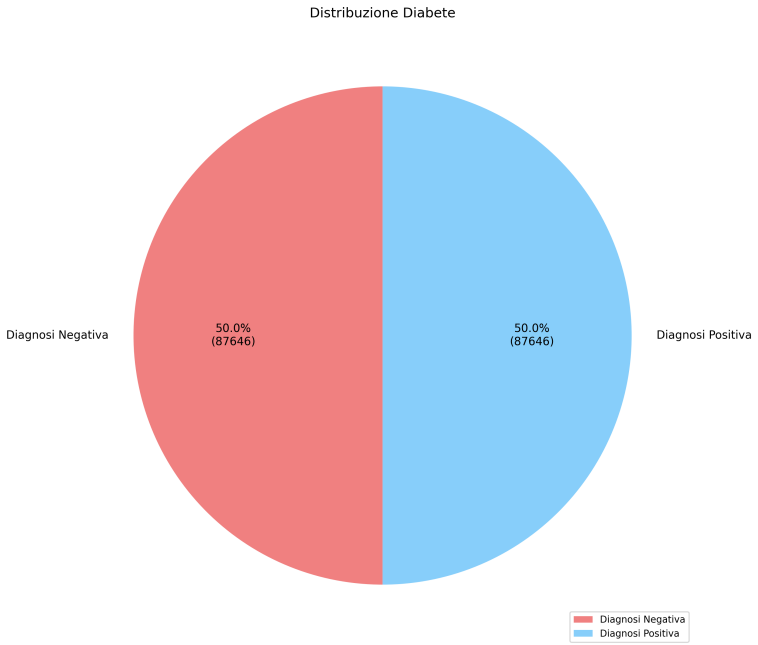


Figure 3.7: *Distribuzione Target con OverSampling*

3.4 Secondo Esperimento

Iperparametri trovati con GridSearch (è possibile osservare alcuni cambiamenti):

Modello	Parametro	Valore
Decision Tree	criterion	gini
	max_depth	12
	min_samples_split	15
	min_samples_leaf	5
Random Forest	n_estimators	100
	max_depth	15
	min_samples_split	15
	min_samples_leaf	13
	criterion	gini
Logistic Regression	C	1
	penalty	l2
	solver	lbfgs
	max_iter	100000
Gradient Boosting Classifier	n_estimators	150
	learning_rate	0.01
	max_depth	10
	min_samples_split	8
	min_samples_leaf	7

Table 3.4: *Tabella degli iper parametri trovati con Grid Search con Over-Sampling*

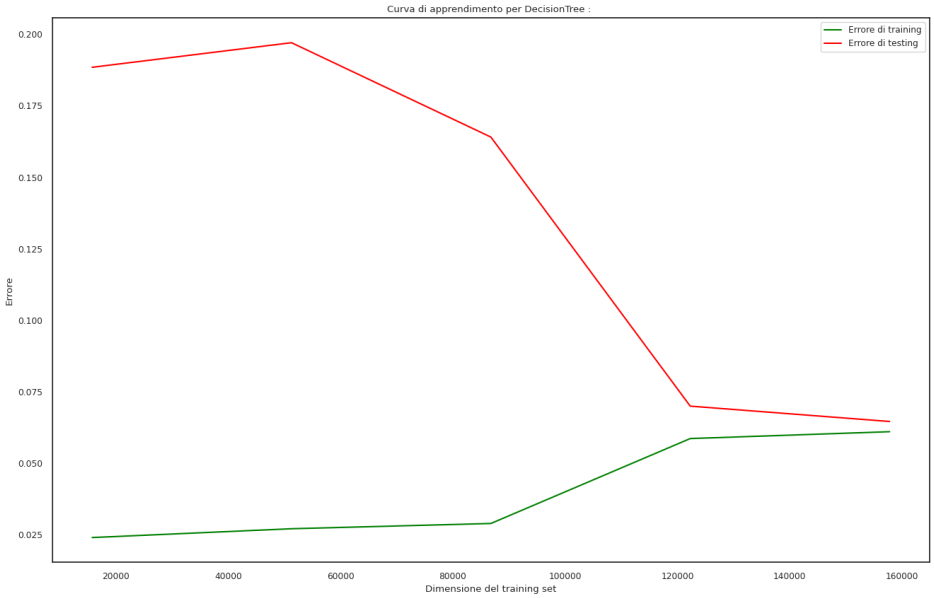


Figure 3.8: *Curve di Apprendimento Decision Tree con OverSampling*

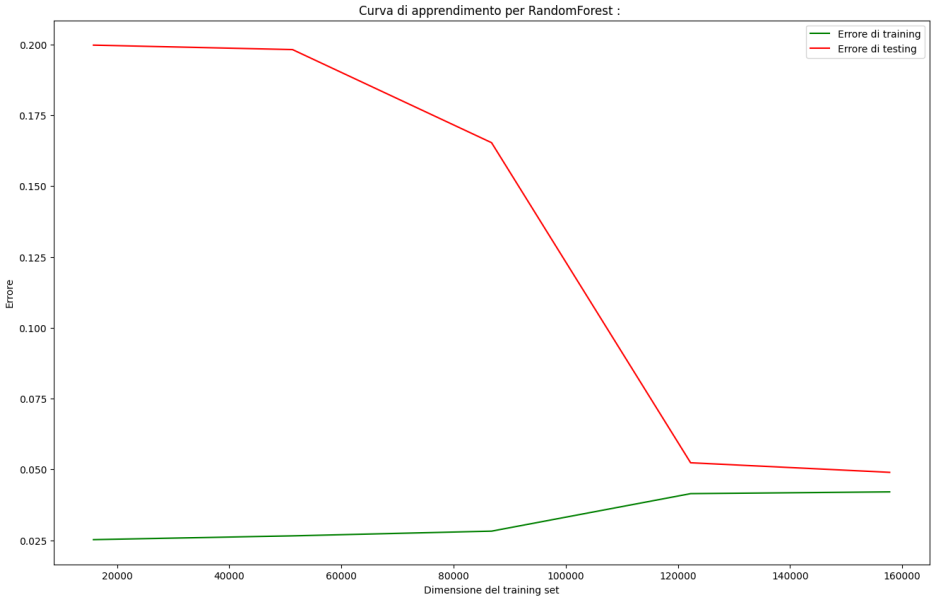


Figure 3.9: *Curve di Apprendimento Random Forest con OverSampling*

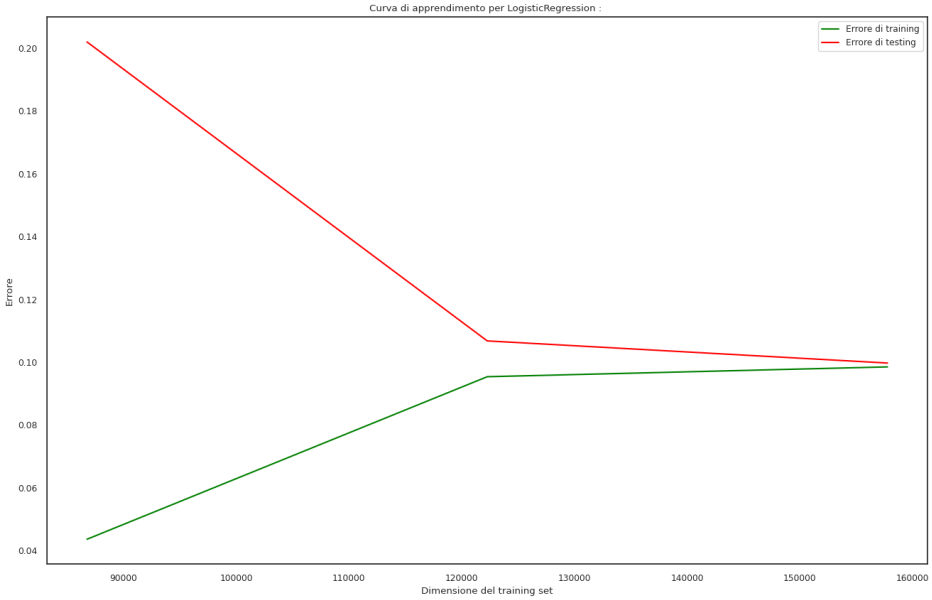


Figure 3.10: *Curve di Apprendimento Logistic Regression con OverSampling*

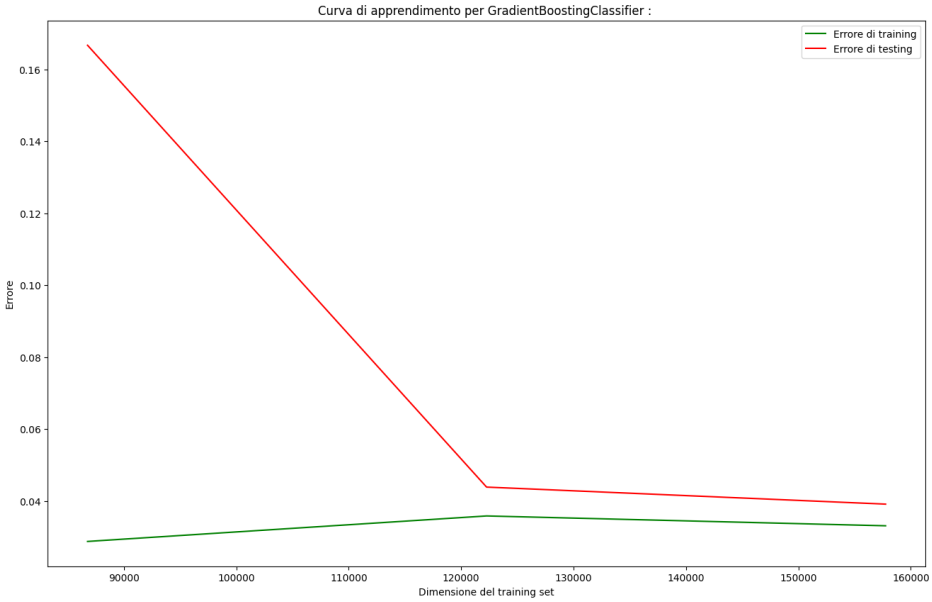
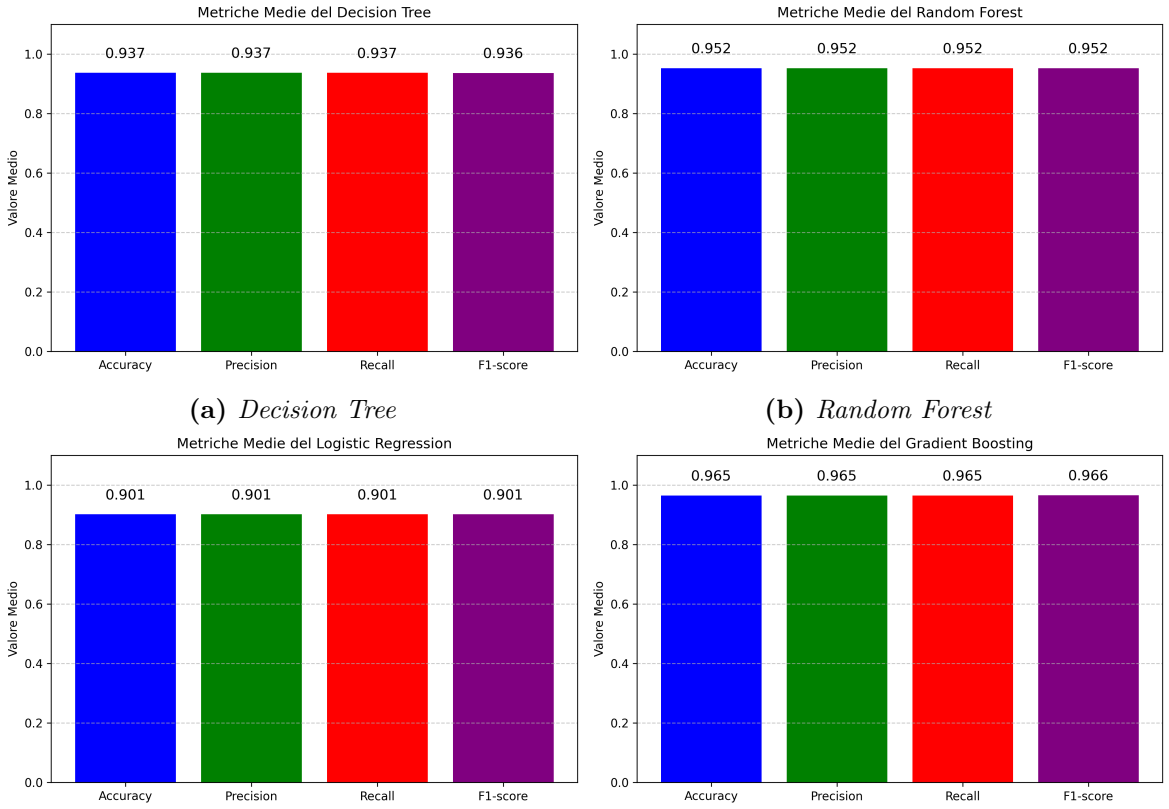


Figure 3.11: *Curve di Apprendimento Gradient Boosting Classifier con OverSampling*



(a) *Decision Tree* **(b)** *Random Forest*

(c) *Logistic Regression* **(d)** *Gradient Boosting*

Figure 3.12: *Metriche di valutazione con OverSampling*

Modello	Train Error Std	Test Error Std
Decision Tree	0.0029997	0.0214524
Random Forest	0.0024160	0.0242266
Logistic Regression	0.0018439	0.0156383
Gradient Boosting	0.0065291	0.0388052

Table 3.5: *Confronto Deviazione Standard dell'Errore per i Modelli Con OverSampling*

Modello	Train Error Var	Test Error Var
Random Forest	8.9982 e-06	4.6020 e-04
Decision Tree	5.8370 e-06	5.8693 e-04
Logistic Regression	3.4000 e-06	2.4455 e-04
Gradient Boosting	4.2629 e-05	1.5058 e-03

Table 3.6: *Confronto Varianza degli Errori per i Modelli con OverSampling*

3.4.1 Considerazioni

Decision Tree :

- l'Accuracy e la Precision sono diminuite rispetto al primo esperimento, tuttavia riflettono meglio la realtà del problema, mentre il Recall è aumentato, segnale che ora la classe minoritaria viene riconosciuta molto meglio e riducendo la tendenza del modello a ignorare la classe minoritaria., di conseguenza anche F1 è aumentata.
- la varianza dell'errore di test è leggermente aumentata rispetto al primo esperimento, probabilmente a causa del bilanciamento del dataset.
- la riduzione del possibile overfitting è evidente, l'errore di training non è più troppo basso, e l'errore di testing si è stabilizzato.

Random Forest :

- come il Decision Tree l'Accuracy e la Precision sono diminuite rispetto al primo esperimento, tuttavia riflettono meglio la realtà del problema, mentre il Recall è aumentato, segnale che ora la classe minoritaria viene riconosciuta molto meglio, di conseguenza anche F1 è aumentata. Le prestazioni sono migliori rispetto al Decision Tree.
- la varianza dell'errore di test è leggermente aumentata rispetto al primo esperimento, probabilmente a causa del bilanciamento delle classi. Tuttavia, l'andamento delle curve di apprendimento è nettamente migliore rispetto al primo esperimento, indicando che il modello è più stabile e continua a generalizzare meglio.
- con l'aumentare dei dati forniti l'errore di training cresce molto lentamente mentre l'errore di testing si è abbassa e le due curve sono dirette verso la convergenza, osserviamo che si avvicinano più rapidamente rispetto al Decision Tree. L'oscillazione nelle curve è minore rispetto al primo esperimento, indicando che il modello è più stabile. La riduzione del possibile overfitting è evidente.

Logistic Regression :

- tutte le metriche sono leggermente migliorate, ma risultano inferiori rispetto agli altri modelli.
- l'errore di training è più alto rispetto al primo esperimento, il che significa che il modello fatica di più a imparare dai dati, mentre l'errore di testing è ancora alto.

Gradient Boosting :

- tutte le metriche sono leggermente migliorate, ed inoltre risultano le migliori tra tutti i modelli

- la deviazione standard dell'errore di training è più alta rispetto agli altri modelli, ciò potrebbe indicare una maggiore sensibilità rispetto ai dati di training
- le curve di apprendimento mostrano un notevole miglioramento rispetto all'esperimento precedente.

Tutti i modelli sembrano aver beneficiato del bilanciamento delle classi.

Tuttavia, con l'overampling sono stati aggiunti un numero elevato di esempi sintetici, che potrebbero non riflettere accuratamente la realtà dei dati. Per mitigare questo problema, è stato sperimentato un approccio combinato utilizzando un OverSampling meno aggressivo, con una strategia di campionamento pari a 0.3, e UnderSampling, cercando di migliorare l'equilibrio del dataset senza introdurre eccessiva distorsione nei dati.

Per l'UnderSampling si è sfruttato un oggetto **RandomUnderSampler**, grazie alla libreria `imblearn.under_sampling`, mentre per l'OverSampling, al contrario dell'esperimento precedente, si è usato sample_strategy pari a 0.3, che indica che la classe minoritaria dovrà avere un numero di campioni pari al 30% della classe maggioritaria.

A seguito dell'OverSampling della classe minoritaria (pazienti con diagnosi positiva) e dell'UnderSampling della classe maggioritaria (pazienti con diagnosi negativa), gli esempi sono così distribuiti rispetto al target :

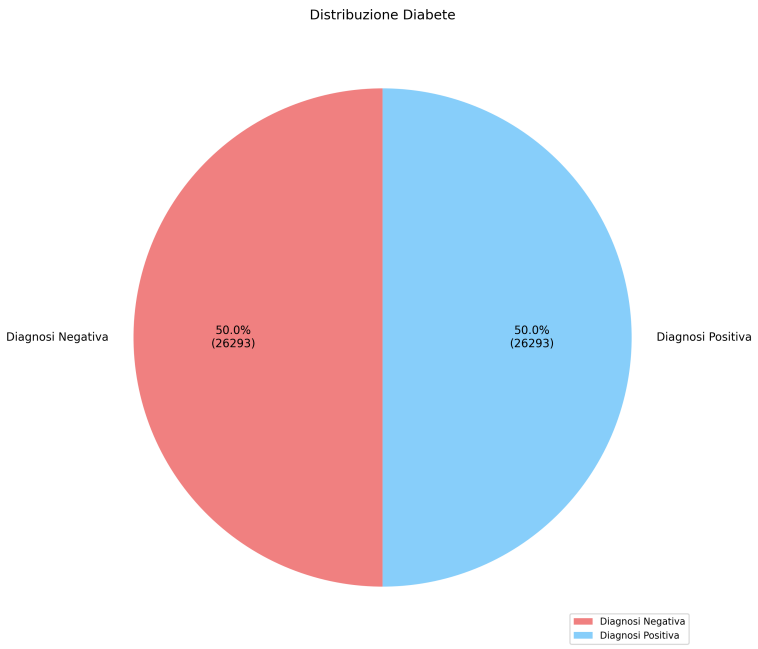


Figure 3.13: *Distribuzione Target con OverSampling e UnderSampling*

3.5 Terzo Esperimento

Iperparametri trovati con Gridsearch :

Modello	Parametro	Valore
Decision Tree	criterion	entropy
	max_depth	12
	min_samples_split	10
	min_samples_leaf	5
Random Forest	n_estimators	50
	max_depth	15
	min_samples_split	15
	min_samples_leaf	3
	criterion	gini
Logistic Regression	C	0.01
	penalty	l2
	solver	liblinear
	max_iter	100000
Gradient Boosting Classifier	n_estimators	150
	learning_rate	0.01
	max_depth	10
	min_samples_split	12
	min_samples_leaf	5

Table 3.7: *Tabella degli iperparametri trovati con Grid Search con OverSampling ed UnderSampling*

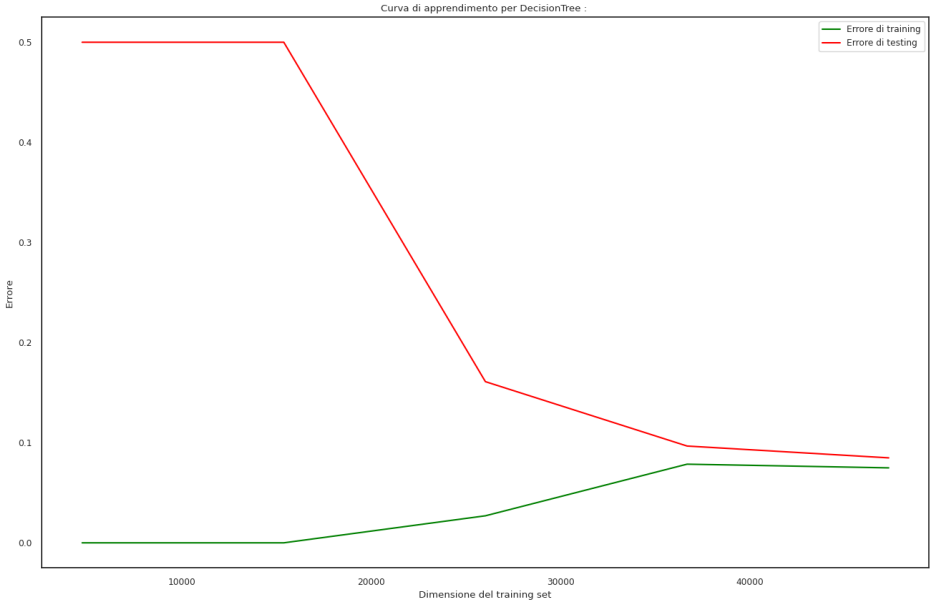


Figure 3.14: *Curva di Apprendimento Decision Tree con OverSampling e UnderSampling*

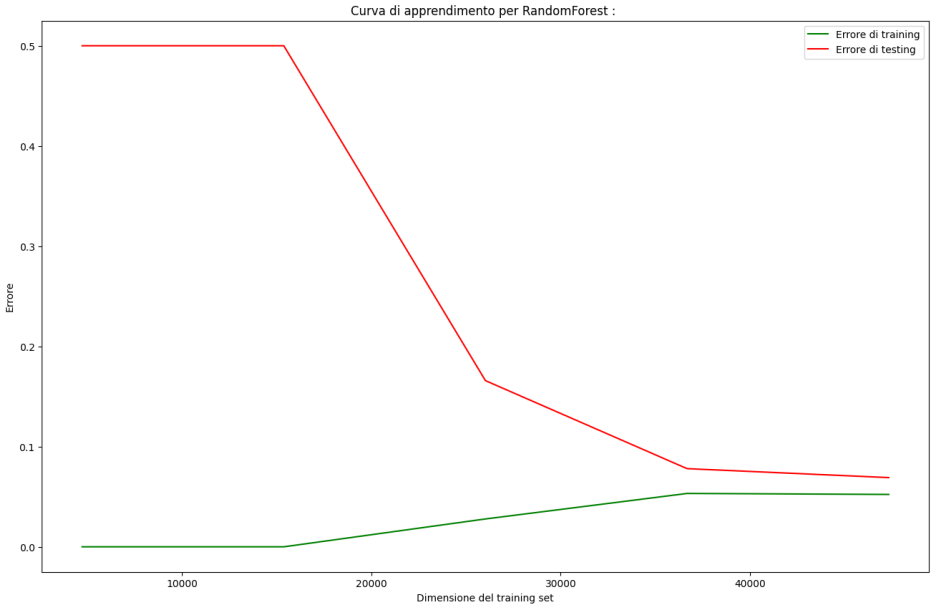


Figure 3.15: *Curva di Apprendimento Random Forest con OverSampling e UnderSampling*

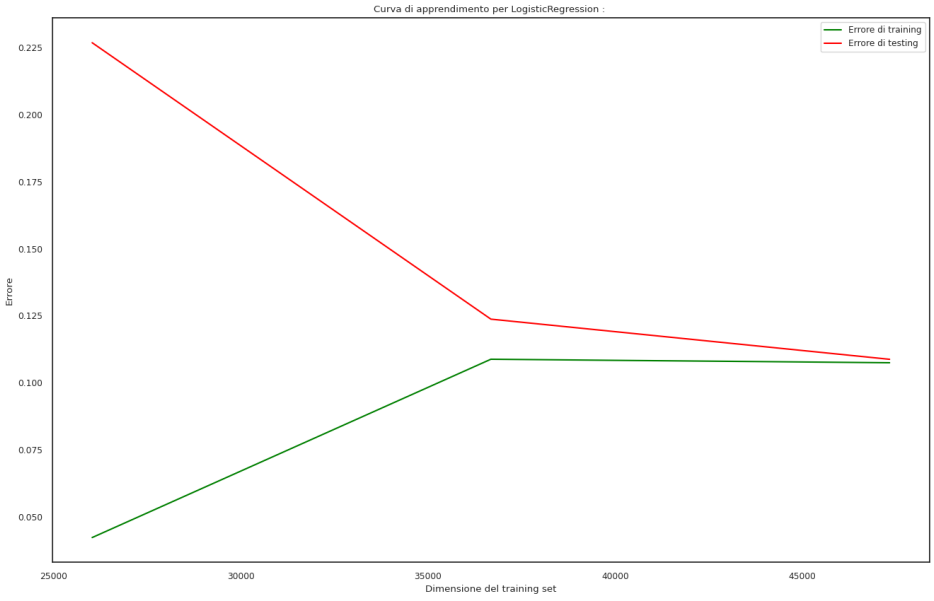


Figure 3.16: *Curva di Apprendimento Logistic Regression con OverSampling e UnderSampling*

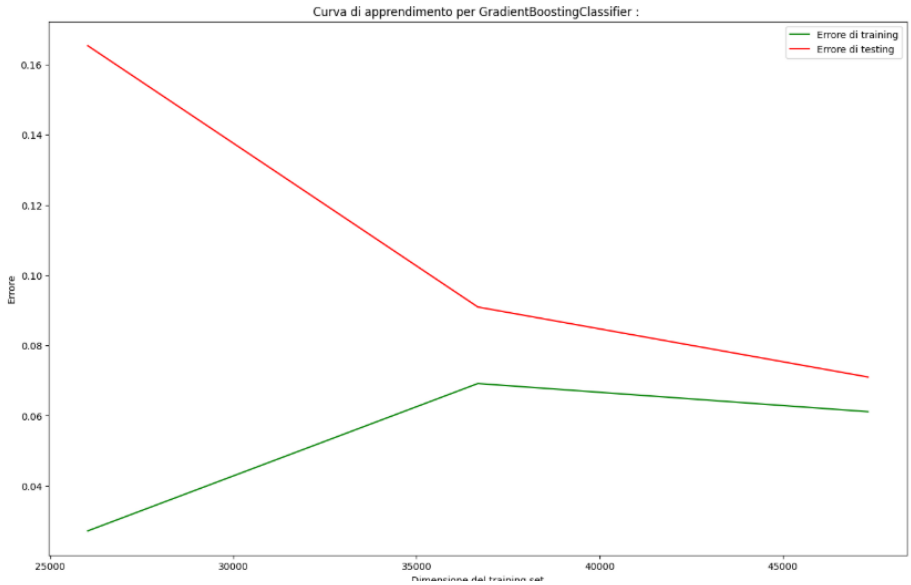
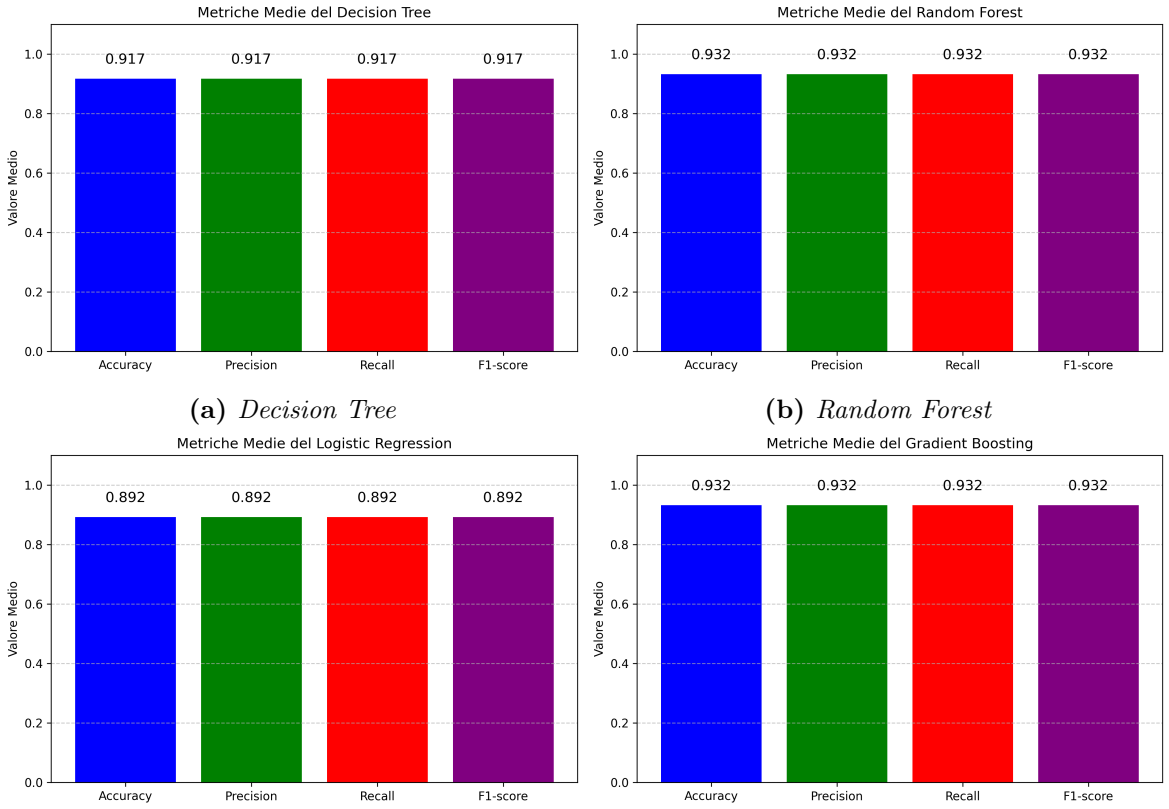


Figure 3.17: *Curva di Apprendimento Gradient Boosting Classifier con OverSampling e UnderSampling*



(a) *Decision Tree* **(b)** *Random Forest*

(c) *Logistic Regression* **(d)** *Gradient Boosting*

Figure 3.18: *Metriche di valutazione con OverSampling e UnderSampling*

Modello	Train Error Std	Test Error Std
Decision Tree	0.0021449	0.0181698
Random Forest	0.0022029	0.0213590
Logistic Regression	0.0012139	0.0111554
Gradient Boosting	0.0049538	0.0244220

Table 3.8: *Confronto Deviazione Standard dell'Errore per i Modelli Con OverSampling e UnderSampling*

Modello	Train Error Var	Test Error Var
Decision Tree	4.6006 e-06	3.3014 e-04
Random Forest	4.8529 e-06	4.5620 e-04
Logistic Regression	1.4737 e-06	1.2444 e-04
Gradient Boosting	2.4540 e-05	5.9643 e-04

Table 3.9: *Confronto Varianza degli Errori per i Modelli con OverSampling e UnderSampling*

3.5.1 Considerazioni

Decision Tree :

- l'Accuracy è ancora leggermente più bassa, ma il modello è molto più bilanciato. La Precision rimane pressochè simile, mentre il Recall è migliorato nuovamente, continua quindi il miglioramento del riconoscimento della classe minoritaria, di conseguenza anche F1 è aumentata.

- l'errore di training è più alto rispetto agli esperimenti precedenti, il che suggerisce che il modello è stato regolarizzato meglio. La stabilizzazione dell'errore di testing indica che il modello sta apprendendo in modo più equilibrato, senza adattarsi troppo ai dati di training.

- analizzando le curve di apprendimento, l'errore di training inzia da un errore più alto rispetto ai due esperimenti, mentre l'errore di testing si avvicina ancora di più a quello di training, suggerendo che il modello generalizza meglio, inoltre la curva dell'errore di testing mostra una discesa più regolare e una maggiore stabilizzazione.

Random Forest :

- l'Accuracy è ancora leggermente più bassa, ma il modello è molto più bilanciato, inoltre è più alta del Decision Tree dello stesso esperimento. La Precision rimane pressochè simile, mentre il Recall è migliorato nuovamente, continua quindi il miglioramento del riconoscimento della classe minoritaria, di conseguenza anche F1 è aumentata.

- l'errore di training è più alto, ma questo è positivo perché indica che il modello sta evitando di adattarsi troppo ai dati di training, indicando una riduzione del possibile overfitting. Il gap tra errore di training e di testing va man mano riducendosi, segnale che il modello ha la migliori capacità di generalizzazione.

Logistic Regression :

- tutte le metriche sono leggermente più basse rispetto all'esperimento precedente, ma si ha migliore bilanciamento, tuttavia sono molto inferiori rispetto agli altri modelli, ciò potrebbe essere dato dal bias del modello, ovvero data la sua semplicità potrebbe non essere in grado di catturare pattern complessi

Gradient Boosting :

- tutte le metriche sono leggermente più basse rispetto all'esperimento precedente, ma si ha migliore bilanciamento. Le sue metriche risultano quelle migliori insieme al Random Forest

- l'andamento delle curve di apprendimento suggerisce che il modello ha una buona capacità di generalizzazione, inoltre l'andamento verso il basso suggerisce che fornendo ulteriori dati il modello ne potrebbe beneficiare.

Il Gradient Boosting sembra essere il modello con il miglior equilibrio tra Recall, Precision e generalizzazione. Anche se le metriche sono leggermente più basse rispetto all'esperimento precedente, il modello offre prestazioni stabili e bilanciate.

Tuttavia, essendo Gradient Boosting un modello composto da Decision Tree in maniera sequenziale, al contrario di Random Forest che sfrutta i Decision Tree in parallelo, richiede molto più tempo di addestramento e risorse computazionali.

Possiamo quindi concludere che, in assenza di limiti di tempo e spazio, Gradient Boosting è il modello migliore in termini di performance complessiva, al contrario conviene ricorrere al Random Forest, che rimane una validissima alternativa.

Capitolo 4

Apprendimento Non Supervisionato

L'apprendimento non supervisionato è una branca del Machine Learning, in cui un modello viene addestrato su un insieme di esempi descritti dalle sole features di input, al contrario dell'apprendimento supervisionato dove è anche presente il target (cosidetto ground truth), con l'obiettivo di scoprire raggruppamenti e pattern nascosti all'interno dei dati.

Nel caso di studio, l'apprendimento non supervisionato, non è stato effettuato sull'intero dataset, poiché non avrebbe avuto alcun senso dato che sono già presenti le effettive diagnosi, bensì è stato applicato al sottoinsieme del dataset composto dai soli pazienti con diagnosi positiva, con lo scopo di verificare se potesse fornire spunti di analisi per l'identificazione di pattern e sotto raggruppamenti tra i pazienti diabetici, cercando di scoprire sottogruppi con caratteristiche simili, ad esempio per cercare di raggruppare i diabetici in vari sottogruppo in base al tipi particolare di diabete da cui sono affetti (Tipo 1, Tipo 2, Gestazionale, ecc...)

4.1 Metodologia

Per l'apprendimento non supervisionato si è scelto di usare il **Kmeans**, un algoritmo di clustering non supervisionato che raggruppa i dati in k cluster basandosi sulla somiglianza tra gli esempi ed i centroidi dei cluster. In particolare, all'inizio dell'algoritmo, i centroidi iniziali vengono scelti casualmente (`init = "random"`), successivamente in modo iterativo ogni esempio viene assegnato al cluster del centroide più vicino, calcolando la distanza tra i due esempi come distanza euclidea, e dopo l'assegnazione degli esempi ai cluster, vengono ricalcolati i centroidi come la media degli esempi che sono stati assegnati al cluster. Il processo continua iterativamente fino alla convergenza, ovvero quando i centroidi smettono di muoversi in modo significativo o viene raggiunto il numero massimo di iterazioni.

Il numero di cluster k è un iperparametro che deve essere scelto attentamente. In tal caso si è sfruttata l'**Elbow Rule**, che permette di individuare un valore ottimale k di cluster analizzando l'inerzia del modello, ossia la somma delle distanze quadratiche

tra ogni punto e il centroide del proprio cluster. In particolare, si calcola l'inerzia per diversi valori di k e si rappresenta graficamente il risultato in un grafico del gomito. L'andamento dell'inerzia tende a diminuire con l'aumentare di k, ma oltre un certo punto la riduzione diventa meno significativa, formando una curva con un "gomito", proprio il valore di k corrispondente a questo gomito è considerato il punto ottimale, poiché rappresenta il miglior compromesso tra compattezza dei cluster e riduzione della varianza intra-cluster senza aggiungere complessità eccessiva al modello.

Per l'implementazione si è usato l'algoritmo **KMeans** della libreria sklearn.cluster, mentre per l'individuazione del miglior k si è usata la funzione **KneeLocator** della libreria `kneed`.

Valutazione

Nell'apprendimento non supervisionato, essendo che non ci sono etichette di riferimento per valutare direttamente le perfomance, per la valutazione considereremo metriche che misurano coerenza interna, separabilità dei cluster o adattabilità alla struttura dei dati, in particolare :

- **Silhouette Score** : misura quanto un punto sia simile al cluster assegnato rispetto agli altri cluster, e può assumere valore `[-1,+1]`, dove `-1` indica il caso peggiore e `+1` quello ottimo.
- **Within-Cluster Sum of Squares** (WCSS) : misura la somma delle distanze quadrate tra i punti e il loro centroide all'interno di ciascun cluster, dove un valore basso indica che i punti all'interno di ciascun cluster sono vicini tra loro, quindi il clustering è più compatto, mentre un valore alto significa che i cluster sono molto dispersi, suggerendo un clustering meno efficace. E' proprio grazie al WCSS che si riesce a trovare il k ottimale con l'Elbow Rule.

4.2 Preparazione del dataset

Prima di tutto sono stati eliminati dal dataset i pazienti con diagnosi negativa (con feature `diabetes = 0`), ed in seguito è stata rimossa la feature `diabetes`, in quanto assume lo stesso valore per ogni paziente diabetico, e quindi non fornisce informazioni utili nella differenziazione tra i pazienti.

Inoltre, essendo l'algoritmo di clustering utilizzato, KMeans, basato sulla distanza euclidea delle caratteristiche degli esempi per l'assegnamento ai cluster, esso potrebbe portare a scarsi risultati se le caratteristiche utilizzassero scale di valori molto diverse tra loro.

A tal fine è stata effettuata la **Normalizzazione** delle feature, in modo portare tutti i valori nell'intervallo $[0, 1]$, effetuando la seguente trasformazione :

$$x' = \frac{x - \min(x)}{\max(x) - \min(x)}$$

Per effettuare la normalizzazione è stato usato, grazie alla libreria sklearn . preprocessing, l'oggetto **MinMaxScaler**.

4.3 Esperimento

Tramite l'Elbow Rule si è trovato che il numero ottimale k di cluster è 3, come mostra il seguente grafico :

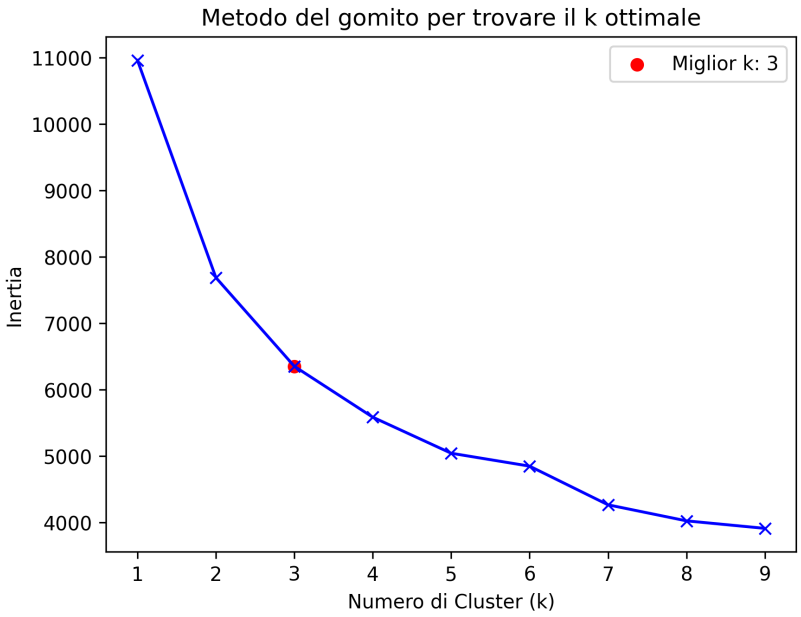
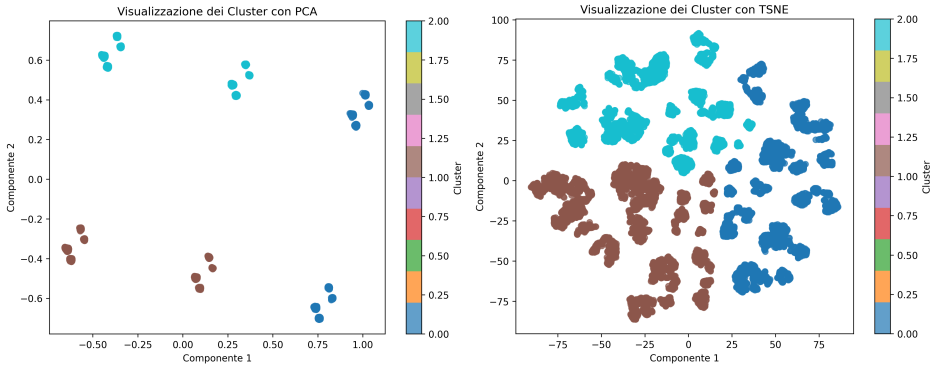


Figure 4.1: *Elbow Rule*

Dopo aver partizionato i dati in cluster grazie all'algoritmo di clustering KMeans con k pari a 3, è stato utile visualizzarli in uno spazio bidimensionale per valutare la separabilità e la distribuzione dei punti. Poiché i dati originali possono avere molte dimensioni, si utilizzano tecniche di riduzione della dimensionalità come PCA e t-SNE, che permettono di rappresentare graficamente i cluster senza modificarne la struttura.

PCA trasforma i dati trovando le componenti principali con la massima varianza e li proietta in uno spazio bidimensionale, offrendo una rappresentazione lineare dei cluster che permette di analizzare la loro distribuzione globale e l'eventuale sovrapposizione.

t-SNE conserva le relazioni di vicinanza tra i punti, enfatizzando la struttura locale dei dati e rivelando eventuali sottogruppi all'interno dei cluster, risultando particolarmente utile quando i cluster presentano forme complesse e non lineari.



(a) *PCA* **(b)** *t-SNE*

Figure 4.2: *Visualizzazone clustering PCA e TSNE*

A seguito del clustering si è ottenuto un WCSS pari a 6352 e Silhouette Score pari a 0.2738. Dal WCSS deduciamo che essendo un valore piuttosto alto, può indicare che i cluster non sono molto compatti, mentre dal Silhouette Score deduciamo che i cluster sono moderatamente distinti, ma non perfettamente separati, ciò si può facilmente notare dalle figure soprastanti.

Pertanto non si proseguirà con analisi e test statistici per l'identificazione di pattern tra i vari cluster ottenuti, ed inoltre sarebbe necessario consultare un esperto del dominio medico per analizzare le distribuzioni delle feature degli esempi nei vari cluster ottenuti.

Capitolo 5

Ragionamento probabilistico e Bayesian Network

Il ragionamento probabilistico è una forma di ragionamento che sfrutta la teoria della probabilità, in particolar modo la dipendenza ed indipendenza tra variabili, ed il teorema di Bayes. Nel ragionamento probabilistico si assegnano probabilità a ipotesi ed eventi e si utilizzano le probabilità a posteriori per i ragionamenti. Un'applicazione del ragionamento probabilistico sono le reti bayesiane.

5.1 Preparazione del dataset

Inizialmente, lasciando il dataset inalterato e provando ad apprendere la struttura della rete bayesiana, è stato riscrontato l'errore `segmentation fault`, a causa di un'eccessiva quantità di memoria richiesta. Ciò è dovuto al fatto che le reti Bayesiane, per calcolare le Conditional Probability Distributions (CPD), che rappresentano le distribuzioni condizionate delle variabili e sono rappresentate sotto forma di Conditional Probability Table (CPT), necessitano di osservare ogni singolo valore di esse, tuttavia, essendo che i valori delle variabili (features) sono continui e non discreti, ciò diventa facilmente impraticabile.

Come prima idea di soluzione si è pensato di suddividere manualmente i domini delle variabili continue in intervalli discreti, tuttavia data la numerosità di features e al fatto di dover stabilire degli intervalli ammissibili, si è fatto ricorso ad una soluzione più rapida, il **KBinsDiscretizer**, dalla libreria sklearn.preprocessing, che converte i valori continui assunti da una variabile in un numero limitato di categorie (bin), in particolare si è utilizzata la tecnica di encoding **ordinale**, per far si che i valori continui vengano mappati a valori ordinali, ed ogni bin sia rappresentato da un numero intero che indica in quale bin ogni valore è stato categorizzato, e strategia **uniforme**, per far si che l'intervallo dei possibili valori continui sia suddiviso in bin di ampiezza uguale.

5.2 Apprendimento della Struttura

Per l'implementazione della Rete Bayesiana si è sfruttata la libreria **pgmpy**, mentre per la visualizzazione della rete si è fatto uso della libreria **networkx**.

La struttura della Rete Bayesiana è stata appresa mediante l'utilizzo di **HillClimbSearch**, un algoritmo di ricerca locale per esplorare le possibili strutture della rete, in particolare parte da una struttura di rete casuale e cerca di migliorarla iterativamente, e come estimator è stata usata la **Maximum Likelihood**.

Di seguito si riporta la struttura della rete:

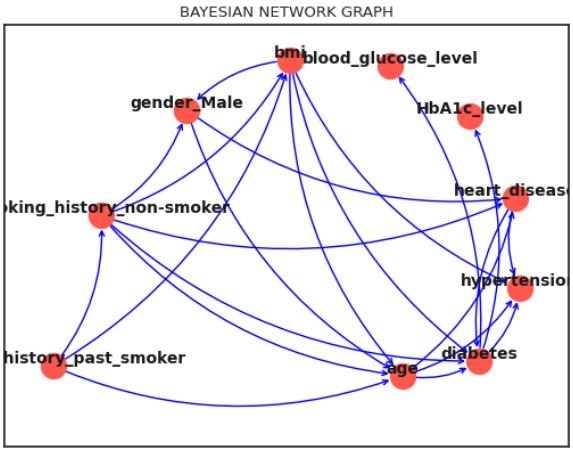


Figure 5.1: *Metriche*

Una volta appresa la struttura della Rete Bayesiana, uno dei suoi principali scopi è quello di effettuare **inferenza probabilistica**, ovvero la rete può essere sfruttata per il calcolo delle probabilità di una o più variabili (incognite), dati alcuni valori di altre variabili osservate (evidenza).

Nel contesto di una rete bayesiana, l'inferenza probabilistica si basa sul **Teorema di Bayes**, che fornisce un modo per aggiornare la probabilità di un evento a seguito di nuove evidenze.

5.3 Esempio : Generazione di campioni e gestione di dati mancanti

Una volta appresa la struttura, tramite le Reti Bayesiane è possibile generare nuovi campioni, ovvero nuove configurazioni di variabili, ciò risulta particolarmente utile per per fare inferenza su nuovi dati, oppure anche per generare nuovi dati da utilizzare per addestrare un modello. Per generare un nuovo campione vengono sfruttate le probabilità apprese dal dataset in fase di addestramento per generare un esempio sintetico credibile.

Ad esempio :

age	3.0
bmi	0.0
gender_Male	0.0
smoking_history_non-smoker	0.0
HbA1c_level	2.0
blood_glucose_level	1.0
heart_disease	0.0
hypertension	0.0
diabetes	4.0

Table 5.1: *Esempio generato dalla Rete Bayesiana*

Ovviamente i valori delle feature per l'esempio randomico vengono generati già discretizzati.

Le reti bayesiane sono in grado di gestire casi in cui una variabile di input non è nota. Questo è utile in quanto i dati reali spesso presentano valori mancanti. La Rete Bayesiana è in grado di fare inferenza su nuovi dati, anche se non tutte le variabili sono note.

Ad esempio, eliminando la variabile `HbA1c_level` dal precedente esempio generato, e provandone a predire il valore per inferenza sulla Rete Bayesiana si ottiene :

HbA1c_level	$\phi(HbA1c\_level)$
HbA1c_level(0.0)	0.0000
HbA1c_level(1.0)	0.0000
HbA1c_level(2.0)	0.6173
HbA1c_level(3.0)	0.1503
HbA1c_level(4.0)	0.2324

Table 5.2: *Risultato inferenza sulla Rete Bayesiana*

Possiamo notare che è un risultato consistente con il valore effettivo, in quanto il valore effettivo è quello con probabilità più alta.

5.4 Esempio : Query

Una volta appresa la struttura, le Reti Bayesiane sono in grado anche di rispondere a query probabilistiche, ovvero di calcolare la probabilità di una variabile dato un insieme di evidenze.

Tuttavia risolvere una query data l'evidenza e fornire una distribuzione di probabilità è computazionalmente costoso e diventa rapidamente intrattabile, soprattutto

in una rete con molte variabili, poichè per calcolare la distribuzione di probabilità è necessario, applicando il teorema di Bayes, moltiplicare i fattori di probabilità e marginalizzare sulle variabili non osservate, ovvero sommare su tutte le loro possibili istanze, ma il numero di combinazioni delle variabili cresce esponenzialmente con il numero di nodi della rete.

Per ridurre tale complessità e fornire risposte alle query in modo efficiente si è sfruttata l'**VariableElimination**, grazie alla libreria `pgmpy.inference`, una tecnica per eseguire inferenza sulle Reti Bayesiane, che calcola la probabilità di una variabile d'interesse eliminando iterativamente le variabili nascoste tramite moltiplicazione dei fattori di probabilità e sommando le probabilità sulle variabili eliminate, riducendo così la complessità computazionale rispetto all'inferenza diretta.

Ad esempio : "Data l'osservazione che un paziente ha un'età discretizzata del valore (4.0), un indice di massa corporea (BMI) pari (3.0), non ha malattie cardiache (0.0), e un livello di HbA1c di (2.0), qual è la distribuzione di probabilità del livello di glucosio nel sangue del paziente?"

```
query_report(infer,
             variables=['blood_glucose_level'],
             evidence={ 'age': 4.0,
                        'bmi': 3.0,
                        'heart_disease': 0.0,
                        'HbA1c_level': 2.0
             }
)
```

Risultato della query :

blood_glucose_level	$\phi(blood\_glucose\_level)$
`blood_glucose_level(0.0)`	0.0922
`blood_glucose_level(1.0)`	0.5620
`blood_glucose_level(2.0)`	0.0761
`blood_glucose_level(3.0)`	0.1019
`blood_glucose_level(4.0)`	0.1678

Table 5.3: *Risultato query sulla Rete Bayesiana*

Capitolo 6

Rete Neurale

Nell'apprendimento supervisionato, la Regressione Logistica, ha fornito risultati accettabili, ma non ottimali, ciò potrebbe essere dovuto a causa di un alto bias del modello, ovvero la semplicità del modello lineare potrebbe non riuscire a catture eventuali relazioni non lineari che si presentano nei dati. Pertanto si è pensanto di introdurre l'uso delle Reti Neurali Artificiali, che rappresentano un'estensione della regressione logistica, in modo da aggiungere strati nascosti e funzioni di attivazioni non lineari, per provare ad apprendere pattern più complessi e migliorare la capacità di generalizzazione del modello, inoltre sono altamente scalabili e possono gestire grandi volumi di dati.

Nel caso di studio si è voluto considerare l'applicazione delle reti neurali per la classificazione della diagnosi del diabete per il paziente, valutando se un'architettura più avanzata possa portare a un miglioramento delle performance rispetto alla regressione logistica.

6.1 Metodologia

La Rete Neurale che si andrà a definire, sarà composta da un layer di input, due hidden layer, ed il layer di ouput. L'idea è di valutare se aggiungendo due hidden layer, definiti da funzioni non lineari, si riescano ad avere performance migliori rispetto al modello lineare della Regressione Logistica già esaminito.

Per quanto riguarda l'architettura della Rete Neurale si è deciso di usare:

- unità del layer di input : 8, essendo 8 le feature di input (escludendo la feature target)
- funzione di attivazione degli hidden layer : `ReLu`, che attiva un neurone e restituisce, se l'input è positivo, esso stesso, altrimenti se negativo 0, ciò serve a introdurre non linearità nel modello e a migliorare l'apprendimento.
- funzione di attivazione del layer di output : `sigmoid`, essendo un task di classificazione binaria, in particolare trasformerà l'output in una probabilità tra 0 e 1, che stabilirà la classificazione
- funzione di loss : `binary_crossentropy`, tipicamente utilizzata nei problemi di classificazione binaria, che penalizza maggiormente le previsioni errate
- epoche : `50`, ovvero nell'addestramento sul dataset verranno effettuate 50 passate complete, tale valore si è fissato e non è stato scelto come iper parametro da ricercare, in quanto con brevi sperimentazioni si è visto che dopo 50 epoche i valori non miglioravano significativamente ma rimanevano stabili, dunque è un buon compromesso (in future implementazioni si potrà fissare un numero di epoche maggiore, ma aggiungendo un early stopping, in modo da trovare l'effettivo compromesso ideale)
- batch size : `128`, ovvero verranno esaminati 128 campioni prima di aggiornare i pesi della rete, anche tale valore è stato fissato e non è stato scelto come parametro da ricerca, in quanto date le dimensioni del dataset rappresenta buon compromesso tra stabilità e velocità dell'addestramento
- learning step : `0.01`, ovvero la dimensione degli step con la quale il modello aggiorna i pesi, in particolare si è scelto 0.01 poichè rappresenta un buon compromesso tra la velocità di addestramento e la precisione del modello.

Mentre per quanto riguarda, il numero di unità nel primo e secondo Hidden Layer, ed il tipo di ottimizzatore, si è deciso di utilizzare la **GridSearch** in combinazione a **CrossValidation** per trovare la migliore configurazione.

In particolare al ricerca è stata fatta tra i seguenti iper parametri :

- **model__hidden_units_1** : numero di unità nel primo hidden layer : `[4, 8, 16, 24, 32]`

- **model__hidden_units_2** : numero di unità nel secondo hidden layer : `[4, 8, 16, 24, 32]`

- **model__optimizer** : tipo di algoritmo che aggiorna i pesi della rete neurale per minimizzare la funzione di loss : `["adam", "sgd", "rmsprop"]`, in particolare `sgd` aggiorna i pesi del modello dopo ogni batch usando il gradiente della loss, `rmsprop`, normalizza il learning rate dividendo per la media quadratica dei gradienti passati, e `adam`, combina i vantaggi di SGD con Momentum e RMSprop, adattando dinamicamente il learning rate per ogni parametro.

Per l'implementazione della Rete Neurale si è sfruttata **SciKeras**, un'interfaccia che collega `Keras` con `Scikit-Learn`, consentendo di usare modelli di deep learning all'interno di strumenti di scikit-learn come la GridSearchCV.

Per la valutazione, si useranno il **report** generato dalla funzione classification_report di sklearn.metrics, che descrive le varie metriche già viste nell'apprendimento supervisionato, e le **curve dell'accuracy e della loss**, della fase di training e validation, al variare del numero delle epoche passate.

Per garantire una corretta ottimizzazione e valutazione del modello, il dataset viene inizialmente suddiviso in 70% per il training e 30% per il testing, e successivamente la parte di training viene ulteriormente suddivisa in 80% di training effettivo e 20% di validation. In particolare, la parte di training effettiva verrà utilizzata per l'addestramento del modello, la parte di validation per l'ottimizzazione degli iperparametri e per monitorare il modello durante il traininge, e la parte di testing per la valutazione finale del modello su dati mai visti. Questa suddivisione ci permette di effettuare una stima affidabile e robusta delle prestazioni del modello.

6.2 Preparazione del dataset

Essendo che le Reti Neurali per apprendere i pesi sfruttano la Discesa di Gradiente, se le features hanno scale diverse può accadere che i pesi associati alle feature con valori più grandi possano avere aggiornamenti più grandi, mentre quelli con valori piccoli saranno trascurati, di conseguenza i gradienti possono diventare troppo grandi o troppo piccoli, causando oscillazioni o rallentando la convergenza.

A tal fine è stata effettuata la **Normalizzazione** delle features, in quanto la Discesa del Gradiente funziona meglio quando i dati sono compresi in intervalli di dimensione uguale. Questo permette di evitare che alcune feature dominino l'apprendimento, rendendo il training più stabile ed efficiente.

6.3 Esperimento

Iperparametri trovati con GridSearch :

`model__hidden_units_1`	24
`model__hidden_units_2`	8
`model__optimizer`	`adam`

Table 6.1: *Iperparametri*

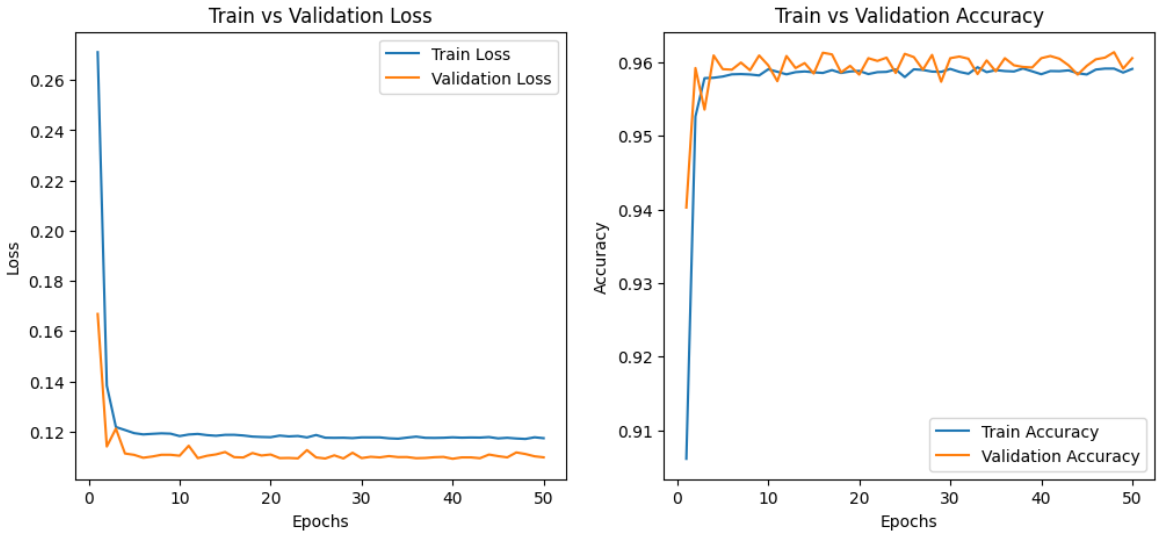


Figure 6.1: *Curve dell'accuratezza e della loss al variare delle epoche*

	Precision	**Recall**	**F1 score**	**Support**
Diagnosi Negativa	0.97	1.00	0.98	12317
Diagnosi Positiva	0.94	0.69	0.80	1141
accuracy			0.97	13458
macro avg	0.95	0.84	0.89	13458
weighted avg	0.97	0.97	0.97	13458

Table 6.2: *Report di classificazione per il validation set*

	Precision	**Recall**	**F1 score**	**Support**
Diagnosi Negativa	0.97	1.00	0.98	26266
Diagnosi Positiva	0.95	0.66	0.78	2573
accuracy			0.97	28839
macro avg	0.96	0.83	0.88	28839
weighted avg	0.97	0.97	0.96	28839

Table 6.3: *Report di classificazione per il test set*

Considerazioni :

- le curva della loss, sia di training che validation, scendono rapidamente durante le prime epoche per poi rimanere abbastanza stabili, in particolare la curva della loss di validation è leggermente più bassa di quella di training, anche se si sta utilizzando un ottimizzatore ciò risulta abbastanza sospetto, ed inoltre le curve risultano un pò troppo stabili e molto "lisce", tuttavia ciò non implica necessariamente che il modello ha perfomance ottime.

- l'accuratezza e la precision sono molto alte, sia nel training che nella validation e sia rispetto ad entrambe le classi, mentre osservando il recall della classe dei pazienti con diagnosi positiva, possiamo notare che è nettamente inferiore rispetto a quella dei pazienti con diagnosi negativa, ciò significa che molti pazienti con effettiva diagnosi positiva non vengono rilevati dalla rete neurale

Il basso recall della classe dei pazienti con diagnosi positiva suggerisce che la rete neurale potrebbe essere influenzata dal fatto che il dataset è sbilanciato rispetto alla diagnosi, di conseguenza potrebbe imparare a classificare maggiormente i pazienti con diagnosi negativa, e a tralasciare quelli con diagnosi positiva. Ciò potrebbe rendere la classificazione più prevedibile, e questa potrebbe essere la causa per cui le curve risultavano molto lisce.

Pertanto come per l'Apprendimento Supervisionato, si è decisio di bilanciare il dataset effettuando una strategia mista, di OverSamplig della classe minoritaria (Diagnosi Positiva) ed UnderSampling della classe maggioritaria (Diagnosi Negativa), per valutare le prestazioni su un dataset non sbilanciato rispetto al target.

Date le osservazioni del secondo esperimento dell'Apprendimento Supervisionato, non verrà presentato per la Rete Neurale l'esperimento con il dataset con strategia di solo OverSampling, ma si passa direttamente al dataset composto come nel terzo esperimento.

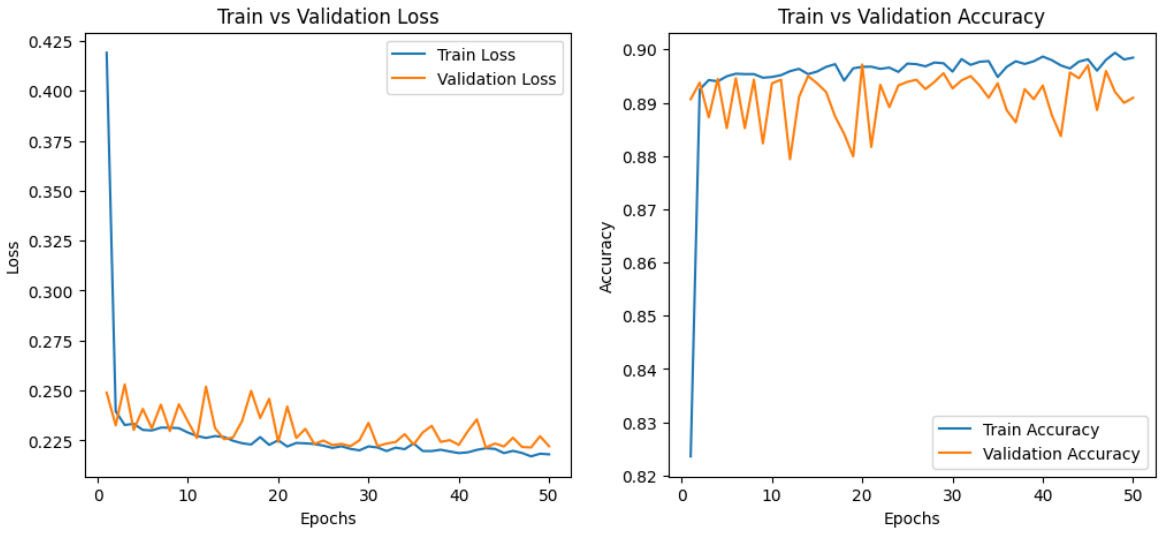


Figure 6.2: *Curve dell'accuratezza e della loss al variare delle epoche con OverSampling ed UnderSampling*

	Precision	**Recall**	**F1 score**	**Support**
Diagnosi Negativa	0.85	0.94	0.89	3679
Diagnosi Positiva	0.93	0.84	0.88	3683
accuracy			0.90	7362
macro avg	0.90	0.90	0.90	7362
weighted avg	0.90	0.90	0.90	7362

Table 6.4: *Report di classificazione per il validation set*

	Precision	**Recall**	**F1 score**	**Support**
Diagnosi Negativa	0.85	0.94	0.89	7872
Diagnosi Positiva	0.93	0.84	0.88	7904
accuracy			0.90	15776
macro avg	0.90	0.90	0.90	15776
weighted avg	0.90	0.90	0.90	15776

Table 6.5: *Report di classificazione per il test set*

Considerazioni :

- la curva della loss di validazione, seppure con qualche picco di alti e bassi, dovuto al bilanciamento delle classi e quindi ad una variabilità maggiore nei dati,

continua a convergere verso la loss di training, segnale che il modello ha un buon livello di generalizzazione e non presenta chiaro overfitting.

- l'accuracy è diminuita rispetto al precedente esempio, tuttavia tale risultato è più equilibrato e stabile, in quanto in tal caso la rete neurale riconosce meglio entrambe le classi, infatti il recall della classe della diagnosi positiva è aumentato notevolmente, di conseguenza anche la metrica F1 ha subito un miglioramento

- possiamo osservare che nei grafici, la scala delle loss (asse y) è diversa, in particolare la loss risulta leggermente più alta in questo esperimento, ciò è dovuto al fatto che la rete neurale osserva in maniera bilanciata gli esempi di entrambe le classi, pertanto ha più probabilità di commettere errori data la variabilità degli esempi.

6.4 Conclusioni

Le Reti Neurali sono un modello complesso, con un elevato costo computazionale, d'altro lato la Regressione Logistica è un modello semplice e meno costoso da addestrare (infatti la ricerca degli iperparametri e l'addestramento della Rete Neurale ha richiesto circa 2 ore, mentre la Regressione Logistca circa 20 minuti).

Tuttavia le Reti Neurali sono in grado di apprendere rappresentazione complesse dei dati e cattuare relazioni non lineari, grazie alle funzioni di attivazione, che il semplice modello della Regressione Lineare non è in grado di fare, pertanto in alcune situazioni potrebbero fornire risultati nettamente migliori.

Confrontando i risultati ottenuti da entrambi i modelli, possiamo osservare che la Rete Neurale non ha fornito un aumento significativo delle performance rispetto alla Regressione Logistica, vista nel capitolo dell'Apprendimento Supervisionato. Quindi l'introduzione della non linearità del modello non ha fornito risultati splendenti, come ci si aspettava.

Pertanto possiamo concludere, che per il problema affrontato dal caso di studio e del dataset analizzato, la Regressione Lostica si rivela una scelta più efficiente, in termini di costi computazionali e performance, fornendo previsioni comparabili a quelle della Rete Neurale.

Capitolo 7

Sviluppi Futuri

I modelli e le tecniche adottate nel caso di studio hanno già dimostrato buone capacità predittive, tuttavia in un scenario reale di supporto medico, le performance diventano un fattore cruciale, pertanto non è concepibile un buon risultato ma si deve aspirare all'ottimizzazione quanto più possibile, in quanto anche minimi errori potrebbero portare ad avere conseguenze significative sulla salute dei pazienti, come nel caso di falsi negativi, cioé pazienti che non sono risultati diabetici ma che in realtà lo sono, che potrebbero portare ad una diagnosi tardiva e complicazione della situazione, e come nel caso di falsi positivi, cioè pazienti che sono risultati diabetici ma che in realtà non lo sono, che potrebbero portare a trattamenti non necessari e compromettere inutilmente la salute del paziente.

Per questo motivo, migliori sviluppi futuri potrebbero includere :

- creazione di un'ontologia, e connessione verso altre ontologie mediche già esistenti e validate, in modo da poter, tramite inferenza, acquisire più conoscenza, e poter fornire previsioni e spiegazioni più accurate.
- valutare l'applicazione di altri modelli di ensambling per l'apprendimento supervisionato, in modo da ricercare modelli che forniscano una diagnosi più affidabile e robusta.

Bibliografia

[1] D. Poole, A. Mackworth: Artificial Intelligence: Foundations of Computational Agents. 3/e. Cambridge University Press