UNIVERSITÀ
DI PARMA

Dipartimento di Ingegneria e Architettura
Corso di Laurea in Ingegneria informatica, elettronica e delle telecomunicazioni

Data Augmentation per la Predizione di Interventi Chirurgici Basata su Parametri Anamnestici

Data Augmentation for Predicting Surgical Interventions Based on Anamnestic Parameters

Relatore:
Chiar.mo Prof. Michele Tomaiuolo

Correlatore:
Dott.Ing Mattia Pellegrino

Tesi di Laurea di:
Giovanni Annaloro

ANNO ACCADEMICO 2023-2024

A chi mi è stato vicino e mi ha supportato in questo percorso

Ringraziamenti

Voglio cominciare ringraziando il mio relatore, il *Prof. Michele Tomaiuolo*, per la disponibilità, i consigli forniti, il supporto costante e la tempestività nel far fronte a tutte le mie richieste.
Un grande ringraziamento va al mio correlatore, il *Dott.Ing Mattia Pellegrino*, per la sua collaborazione e le preziose indicazioni fornite durante la scrittura della tesi. Il suo aiuto è stato per me di fondamentale importanza.
Ringrazio poi i miei genitori, mia madre *Leandra* e mio padre *Manlio*, per essere sempre stati un punto di riferimento e una fonte di fiducia e supporto durante tutto il mio percorso di studi.
Infine, un ringraziamento speciale va alla mia fidanzata *Agnese*, per essermi sempre stata accanto e aver reso questo difficile percorso più leggero e sopportabile.

Indice

Elenco delle figure

Elenco delle tabelle

Elenco degli algoritmi

Introduzione

Grazie ai recenti progressi nel campo del Machine Learning, l'intelligenza artificiale sta trovando sempre più applicazioni nel settore della scienza e della tecnologia medica. Uno degli utilizzi più promettenti riguarda il supporto alle decisioni nel processo di diagnosi, scelta del trattamento e pianificazione delle cure. Tuttavia, l'implementazione di modelli di Machine Learning prestazionali richiede l'accesso a una grande quantitá di dati, il che può essere problematico, specialmente nel contesto medico, dove questi sono meno facilmente disponibili per via della severa normativa sulla privacy e della reticenza del pubblico a condividerli.
Una strategia efficace per affrontare la carenza di dataset sufficientemente ampi e diversificati consiste nella generazione di nuovi dati sintetici a partire da quelli esistenti, al fine di ampliare e variare il dataset originale. Questa tecnica, nota come data augmentation, consente di migliorare significativamente le prestazioni e la capacità di generalizzazione dei modelli di Machine Learning. L'obiettivo di questo lavoro di tesi è ottimizzare le prestazioni di un classificatore progettato per prevedere la tipologia di intervento chirurgico a cui un paziente verrà sottoposto. Per fare ciò si è prima ricercato il modello di ML più performante sul dataset originario, un dataset tabulare fornito dall'Azienda Ospedaliero-Universitaria di Parma, contente diversi parametri anamnestici raccolti da pazienti operati nella struttura. Successivamente sono stati generati diversi dataset aumentati, tra questi, sono stati selezionati quelli che hanno prodotto i modelli più performanti quando utilizzati per addestrare il classificatore. Infine sono stati confrontati i modelli addestrati

sul dataset originale con quelli addestrati sui dataset aumentati per verificare se le tecniche utilizzate possano apportare delle migliorie, in termini di performance, al modello di classificazione.

Capitolo 1

Stato dell'arte

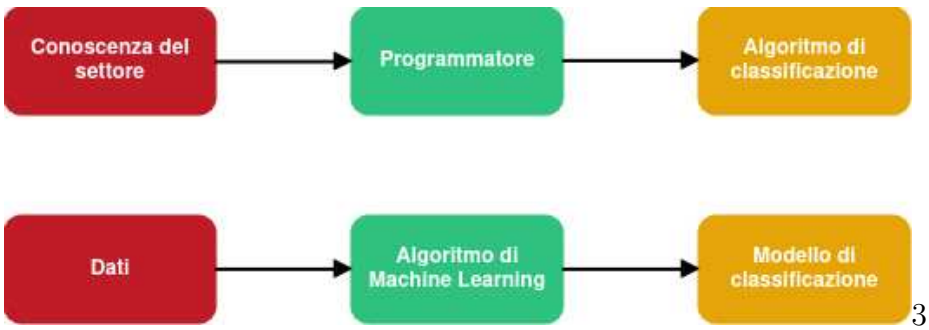
In questo capitolo verranno illustrate le tecnologie che costituiscono lo stato dell'arte nell'ambito della classificazione automatica e della data augmentation

1.1 Machine Learning

Il Machine Learning, d'ora in poi ML, è quella disciplina dell'informatica che si occupa di sviluppare tecniche e algoritmi che permettono ai calcolatori di risolvere problemi senza essere programmati esplicitamente. L'idea alla base del ML è di utilizzare esempi, in forma di dati forniti al calcolatore, per far sì che questo possa elaborare autonomamente una strategia di risolutiva per il problema dato. Per capire la differenza tra un approccio convenzionale e uno che fa uso del ML si supponga ad esempio di dover automatizzare la risoluzione di un problema di *classificazione*, ossia di dover sviluppare un software che assegni una *istanza* a una delle possibili *classi* di appartenenza .

Una prima strategia potrebbe essere quella di sviluppare un *sistema esperto*, ossia utilizzare la specifica conoscenza del settore per estrarre delle regole con cui poi sviluppare un algoritmo che imiti il processo di decision-making di un esperto.

Qualora si volesse invece risolvere il problema usando il ML sarebbe sufficiente avere un dataset contenente un numero sufficiente di istanze, descritte da *features* e assegnate a una classe attraverso un *label*. In questo caso la conoscenza specifica del settore necessaria per effettuare correttamente una classificazione sarebbe implicitamente contenuta nel dataset e verrebbe trasferita a un *modello* creato da un *algoritmo di ML* eseguito sui dati.



3

Figura 1.1: Schema sviluppo modello Machine Learning

Il grande vantaggio dell'utilizzare il ML è quello di poter automatizzare tutti quei compiti il cui processo esecutivo è difficile da definire esplicitamente attraverso regole o procedure programmabili manualmente. Si pensi ad esempio alla classificazione di immagini, attività facilmente eseguibile per un essere umano ma il cui svolgimento è molto difficile da descrivere algoritmicamente.

1.1.1 Ensemble learning

L'*Ensemble learning* è una famiglia di algoritmi di ML che prevede l'addestramento e l'aggregazione di più modelli di classificazione o regressione. L'idea alla base di questi algoritmi è che la combinazione di output prodotti da più *modelli deboli*, ossia modelli che hanno una capacità di predizione leggermente superiore a una assegnazione casuale, sia migliore dell'output prodotto da un singolo modello forte. Esiste una grande variet'a di algoritmi

di enseble learning. Questi differiscono nelle modalitá con le quali i modelli deboli vengono addestrati e aggregati per formare un modello forte. Tra i più noti e utilizzati vi sono algoritmi di *bagging* e *boosting*.

Algoritmi di bagging

Gli algoritmi di *bootstrap-aggregating* o brevemente *bagging* vengono largamente utilizzati per migliorare le performance di modelli particolarmente soggetti a overfitting, ossia ad adattarsi troppo al dataset di addestramento, perdendo capacità di generalizzazione. L'esecuzione di un algoritmo di bagging prevede due fasi:

- Bootstrap: Nella fase di bootstrap si producono n dataset a partire dal dataset originale e si addestra un modello debole per ogni dataset generato. La generazione avviene effettuando m estrazioni con ripetizione dal dataset originale.
- Aggregazione: Nella fase di aggregazione gli n modelli deboli addestrati vengono uniti in un unico modello forte. Questo effettua una predizione combinando le predizioni dei modelli deboli.

Algoritmi di boosting

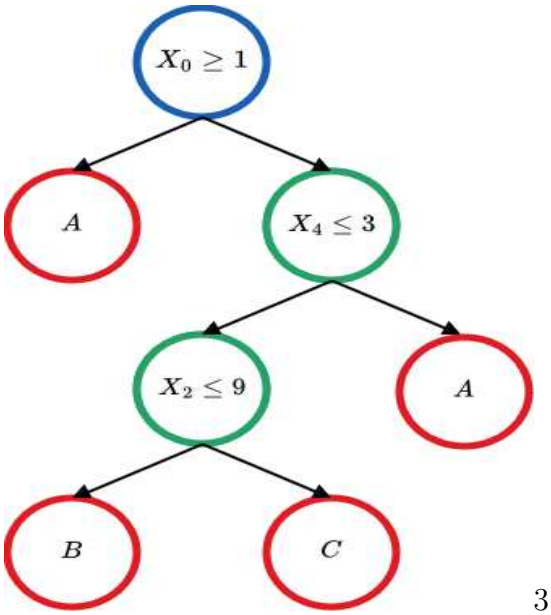
Un algoritmo di *boosting* prevede solitamente due fasi di esecuzione

- Generazione dataset pesati: In questa fase si generano n dataset differenti. Il primo dataset generato è uguale al dataset originario, questo viene usato per addestrare un modello debole. Il dataset successivo viene generato pesando i dati contenuti dal dataset originario in base agli errori commessi dal modello debole, il valore del peso è direttamente proporzionale alla misura dell'errore compiuto. I dataset successivi vengono generati con le stesse modalità.
- Aggregazione: In questa fase i modelli deboli addestrati vengono aggregati con modalità simili a quelle seguite negli algoritmi di bagging.

1.1.2 Random Forest

Random Forest (RF) [1] è uno degli algoritmi di *Ensemble Learning* più noti e utilizzati per task di classificazione e regressione. L'idea alla base del suo funzionamento, e di tutti gli algoritmi di Ensemble Learning, è che la combinazione di output prodotti da più modelli deboli, ossia modelli che hanno una capacità di predizione leggermente superiore a una assegnazione casuale, sia migliore dell'output prodotto da un singolo modello forte. Per comprendere pienamente RF si deve prima parlare del modello debole che utilizza, l'*albero decisionale* Un albero decisionale è una struttura gerarchica ad albero binario composta da :

- *Nodo radice*
- *Nodi interni*
- *Nodi foglia*
- *Rami*



3

Figura 1.2: Albero decisionale

Si supponga di avere un dataset con n feature: $X_0, X_1, ..., X_n$ e di volerne classificare una istanza con un albero decisionale. La classificazione avviene attraverso un numero finito di spostamenti all'interno dell'albero. La direzione di ogni spostamento dipende dal valore di una delle X_i feature dell' istanza e da un valore soglia Y_i associato al *nodo*. Lo spostamento tra i nodi avviene attraverso i *rami*, questi portano dal nodo radice a uno dei *nodi foglia*. I nodi foglia rappresentano la classe a cui viene assegnata l'istanza e hanno cardinalità identica a quella delle classi.
Il processo di apprendimento consiste nella scelta del numero di nodi interni e nelle coppie (X_i, Y_i) associate a ognuno di essi. Per ogni nodo si inizializzano casualmente molte coppie (Xi, Yi) per poi selezionare quella che massimizza una metrica di correttezza. Quest'ultima costituisce un *iperparametro* del modello. Tra le più utilizzate vi sono:

- *Information gain*
- *Gini index*

RF genera una *foresta casuale* di alberi decisionali $\{T_1, T_2, \ldots, T_n\}$, ossia un insieme di alberi generati casualmente che vengono utilizzati come un unico modello di classificazione. L'algoritmo generativo è il seguente:

Algoritmo 1 Generazione Foresta Casuale

1: **Input**: Dataset D, numero di alberi n, numero di feature da selezionare m
2: **Output**: Lista di alberi di decisione $\{T_1, T_2, \ldots, T_n\}$
3: **for** $i = 1$ **a** n **do**
4: $D_i \leftarrow$ Campionamento casuale di D con sostituzione
5: $F_i \leftarrow$ Selezione casuale di m feature da D
6: $T_i \leftarrow$ Costruzione di un albero di decisione usando D_i e F_i
7: **restituisci** $\{T_1, T_2, \ldots, T_n\}$

Dove $\boldsymbol{n}$ e $\boldsymbol{m}$ sono iperparametri del modello.
La foresta generata viene poi utilizzata per classificare un'istanza selezionan-

do la classe predetta dalla maggioranza relativa degli alberi che la compongono.

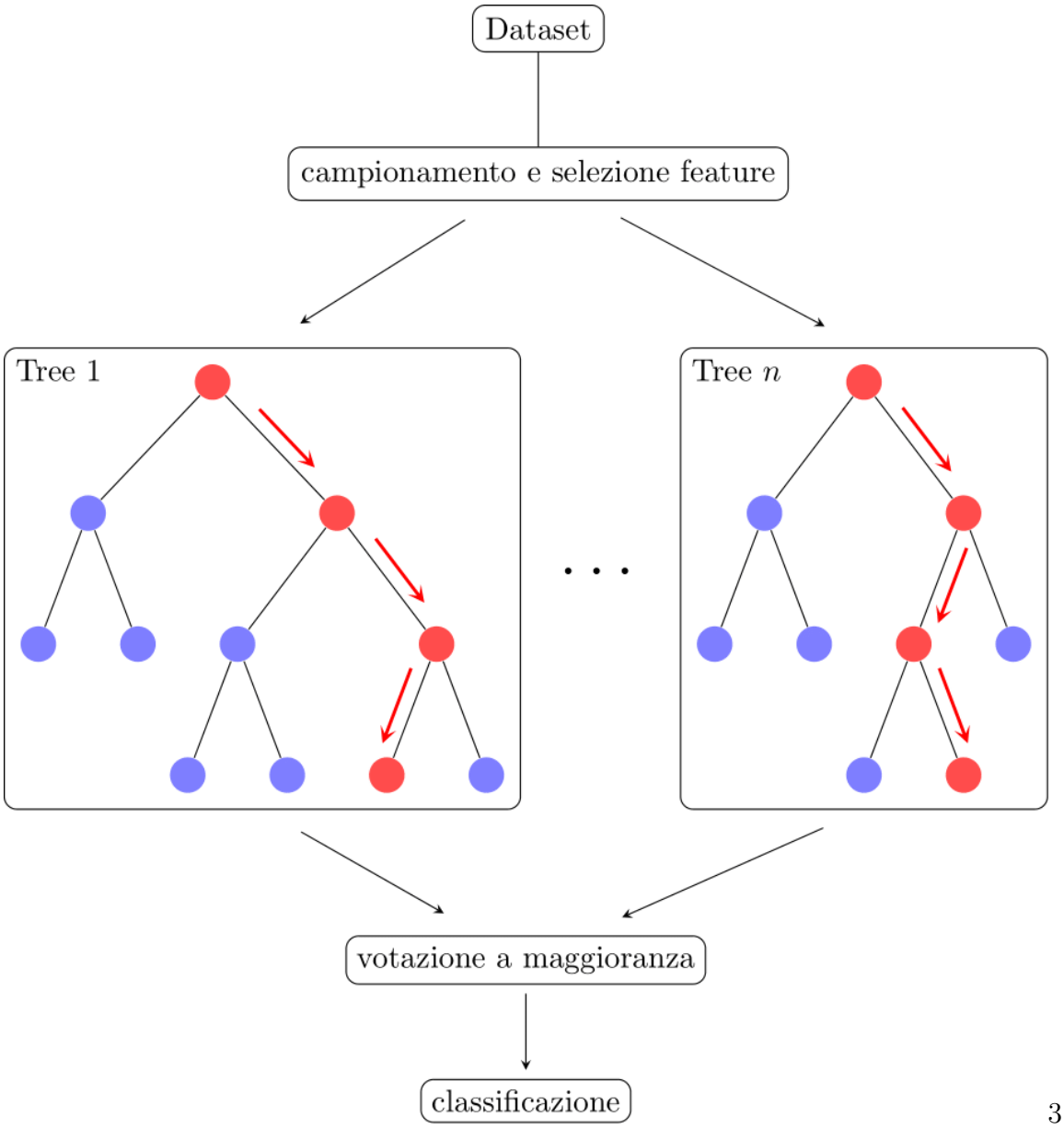


Figura 1.3: Random Forest

1.1.3 AdaBoost

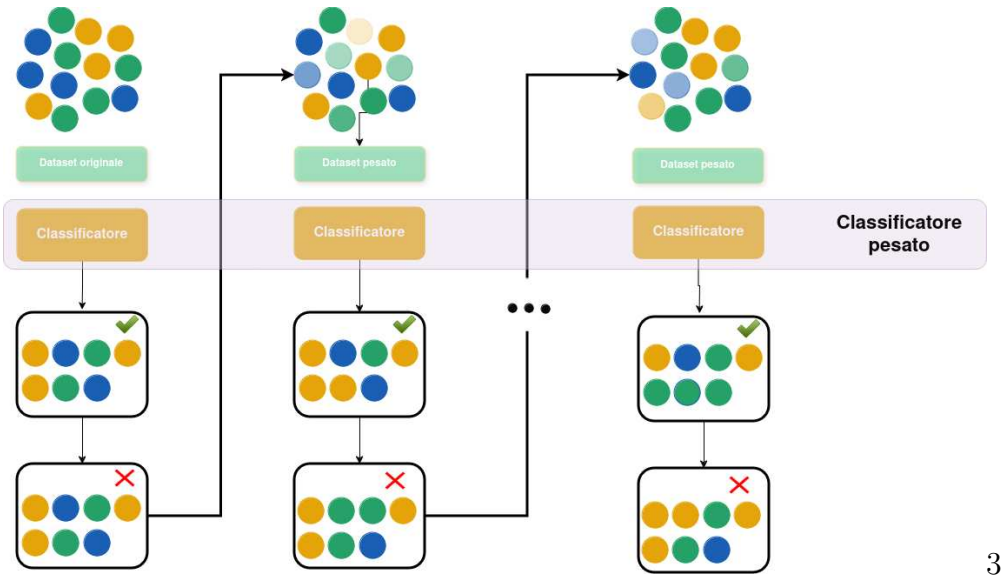
AdaBoost [2] , acronimo di Adaptive Boosting, è un algoritmo di Ensemble Learning ampiamente utilizzato per task di classificazione . La sua peculiarità risiede nel modo in cui viene costruita la combinazione di modelli deboli per formare un modello forte.
L'idea chiave di AdaBoost è quella di assegnare dei pesi ai dati di addestramento in modo iterativo, concentrandosi sui dati più difficili da classificare. In ogni iterazione, si costruisce un modello debolmente predittivo su un sottoinsieme dei dati, assegnando maggior peso ai dati che sono stati classificati erroneamente nelle iterazioni precedenti. Alla fine del processo di addestramento, si combinano i modelli deboli assegnando loro dei pesi in base alla loro accuratezza.
La generazione di un modello di AdaBoost può essere descritto dal seguente algoritmo:

Algoritmo 2 Generazione modello di AdaBoost

1: **Input**: Dataset D di dimensione N, numero di modelli deboli M, numero di classi K
2: **Output**: Modello addestrato $H(x)$
3: Inizializza i pesi dei dati $w_i = \frac{1}{N}$ per $i = 1, \ldots, N$
4: **for** $m = 1$ **a** M **do**
5: Addestra un modello debole $h_m(x)$ su D pesato con w
6: Calcola l'errore pesato $e_m = \sum_{i=1}^{N} w_i * I((h_m(x_i) \neq y_i)$
7: Calcola il peso del modello $\alpha_t = \frac{1}{2} \ln\left(\frac{1-e_m}{e_m}\right)$
8: Aggiorna i pesi dei dati: $w_i \leftarrow A(w_i, y_i, h_m(x_i))$
9: Normalizza i pesi: $w \leftarrow \frac{w}{\sum_{i=1}^{N} w_i}$
10: **restituisci** il modello addestrato $H(x) = \text{argmax}_k \left(\sum_{m=1}^{T} \alpha_t I(h_m(x) = k)\right)$

Dove $\boldsymbol{I}$ è la funzione indicatrice, vale 1 se il suo argomento è un espressione vera e 0 diversamente, $\boldsymbol{A}$ è la funzione di aggiornamento dei pesi e $\boldsymbol{M}$

numero di modelli deboli è un iperparametro dell'algoritmo.



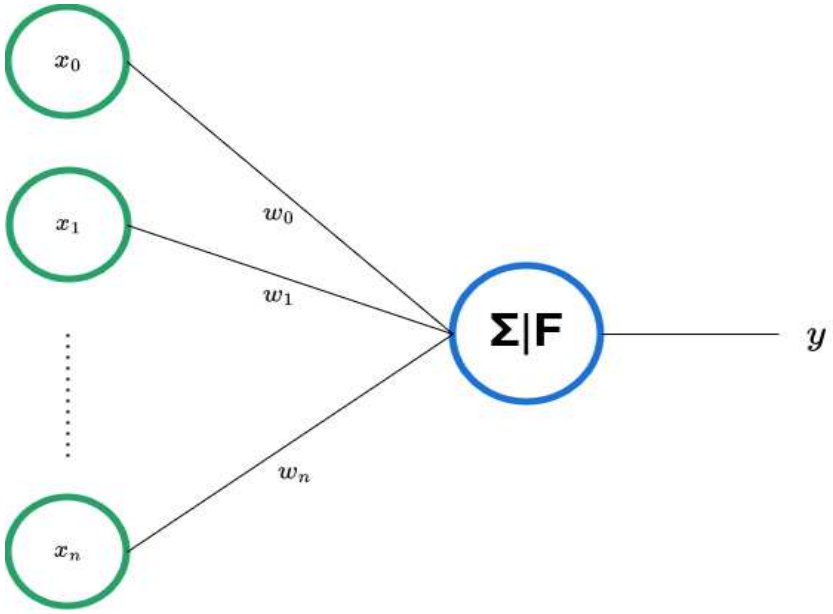
3

Figura 1.4: Adaboost

Una volta addestrato, il modello $H(x)$ può essere utilizzato per classificare nuovi dati assegnando loro l'etichetta predetta dalla classe che massimizza la somma pesata dei voti dei modelli deboli.

1.1.4 Rete Neurale

La *Rete Neurale* è senza dubbio il modello di ML che ha permesso di ottenere i risultati più significativi nel campo dell'intelligenza artificiale.
La sua architettura, ispirata a quella di un circuito neurale biologico è basata sulla interconnessione tra più unità di neuroni artificiali.
Un *neurone artificiale* è una struttura ispirata a un neurone biologico. Matematicamente può essere descritto come una funzione che riceve uno o più input e restituisce un singolo output.



3

Figura 1.5: Neurone artificiale

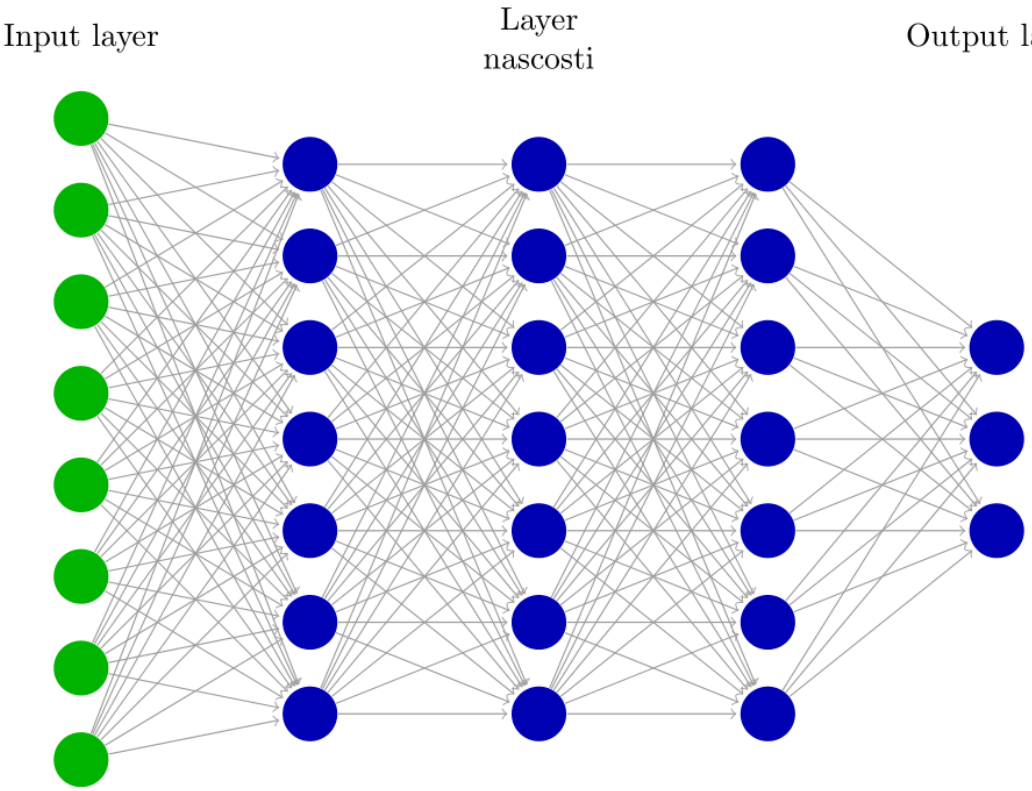
Gli input del neurone sono costituiti da output di altri neuroni moltiplicati per dei *pesi* $(w_0, w_1, ..., w_n)$. Gli input pesati vengono sommati a un *bias* e forniti in input a una *funzione di attivazione*, generalmente non lineare, il cui output costituisce quello del neurone stesso.

$$y = F\left(\sum_{i=1}^{n} w_i x_i + b\right)$$

Dove:

- x_i sono gli input del neurone,
- w_i sono i pesi associati agli input,
- $F()$ è la funzione di attivazione:
- b è il bias
- y è l'output del neurone

Una rete neurale è formata da *strati di neuroni* collegati tra loro.



3

Figura 1.6: Rete neurale profonda

La "conoscenza" della rete è costituita dai pesi e bias dei suoi neuroni. Il processo di apprendimento consiste nell'individuazione di un insieme di pesi e bias che minimizzano una *funzione di costo*, misura di quanto la rete tenda a compiere errori.
La tecnica più utilizzata per addestrare la rete consiste nell'applicare l'algoritmo di *backpropagation*. Questo algoritmo aggiorna il valore di pesi e bias sommando a questi il gradiente della funzione di costo della rete moltiplicato per un *learning rate*. Quest'ultimo è un importante iperparametro del modello la cui scelta influenza pesantemente le performance della rete.
La potenza e la versatilità di questo modello è evidenziato da un importante

risultato teorico: il *Teorema di approssimazione universale*. Questo afferma che con una adeguata configurazione una rete neurale possa approssimare un'amplissima classe di funzione con precisione arbitraria.

1.2 Data augmentation

La *data augmentation* è un insieme di tecniche che permette di creare dati sintetici che riproducono la struttura e la distribuzione statistica appartenente a dei dati reali. I dati generati possono poi essere utilizzati da un modello di ML durante la fase di apprendimento. Utilizzare dati generati sintetici presenta molti vantaggi:

- Migliorare la generalizzazione del modello: L'aggiunta di dati sintetici puó aiutare il modello di ML a generalizzare meglio, ovvero a fare previsioni accurate su dati non visti.
- Aumentare la robustezza del modello: Esponendo il modello a una varietÃ più ampia di dati attraverso la data augmentation, si può rendere il modello più robusto alle variazioni nei dati di input e alle condizioni ambientali.
- Ridurre il rischio di overfitting: L'overfitting si verifica quando il modello memorizza il rumore nei dati di addestramento anzichè imparare i pattern rilevanti. L'aggiunta di dati sintetici può aiutare a ridurre questo rischio, specialmente quando si lavora con dataset limitati.
- Affrontare problemi di sbilanciamento delle classi: La data augmentation può essere utilizzata per bilanciare le classi in uno scenario di classificazione dove alcune classi sono sottorappresentate, migliorando cos'i le prestazioni del modello.
- Proteggere la privacy dei dati: Utilizzando dati generati sinteticamente anzichè dati reali, è possibile proteggere la privacy dei dati sensibili.

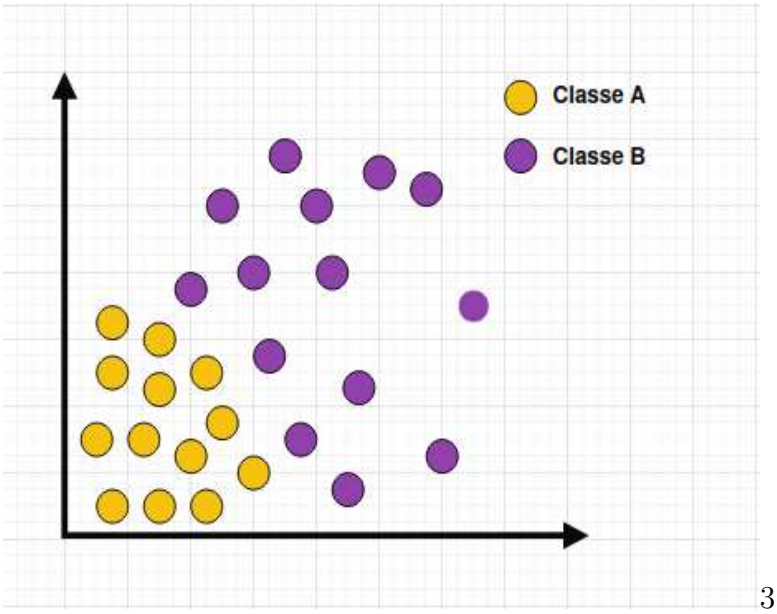
Questo è particolarmente importante in contesti in cui la privacy dei dati è una preoccupazione, come nel caso dei dati sanitari o finanziari.

Le tecniche più utilizzate possono essere raggruppate in due grandi categorie:

- Generazione di dati tramite algoritmi di sovracampionamento
- Generazione di dati tramite modelli generativi

1.2.1 Smote

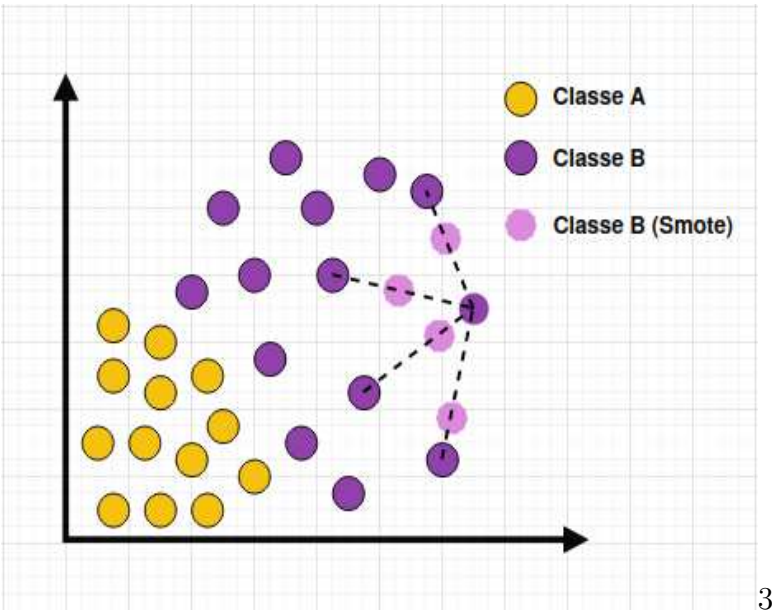
Synthetic minority over-sampling technique *(SMOTE)* [3] è una delle tecniche di sovracampionamento più note. Viene utilizzata sopratutto per ribilanciare un dataset con label categoriche in cui una o più classi sono sottorappresentate, ma può anche essere sfruttata per andare semplicemente ad aumentare le dimensioni del dataset. L'idea alla base del suo funzionamento è che rappresentando geometricamente un insieme di dati, sia possibile individuare zone nello spazio contenenti principalmente istanze di una certa classe.



3

Figura 1.7: Classi in un piano geometrico

Si supponga di voler creare k nuove istanze della classe B, per prima cosa si seleziona casualmente uno dei punti appartenenti a B. Successivamente si individuano i k punti più vicini a quello selezionato e per ognuno di essi si determina la retta che li congiunge al punto originale. Infine, per ogni retta, si seleziona casualmente un punto che giace su di essa. I punti selezionati costituiscono le k nuove istanze della classe.



3

Figura 1.8: Generazione di nuove istanze sintetiche con Smote

In generale se si vuole sovraccampionare una classe C_i si esegue il seguente algoritmo:

Algoritmo 3 Synthetic Minority Over-sampling Technique

Input: Dataset di addestramento D, fattore di sovracampionamento N, numero di vicini k, classe da sovracampionare C_i
Output: Dataset sovracampionato D_{SMOTE}
for ogni campione x_i della classe C_i **do**
    Trova i k vicini più vicini di x_i
    for $n = 1$ **a** N **do**
        Scegli casualmente un vicino x_{nbor} tra i k vicini di x_i
        Calcola il vettore differenza diff $= x_{nbor} - x_i$
        Genera un nuovo campione sintetico $x_{\text{syn}} = x_i + \text{diff} \times \text{rand}(0, 1)$
        Aggiungi x_{syn} a D_{SMOTE}
restituisci D_{SMOTE}

1.2.2 Modelli generativi

Un modello generativo è un particolare tipo di modello di ML, generalmente una rete neurale, capace di apprendere la distribuzione dei dati di allenamento e di produrre nuovi dati sintetici che rispettino questa distribuzione. Esistono diverse architetture di pensate per questo scopo. Tra le più utilizzate vi sono: Dove:

- Variational autoencoder
- Generative adversial network
- Transformer
- Normalizing flow

Nei seguenti capitoli verranno trattati nel dettaglio le architetture utilizzate all'interno di questa tesi.

Variational autoencoder

Un *Variational autoencoder (VAE)* [4] è un modello basato su un architettura encoder-decoder. Queste sono due componenti distinte con funzioni opposte:

- Encoder: L'encoder converte l'input in una rappresentazione latente o compressa. È costituito da uno o più strati di neuroni che trasformano l'input in uno spazio di rappresentazione più compatto e significativo. La compressione viene ottenuta riducendo il numero di neuroni per strato fino a raggiungere un collo di bottiglia.
- Decoder: Il decoder prende la rappresentazione latente generata dall'encoder e la decodifica per generare l'output desiderato. È composto da uno o più strati di neuroni che trasformano la rappresentazione latente in un formato appropriato per l'output desiderato. La decompressione viene ottenuta aumentando il numero di neuroni per strato fino a raggiungere la dimensionalità dell'input.

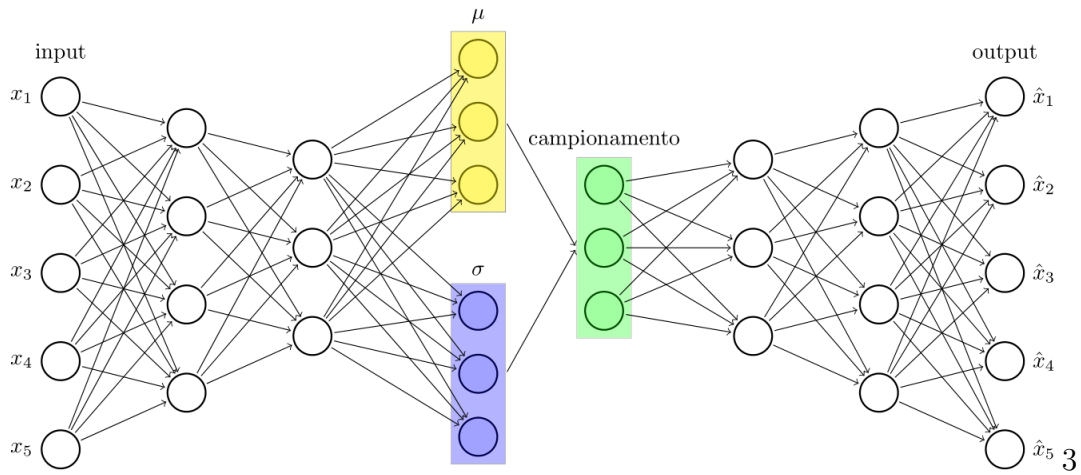


Figura 1.9: Variational autoencoder

La particolarità del VAE è di associare a ogni input una distribuzione normale, gli input vengono infatti compressi in due vettori: Un *vettore media* $\boldsymbol{\mu}$ e un *vettore varianza* $\boldsymbol{\sigma}$ che dopo essere campionati costituiranno l'input del

decoder.
Associare una distribuzione a un input permette di poter creare uno spazio latente "regolare", ossia uno spazio in cui input con caratteristiche simili vengono mappati in punti geometricamente vicini. La regolarità viene ottenuta utilizzando una funzione di loss composta da un termine di ricostruzione e un termine di regolarizzazione. Quest'ultimo è costituito dalla misura di divergenza di *Kullback-Leibler* tra la distribuzione che costituisce l'output dell'encoder e una gaussiana centrata nell'origine dello spazio latente:

$$D_{\mathrm{KL}}(P||Q) = \int_{-\infty}^{\infty} p(x) \log \frac{p(x)}{q(x)}\, dx$$

Uno spazio latente regolare rende la rete particolarmente adatta alla produzione di nuovi dati sintetici. Per produrre un nuovo dato è infatti sufficiente campionare casualmente lo spazio latente e riassociare un dato al punto attraverso il decoder.

Normalizing flow

Un *Normalizing flow (NF)* [5], ossia un modello basato sulla normalizzazione del flusso sfrutta il *Teorema di trasformazione delle variabili aleatorie* per trasformare una distribuzione semplice in una distribuzione complessa che approssimi quella di un dataset fornito in input.
In questo modello la fase di generazione consiste nel campionamento della distribuzione complessa generata dalle trasformazioni applicate.
Le trasformazioni applicate alla variabile aleatoria, difficilmente determinabili a priori, sono effettuate da una rete neurale. Durante il training del modello vengono scelti i parametri che implementano le trasformazioni migliori, ossia quelli che permettono di generare la distribuzione più fedele a quella dei dati di addestramento.

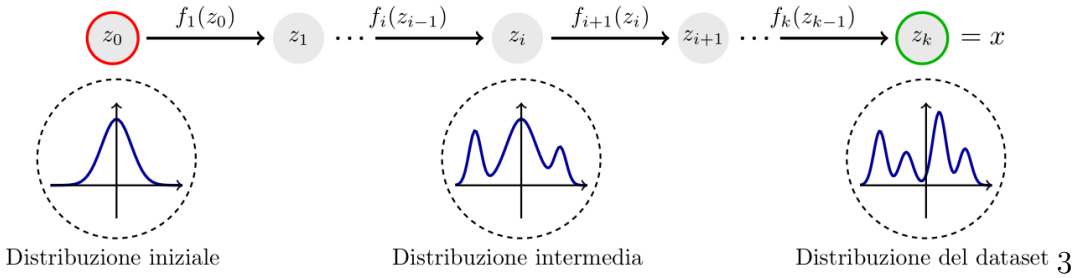


Figura 1.10: Normalizing flow

Capitolo 2

Implementazione

2.1 Strumenti di sviluppo

Per sviluppare il codice necessario al lavoro di tesi è stato utilizzato *Python*, linguaggio interpretato molto utilizzato in ambito ML per la sua semplicitá di utilizzo e l'abbondanza di librerie dedicate.
È stato fatto un utilizzo estensivo di *Scikit-learn*, libreria open-source di machine learning, inizialmente sviluppata da David Cournapeau come parte del suo lavoro di tesi presso il Laboratoire de Recherche en Informatique (LRI) dell'Universitè Paris-Sud, tra le più utilizzate e popolari. Scikit-learn offre moltissimi strumenti utili in ambito ML come algoritmi di classificazione e regressione, metriche, algoritmi di data augmentation e preprocessing.
Tutte le reti neurali utilizzate in questa tesi sono state create e addestrate utilizzando *Tensorflow* e *Keras*. Tensorflow è una libreria open-source sviluppata da Google-Brain nel 2015. Fornisce tutti gli strumenti di basso livello necessari per implementare algoritmi di deep learning. Keras, creata da Francois Chollet, fornisce API di alto livello per la costruzione di reti neurali. È basata su Tensorflow e permette di creare reti in poche righe di codice.
Le tecniche di data augmentation basate su reti generative profonde sono state implementate utilizzando *Synthcity* [6]. Synthcity è una libreria open-

source basata su Pytorch sviluppata dal Vanderschaar-lab di Oxford. Permette di creare e valutare dati tabulari sintetici generati attraverso algoritmi di deep learning allo stato dell'arte. Ad oggi la libreria dispone dipiù di 25 modelli. Tra questi alcuni possono essere di grande utilità nella data augmentation in ambito medico in quanto incentrati sul livello di privacy dei dati sintetici prodotti.

2.2 Dataset

Il dataset utilizzato è stato creato grazie a una collaborazione tra l'Università di Parma e l'Azienda ospedaliero-universitaria di Parma. Il dataset è composto da dati provenienti da *1121 pazienti* operati presso l'Azienda ospedaliero-universitaria di Parma. A ogni paziente sono state associate *29 feature*, queste sono costituite da informazioni anagrafiche, dati provenienti dalla sua cartella clinica o misurazioni effettuate dal personale dell'ospedale durante la degenza del paziente. Delle 29 feature, riportate in tabella 2.1, 4 sono numeriche e 25 categoriche. Il fatto che le feature del dataset siano sia categoriche che numeriche rende più difficile la fase di data augmentation.

Tabella 2.1: Feature del dataset

Feature	tipo
Età	numerico
Sesso	numerico
Peso	numerico
Altezza	numerico
Bmi	categorico
Fumo	categorico
Osas	categorico
Bpco	categorico
Ipertensione arteriosa	categorico
Cardiopatia ischemica cronica	categorico
Pregresso infarto miocardio	categorico
Pregresso SCC	categorico
Ictus	categorico
Pregresso TIA	categorico
Altro comorbidita	categorico
Antipertensivi	categorico
Broncodilatatori	categorico
Antiaritmici	categorico
Anticoagulanti	categorico
Antiaggreganti	categorico
Tigo	categorico
Insulina	categorico
Altro terapia	categorico
Diabete mellito tipo 2	categorico
Diabete mellito tipo 1	categorico
Aritmia NO	categorico
Aritmia FA	categorico
Aritmia TACHI	categorico
Aritmia TPSV	categorico

Le operazioni a cui sono stati sottoposti i pazienti sono di tre tipi:

- Interventi sul sistema endocrino
- Interventi sull'apparato digerente
- Interventi sul sistema cardiovascolare

Le tipologie di intervento costituiscono le *3 variabili target* del dataset.
Il numero e la distribuzione delle operazioni effettuate appare nella tabella 2.2

Tabella 2.2: Distribuzione del dataset

Numero pazienti	Operazione effettuata	Percentuale
784	Operazione apparato digerente	70%
207	Operazione sistema endocrino	19%
130	Operazione sistema cardiovascolare	11%

Appare evidente il forte sbilanciamento tra le classi del dataset, con quasi il 75% dei casi appartenenti alla classe riguardante l'apparato digerente. Uno sbilanciamento così forte rende difficile ottere un classificatore con prestazioni ottimali. Visto che la maggiorparte delle istanze sono appartenenti alla classe apparato digerente durante la fase di training il classificatore potrebbe presentare un fenomeno di overfitting su questa particolare classe. Lo sbilanciamento costituisce inoltre un problema per il calcolo stesso delle prestazioni: i dataset utilizzati per il calcolo delle metriche sono creati con istanze appartenti al dataset originale, si avrà allora necessariamente che anche questi dataset saranno sbilanciati. Per questo sono state utilizzate delle metriche che risultino affidabili anche in caso di sbilanciamento.

2.3 Preprocessing

Il preprocessing dei dati è di fondamentale importanza nella riuscita di un processo di ML. In questo capitolo verranno riportate tutte le trasformazioni effettuate sul dataset.

2.3.1 Divisione del dataset in train, validation e test set

Affinchè un dataset possa essere utilizzato per allenare e testare un modello, questo va diviso in 3 parti distinte

- Train set: La parte del dataset sul quale il modello viene allenato
- Validation set: La parte del dataset utilizzata per effettuare l'hyperparameter tuning del modello e la selezione del modello addestrato con le prestazioni migliori
- Test set: La parte del dataset utilizzata per calcolare le prestazioni del modello

La distribuzione del dataset dopo la divisione in train, validation e test set è la seguente:

Tabella 2.3: Distribuzione del dataset

Numero pazienti	Train	Validation	Test
Totale	826	207	88
Operati apparato digerente	599	155	30
Operati apparato endocrino	147	32	28
Operati sistema cardiovascolare	80	20	30

2.3.2 Normalizzazione

La normalizzazione è una tecnica che viene utilizzata per portare i valori delle feature di un dataset in un range limitato, solitamente [0,1]. Molti modelli, tra cui ad esempio quelli basati sul calcolo del gradiente della funzione di costo, risultano piú stabili in fase di addestramento se le feature hanno valori compresi nell'intervallo [0,1] . Nel dataset in questione sono state normalizzate le sole feature numeriche, nello specifico è stato utilizzato il MinMaxScaler di Scikit-learn che associa a una features x un valore scalato calcolato secondo la seguente formula:

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

Dove:

- $x_{\min}$: è il valore minimo assunto dalla feature
- $x_{\max}$: è il valore massimo assunto dalla feature

Train , validation e test set sono stati normalizzati *separatamente*. Normalizzare il dataset prima di averlo diviso potrebbe falsificare la valutazione del modello. Si supponga di avere un dataset in cui per una feature x il valore massimo $x_{\max}$ o minimo $x_{\min}$ appartenga al test set, questo valore verrebbe utilizzato per normalizzare il training set, trasferendo a quest'ultimo informazione proveniente dal test set.

2.4 Metriche utilizzate

In questa sezione sono riportate le metriche selezionate per valutare le performance di ogni modello. Le metriche utilizzate nei task di classificazione sono numerose e la loro validità dipende fortemente dal modello e dal dataset che si utilizza.

2.4.1 Accuracy

L'accuracy, la metrica in assoluto più semplice, è la percentuale di classificazioni corrette su totale:

$$\text{Accuracy} = \frac{\text{Classificazioni corrette}}{\text{Totale classificazioni}}$$

L'accuracy è valida solo quando il dataset che si sta utilizzando è bilanciato, ossia quando il numero di istanze per classe è all'incirca lo stesso .

2.4.2 Precision e recall

Precision e recall sono metriche utilizzate nella classificazione binaria.
Si calcolano a partire da: :

- TP (True positive) : Numero di istanze correttamente etichettate dal modello come appartenenti alla classe positiva
- FP (False positive) : Numero di istanze erroneamente etichettate dal modello come appartenenti alla classe positiva
- FN (False negative) : Numero di istanze correttamente etichettate dal modello come non appartenenti alla classe positiva [1]

La precision è pari al rapporto tra veri positivi e veri positivi più falsi positivi

$$\text{Precision} = \frac{TP}{TP+FP}$$

La recall è pari al rapporto tra veri positivi e veri positivi più falsi negativi

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

L'utilizzo di precision e recall può essere esteso alla classificazione multiclasse valutando le metriche per ogni classe e poi calcolandone media e media ponderata.

[1] Il termine *positivo* si utilizza per riferirsi alla classe di cui si stanno calcolando le metriche

2.4.3 F1Score

Precision e recall sono metriche tra le queli esiste un trade-off. Un modello che genera molti falsi negativi tende a generare pochi falsi positivi e viceversa, creando un rapporto di proporzionalità inversa tra le due metriche. Se risulta necessario massimizzare sia precision che recall si può monitorare la media armonica tra le due metriche. Questa costituisce la F1Score:

$$\text{F1Score} = \frac{\text{precision*recall}}{\text{precision+recall}}$$

Differentemente dalla accuracy la F1Score risulta valida anche quando viene valutata su dataset non bilanciati.

2.4.4 Matrice di confusione

Una matrice di confusione constiste in una tabella con una disposizione specifica utilizzata per visualizzare con semplicitÃ le prestazioni di un classificatore. Pur non essendo formalmente una metrica, la matrice di confusione rappresenta uno strumento molto utilizzato per l'analisi delle prestazioni di un classificatore. Le *righe* della matrice rappresentano le *classi attuali* di ogni istanza, le *colonne* rappresentano invece le *classi predette* dal modello. In ogni posizione la matrice contiene il numero di istanze appartenti alla classe indicata della riga e classificate come appartententi alla classe indicata dalla colonna.

		Predette	
		Classe 1	*Classe 2*
Vere	*Classe 1*	50	10
	Classe 2	5	35

Tabella 2.4: Confusion Matrix

Una matrice di confusione con molti elementi nella diagonale e pochi elementi altrove è un buon indicatore di un modello robusto.

Capitolo 3

Architettura del sistema realizzato

3.1 Algoritmi utilizzati

Selezionare a priori l'algoritmo più adatto per un task di classificazione non è un compito banale. Di conseguenza sono stati valuatati diversi algoritmi:

- Random Forest
- AdaBoost
- Rete neurale

Per ogni algoritmo si procederà prima a ricercare gli *hyperparameter* migliori e successivamente si selezioneranno i modelli più prestazionali

3.2 Hyperparameter tuning

Gli hyperparameter di un algoritmo sono parametri che regolano il processo di apprendimento e l' architettura del modello. Questi paramteri non possono essere appresi durante l'allenamento del modello e vanno quindi selezionati a priori.

Nel processo di ML la fase di selezione degli hyperparameter prende il nome di*hyperparameter tuning*.
Per effettuare l'hyperparameter tuning dei tre algoritmi selezionati è stata utilizzata la *GridSearch*. Questa tecnica consiste in una *ricerca esaustiva* in uno spazio definito di hyperparameter. Per ogni possibile combinazione di hyperparameter si addestra un modello e si valutano le prestazioni con una metrica di riferimento. Nel caso specifico, la ricerca è stata effettuata addestrando i modelli sul train set e testandoli sul validation set. Come è possibile osservare dalla tabella 2.3 il validation test presenta un forte sbilanciamento, di conseguenza si è scelto di prendere la *Macro F1Score* come metrica di riferimento, quest'ultima risulta infatti affidabile anche se utilizzata con dataset sbilanciati. Nel caso di AdaBoost e Random Forest è stato utilizzata la funzione *GridSearchCV* di Scikit-learn, per la rete neurale è stata invece utilizzata la funzione *GridSearch* di Keras tuner.
Gli spazi degli hyperparameter dei tre algoritmi sono riportati nella tabella 3.1

Tabella 3.1: Spazio di ricerca degli hyperparameter

Modello	*Iperparametro*	*Valori*
Rete neurale	Learning rate	0.0001, 0.001
	Numero neuroni per layer	30
	Momento	0.0001, 0.001
	Numero di layers interni	1, 2, 3
	Funzione di attivazione layer input	ReLU
	Funzione di attivazione layers interni	ReLU
	Funzione di attivazione layer output	Softmax
	Dropout layers interni	0.3, 0.4, 0.5, 0.6
	Dropout layer input	0.1, 0.2, 0.3
	Metodo di ottimizzazione	SGD
AdaBoost	Algoritmo modello debole	GaussianNaiveBayes ,StochasticGradientDescent ,DecisionTree
	Learning rate	0.001, 0.001, 0.01, 0.1
	Algoritmo	SAMME
	Numero modelli deboli	150, 200 ,250 ,300 ,350 ,400 ,450
Random Forest	Numero di Alberi	150, 200, 250, 300, 350, 400, 450, 500
	Criterio selezione nodi	Gini, Entropia, Logloss
	Criterio numero massimo di feature	Sqrt, Log2

Quando non esplicitamente specificato vengono utilizzati gli hyperparameter di default delle implementazioni di Adaboost e Random Forest della libreria Scikit-learn

Gli hyperparameter selezionati sono riportati nella tabella 3.2

Tabella 3.2: hyperparameter selezionati

Modello	*Iperparametro*	*Valori*
Rete neurale	Learning rate	0.001
	Momento	0.001
	Numero di layers interni	2
	Funzione di attivazione layer input	ReLU
	Funzione di attivazione layers interni	ReLU
	Funzione di attivazione layer output	Softmax
	Dropout layers interni	0.4
	Dropout layer input	0.2
	Metodo di ottimizzazione	SGD
AdaBoost	Algoritmo modello debole	StochasticGradientDescent
	Learning rate	0.01
	Algoritmo	SAMME
	Numero modelli deboli	300
Random Forest	Numero di Alberi	300
	Criterio selezione nodi	Entropia
	Criterio numero massimo di feature	Sqrt

Tutti i modelli di Adaboost, Randomforest e rete neurale verranno inizializzati utilizzando questi parametri.

3.2.1 Training e selezione dei modelli migliori

Ottenuti gli hyperparameter migliori per la rete neurale, Random Forest e AdaBoost é possibile procedere con il training e la selezione dei modelli migliori.

Rete neurale

La rete neurale è stata allenata utilizzando *Stochastic gradient descent (SGD)*, questo metodo di ottimizzazione permette di calcolare il gradiente della funzione di costo e di aggiornare i parametri su *batch* (sottoinsiemi di pari dimensioni del dataset), rendendo il processo molto meno costoso computazionalmente.
La funzione di costo utilizzata è la *Categorical crossentropy*:

$$H(y, \hat{y}) = -\sum_{i=1}^{N} y_i * log(\hat{y}_i)$$

Dove:

- y è il vettore delle etichette reali
- $\hat{y}$ è il vettore delle probabilità predette dal modello
- N è il numero delle classi

L'addestramento è stato effettuato per 100 *epoch* (numero di volte in cui l'intero dataset viene presentato al modello) con un batch di dimensione 16. Per evitare che il modello vada in *overfitting* è stata utilizzata la tecnica dell' *early stopping*. Questa consiste nel fermare l'addestramento non appena la funzione di costo comincia ad aumentare e nel recuperare i parametri della rete che hanno minimizzato la loss.

3.2.2 AdaBoost e Random Forest

I modelli basati su Adaboost e Random Forest sono stati addestrati utilizzando rispettivamente gli algoritmi descritti nei capitoli 1.1.3 e 1.1.2. I modelli

di entrambi gli algoritmi sono stati implementati e addestrati utilizzandone le implementazione presenti su *scikit-learn*

3.2.3 Selezione dei modelli

Nei tre algoritmi utilizzati alcuni parametri dei modelli vengono inizializzati casualmente. Nella rete neurale pesi e biases vengono inizializzati casualmente per poi essere modificati durante l'esecuzione di SGD. In Random Forest vengono selezionate casualmente le feature e i valori soglia utilizzati dai nodi degli alberi della foresta e in AdaBoost il modello debole determinato durante la fase di hyperparameter tuning, ossia SGDClassifier, viene inizializzato con dei pesi casuali similmente alla rete neurale.
I valori casuali che assumono i parametri possono avere un grande impatto sulle prestazioni dei modelli addestrati. Di conseguenza, per ottenere prestazioni più elevate è necessario inizializzare e allenare più modelli per poi selezionare i migliori.
Per ognuno degli algoritmi sono stati addestrati *50 modelli*, ognuno dei quali è stato poi testato sul *validation set*. Infine, per ogni algoritmo è stato selezionato il modello che massimizzava la *Macro F1Score*, ossia la media aritmetica della F1Score di ogni classe.
Nella seguente tabella sono riportate le metriche dei tre modelli con i parametri ottimali che sono stati selezionati con la procedura descritta. Tutte le metriche sono state calcolate utilizzando il validation set. F1Score, precision e recall sono state calcolate in modalitá macro.

Tabella 3.3: Metriche dei modelli selezionati con i migliori hyperparameter

Modello	*Accuracy*	*F1Score*	*Precision*	*Recall*
Rete neurale	0.64	0.45	0.47	0.49
Adaboost	0.74	0.41	0.54	0.40
Randomforest	0.75	0.41	0.74	0.40

I risultati migliori per ogni metrica sono evidenziati in verde

Come è possibile osservare la *F1Score* dei tre modelli è molto bassa nonostante siano stati selezionati gli hyperparameter migliori. Si può notare una grande differenza tra i valori di F1Score e accuracy, questa è dovuta al marcato sbilanciamento di classe del validation set. La accuracy è una metrica molto sensibile al bilanciamento delle classi del dataset su cui viene calcolata.

3.3 Data augmentation

Come è possibile osservare nella tabella il training set utilizzato per allenare i modelli è estremamente sbilanciato oltre a essere di dimensioni contenute per un task di ML.

Tabella 3.4: Distribuzione del train set

Numero pazienti	Operazione effettuata	Percentuale
599	Operazione apparato digerente	72%
147	Operazione sistema endocrino	18%
80	Operazione sistema cardiovascolare	10%

È lecito aspettarsi che i modelli allenati non potranno avere prestazioni particolarmente elevate.
In questo capitolo si procederà quindi ad aumentare il train set generando nuovi dati sintetici tramite le tecniche descritte nei capitoli 1.2.1 e 1.2.2.

3.3.1 Generazione con Smotenc

Visto che le feature del dataset sono sia categoriche che numeriche non sarebbe in principio possibile adoperare Smote per la generazione di nuovi dati sintetici. Questo problema è stato risolto utilizzando *SMOTENC (Smote nominal-continuous)*, una variante di Smote con cui è possibile generare dati

in cui sono presenti sia feature categoriche che numeriche. È stata in particolare utilizzata l'implementazione dell'algoritmo presente all'interno della libreria Scikit-learn.
Il processo di data augmentation è stato diviso in due fasi. Una fase di *oversampling* del dataset e di *undersampling*. La fase di oversampling costituisce la fase di sintesi di nuovi dati nel processo di data augmentation, per ogni classe vengono generate un numero arbitrario di nuove istanze. La fase di undersampling consiste invece in una eliminazione casuale di un numero arbitrario di istanze per ogni classe. Effettuare prima oversampling e successivamente undersampling permette di generare un dataset di dimensione fissata preservando il massimo numero possibile di istanze reali.
Nella fase di oversampling sono state generate *755* istanze della classe *operazione apparato digerente* e *300* istanze della classe *operazione sistema endocrino* e *operazione sistema cardiovascolare*, in tutti e tre i casi utilizzando un k pari a *9* . Il dataset creato conterrebbe ora *1355* istanze ma risulterebbe ancora molto sbilanciato in favore della classe operazione all'apparato digerente. Si è proceduto allora a un undersampling casuale differenziato per ogni classe, questo porta alla distribuzione di classi riportata in tabella 3.5

Tabella 3.5: Distribuzione del train set sintetico

Classe	Numero istanze
Operazione apparato digerente	300
Operazione sistema endocrino	300
Operazione sistema cardiovascolare	300

3.3.2 Selezione del dataset migliore

Durante sia la fase di oversampling che quella di undersampling, si introduce una componente casuale nella generazione del dataset. Ciò implica che ogni esecuzione della procedura descritta produce un dataset diverso. Per cercare di generare il miglior dataset possibile si è allora proceduto ad allenare

più modelli di adaboost per ogni dataset, per ogni modello si è calcolata la *F1Score macro* sul validation set e infine per ogni dataset si è calcolata la media delle F1Score. Infine si é selezionato il dataset a cui è associata la media maggiore. Nel dettaglio la procedura di selezione é la seguente:

Algoritmo 4 Selezione del dataset migliore generato con Smotenc

1: **for** $k = 1$ 10 **do**
2: Genera un dataset d_k
3: **for** $i = 1$ 10 **do**
4: Allena un modello di AdaBoost su d_k
5: Calcola e salva la F1Score F_i del modello sul validation set
6: Calcola la media degli F1Score del modello AVG_k
7: Restituisci il dataset che massimizza AVG_k

Nel seguente istogramma è riportata la media dei 10 modelli di adaboost addestrati per ogni dataset generato con smotenc

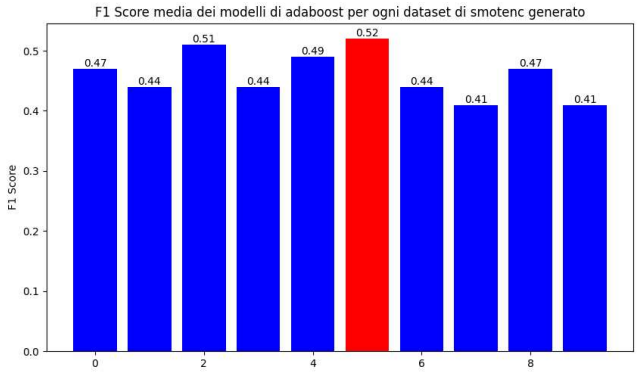


Figura 3.1: F1Score dei modelli addestrati su dataset aumentati con Smotenc

Il dataset selezionato, ossia quello per cui la media della F1Score dei 10 modelli di Adaboost addestrati è maggiore presenta una *F1Score pari a 0.52*. Questo può essere interpretato come indice di un impatto positivo della

procedura di data augmentation effettuata sulle prestazioni dei modelli. La F1Score del miglior modello di Adaboost con gli hyperparameter migliori addestrato sul dataset di training originale era pari *0.41*, come è possibile osservare dalla tabella 3.3.

3.3.3 Generazione con Synthcity

Synthcity offre un ampia gamma di modelli generativi per dati tabulari, questi vanno allenati su un train set di riferimento per poi poter essere utilizzati per generare nuovi dati sintetici. Per aumentare il dataset in questione sono stati scelti *Tvae* [7] e *Nflow* [8].

- *Tvae* (Triplet-Based Variational Autoencoder) : é una architettura di deep learning che adopera tecniche di deep metric learning per generare dati sintetici tabulari in cui coesistono feature numeriche e feature categoriche. Una volta fornito un dataset e indicata la tipologia di ogni feature è possibile addestrare un modello di Tvae sul dataset e generare nuovi dati sintetici

- *Nflow (Neural spline flow)* : è una architettura basata su normalizing flow. Similmente a Tvae permette di generare sia dati numerici che categorici.
  I modelli di entrambi gli algoritmi sono stati addestrati utilizzando il miglior dataset ottenuto durante il processo di data augmentation con Smotenc. È stato scelto di utilizzare un dataset di training già aumentato e sopratutto già *bilanciato* per cercare di massimizzare il bilanciamento delle classi delle istanze sintetiche generate. Sia i modelli di Tvae che di Nflow sono stati addestrati per *100 epoch*, utilizzando un *batch size pari a 32*. In entrambi i casi sono stati utilizzati gli hyperparameter di default delle implementazioni presenti in Synthcity:

Tabella 3.6: hyperparameter modello di Tvae

Parametro	*Descrizione*	*Valori*
n_iter	Numero di iterazioni	1000
n_units_embedding	Numero di unità embedding	500
lr	Learning rate	0.001
weight_decay	Decadimento del peso	1e-05
batch_size	Dimensione del batch	200
random_state	Stato random	0
decoder_n_layers_hidden	Numero di strati nascosti nel decoder	3
decoder_n_units_hidden	Numero di unità nascoste nel decoder	500
decoder_nonlin	Funzione non lineare decoder	leaky_relu
decoder_dropout	Dropout decoder	0
encoder_n_layers_hidden	Numero di strati nascosti nell'encoder	3
encoder_n_units_hidden	Numero di unitàâ nascoste nell'encoder	500
encoder_nonlin	Funzione non lineare dell'encoder	leaky_relu
encoder_dropout	Dropout dell'encoder	0.1
loss_factor	Fattore di perdita	1
data_encoder_max_clusters	Numero massimo di cluster nel data encoder	10
dataloader_sampler	Campionatore dataloader	Nessuno
clipping_value	Valore di clipping	1
n_iter_print	Iterazioni per stampa	50
n_iter_min	Numero minimo di iterazioni	100
patience	Pazienza	5
sampling_patience	Pazienza campionamento	500

Tabella 3.7: hyperparameter modello di Nflow

Parametro	*Descrizione*	*Valori*
n_iter	Numero di iterazioni	1000
n_units_embedding	Numero di unità nell'embedding	500
lr	Learning rate	0.001
weight_decay	Decadimento del peso	1e-05
batch_size	Dimensione del batch	200
random_state	Stato random	0
decoder_n_layers_hidden	Numero di strati nascosti nel decoder	3
decoder_n_units_hidden	Numero di unità nascoste nel decoder	500
decoder_nonlin	Funzione non lineare del decoder	leaky_relu
decoder_dropout	Dropout del decoder	0
encoder_n_layers_hidden	Numero di strati nascosti nell'encoder	3
encoder_n_units_hidden	Numero di unità nascoste nell'encoder	500
encoder_nonlin	Funzione non lineare dell'encoder	leaky_relu
encoder_dropout	Dropout dell'encoder	0.1
loss_factor	Fattore di perdita	1
data_encoder_max_clusters	Numero massimo di cluster nel data encoder	10
dataloader_sampler	Campionatore del dataloader	Nessuno
clipping_value	Valore di clipping	1
n_iter_print	Iterazioni per stampa	50
n_iter_min	Numero minimo di iterazioni	100

Per ogni algoritmo sono stati addestrati due modelli, ognuno è stato poi utilizzato per generare 5 dataset composti da 1000 istanze. Per ogni modello è stato poi selezionato il miglior dataset generato. La procedura di generazione e selezione dei dataset è descritta in dettaglio dal seguente algoritmo:

Algoritmo 5 Selezione del dataset migliore generato con Synthcity

for algorithm in [Tvae, Nflow] **do**
    for $j = 1\ 2$ **do**
        allena un modello M_j
        for $k = 1\ 5$ **do**
            Genera un dataset d_k
            for $i = 1\ 10$ **do**
                Allena un modello di AdaBoost su d_k
                Calcola la F1Score F_i del modello Adaboost sul validation set
            Calcola la media degli F_i del modello Adaboost AVG_{jk}
    Restituisci il dataset che massimizza AVG_{jk}

I due istogrammi seguenti riportano la media delle F1Score macro dei modelli di Adaboost calcolati sui dataset generati con Nflow e Tvae

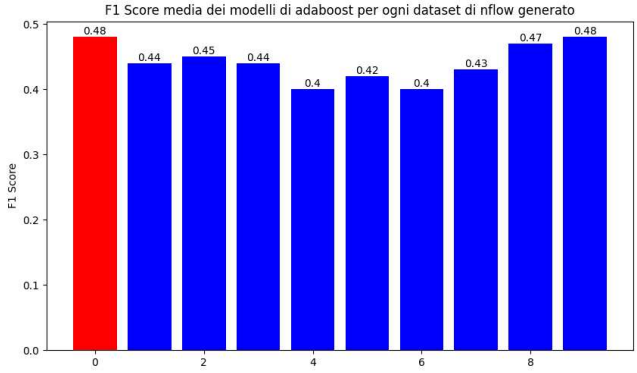


Figura 3.2: Istogramma della media delle F1Score per modelli addestrati su dataset aumentati con Nflow

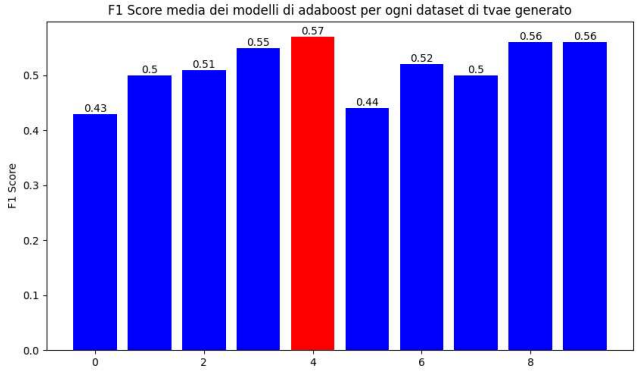


Figura 3.3: Istogramma della media delle F1Score per modelli addestrati su dataset aumentati con Tvae

Come è possibile osservare il la media delle F1score dei modelli di Adaboost addestrati con il miglior dataset aumentato con Nflow è pari a *0.48* mentre il miglior dataset aumentato con Tvae ha addirittura permesso di ottenere una F1Score pari a *0.57*.

Tabella 3.8: Distribuzione dei dataset generati con Synthcity

Dataset aumentato con Nflow		
Numero pazienti	Operazione effettuata	Percentuale
344	Operazione apparato digerente	34%
296	Operazione sistema endocrino	30%
360	Operazione sistema cardiovascolare	36%
Dataset aumentato con Tvae		
Numero pazienti	Operazione effettuata	Percentuale
341	Operazione apparato digerente	34%
306	Operazione sistema endocrino	31%
353	Operazione sistema cardiovascolare	35%

É inoltre possibile osservare dalla tabella 3.8 che i due dataset risultato sostanzialmente bilanciati.

Capitolo 4

Risultati

In questo capitolo vengono presentate e comparate le performance ottenute dai modelli addestrati sul dataset originale e sui dataset aumentati tramite le tecniche discusse nei capitoli precedenti. Per ogni algoritmo utilizzato sono riportate le metriche menzionate nel capitolo 2.4 calcolate globalmente sul *test set*.
La metrica che verrà presa maggiormente in considerazione per confrontare le performance dei modelli è la *F1Score macro*, ossia la media *non pesata* delle F1Score calcolate per ogni classe.

4.1 Modelli addestrati sul dataset non aumentato

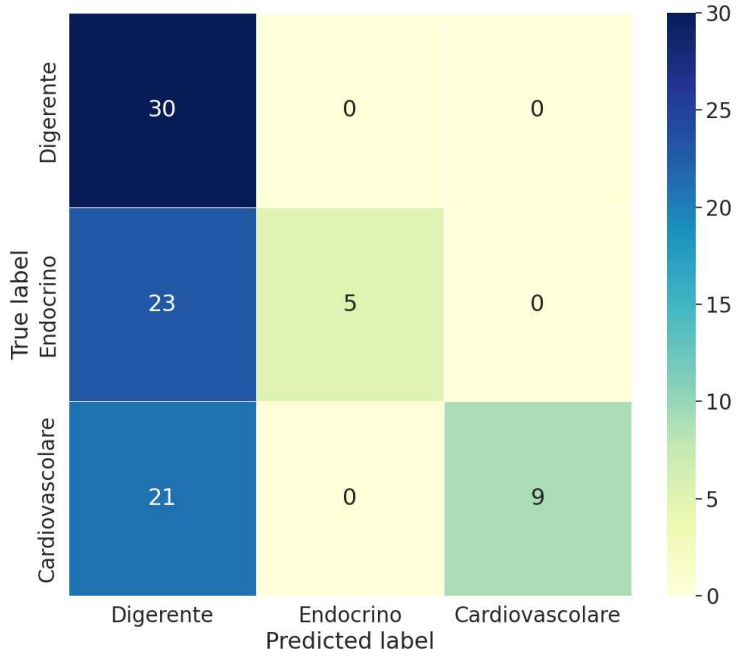
Nella seguente tabella sono riportate le metriche dei modelli migliori addestrate sul dataset di training non aumentato. Tutte le metriche sono state calcolate sul dataset di test. F1Score, precision e recall sono state riportate in forma macro, ossia le metriche sono state prima calcolate per ogni singola classe e si è poi riportata la loro media non pesata.

Tabella 4.1: Metriche dei modelli addestrate sul dataset non aumentato

Modello	*Accuracy*	*F1Score*	*Precision*	*Recall*
Rete neurale	0.43	0.40	0.50	0.43
Adaboost	0.50	0.45	0.80	0.49
Randomforest	0.48	0.42	0.79	0.47

I risultati migliori per ogni metrica sono evidenziati in verde

Dalla tabella emerge chiaramente che nessuno dei modelli ha ottenuto prestazioni soddisfacenti. Adaboost è risultato essere il modello migliore secondo ogni metrica. Di seguito é riportata la matrice di confusione:



3

Figura 4.1: Matrice di confusione di Adaboost

Come è possibile osservare dalla matrice di confusione il modello ha una

forte tendenza a classificare ogni istanza come appartente alla classe delle operazioni all' apparato digerente. Tutte le instanze effettivamente appartenenti a questa classe sono state classificate correttamente, alcontempo peró sono state classificate come operazioni all' apparato digerente la maggiorparte delle operazioni al sistema cardiovascolare e al sistema endocrino.
Si può constatare dalle matrici di confusione dei modelli di Randomforest e rete neurale :

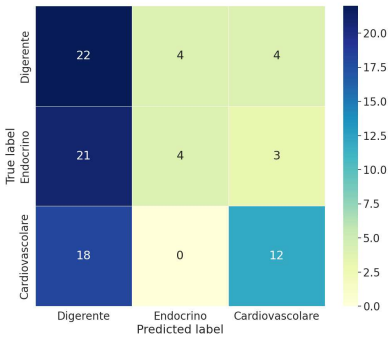


Figura 4.2: Matrice di confusione ottenuta addestrando la rete neurale sul dataset originale

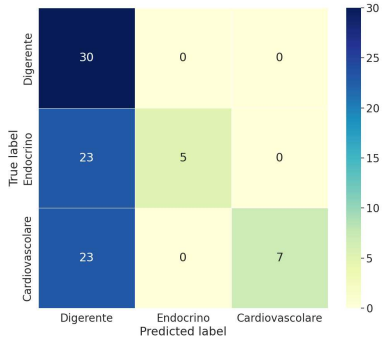


Figura 4.3: Matrice di confusione ottenuta addestrando Randomforest sul dataset originale

che questa tendenza non riguarda solo il modello di Adaboost . Questo comportamento è molto probabilmente causato dal marcato sbilanciamento del training set in favore della classe apparato digerente che come riportato in tabella 2.3 costituisce 599 delle 826 istanze totali.

4.2 Modelli addestrati sui dataset aumentati

Di seguito, per ogni dataset aumentato generato nel capitolo 1.2.2 sono riportate le performance ottenute dai tre modelli sul test set. Come

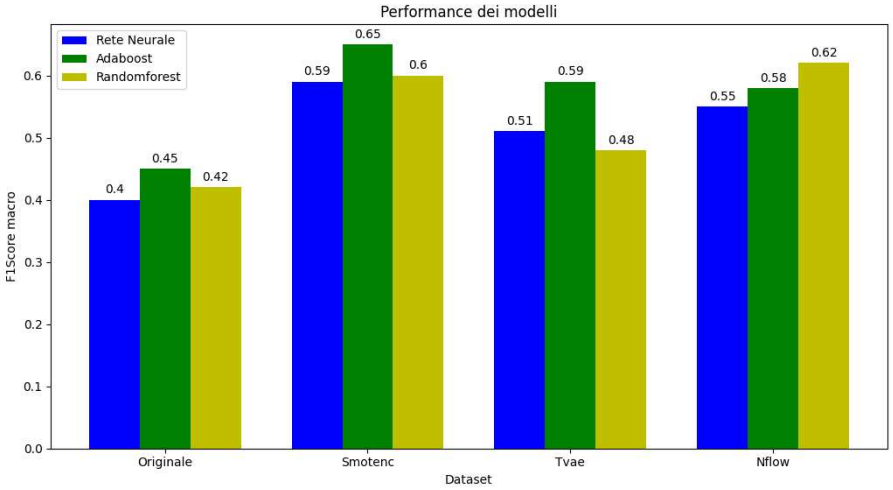
per la tabella 4.2 sono state riportate tutte le metriche in forma macro:

Tabella 4.2: Metriche dei modelli addestrate sui dataset aumentati con Nflow, Smotenc e Tvae

Aumento	*Modello*	*Accuracy*	*F1Score*	*Precision*	*Recall*
Nflow	Rete neurale	0.57	0.55	0.57	0.56
	Adaboost	0.60	0.58	0.58	0.60
	Randomforest	0.62	0.62	0.62	0.62
Smotenc	Rete neurale	0.64	0.59	0.67	0.64
	Adaboost	0.65	0.65	0.66	0.65
	Randomforest	0.60	0.60	0.73	0.60
Tvae	Rete neurale	0.53	0.51	0.52	0.53
	Adaboost	0.60	0.59	0.58	0.60
	Randomforest	0.50	0.48	0.48	0.50

I risultati migliori per ogni metrica sono evidenziati in verde

Come è possibile osservare dalle tabella, per ogni dataset generato la F1Score macro di ognuno dei modelli ha subito un incremento rispettivamente alla stessa calcolata dai risultati prodotti dal modello addestrato sul dataset non aumentato.
Nel seguente istogramma viene rappresentata la F1Score macro di ogni modello per ogni dataset generato:



3

Figura 4.4: Istogramma prestazione dei modelli

Dall'istogramma emerge un chiaro aumento di prestazioni nei modelli addestrati su dataset aumentati. L'aumento maggiore per Adaboost e la rete neurale è stato ottenuto con l'utilizzo di Smotenc, mentre per quanto riguarda Randomforest è stato raggiunto utilizzando Nflow. Il modello con la F1Score maggiore risulta essere Adaboost addestrato sul dataset aumentato con Smotenc. Il modello in questione ha ottenuto una *F1Score pari a 0.65* che costituisce un *aumento del 44% delle prestazioni* rispetto al modello di Adaboost addestrato sul dataset non aumentato.

In seguito è riportata la matrice di confusione del modello in questione addestrato sul dataset originale e sul dataset aumentato con Smotenc:

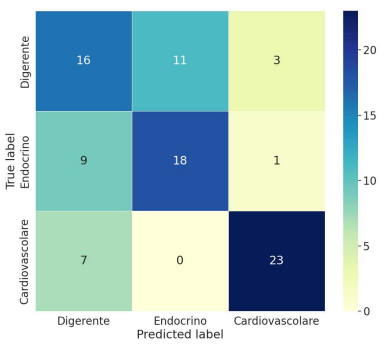


Figura 4.5: Matrice di confusione ottenuta addestrando Adaboost sul miglior dataset aumentato con Smotenc

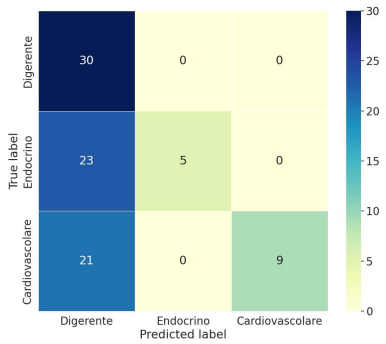


Figura 4.6: Matrice di confusione ottenuta addestrando Adaboost sul dataset originale

Confrontando la nuova matrice di confusione con quella ottenuta dal modello addestrato sul dataset non aumentato risulta un evidente aumento della capacità di classificazione del modello. Questo aumento delle capacità di classificazioni è anche riscontrabile dal confronto tra le metriche del miglior modello addestrato sul dataset non aumentato e quelle del miglior modello addestrato sul dataset aumentato con Smotenc.

Tabella 4.3: Confronto metriche dei modelli di Adaboost

Dataset	*Accuracy*	*F1Score*	*Precision*	*Recall*
Non aumentato	0.50	0.45	0.80	0.49
Smotenc	0.60	0.65	0.66	0.65

I miglioramenti sono evidenziati in verde, in rosso i peggioramenti

La tendenza a classificare ogni istanza come appartanente alla classe delle operazioni all'apparto digerente, riscontrata nel modello di Adaboost addestrato sul set non aumentato, risulta fortemente ridotta.

Questo risultato, riscontrabile anche nei modelli migliori di rete neurale e Randomforest addestrati sul dataset aumentato con Smotenc ,è probabilmente dovuto al ribilanciamento del dataset effettuato tramite le nuove istanze sintetiche generate.

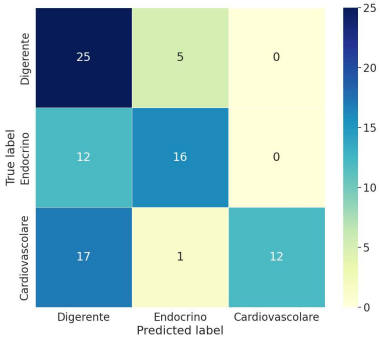


Figura 4.7: Matrice di confusione ottenuta addestrando il modello di Randomforest sul miglior dataset aumentato con Smotenc

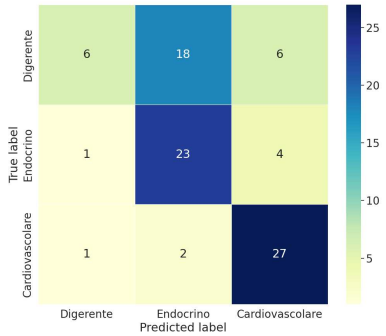


Figura 4.8: Matrice di confusione ottenuta addestrando il modello di rete neurale sul miglior dataset aumentato con Smotenc

Dall'analisi delle matrici di confusione dei tre modelli appare però evidente una ridotta capacità di differenziazione tra la classe delle operazioni all'apparato digerente e la classe delle operazioni al sistema endocrino, questo suggerisce una forte sovrappobilitá delle caratteristiche delle due classi ai fini della classificazione.

Capitolo 5

Conclusioni

Visti i risultati ottenuti l'obiettivo prefissato per questa tesi si può considerare raggiunto. Grazie alle tecniche di data augmentation adoperate é stato possibile *aumentare del 44%* la *F1Score* del miglior modello di predizione addestrato sul dataset originale. Il miglioramento delle prestazioni ottenuto suggerisce che il processo di data augmentation costituisca un valido strumento per far fronte alla carenza di dati tabulari come quelli oggetto del lavoro di questa tesi, molto utilizzati per applicazioni di ML in ambito medico . Un fatto notevole che emerge dall'analisi dei risultati ottenuti è come nella maggior parte dei casi le prestazioni ottenute dai modelli addestrati su dataset aumentati con Smotenc siano state sensibilmente maggiori da quelle ottenute da modelli addestrati su dataset aumentati con i due algoritmi di deep learning generativi utilizzati. Questo fatto, che trova riscontro in letteratura almeno per quanto riguarda la classificazione binaria per dati tabulari [9], suggerisce la possibilità di proseguire il lavoro di tesi fin qui svolto andando a esplorare ulteriori approcci generativi per la data augmentation e verificando se attraverso questi sia possibile ottenere risultati migliori rispetto a quelli ottenibili con algoritmi di data augmentation non basati su architetture di deep learning generative come Smotenc.

Bibliografia

[1] L Breiman. Random forests. 2001.

[2] Robert E. Schapire Yoav Freund. A short introduction to boosting. *Proceedings of the 16th international joint conference on Artificial intelligence*, 2July 1999:1401â1406, 1999.

[3] L. O. Hall W. P. Kegelmeyer N. V. Chawla, K. W. Bowyer. Smote: Synthetic minority over-sampling technique. 2002.

[4] Max Welling Diederik P Kingma. Auto-encoding variational bayes. 2013.

[5] Shakir Mohamed Danilo Jimenez Rezende. Variational inference with normalizing flows. 2015.

[6] Zhaozhi Qian, Bogdan-Constantin Cebere, and Mihaela van der Schaar. Synthcity: facilitating innovative use cases of synthetic data in different data modalities. 2023.

[7] Haleh Akrami, Sergul Aydore, Richrd Leahy, and Anand Joshi. Robust variational autoencoder for tabular data with beta divergence. 2020.

[8] Iain Murray George Papamakarios Conor Durkan, Artur Bekasov. Neural spline flows. 2019.

[9] Sergul Aydore Dionysis Manousakas. On the usefulness of synthetic tabular data generation. 2023.

[10] Raghuram Iyengar Raina M Merchant David A Asch Meghana Sharma Carolyn C Cannuscio David Grande, Nandita Mitra. Consumer willingness to share personal digital information for health-related uses. 2022.