Alma Mater Studiorum · Università di Bologna

SCUOLA DI SCIENZE
Corso di Laurea in Ingegneria e Scienze Informatiche

Studio e Sperimentazione di Tecnologie di Machine Learning in Ambito Genomico ed Healthcare

Relatore:
Prof. Gianluca Moro

Presentata da:
Francesco Montelli

Correlatore:
Prof. Roberto Pasolini

Sessione: I
Anno Accademico: 2018/2019

Parole Chiave

- machine learning
- python
- genomica
- gene ontology
- microbiota
- patologie

Indice

Capitolo 1

Introduzione

Le applicazioni di intelligenza artificiale stanno rivoluzionando la nostra società, rendendo quasi ordinari compiti che fino a qualche anno fa erano considerati inaffrontabili [28]. Casi di successo eclatanti sono all'ordine del giorno nei più disparati settori disciplinari che vanno dall'astronomia, ai sistemi produttivi e logistici, all'arte, fino alla medicina.

Questo sviluppo è anche legato alla contemporanea diffusione di sistemi di calcolo sempre più potenti, a basso costo e miniaturizzati che incrementano anche la raccolta capillare di grandi masse di dati, come ad esempio le tecnologie Internet of Anything e wearable[1] [18], che stanno aprendo nuove possibilità in ambito medicale, dalla diagnosi [26], allo sviluppo di nuovi farmaci [19] fino allo sviluppo di terapie personalizzate [4]. In particolare in ambito healthcare, negli ultimi anni, si stanno moltiplicando gli studi[2] riguardanti il microbiota (o microbioma) intestinale, un aggregato di microorganismi dal peso complessivo di circa 2Kg [3] che vivono in simbiosi con il tratto intestinale. Secondo recenti studi il microbiota ci caratterizza in modo similare al genoma [34], ed è ritenuto così importante da essere ormai considerato un nuovo organo [5]. Alcuni nuovi studi su riviste scientifiche prestigiose, tra cui diverse del gruppo Nature come, ad esempio, Nature Communication, hanno evidenziato correlazioni significative tra composizione del microbiota e numerose gravi patologie come malattie autoimmuni degenerative [32], cardiovascolari [22], tumori del tratto intestinale [25], Parkinson [21], ed alcune forme di depressione [33]. Altri studi sono riusciti ad isolare componenti del microbiota che caratterizzano l'etnia di origine della persona [8].

Altri settori in cui applicazioni intelligenti sono utilizzate con sempre maggior successo, comprendono la genomica[4], un settore dove uno dei problemi è la grandissima

[1] dispositivi e sensori indossabili con capacità computazionale e di comunicazione

[2] Google Trends mostra, negli ultimi 5 anni, un interesse crescente in questo ambito. Da Luglio 2014 si è passati da un fattore di interesse di 25 ad un fattore di interesse di 89. Il fattore di interesse è un valore compreso tra 0 e 100 che fornisce una indicazione sul volume delle ricerche effettuate.

[3] https://www.gutmicrobiotaforhealth.com/en/about-gut-microbiota-info/

[4] branca della biologia molecolare che si occupa dello studio del genoma

quantità di dati da gestire, rappresentare ed elaborare. In questo settore, in particolare nella genomica funzionale[5], è diventata prassi comune sfruttare algoritmi di machine learning per ricevere suggerimenti per indirizzare la ricerca in modo da ottimizzare tempo e costi. Questo approccio è reso possibile anche dalla nascita di banche dati come Gene Ontology [11] [3] che mettono a disposizione, in modo standardizzato e indipendente dalla specie, informazioni riguardanti geni e le loro funzioni.

Applicare l'intelligenza artificiale a questo tipo di analisi è fondamentale, soprattutto a fronte del crescente volume di dati che si hanno a disposizione, ad esempio, i macchinari di sequenziamento di nuova generazione usati per eseguire la lettura del genoma partendo da un campione possono produrre fino a 120GB di dati per campione[6]. É bene ricordare che comunque queste tecniche non sono utilizzate per sostituire esperti umani ma per potenziare la ricerca e fare avanzare la conoscenza.

Lo scopo di questa tesi è applicare a due casi di studio diversi algoritmi di learning e produrre confronti tra vari modelli evidenziando come i metodi ensemble ottengono risultati in alcuni casi migliori dei lavori in letteratura. I casi di studio presi in esame riguardano sia lo sviluppo di esperimenti per la scoperta di annotazioni Gene Ontology, ossia di nuove funzione biologiche di geni, sia di esperimenti riguardanti l'associazione tra composizione batterica del microbiota ed alcune gravi patologie, come cancro colon-rettale, morbo di Chron, morbo di Parkinson, disturbi dello spettro autistico ed artrite reumatoide.

[5]la genomica funzionale è una branca della genomica che si occupa dello studio delle funzioni biologiche dei geni

[6]https://emea.illumina.com/systems/sequencing-platforms.html

Capitolo 2

Panoramica

2.1 Machine learning e Data Science

L'intelligenza artificiale può essere definita come lo studio di sistemi informatici in grado di portare a termine compiti che normalmente richiederebbero l'uso di una intelligenza di tipo umano.[2]

La data science è un settore multidisciplinare che comprende concetti di statistica, machine learning e conoscenza del dominio in cui si opera con lo scopo di ottenere informazioni dalla grande mole di dati grezzi che si hanno a disposizione.

Il machine learning è una branca dell'intelligenza artificiale che ha lo scopo di realizzare agenti intelligenti in grado di apprendere autonomamente il modo migliore di portare a termine un compito trovando pattern nei dati. Osservando il mondo esterno l'agente è in grado di apprenderne una rappresentazione e sfruttarla per eseguire predizioni e stime su nuovi dati senza la necessità di descrivere in modo formale il suo comportamento.

Nel machine learning esistono diversi modi in cui gli agenti possono imparare:

- *Apprendimento supervisionato*
  Si parla di apprendimento supervisionato nel momento in cui all'algoritmo, insieme ai dati, vengono fornite delle indicazioni esplicite, dette label, sul risultato atteso. Lo scopo sarà di individuare una funzione non nota in partenza che sia in grado di approssimare il risultato minimizzando l'errore compiuto.

- *Apprendimento non supervisionato*
  Nell'apprendimento non supervisionato l'algoritmo non riceve nessuna indicazione sul risultato atteso e deve trovarlo in modo autonomo. Un possibile esempio di applicazione di queste tecniche è la separazione dei dati in gruppi non noti a priori.

- *Apprendimento per rinforzo*
  Il reinforcement learning è una tecnica di machine learning in cui l'algoritmo impara tramite una serie di ricompense che si basano sull'esito delle azioni che compie

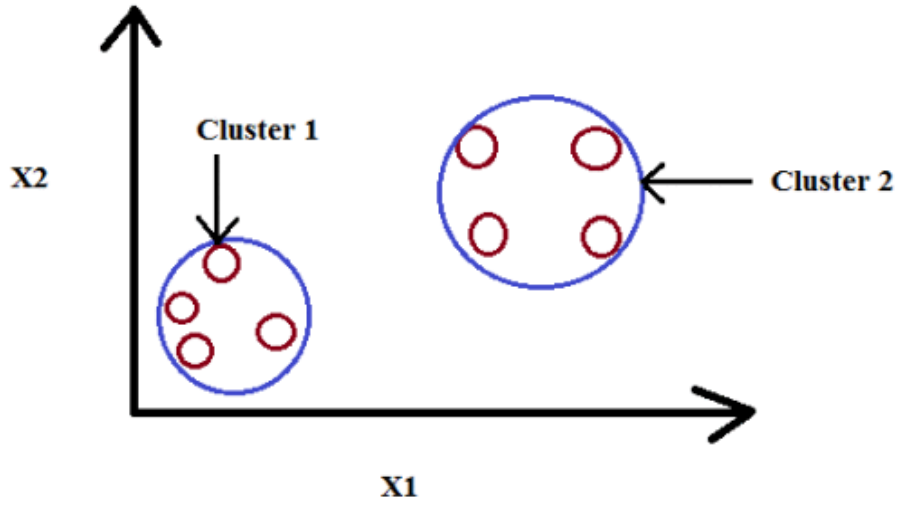


Figura 2.1: Scoperta di raggruppamenti non noti a priori

all'interno dell'ambiente in cui deve operare, ad esempio un simulatore. Recentemente in questo ambito sono emersi risultati eccezionali come AlphaGO, il primo modello in grado di sconfiggere un maestro nel gioco del GO[1]; questo problema era considerato inaffrontabile fino a prima di questo risultato [20].

Negli ultimi tempi la necessità di applicare algoritmi di learning è cresciuta moltissimo grazie alla crescente disponibilità di dati, soprattutto di tipo non strutturato come, ad esempio, testo ed immagini. Da una recente indagine prodotta da LinkedIn [27] emerge che nei soli Stati Uniti sono vacanti circa 150.000 posti di lavoro nel settore della data science.

2.2 Workflow di un progetto data science

Il processo di sviluppo di un progetto di data science è un ciclo di diverse fasi:

1. *Comprensione del problema e del contesto*
   Come in ogni progetto occorre definire in modo chiaro lo scopo dell'analisi e il problema che si prefigge di affrontare. All'interno di questa fase è importante prendere contatto con il dominio applicativo dato e con il committente in modo da capire il suo scopo. In questo contesto deve anche essere svolto uno studio di fattibilità al fine di capire se il problema che ci si è prefissati è effettivamente risolvibile.

[1] https://it.wikipedia.org/wiki/Go_(gioco)

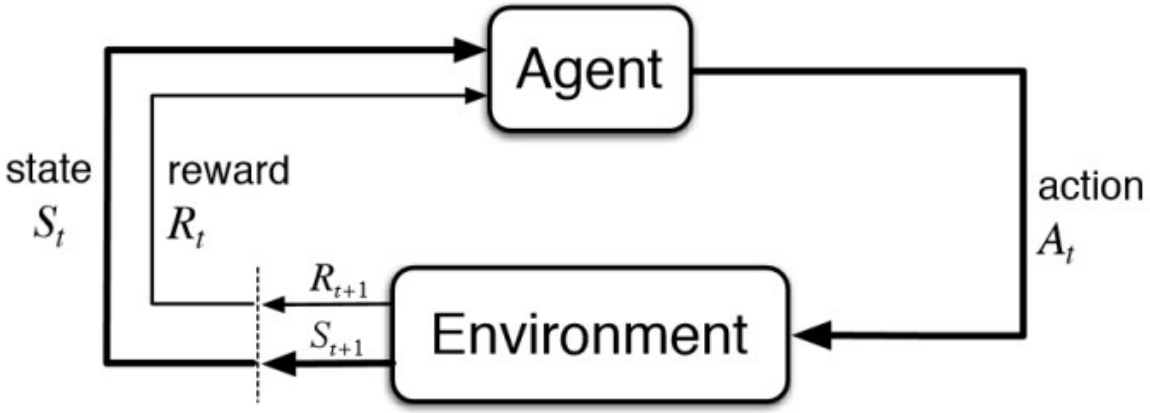


Figura 2.2: Visualizzazione del processo di reinforcement learning

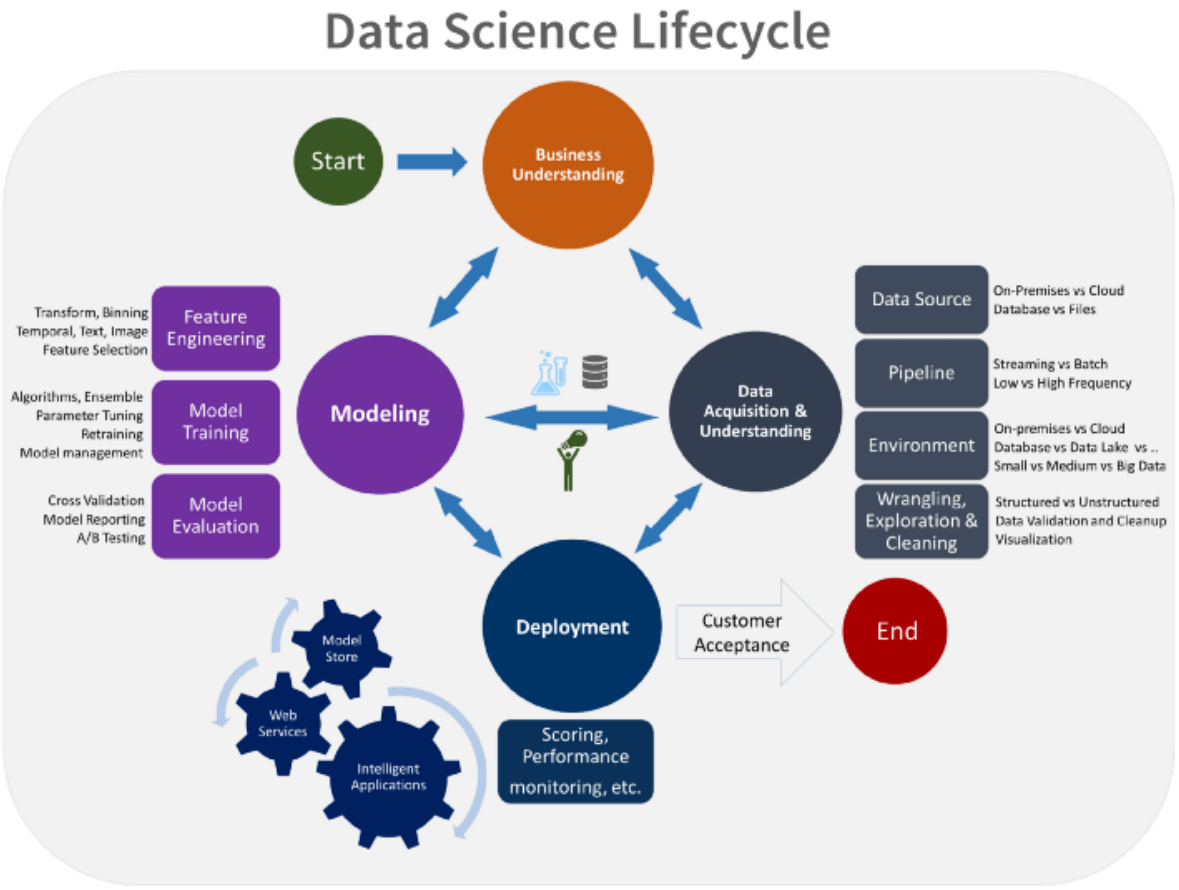


Figura 2.3: Esempio di lifecycle di un progetto di data science [29]

2. *Collezione dei dati*
   All'interno di questa fase vengono recuperati i dati provenienti, eventualmente, da sorgenti eterogenee tenendo conto dello scopo che si vuole raggiungere.

3. *Preparazione dei dati*
   Difficilmente i dati sono usabili nella forma in cui sono stati raccolti. Lo scopo di questa fase è ottenere una rappresentazione uniforme ed usabile all'interno del progetto. É importante investire tempo e risorse in questa fase in modo da risparmiarne nelle fasi successive. I dati vengono poi suddivisi in due gruppi: *training set* e *test set*, usati rispettivamente per addestrare e valutare il modello su dati mai visti prima.

4. *Analisi dei dati*
   Spesso eseguita di pari passo alla precedente; in questo contesto vengono compiute analisi preliminari alla modellazione. Uno degli scopi principali è identificare dati mancanti o errati e trovare correlazioni tra essi. Dato che i risultati devono essere comunicati anche a persone con un background non tecnico è importante realizzare visualizzazioni che permettano di capire a colpo d'occhio quanto scoperto.

5. *Modellazione*
   Dopo avere raccolto, pulito e analizzato i dati è possibile procedere con la realizzazione di modelli predittivi. Dato che non è possibile sapere a priori quale algoritmo renderà meglio degli altri è necessario eseguire diverse prove per potere determinare il migliore.

6. *Ricerca degli iperparametri*
   Gli iperparametri sono dei parametri che regolano gli algoritmi; devono essere indicati dallo sviluppatore dal momento che non vengono appresi durante la fase di learning. In questa fase si usano approcci come la GridSearch[2] e la RandomSearch[3] per automatizzare il processo di ricerca degli iperparametri migliori.

7. *Verifica e comunicazione dei risultati*
   Dopo avere ottenuto delle predizioni è fondamentale verificare la loro bontà confrontando delle predizioni eseguite su dati inediti per il modello e confrontarli con la realtà; se possibile è anche opportuno in base a quali criteri il modello di conoscenza ottenuto stia operando. Al termine delle verifiche è possibile procedere con la redazione di reportistica che includa anche i risultati ottenuti

Le fasi presentate, come è possibile vedere nella figura 2.3, possono anche essere seguite in modo non lineare tra le varie iterazioni dato che mano a mano che si procede con l'analisi possono emergere nuove intuizioni o problematiche non previste inizialmente.

[2]Ricerca Esaustiva dei possibili iperparametri
[3]Ricerca randomica nello spazio dei possibili iperparametri

Capitolo 3

Analisi dei Dati e Modellazione del Problema

Come descritto precedentemente sono stati analizzati due casi di studio:

- analisi su MicrobiomeHD [16]
- analisi su Gene Ontology

3.1 MicrobiomeHD

MicrobiomeHD [16] è un dataset prodotto nel contesto di una ricerca volta ad indagare la presenza di risposte del microbiota intestinale ad uno stato di salute alterato. [15] Questo dataset è stato creato con lo scopo di rendere più facile eseguire analisi comparate tra diversi studi che, a causa di metodologie variegate e mancanza di pipeline di elaborazione standardizzate, sarebbero altrimenti difficili da confrontare in modo diretto.
Ogni dataset si compone di 4 elementi principali:

- *summary_file.txt*: contiene le informazioni sulla configurazione usata per la pipeline di elaborazione
- *RDP/classification_denovo.txt*: contiene la classificazione tassonomica di ogni organismo individuato durante le analisi; ogni organismo viene indicato con un ID univoco.
- *datasetID.otu_seqs.100.fasta*: contiene la OTU[1] table, una tabella che associa ad ogni coppia paziente/organismo un valore che rifletta la presenza di tale organismo nel microbiota del paziente

[1] Operational Taxonomic Unit

	A	B	C	D	E	F
1		sample01	sample01	sample02	sample02	sample03
2	OTU0001	528	68	1755	773	2167
3	OTU0002	138	68	559	2588	1198
4	OTU0003	36	533	673	351	815
5	OTU0004	1	2618	5	17	19
6	OTU0005	224	237	81	271	313

Figura 3.1: Esempio di otu table

Avere la possibilità di lavorare con dei dati preprocessati è di fondamentale importanza. Eseguire le analisi partendo dai dati grezzi derivata dall'analisi dei singoli campioni è una operazione molto complessa che, nonostante la presenza di numerosi tool [7] [17], richiede una conoscenza del dominio elevata ed elevati costi computazionali dato che i dati grezzi possono essere nell'ordine delle centinaia di GByte.

3.1.1 OTU table

Le OTU table sono matrici usate per indicare la quantità di letture diverse presenti all'interno di ogni campione. Con letture si intende una sequenza di nucleotidi con una similarità, generalmente, del 97% [2]. Tuttavia è possibile adoperare raggruppamenti più precisi. In questo caso i ricercatori hanno utilizzato un raggruppamento al 100% di similarità, questo equivale a creare una riga per ogni sequenza univoca di nucleotidi.

3.1.2 Preparazione dei dati

MicrobiomeHD mette a disposizione, per ogni studio, un file compresso in formato *.tar.gz*. All'interno dell'archivio sono organizzati una serie di file che contengono le informazioni di interesse; ogni entità, sia essa un paziente o un organismo, viene identificata in modo univoco da un ID. Per rendere più semplice lavorare con questi dati per ogni dataset è stato prodotto un dataframe pandas in cui le righe vengono usate per rappresentare i pazienti, mentre le colonne rappresentano i vari generi degli organismi trovati e l'informazione sullo stato di salute. All'interno delle celle è possibile leggere il quantitativo di organismi di un certo genere contenuti nel campione del paziente indicato dalla riga. Per

[2]https://amplicon-sequencing-pipeline.readthedocs.io/en/latest/otu-tables.html

quanto riguarda l'informazione sulla salute è stata inserite una 'H' per i pazienti sani ed un codice indicante la patologia per i pazienti non sani.

Un esempio del risultato finale è il seguente:

	Acanthopleuribacter	Acetanaerobacterium	Acetivibrio	Acetobacterium	Acidaminococcus	Acidovorax	Akkermansia	Alkalibaculum	Allisonella	Allobacillus	...	Thermicanus
N01	0	41	3	0	0	0	125	0	0	0	...	0
N02	2	12	0	0	0	0	188	0	0	0	...	0
N03	0	0	4	0	0	0	0	2	0	0	...	0
N04	0	3	11	0	0	0	204	8	0	0	...	2
N05	0	0	8	0	0	0	7	1	0	0	...	0
N06	0	1	4	0	0	0	0	9	0	0	...	0
N07	0	6	0	0	0	0	0	0	0	0	...	0
N08	0	9	22	0	0	0	2	18	0	0	...	0
N09	0	1	0	0	0	0	7	0	0	0	...	0
N10	2	22	4	0	0	0	2	0	1	0	...	0

Per ottenere i dati formattati in questo modo sono state eseguite le seguenti operazioni:

1. lettura dei metadati
2. lettura della otu table
3. lettura della classificazione tassonomica
4. raggruppamento dei dati della otu table in base alla classificazione tassonomica (sommate le letture degli organismi appartenenti allo stesso genere)
5. eseguita la trasposizione
6. aggiunta dei metadati relativi alla salute

3.1.3 Analisi

Nel processo di data science il primo passo da compiere è l'analisi dei dati; in questa fase si inizia a prendere conoscenza dei dati con cui si dovrà lavorare nelle successive fasi.

All'interno del progetto è stata usata la seguente codifica per indicare le patologie in esame:

- ASD: Autistic Spectrum Disorder
- CDI: Clostridium Difficile Infection
- CRC: ColoRectal Cancer
- EDD: Eosinophilic Digestive Disease
- HIV: Human Immunodeficiency Virus

- IBD: Inflammatory Bowel Diseases
- MHE: Minimal Hepatic Encephalopathy
- NASH: NonAlcoholic SteatoHepatitis
- OB: Obesity
- PAR: Parkinson
- RA: Rheumatoid Arthritis
- T1D: Type 1 Diabetes

Indagando la colonna 'DiseaseState', che riporta le informazione sullo stato di salute, è possibile notare che, per alcuni dataset, i pazienti sani sono stati indicati non con la lettera H usata comunemente ma, con la dicitura *non* seguita dalla sigla della patologia. Questa colonna potrebbe presentare anche valori nulli.

Avendo a disposizione queste informazioni si decide di procedere eliminando dal dataset le righe che presentano una mancanza di informazioni relative allo stato di salute e, nel conteggio dei pazienti sani, sommare le persone il cui stato di salute è H e quelle in cui è presente la stringa *non*.

3.1.4 Data augmentation

Per arricchire le analisi effettuabili si è deciso di raggruppare i dataset di riferimento per le varie malattie in modo da potere studiare ogni patologia nel complesso e arricchire i dati sfruttando soggetti provenienti da gruppi eterogenei.

Infine vengono riportati i dati relativi all'unione di tutti i dataset

3.1.5 Analisi "numerica" dei dati

Un'analisi importante da eseguire è quella volta a capire la varianza dei dati, la misura di quanto i dati si discostano dal valore medio; a livello intuitivo è possibile affermare che più la varianza è alta più i dati si discostano dalla media. Avere informazioni su questo aspetto è fondamentale nel caso si usino algoritmi basati sul computo della distanza euclidea, come le SVM, usate in questi esperimenti.

$$\sigma^2 = \frac{\sum_{i=1}^{n} x_i - \bar{x}}{n} \tag{3.1}$$

Questi test sono eseguiti sul dataset ottenuto dall'unione dei vari studi in modo da avere uno sguardo sulla situazione generale ed evitare di ripetere le analisi per tutti gli esperimenti coinvolti.

Tabella 3.1: Composizione dataset singoli

Name	Healty	Not-Healty	Total
asd_kb	20	19	39
asd_son	44	59	103
cdi_schubert	243	93	336
cdi_vincent	25	25	50
cdi_youngster	18	82	100
crc_xiang	22	21	43
crc_zackular	58	30	88
crc_zeller	75	41	116
crc_zhao	56	46	102
edd_singh	82	222	304
hiv_dinh	15	21	36
hiv_lozupone	21	30	51
hiv_noguerajulian	57	293	350
ibd_alm	24	67	91
ibd_engstrand_maxee	46	68	114
ibd_gevers	16	146	162
ibd_huttenhower	27	201	228
mhe_zhang	26	51	77
nash_chan	22	32	54
nash_ob_baker	41	22	63
ob_escobar	10	20	30
ob_goodrich	451	528	979
ob_gordon	61	220	281
ob_jumpertz	35	26	61
ob_ross	26	37	63
ob_wu	335	394	729
ob_zeevi	335	394	729
ob_zupancic	127	128	255
par_scheperjans	74	74	148
ra_littman	28	86	114
t1d_alkanani	55	57	112
t1d_mejialeon	8	21	29

Conteggio dei soggetti sani e malati per ogni dataset.

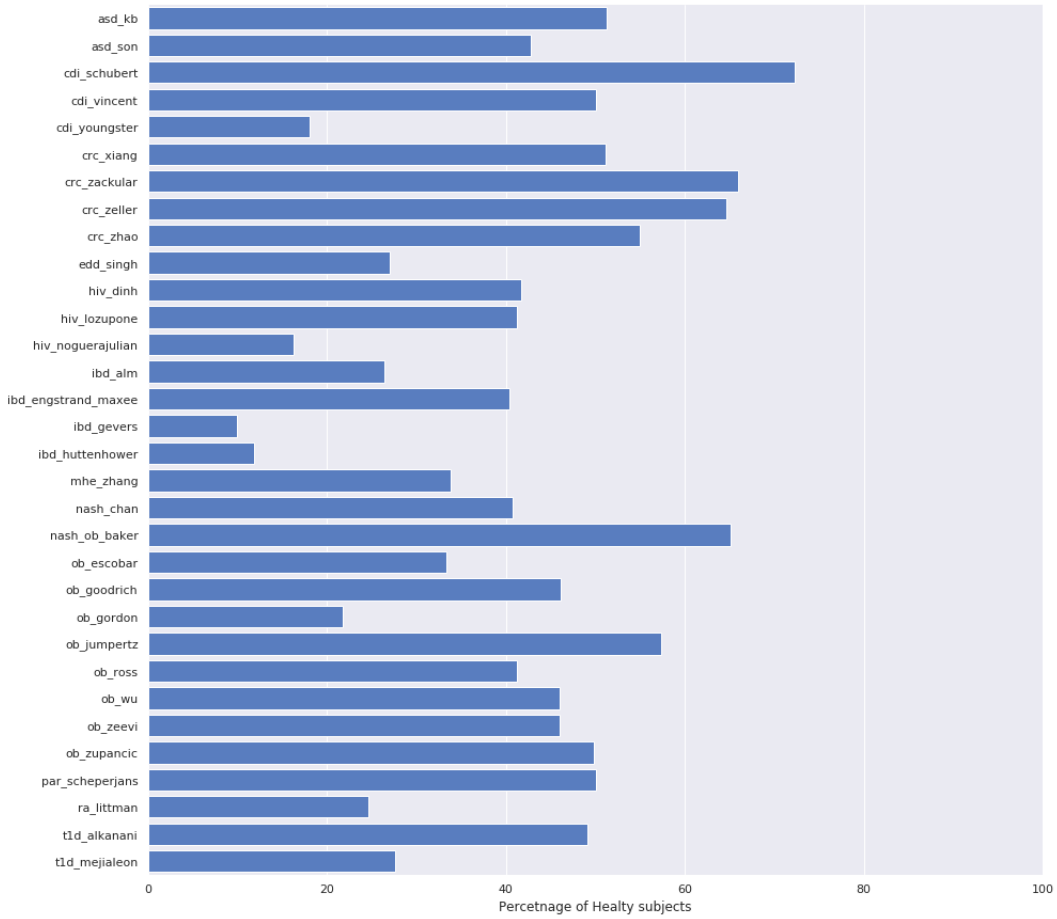


Figura 3.2: Percentuale di soggetti sani all'interno di ogni studio

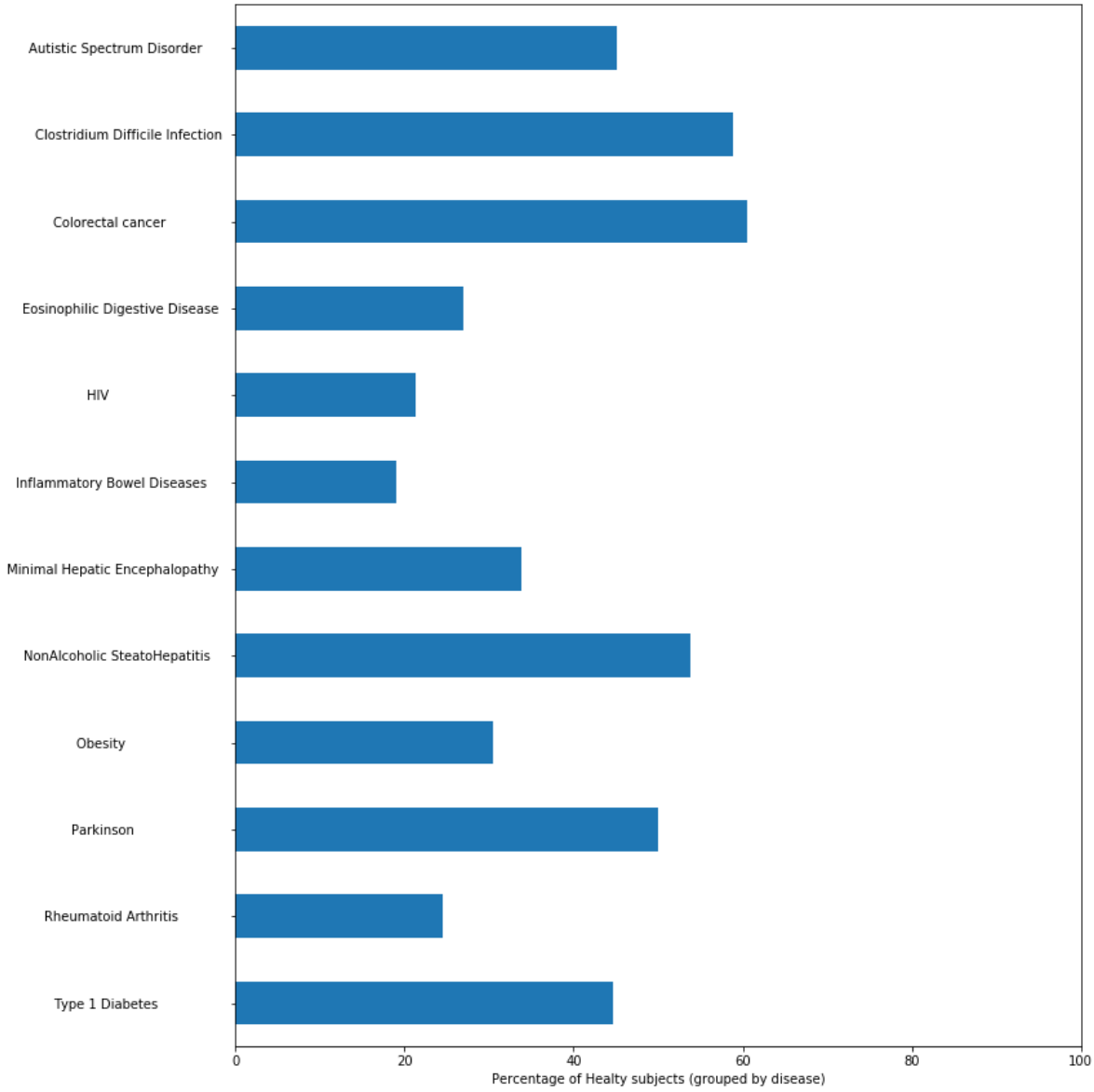


Figura 3.3: Percentuale di soggetti sani raggruppati per patologia

Name	Healty	Not-Healty	Total
Autistic Spectrum Disorder	64	78	142
Clostridium Difficile Infection	286	200	486
Colorectal cancer	211	138	349
Eosinophilic Digestive Disease	82	222	304
HIV	93	344	437
Inflammatory Bowel Diseases	113	482	595
Minimal Hepatic Encephalopathy	26	51	77
NonAlcoholic SteatoHepatitis	63	54	117
Obesity	751	1710	2461
Parkinson	74	74	148
Rheumatoid Arthritis	28	86	114
Type 1 Diabetes	63	78	141

Tabella 3.2: Composizione dataset aggregati

Name	Healty	Not-Healty	Total
All	1796	3489	5285

Tabella 3.3: Composizione dataset aggregati

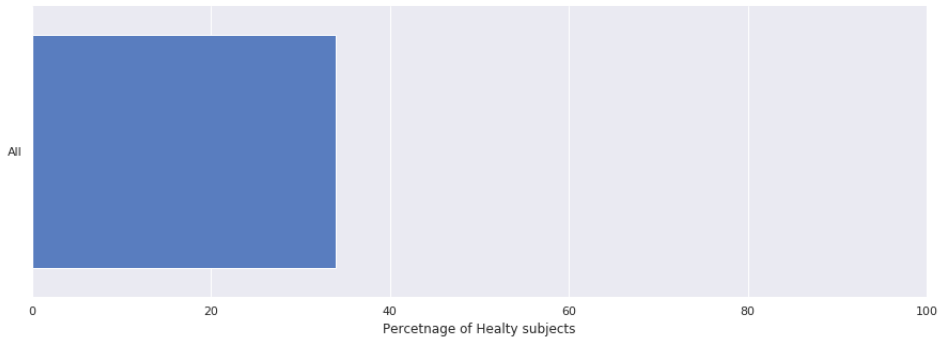


Figura 3.4: Percentuale di soggetti sani globale

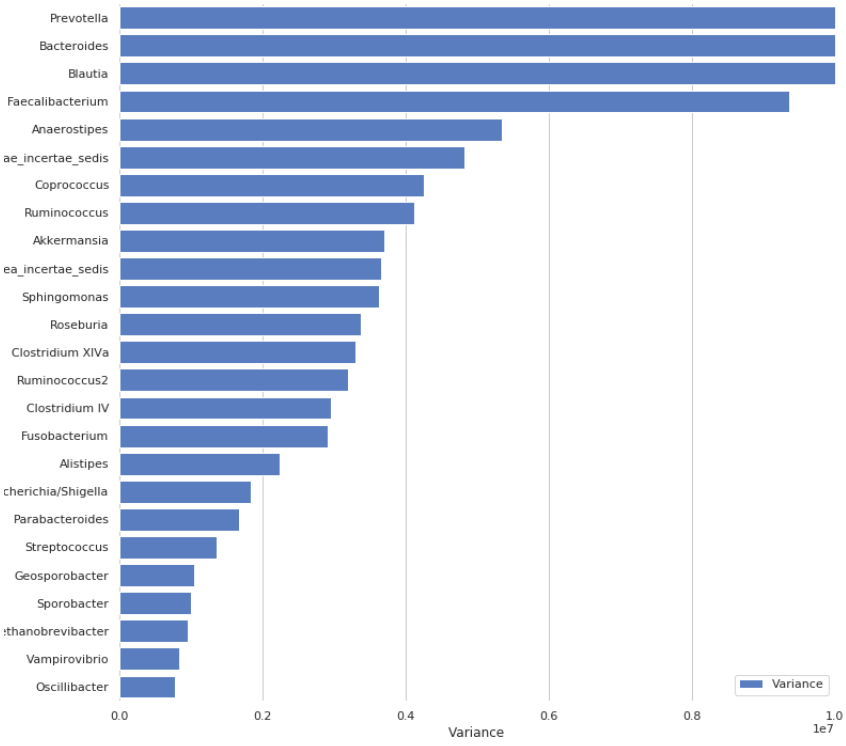


Figura 3.5: Rappresentazione della varianza (top 25)

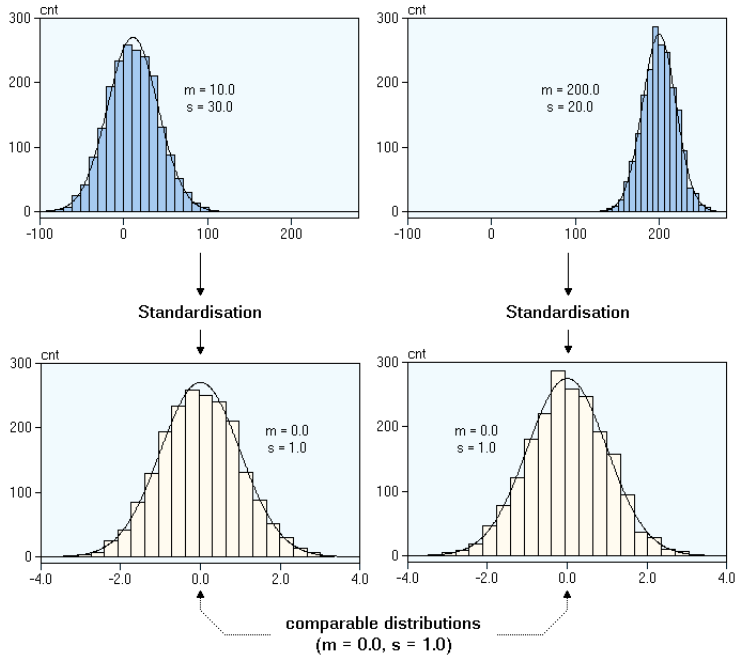


Figura 3.6: L'operazione di standardizzazione

Come è possibile vedere dal grafico 3.5 ci sono molti generi che rappresentano una varianza considerevole. Alcuni modelli, ad esempio le SVM, potrebbero performare male in casi come questo; sarà pertanto opportuno valutare i modelli anche dopo avere applicato un processo di standardizzazione sui dati.

La standardizzazione è una operazione che modifica i valori in modo che la loro media (μ) sia 0 e lo scarto quadratico medio (σ) sia 1,:

$$X_{\text{new}} = \frac{x_i - \mu}{\sigma} \tag{3.2}$$

Nella figura 3.6 è possibile vedere una rappresentazione grafica di questa procedura

3.1.6 Modellazione del problema

I dati di MicrobiomeHD possono essere usati per eseguire una classificazione sano-malato. I modelli addestrati possono poi essere analizzati per determinare quali sono i microorganismi più determinanti per eseguire l'assegnamento ad una delle due classi.

3.2 Gene Ontology

Gene Ontology (da ora abbreviato in *GO*) è un progetto nato con lo scopo di realizzare una nomenclatura comune per i geni e le funzioni biologiche dei vari organismi in modo da facilitare la ricerca e la condivisione dei dati provenienti da un insieme eterogeneo di specie. GO si compone di 3 antologie i cui termini sono legati da una relazione gerarchica espressa mediante un DAG[3].

Il grafo permette di rappresentare relazioni tra i vari termini; le relazioni possono essere:

- *Is_A*: il termine "A" è un termine "B"
- *Part_Of*: il termine "A" fa parte del termine "B"
- *Has_Part*: relazione complementare di Part_Of
- *Regulates*: il termine "A" regola il termine "B"

3.2.1 Matrici delle annotazioni

Le annotazioni vengono rappresentate mediante matrici binarie in cui le righe rappresentano i geni mentre le colonne rappresentano le funzioni biologiche. L'associazione tra il gene i e la funzione j viene rappresentata mediante l'inserimento di un 1 in corrispondenza della cella *i,j*. In questo modo viene indicato che il gene i svolge la funzione descritta dal termine *j*. Oltre a queste annotazioni esplicite è possibile anche definire un insieme di annotazioni implicite che possono essere ricavate dalla gerarchia GO: se la funzione j è annotata per il gene i è possibile procedere ad annotare anche le funzioni ancestor di quella correntemente in esame.

Queste relazioni permettono di costruire una gerarchia in cui, per ogni termine, possono essere indicati termini "ancestor" (superiori a livello di gerarchia quindi più generici) e "descendant" (inferiori a livello di gerarchia quindi più "specifici").

Oltre a queste informazioni per ogni annotazione sono inseriti dei codici che indicano come è stata prodotta; ad esempio ND(No biological Data avaiable) e IEA(Inferred from Electronic Annotation). Nelle analisi eseguite in questo contesto sono state considerate solo le annotazioni che sono state validate da un esperto umano, ovvero quelle che posseggono codici diversi da ND e IEA.

Questo tipo di annotazioni possono contenere errori ed essere incomplete; in questo contesto è importante avere a disposizione metodi affidabili in grado di rilevare possibili correzioni, soprattutto perché non è sempre possibile fare affidamento sulla possibilità di avere a disposizione un curatore umano in grado di verificare la correttezza delle informazioni contenute negli archivi.

[3]Directed Acyclic Graph

gene	GO:0042288	GO:0043231	GO:0045935	GO:0016782	GO:0044763	GO:0005623	GO:0044267	GO:0044422	GO:0003674	GO:0031974	...	GO:0008252	GO:0000808	GO:0043539	GO:0032909	GO:0008253	GO:0002162	GO:0001977	GO:0032389	GO:0048536	GO:0031579
LILRB1	1	0	1	0	1	1	0	0	1	0	...	0	0	0	0	0	0	0	0	0	0
BAIAP2L1	0	1	0	0	1	1	0	0	1	0	...	0	0	0	0	0	0	0	0	0	0
UBA52	0	1	1	0	1	1	1	1	1	1	...	0	0	0	0	0	0	0	0	0	0
HS3ST3A1	0	1	0	1	0	1	0	1	1	0	...	0	0	0	0	0	0	0	0	0	0
NTRK2	0	0	1	0	1	1	1	0	1	0	...	0	0	0	0	0	0	0	0	0	0
SERPINA3	0	1	0	0	0	1	0	0	1	0	...	0	0	0	0	0	0	0	0	0	0
MUC19	0	1	0	0	0	1	1	1	0	1	...	0	0	0	0	0	0	0	0	0	0
ZNF638	0	1	0	0	0	1	0	1	1	1	...	0	0	0	0	0	0	0	0	0	0
PTPRN2	0	0	0	0	0	1	1	0	1	0	...	0	0	0	0	0	0	0	0	0	0
UBE2I	0	1	0	0	0	1	1	1	1	1	...	0	0	0	0	0	0	0	0	0	0
FUCA1	0	0	0	0	0	1	0	0	1	0	...	0	0	0	0	0	0	0	0	0	0
STOML2	0	1	1	0	1	1	1	1	1	1	...	0	0	0	0	0	0	0	0	0	0
FZD7	0	0	1	0	1	1	0	0	1	0	...	0	0	0	0	0	0	0	0	0	0
ACLY	0	1	0	0	0	1	0	0	1	0	...	0	0	0	0	0	0	0	0	0	0
GNB2	0	0	0	0	1	1	0	0	1	0	...	0	0	0	0	0	0	0	0	0	0
LYN	0	1	0	0	1	1	1	0	1	0	...	0	0	0	0	0	0	0	0	0	0
PTCH1	0	1	0	0	1	1	0	0	1	0	...	0	0	0	0	0	0	0	0	0	0
PGM2	0	0	0	0	0	1	0	0	1	0	...	0	0	0	0	0	0	0	0	0	0
IL21	0	0	0	0	1	0	0	0	1	0	...	0	0	0	0	0	0	0	0	0	0
NUP54	0	1	0	0	1	1	0	1	0	1	...	0	0	0	0	0	0	0	0	0	0
ATP10D	0	1	0	0	1	1	0	0	1	0	...	0	0	0	0	0	0	0	0	0	0
SLC1A1	0	0	0	0	1	1	0	0	1	0	...	0	0	0	0	0	0	0	0	0	0
MED12	0	1	1	0	1	1	0	1	1	1	...	0	0	0	0	0	0	0	0	0	0
RUNX3	0	1	0	0	1	1	1	1	1	1	...	0	0	0	0	0	0	0	0	0	0
HIST1H4L	0	1	0	0	1	1	0	1	1	1	...	0	0	0	0	0	0	0	0	0	0
KIF23	0	1	0	0	1	1	0	1	1	1	...	0	0	0	0	0	0	0	0	0	0
SEC31A	0	1	0	0	1	1	1	1	1	0	...	0	0	0	0	0	0	0	0	0	0
ADRA2B	0	0	0	0	1	1	1	0	1	0	...	0	0	0	0	0	0	0	0	0	0
SULT1E1	0	1	0	1	1	1	0	1	1	0	...	0	0	0	0	0	0	0	0	0	0
BRPF3	0	1	0	0	1	1	1	1	1	1	...	0	0	0	0	0	0	0	0	0	0
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
ZNF132	0	0	0	0	0	0	0	0	1	0	...	0	0	0	0	0	0	0	0	0	0
DDT	0	0	0	0	0	0	0	0	1	0	...	0	0	0	0	0	0	0	0	0	0
MYOM2	0	0	0	0	0	0	0	0	1	0	...	0	0	0	0	0	0	0	0	0	0
CYP3A7-CYP3AP1	0	0	0	0	0	0	0	0	1	0	...	0	0	0	0	0	0	0	0	0	0
PKIA	0	0	0	0	0	0	0	0	1	0	...	0	0	0	0	0	0	0	0	0	0
LRP1B	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	0
SPHKAP	0	0	0	0	0	0	0	0	1	0	...	0	0	0	0	0	0	0	0	0	0
CTRC	0	0	0	0	0	0	0	0	1	0	...	0	0	0	0	0	0	0	0	0	0
TSACC	0	0	0	0	0	0	0	0	1	0	...	0	0	0	0	0	0	0	0	0	0
INA	0	0	0	0	0	0	0	0	1	0	...	0	0	0	0	0	0	0	0	0	0
DSC1	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	0

Figura 3.7: Esempio di matrice delle annotazioni

	# Geni	# Termini	# Annotazioni	Annotazioni per gene
bt_2009	735	793	24943	33.93
bt_2013	2242	2286	115187	51.37
dd_2009	2277	1160	83149	36.51
dd_2013	3821	1475	124183	32.50
gg_2009	415	432	11350	27.34
gg_2013	1136	1378	57502	50.61
hs_2009	10773	3507	386447	35.87
hs_2013	13426	6346	886424	66.02
mm_2009	9818	3470	352721	35.92
mm_2013	14503	7458	999550	68.92

Tabella 3.4: Dimensioni delle tabelle annotazioni reperite

3.2.2 Analisi delle matrici delle annotazioni

Le matrici delle annotazioni possono essere reperite da Genomic and Proteomic Data Warehouse (GPDW); gli organismi selezionati sono i seguenti:

- *Mus Musculus (MM)*
- *Bos Taurus (BT)*
- *Gallus Gallus (GG)*
- *Dictyostelium Discoideum (DD)*
- *Homo Sapiens (HS)*

Per tutti gli organismi sono state scaricate due versioni delle matrici, provenienti da due anni diversi: 2009 e 2013.

Nella tabella 3.4 sono riportati i dati relativi ai vari organismi in esame; i dati fanno riferimento alle matrici con solo annotazioni validate da un ricercatore. Le relazioni implicite determinate dalla gerarchia sono state esplicitate.

3.2.3 Modellazione del problema

Nel corso degli anni sono stati proposti diversi approcci al problema, tra cui un approccio basato su una classificazione multi-label [10] in cui ad ogni gene possono essere associate più funzioni biologiche. Questo approccio permette di usare i classici algoritmi di learning sfruttando in rotazione ogni funzione come label da predire per poi concatenare le varie predizioni per ottenere una matrice finale delle predizioni. In aggiunta a questo

procedimento sono stati proposti anche approcci cross-domain [13] in cui vengono sfruttati i dati relativi alle annotazioni di un organismo maggiormente studiato per ricavare le predizioni per le associazioni di un organismo meno studiato.

Capitolo 4

Algoritmi di Machine Learning

I due task più comuni affrontati nel machine learning riguardano l'estrazione di modelli di conoscenza da masse di dati con metodi basati sulla regressione o con metodi di classificazione. I problemi di regressione consistono nel predire il valore di una variabile continua in base al valore assunto da altre variabili, dette feature, che definiscono le caratteristiche delle istanze del problema. I task di classificazione, invece, predicono il valore di una variabile discreta che rappresenta la classe a cui ogni istanza del problema appartiene in base alle sue caratteristiche. All'intero di questa tesi i problemi affrontati sono di classificazione binaria multivariata, ossia problemi di classificazione in cui ogni istanza può appartenere ad una ed una sola classe tra le due messe a disposizione sulla base del valore che le feature assumono.

4.1 Regressione

Come accennato in precedenza la regressione consente di stimare il valore di una variabile continua dipendente y sulla base di una o più variabili indipendenti x. Nel metodo lineare si assume che le x siano tra loro indipendenti.

$$\hat{y} = \alpha x + \beta \tag{4.1}$$

L'obiettivo della regressione è determinare i parametri che permettano di ottenere la stima migliore possibile del valore y. E' possibile eseguire sia una regressione univariata che multivariata, la formulazione rimane uguale per entrambi i casi.

4.1.1 Discesa del gradiente

La discesa del gradiente è il processo con il quale i metodi supervisionati sono in grado di apprendere; è un problema di ottimizzazione in cui i parametri del modello di conoscenza sono modificati in modo da minimizzare gli errori compiuti sui dati da cui viene appreso.

Nel caso della regressione l'errore compiuto usando i parametri α e β può essere definito come:

$$E(\alpha, \beta) = \frac{1}{m} \sum_{i=1}^{m} (\alpha x + b - y)^2 \tag{4.2}$$

L'errore può essere rappresentato in un grafico in $n+1$ dimensioni, una per ogni variabile del problema più una per l'errore; il nostro scopo è trovare la combinazione dei parametri che permettono di minimizzare quest'ultimo.

Il gradiente di una funzione f è il vettore delle derivate parziali corrispondenti a ciascuna delle sue variabili. A livello intuitivo è possibile affermare che il gradiente in un punto rappresenta l'inclinazione della curva in quel punto. Sfruttando questa definizione sul grafico dell'errore è possibile utilizzare un processo iterativo per aggiornare i parametri:

1. Si parte da un punto x_0 fissando un valore iniziale per i parametri
2. Al punto x_k si determina l'errore
3. Determinazione del gradiente
4. Al punto x_k si sottrae un valore proporzionale al gradiente in quel punto per ottenere un punto x_{k+1} che consente di avere un tasso di errore minore [1]
5. $k = k + 1$

Questi passaggi vengono ripetuti fino alla convergenza, ovvero fino all'identificazione di un punto di minimo. Questo approccio è alla base di moltissimi tipi di algoritmi di learning.

4.2 Classificazione lineare mediante iperpiani

I metodi che appartengono a questa categoria sfruttano la creazione di iperpiani per separare in modo lineare i dati su cui vengono addestrati. Un iperpiano $wx + b = 0$ divide lo spazio in due partizioni, ognuna occupata dai membri di una classe specifica, ed è definito da due parametri:

- w: vettore unitario perpendicolare all'iperpiano
- b: l'intercetta definisce la distanza dell'iperpiano dall'origine.

Le istanze $x^{(+)}$ tali che $wx^{(+)} > 0$ appartengono alla classe al di sopra dell'iperpiano. Le istanze x^{-} tali che $wx^{(-)} < 0$ appartengono alla classe al di sotto dell'iperpiano.

[1] L'entità degli aggiornamenti è chiamata *learning rate* e può essere regolata tramite un iperparametro

4.2.1 Determinazione di un iperpiano di separazione

Ogni istanza viene etichettata come $+1$ oppure come -1 a seconda della classe di appartenenza, pertanto l'iperpiano di classificazione può essere riscritto in forma compatta come $-y(b + wx) < 0$. Se una istanza viene classificata in modo corretto la precedente definizione assumerà valore negativo, positivo altrimenti. Questa proprietà può essere usata per definire la seguente funzione che restituisce 0 in caso di classificazione corretta, altrimenti un valore positivo:

$$\max(0, -y(b + wx)) \tag{4.3}$$

A questo punto 4.3 può essere usata per definire la funzione di errore da minimizzare rispetto a b e w per le m istanze di training.

$$\underset{b,w}{\text{minimize}} \sum_{i=1}^{m} \max(0, -y_i h_w(x_i)) \tag{4.4}$$

Dove $h_w(x_i) = b + wx_i$.

L'equazione 4.4 non è derivabile quindi non è usabile per sfruttare la discesa del gradiente, è pertanto necessario sostituirla con una sua approssimazione, identificabile nella funzione softmax:

$$\text{softmax}(0, s) = \log(1 + e^s) \tag{4.5}$$

Perciò minimizzando la somma di 4.5 rispetto a w e b si ottiene:

$$\underset{b,w}{\text{minimize}} \sum_{i=1}^{m} \log\left(1 + e^{-y_i h_w(x_i)}\right) \tag{4.6}$$

Quanto definito in 4.6 è noto come *logloss*.

4.2.2 Regressione Logistica

$$\sigma(t) = \frac{1}{1 + e^{-(b+wt)}} \tag{4.7}$$

La funzione σ descritta in 4.7 è una approssimazione di una funzione a gradino ed il suo risultato è interpretabile come la probabilità di una istanza t di appartenere ad una delle classi. I parametri w e b sono quelli dell'iperpiano di separazione determinato. Come è possibile vedere nella figura 4.1 la funzione ha codominio $\mathbb{R} \to [0 : 1]$.

4.3 SVM

Le macchine a vettori di supporto sono un algoritmo di classificazione descritto per la prima volta da Vapnik [12] nel 1995; lo scopo di questo algoritmo è individuare il

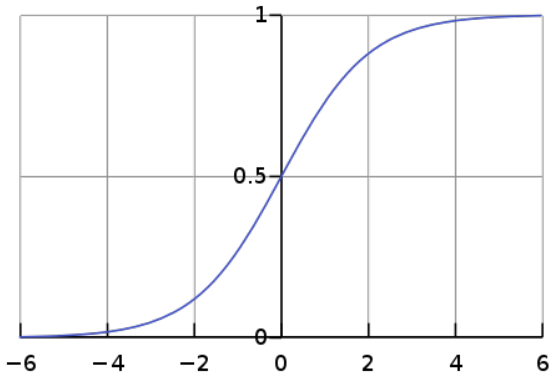


Figura 4.1: Regressione logistica

migliore piano di separazione tra tutti i possibili iperpiani che sono in grado di separare i dati. Nel caso i dati non fossero linearmente separabili il problema può essere reso affrontabile sfruttando una mappatura dei dati in uno spazio con una dimensionalità maggiore sfruttando le funzioni kernel.

L'iperpiano migliore viene identificato in quello che massimizza il margine di separazione tra le istanze in modo da ridurre l'overfitting.

I punti da cui si vuole calcolare la distanza da massimizzare sono detti *support vector* e, dato che sono punti di confine, risultano i punti più complessi da classificare in modo corretto. Dato che il modello di conoscenza si basa sui vettori di supporto non sono necessarie molte istanze di addestramento, difatti basterebbero i soli vettori di supporto per addestrare una SVM.[2]

4.3.1 Classificazione soft margin

Occorre gestire il trade-off tra massimizzazione del margine e minimizzazione degli errori, questo può essere portato a termine mediante l'introduzione di una tolleranza agli errori nel training-set. É possibile introdurre nuove variabili dette di *slack* che permettono di spostare le istanze classificate in modo errato sul vettore di supporto della classe a cui appartengono. Con questa procedura vengono ottenute soluzioni ottime degeneri ma con un margine maggiore. La somma delle variabili di slack deve essere minimizzata all'interno della funzione obiettivo; il peso di questa sommatoria viene regolato tramite iperparametro: un maggiore peso comporta un margine minore e viceversa.

[2]Rimuovendo dal set di addestramento tutti i punti tranne i vettori di supporto si otterrebbe lo stesso risultato.

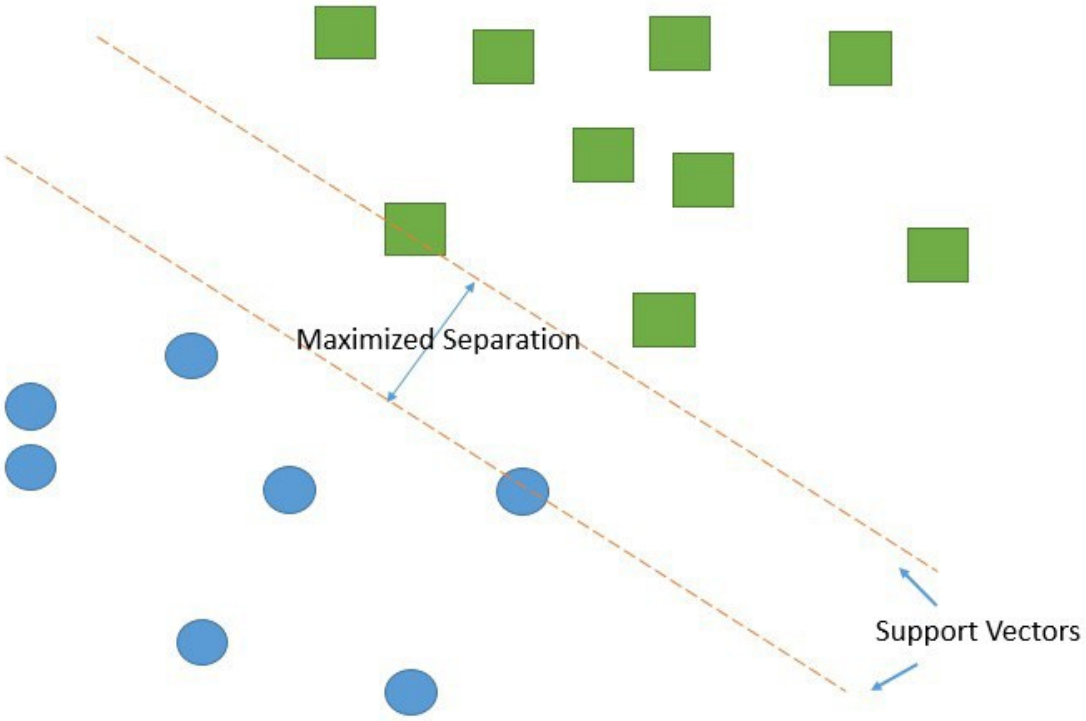


Figura 4.2: Esempio di SVM

4.3.2 Funzionamento

Come detto in precedenza le SVM sfruttano un iperpiano di decisione per decidere a quale classe ogni istanza deve appartenere, tale iperpiano è definito nel seguente modo:

$$\hat{y} = \begin{cases} 0, & \text{if } w^T w + b < 0, \\ 1, & \text{if } w^T w + b \geq 0 \end{cases} \tag{4.8}$$

L'iperpiano di separazione è formato dai punti che rispettano $w^T x_i + b = 0$ mentre i vettori di supporto sono composti dai punti che rispettano:

$$w^t x_i + b = 1 \tag{4.9}$$

per le istanze che appartengono alla classe 0

$$w^t x_i + b = -1 \tag{4.10}$$

per i punti che appartengono alla classe 1.

I vettori di supporto sono paralleli all'iperpiano di classificazione pertanto il margine totale può essere definito come:

$$\rho = \frac{|w^T s_1 + b|}{||w||} + \frac{|w^T s_2 + b|}{||w||} = \frac{1}{||w||} + \frac{1}{||w||} = \frac{2}{||w||} \tag{4.11}$$

Da 4.11 è possibile notare come il margine aumenti al diminuire di w; questo ci permette di vedere la massimizzazione del margine come la minimizzazione di $\frac{||w||}{2}$ che può essere approssimata a $\frac{w^T w}{2}$.

A questo punto il problema può essere trattato come un problema di ottimizzazione:

$$\begin{aligned} &\underset{w,b}{\text{minimize}} \quad \frac{1}{2} w^t w \\ &\text{subject to} \quad t^{(i)}(w^t x^{(i)} + b) \geq 1 \text{ for } \; i = 1, \ldots, m. \end{aligned} \tag{4.12}$$

I vincoli di 4.12 sanciscono che ogni istanza sia classificata in modo corretto; questo approccio è definito *hard margin*. Per permettere che una qualche istanza sia classificata in modo errato durante il trainig è possibile sfruttare l'approccio soft margin introdotto in precedenza:

$$\begin{aligned} &\underset{w,b}{\text{minimize}} \quad \frac{1}{2} w^t w + C \sum_{i=1}^{m} \zeta^{(i)} \\ &\text{subject to} \quad t^{(i)}(w^t x^{(i)} + b) \geq 1 - \zeta^{(i)} \text{ and } \zeta^{(i)} \geq 0 \text{ for } \; i = 1, \ldots, m. \end{aligned} \tag{4.13}$$

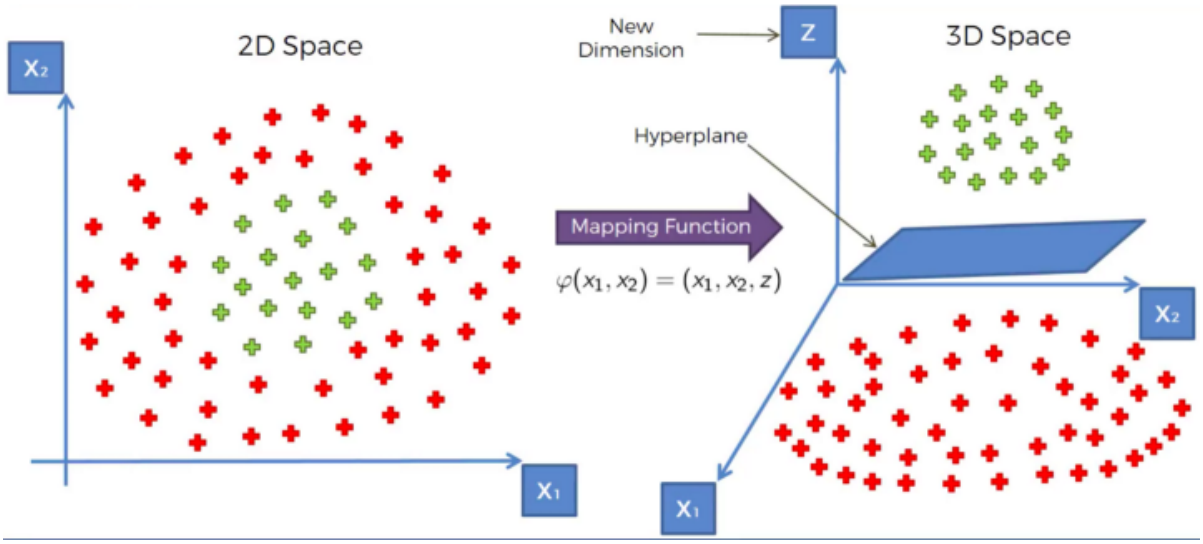


Figura 4.3: Esempio di SVM non lineare

4.3.3 SVM non lineari

Le SVM possono essere usate anche per classificare dati non linearmente separabili mediante una operazione di mapping in uno spazio ad alta dimensionalità, dove i dati sono separabili in modo lineare.

Il mapping avviene attraverso l'applicazione di una funzione non lineare che trasporta i punti in uno spazio composto da un numero maggiore di dimensioni in cui i dati risulteranno separabili mediante iperpiani. Il mapping avviene mediante delle funzioni dette *kernel* che consentono di ottenere i punti mappati nelle nuove dimensioni senza dovere effettivamente memorizzarli.

4.3.4 Principali iperparametri

- kernel: permette di determinare la funzione kernel usata per eseguire il mapping dei dati nello spazio ad alta dimensionalità
- gamma: consente di determinare quanto il modello di conoscenza deve essere "aderente" ai dati, solo per kernel non lineare
- C: peso della penalità per gli errori
- degree: usato nei kernel polinomiali per indicare il grado del polinomio
- probability: permette di ottenere la confidenza della classificazione fatta

4.4 Alberi di decisione

Un albero di decisione è un modello di conoscenza in cui ogni nodo rappresenta una feature e ogni ramificazione rappresenta una regola di decisione appresa dai dati; le foglie contengono le previsioni, ossia nella situazione ideale senza errori di classificazione ogni foglia contiene solo istanze di una data classe.

Gli alberi di decisione permettono di portare a termine sia dei task di regressione che di classificazione. Uno degli aspetti più vantaggiosi degli alberi di decisione è la facilità di interpretazione: dal momento che l'algoritmo di addestramento produce delle regole è possibile capire in modo molto diretto cosa è stato appreso.

4.4.1 Costruzione

Determinare l'albero di decisione ottimale è un problema molto complesso da trattare, per questo motivo vengono usate euristiche per rendere trattabile il problema. Una delle euristiche più usate per la costruzione è un approccio greedy top-down in cui, ad ogni passo, si cerca di massimizzare la purezza della classificazione, ovvero il numero di istanze classificate in modo corretto; per ogni nodo di decisione si procede in modo ricorsivo nel determinare gli split successivi. Un approccio di questo tipo è molto prono all'overfitting pertanto è necessario regolare tramite iperparametri alcuni criteri di arresto.

Dalla figura 4.4 è possibile vedere come un albero di decisione determini una serie di iperpiani di separazione perpendicolari tra loro; questa caratteristica causa una alta sensibilità ai cambiamenti dei dati. Considerando anche l'approccio euristico utilizzato durante la costruzione è possibile ottenere alberi diversi utilizzando gli stessi dati di addestramento.

4.4.2 Principali iperparametri

La costruzione di un albero di decisione può essere controllata mediante diversi iperparametri. In generale gli iperparametri consentono di regolare il grado di libertà che l'albero possiede durante la sua crescita; lo scopo è trovare un compromesso tra l'adattamento e l'overfitting, ovvero è necessario concedere all'albero la possibilità di adattarsi ai dati sottostanti senza però imparare a memoria.

- max_depth: indica la profondità massima dell'albero; al crescere della profondità possono essere eseguiti un numero maggiore di split e, di conseguenza, verranno apprese maggiori condizioni
- min_samples_split: indica il numero minimo di esempi in ogni nodo interno; in questo modo vengono ridotti il numero di split che possono essere eseguiti

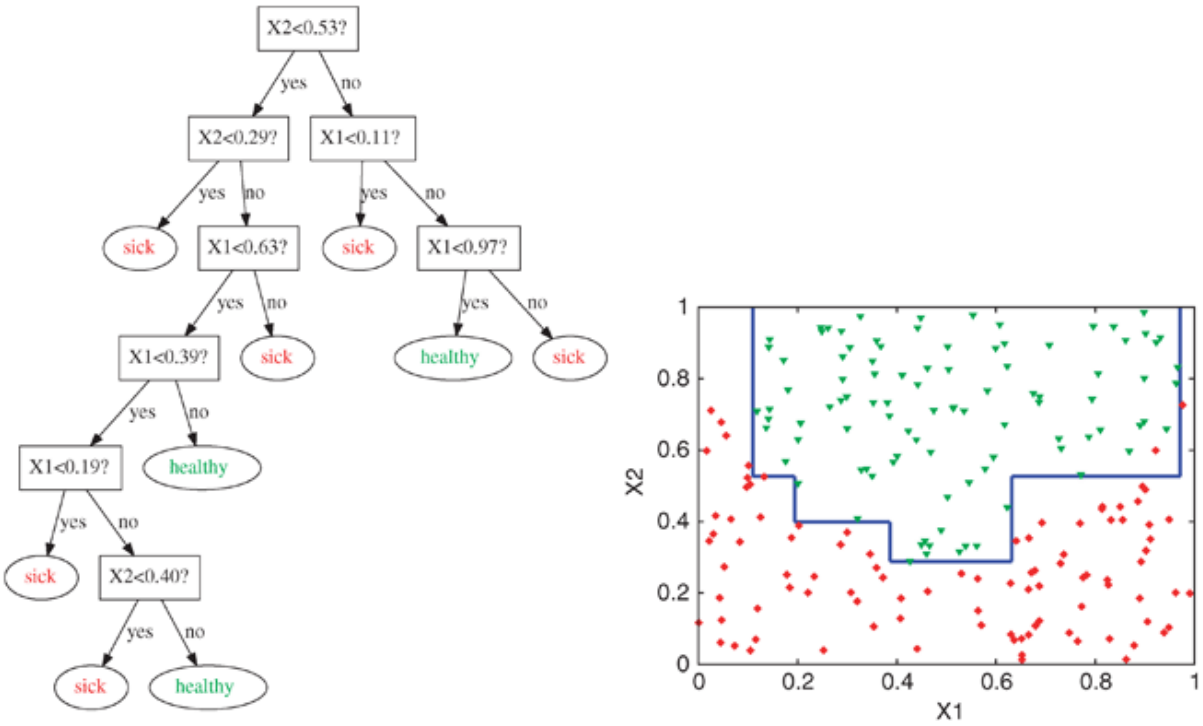


Figura 4.4: Esempio di albero di decisione

4.5 Random Forest

Le foreste sono un esempio di ensemble learning, una tecnica che consente di unire più modelli di conoscenza "deboli" per avere predizioni più accurate. Random forest apprende insiemi di alberi di decisione mediante alcune scelte casuali:

- Randomizzazione a livello di riga: ognuno degli alberi di decisione viene addestrato su un sottoinsieme casuale delle righe. Una stessa riga può essere usata più volte per eseguire l'addestramento del modello; questo approccio è chiamato *bagging*.
- Randomizzazione a livello di colonne: ogni albero viene addestrato solamente su un sottoinsieme delle colonne

Random forest sfrutta l'instabilità ai cambiamenti degli alberi per costruirne un insieme eterogeneo in modo da ottenere dei predittori molto variegati tra loro e quindi complessivamente meno affetti da overfitting.

Una delle caratteristiche più importanti di random forest è la possibilità di determinare quali sono le feature più importanti: dopo la fase di addestramento, per ogni feature, viene determinata una misura che indica quanto essa venga usata all'interno dei vari nodi per ridurre l'impurità degli stessi.

4.5.1 Principali iperparametri

Un modello di conoscenza appreso da random forest possiede gli stessi parametri che possiede un albero di decisione dato che il modello di random forest è costituito da tanti alberi di decisione. In aggiunta è possibile regolare il numero di alberi che compongono la foresta e imporre un limite al numero di feature che è necessario indagare per determinare lo split migliore. Quest'ultimo parametro è presente anche all'interno dei semplici alberi, ma qui assume una maggiore rilevanza dato che si lavora con un aggregato di alberi.

4.6 XGBoost, eXtreme Gradient Boosting

XGBoost [9] è un algoritmo che recentemente ha dominato il mondo delle competizioni online di machine learning[3] [4]

XGBoost, come le random forest, si basa sul concetto di ensemble, ovvero sfrutta una aggregazione di modelli di conoscenza deboli per ottenere una predizione più accurata; ogni nuovo albero viene addestrato sugli errori del precedente in un processo detto *boosting*.

Dal punto di vista matematico il modello di conoscenza può essere rappresentato come una addizione di K funzioni, una per ogni albero:

$$\hat{y}_i = \phi(X_i) = \sum_{k=1}^{K} f_k(x_i) \tag{4.14}$$

Le funzioni che compongono il modello di conoscenza vengono imparate minimizzando il seguente obiettivo regolarizzato:

$$\begin{gathered} \mathcal{L}(\phi) \sum_i l(\hat{y}_i, y_i) + \sum_k \Omega(f_k) \\ \text{dove } \Omega(f) = \gamma \mathcal{T} + \frac{1}{2}\lambda ||\omega||^2 \end{gathered} \tag{4.15}$$

L'equazione (4.15) include l, una funzione convessa che misura la differenza tra la predizione e il valore attuale. Il secondo termine (la regolarizzazione) penalizza la creazione di modelli di conoscenza complessi, T rappresenta il numero delle foglie mentre ω sono i parametri appresi. A livello intuitivo è possibile affermare che ogni modello è un compromesso tra correttezza e complessità. Un modello molto complesso, e quindi aderente ai dati di addestramento, è prono ad ottenere risultati ottimi su quest'ultimi e fallire su dati inediti.

Il modello definito da 4.15 include funzioni come parametri; questo non permette di sfruttare le tecniche classiche che operano nello spazio euclideo. Per venire incontro

[3] https://www.kaggle.com/

[4] https://www.kaggle.com/shivamb/data-science-trends-on-kaggle

a questa limitazione il modello viene generato mediante un metodo di addestramento additivo: ad ogni iterazione viene aggiunto in modo greedy una funzione di classificazione, quindi un albero, con lo scopo di correggere gli errori compiuti nell'iterazione precedente

$$\mathcal{L}^{(t)} = \sum_{i=1}^{n} l(y_i, \hat{y_i}^{t-1}) + f_t(x_i) + \Omega(f_t) \tag{4.16}$$

4.6.1 Shrinkage and Column Subsampling

Oltre alla regolarizzazione inserita in (4.15) sono previste altre due accortezze per prevenire l'overfitting del modello. Una delle tecniche è il subsampling delle colonne durante la determinazione del migliore split: come nelle random forest solo un sottoinsieme delle colonne viene preso in considerazione per determinare lo split. Per evitare l'esplosione del gradiente i valori ottenuti vengono scalati di un valore η dopo ogni passo; questo approccio è detto Shrinkage.

4.6.2 Sparsity-Aware Split Finding

Uno degli altri punti di forza di XGBoost è la gestione nativa dei dati mancanti. Per ogni split viene determinata anche la migliore direzione di default, ovvero una direzione che viene seguita quando non sono presenti dati per una feature. Questa capacità è molto importante perché spesso i dati con cui ci si trova a lavorare sono sparsi. Tra le possibili cause si possono evidenziare sia l'effettiva assenza di un dato ma anche l'elevato numero di zeri generati da procedure come one-hot-encoding[5]

4.6.3 Principali iperparametri

In aggiunta ai parametri classici previsti per le foreste vengono introdotti dei parametri per regolare il processo di boosting come ad esempio il *learning_rate*, ovvero il peso con cui considerare il gradiente nel momento in cui si indagano gli errori di classificazione per determinare il nuovo classificatore. Un learning rate basso costringerà ad usare un numero di alberi maggiore per avere una modellazione efficace, mentre un learning rate elevato porta ad avere un modello poco stabile a causa di repentini cambiamenti.

Un altro parametro legato al boosting è *gamma*, un fattore che determina la soglia minima di riduzione della funzione di errore per aggiungere uno split.

[5]one-hot-encoding è una procedura di codifica delle variabili categoriche (non numeriche) che consiste nell'aggiungere una feature per ogni valore assunto dalla variabile per poi indicare con un 1 la colonna che conteneva il valore

4.7 LigthGBM

LigthGBM [24] è un algoritmo proposto da Microsoft che ha come scopo massimizzare la velocità di apprendimento; come XGBoost si basa sul concetto di boosting.

LigthGBM raggiunge i suoi scopi mediante due accorgimenti: GOSS (Gradient Based One Side Sampling) ed EFB (Exclusive Feature Bundling)

4.7.1 GOSS

GOSS permette di ridurre il numero di istanze su cui viene calcolato il gradiente in modo da ridurre i tempi di calcolo. Verranno considerate sole le prime $a\%$ con gradiente maggiore; tra le rimanenti vengono selezionate in modo casuale il $b\%$ di esse.

Le istanze con un piccolo gradiente vengono "amplificate" secondo una costante $\frac{1-a}{b}$ nel momento in cui l'algoritmo è chiamato a costruire il nuovo predittore del modello di conoscenza.

Sarebbe possibile procedere anche con il semplice scartare le variabili poco importanti ma questo comporterebbe una perdita di informazione.

4.7.2 EFB

Gli spazi con un elevato numero di dimensioni sono spesso molto sparsi e molto spesso le feature sono mutualmente esclusive [6]. Queste feature sparse e mutualmente esclusive possono essere collassate in una singola feature, detta *exclusive feature bundle*

4.8 CatBoost

CatBoost [14] è un algoritmo progettato da Yandex con lo scopo di gestire in modo efficace le feature categoriche [7] senza dovere ricorrere a procedere come l'encoding che comportano la nascita di uno spazio ad alta dimensionalità.

Uno dei possibili approcci per gestire le feature categoriche senza ricorrere all'encoding è il computo di statistiche che dipendono dalla label come ad esempio, per ogni feature categorica, sostituire il valore con la media delle label che posseggono quel valore.

$$x_{i,k} = \frac{\sum_{j=1}^{n} [x_{j,k} = x_{i,k}] \cdot Y_j}{\sum_{j=1}^{n} [x_{j,k} = x_{i,k}]} \tag{4.17}$$

Un soluzione come (4.17) è prona all'overfitting, ad esempio se una unica istanza appartenesse ad una categoria essa verrebbe sostituita con la label stessa. Per ovviare a al problema una parte del dataset dovrebbe essere usata solamente per il calcolo

[6] questo si verifica, ad esempio, quando si esegue one-hot-encoding

[7] Feature non numeriche non direttamente confrontabili tra di loro

di queste statistiche ma questo comporterebbe una riduzione dei dati disponibili per l'addestramento e la validazione.

Per risolvere il problema CatBoost esegue una permutazione dell'intero dataset e, per ogni riga, calcola la media considerando le istanze che la precedono.

4.8.1 Principali iperparametri

CatBoost consente di regolare, oltre ai parametri legati alle foreste e al boosting già visti, la gestione interna delle feature categoriche. Nello specifico è possibile indicare il numero massimo di valori per cui applicare una conversione interna mediante one-hot-encoding piuttosto che convertire le categorie in interi.

4.9 Voting

Il voting è una tecnica di ensemble learning in cui più modelli di conoscenza sono chiamati ad eseguire una votazione sul risultato della classificazione. In queste prove è stato usato un voting di tipo *soft* in cui il voto di ogni modello di conoscenza è pesato dalla sua confidenza. In questo modo il voto di un modello di conoscenza molto sicuro della propria predizione ha un peso maggiore di quello di un modello di conoscenza meno performante.

Capitolo 5

Metriche di Valutazione

La valutazione di modelli di conoscenza appresi è una parte fondamentale di ogni progetto di data science. Questa fase valuta le prestazioni dei modelli appresi dagli algoritmi nella fase di addestramento. Di seguito sono descritti i principali concetti e le metriche usate in questa tesi.

5.1 Matrice di confusione

La matrice di confusione permette di mostrare i risultati del classificatore in modo intuitivo mediante una rappresentazione matriciale; partendo da questa rappresentazione è possibile estrarre diverse metriche di valutazione.

Dai valori contenuti nelle celle delle matrici è possibile estrarre le seguenti misure:

- True Positive (TP): numero di predizioni corrette di un caso positivo
- True Negative (TN): numero di predizioni corrette di un caso negativo
- False Positive (FP): errore di tipo 2, un caso negativo è stato segnalato come positivo
- False Negative (FN): errore di tipo 1, un caso positivo è stato segnalato come negativo

Dalla matrice è possibile ricavare altre metriche con lo scopo di indagare in modo più approfondito i risultati.

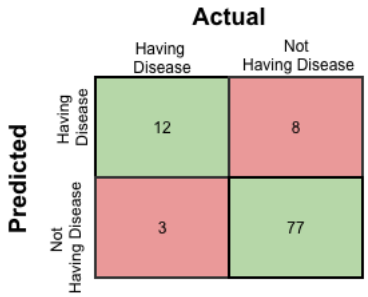


Figura 5.1: Esempio di matrice di confusione

5.2 Metriche

5.2.1 Accuracy

L'accuratezza rappresenta il numero di predizioni corrette effettuate, ad esempio una accuratezza del 90% significa che il modello ha eseguito 90 predizioni corrette su 100.

$$\frac{\mathrm{TP} + \mathrm{TN}}{\mathrm{TP} + \mathrm{TN} + \mathrm{FP} + \mathrm{FN}} \tag{5.1}$$

Non è una metrica adeguata per dataset non bilanciati perché la classificazione corretta della classe maggiormente rappresentata maschererebbe gli errori fatti sulla minoranza.

5.2.2 Precision e Recall

La precision rappresenta la misura delle classificazioni positive corrette prodotte; se un modello ha una precision del 90% significa che se viene prodotta una predizione vera allora questa è corretta con una probabilità del 90%.

$$\mathrm{Precision} = \frac{\mathrm{TP}}{\mathrm{TP} + \mathrm{FN}} \tag{5.2}$$

La recall misura la porzione delle predizioni positive corrette effettuate; è una misura che consente di stimare la presenza di falsi negativi: le istanze vere erroneamente classificate come false.

$$\mathrm{recall} = \frac{\mathrm{TP}}{\mathrm{TP} + \mathrm{FN}} \tag{5.3}$$

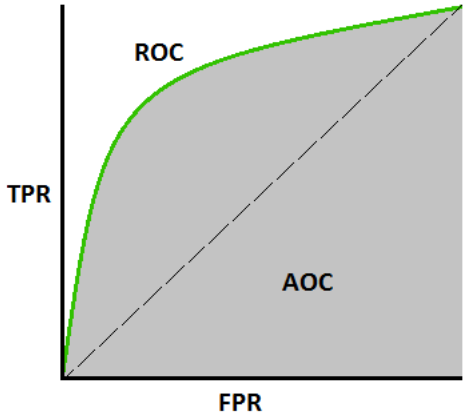


Figura 5.2: Esempio di curva ROC e relativa AUC, la linea tratteggiata rappresenta un classificatore casuale

Per valutare in modo efficace le prestazioni di un modello sia precision che recall devono essere prese in esame. Ottimizzare per una delle due porta degli scompensi per l'altra misura; a seconda del problema occorre impostare gli iperparametri dell'algoritmo di learning in modo che l'errore più grave venga penalizzato.

Un modo per regolare questi parametri è anche impostare una soglia di confidenza per accettare come "vera" la predizione del modello, ad esempio se è importante evitare falsi negativi è possibile accettare una predizione come negativa solo se il modello ripone una confidenza nel risultato superiore ad una certa soglia.

Per avere una idea delle prestazioni complessive è possibile utilizzare una ulteriore metrica: *F1 score*, la media armonica di precision e recall.

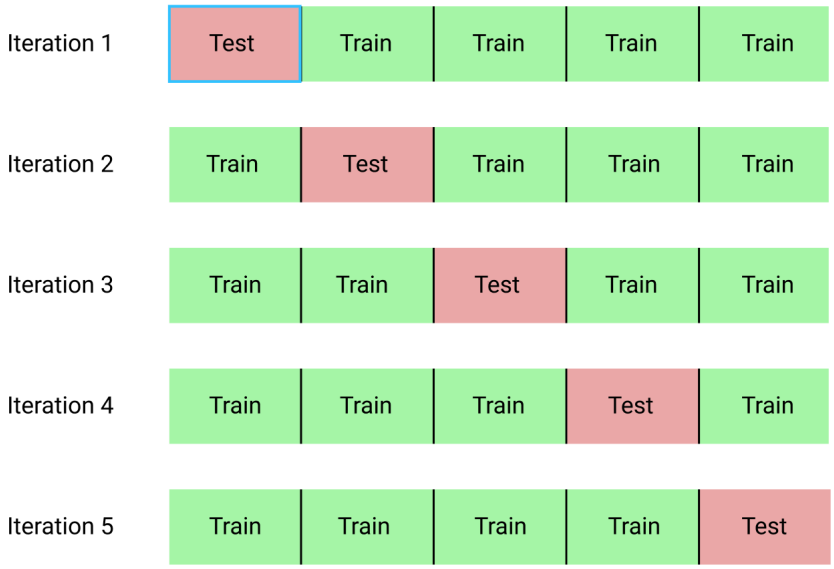
$$F1 = (\frac{\text{recall}^{-1} + \text{precision}^{-1}}{2})^{-1} \tag{5.4}$$

5.3 Curva ROC e AUC

L'area sotto la curva ROC[1], chiamata AUC[2], è una misura grafica della bontà del classificatore, usata in ambito medico fin dal 1970. Recentemente è stata introdotta come misura per la qualità di modelli [23].

La curva ROC viene ottenuta realizzando un grafico che mostra il rapporto tra il numero di veri positivi e falsi positivi al variare della soglia di confidenza della predizione. Fornisce una informazione su quanto il modello è capace di distinguere tra le classi. Maggiore è l'area al disotto della curva, maggiore è la bontà del classificatore. Per avere

[1] Receiver operating characteristic
[2] Area Under the Curve



un termine di confronto è possibile notare che l'AUC di un classificatore randomico è 0.5.

5.4 K-Fold Cross Validation

La K-Fold CV consiste nel dividere il dataset in K gruppi e, mediante un processo di rotazione, utilizzarne K-1 per eseguire l'addestramento; il rimanente, che conterrà dati inediti per il modello, viene usato per ottenere le metriche di valutazione.

Questo approccio è utile per avere risultati più stabili e meno dipendenti da come è stata effettuata la divisione tra train e test set.

Dopo avere calcolato le metriche per ognuno dei K gruppi è possibile ottenere una valutazione globale determinando media e varianza[3] dei risultati ottenuti

5.5 Gene Ontology

Questo caso di studio necessita di precisazioni per specificare alcuni accorgimenti che verranno adottati in fase di valutazione. Come metriche sono state prese in considerazione precision, recall ed F1 escludendo accuracy a causa del rapporto imbilanciato tra annotazioni presenti e dimensione delle matrici che le contengono. Visto che il problema

[3]La varianza rappresenta quanto i valori si discostano dal valore medio, se la varianza è bassa è possibile affermare che sono stati ottenuti risultati stabili

sarà modellato come una classificazione multilabel si utilizzeranno versioni leggermente riviste di precision, recall ed F1 che permettono di ottenere un valore medio di tali metriche; la media viene calcolata in due versioni:

- *macro*: media calcolata a livello di sample
- *micro*: media calcolata a livello globale

Oltre a queste metriche verranno valutate separatamente le transizioni, ovvero le annotazioni che nella versione iniziale erano impostate a 0, ma che sono state trasformate in 1 da parte del modello. Per le transizioni verrà prima di tutto determinato il numero di esse e poi quantificata una percentuale che indica quante sono corrette, ovvero confermate nella matrice dell'anno successivo.

Capitolo 6

Interpretazione dei Modelli di Conoscenza

6.1 Importanza dell'interpretazione di un modello

I risultati ottenuti, per quanto buoni, non devono essere l'unico criterio di decisione nella scelta del modello finale. La capacità di interpretare un modello è uno dei fattori da tenere in conto quando si vuole proporre un modello al pubblico; difficilmente in settori in cui la vita o le finanze delle persone sono a rischio è possibile accettare delle valutazioni non motivate. Con interpretabilità si intende la capacità di un soggetto umano di capire i criteri di decisioni del modello usato per produrre un risultato. Maggiore l'interpretabilità, maggiore la possibilità di capire le decisioni che hanno portato ad ottenere l'esito. Solitamente si parla di *tradeoff tra complessità ed interpretabilità*.

Un modello ottenuto mediante regressione lineare è molto semplice da capire: più i coefficienti associati alle variabili sono alti più la variabile sarà determinante per determinare il risultato. Di contro modelli ottenuti tramite algoritmi di learning più complessi, come le reti neurali, sono più complessi da interpretare.

Esistono diverse tecniche di analisi ma è possibile raggrupparle in due macro-categorie:

- *Analisi a livello globale*: determinare le feature di maggiore importanza analizzando le previsioni fatte sull'intero dataset. Ad esempio in algoritmi basati sulle foreste si possono ottenere stime di importanza usando metodi che forniscano informazioni circa il numero di volte in cui una feature è stata usata all'interno di uno split. A livello intuitivo è possibile affermare che una feature che viene spesso utilizzata per eseguire split è determinante per distinguere le varie classi o determinare un valore.

- *Analisi a livello locale*: in queste tecniche si determina, per ogni predizione, quali sono state le variabili che hanno avuto un peso maggiore per quella specifica istanza.

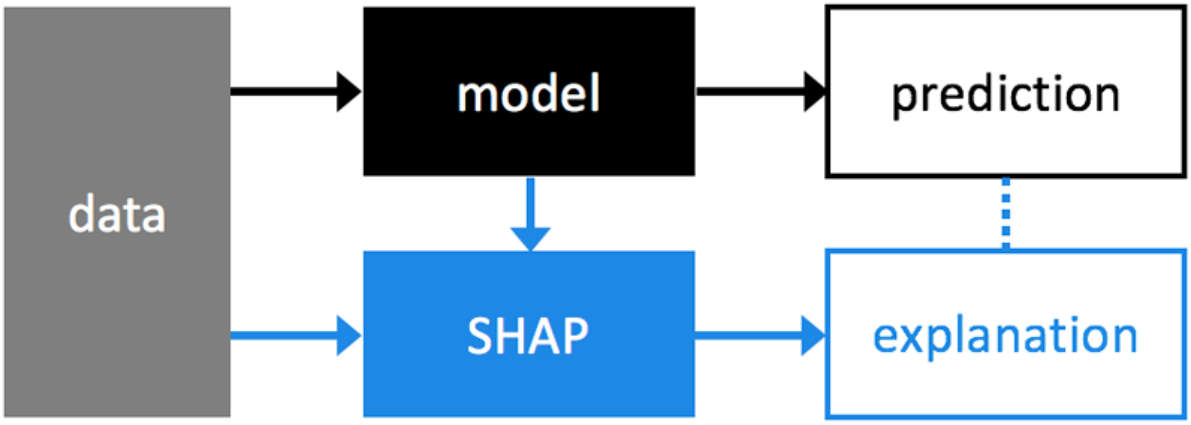


Figura 6.1: Funzionamento della libreria SHAP

6.2 SHAP

SHAP [1] [2] è una libreria pensata per eseguire analisi sui modelli in modo da poterli interpretare in modo rigoroso. Per eseguire le analisi sul modello ogni predizione viene esaminata nel dettaglio per determinare quali fattori hanno spinto verso una certa decisione. Per raggiungere questo scopo viene addestrato un modello lineare per ogni predizione.

6.2.1 Shapley Values

I valori SHAP nascono nel contesto della teoria dei giochi. Sono una tecnica usata per determinare quanto ogni giocatore abbia contribuito al successo in un gioco collaborativo, in questo contesto la predizione. Per la suddivisione dei punti, ovvero dell'importanza, si seguono le seguenti regole:

1. La somma dell'importanza delle singole feature deve essere pari al totale
2. Se due feature hanno contribuito in egual modo devono ricevere la medesima importanza
3. Se una feature non ha contribuito non deve ricevere importanza
4. La ricompensa totale dopo due giochi deve essere pari alla somma dei due compensi ricevuti

[1] https://github.com/slundberg/shap
[2] SHapley Addittive exPlanation

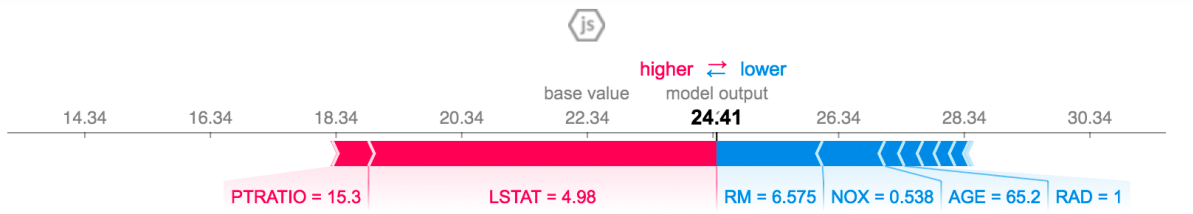


Figura 6.2: Prediction Explainer

Questi coefficienti possono essere calcolati, per ogni predizione, come descritto in 6.1 e vengono poi usati come parametri per il modello di learning usato per spiegare le predizioni.

$$\Phi_i(p) = \sum_{S \subseteq N/i} \frac{|S|!(n - |S| - 1)!}{n!}(p(S \cup i) - p(S)) \tag{6.1}$$

Ad alto livello è possibile affermare che viene calcolata la differenza tra la predizione di un modello che utilizza la feature e di uno che non la utilizza. Maggiore la differenza maggiore l'importanza della feature in esame.

Dato che l'ordine in cui le funzioni vengono testate porta ad ottenere risultati differenti si è costretti a lavorare con tutte le possibili permutazioni delle feature in esame.

6.3 Interpretazione dei grafici

Di seguito è inserita una spiegazione sui vari tipi di grafici che è possibile creare e come interpretarli. Come esempio viene usato un modello addestrato secondo l'algoritmo XGBoost sul dataset *Boston Housing*[3]

6.3.1 Prediction Explainer

Il grafico 6.2 mostra, per una specifica predizione, che ruolo hanno assunto le feature. Nel grafico viene indicato il valore base e il peso che assumono le diverse variabili per spingere verso l'alto (rosse) o verso il basso (blu) il risultato.

Un grafico di questo tipo può essere utile per analizzare nel dettaglio le predizioni sbagliate e per rendersi conto se una certa feature è determinante in modo sistematico all'interno degli errori compiuti.

[3]Boston Housing è un famoso dataset che viene usato per introdurre gli algoritmi di regressione, si compone diverse variabili usate per stimare il prezzo di una casa.

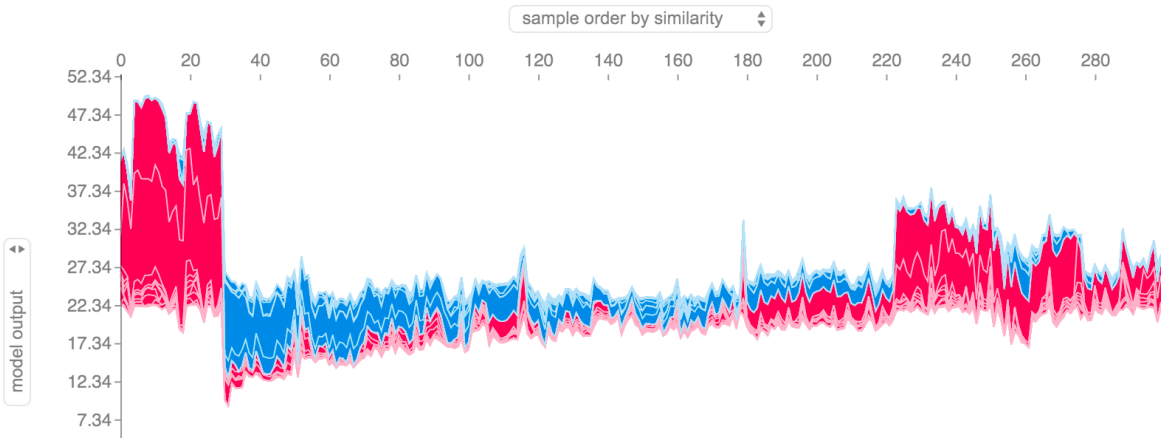


Figura 6.3: Model Explainer

6.3.2 Model Explainer

Il grafico in figura 6.3 ha lo scopo di rappresentare la spiegazione dell'intero modello. Sull'asse X sono inserite tutte le predizioni numerate secondo il loro ordine, mentre sull'asse Y il valore predetto. Interagendo con il grafico all'interno dell'ambiente di sviluppo viene evidenziato il dettaglio delle variabili. I colori rosso e blu assumono lo stesso significato del caso precedente.

6.3.3 Summary Plot

I grafici 6.4 e 6.5 sono due visualizzazioni per lo stesso output: una valutazione riassuntiva dell'intero modello. Il grafico 6.4 fornisce maggiori informazioni perché, per ogni predizione, viene inserito un punto. Il grafico 6.5 rappresenta una visualizzazione semplificata in cui, per ogni feature, viene inserito il valore di impatto medio.

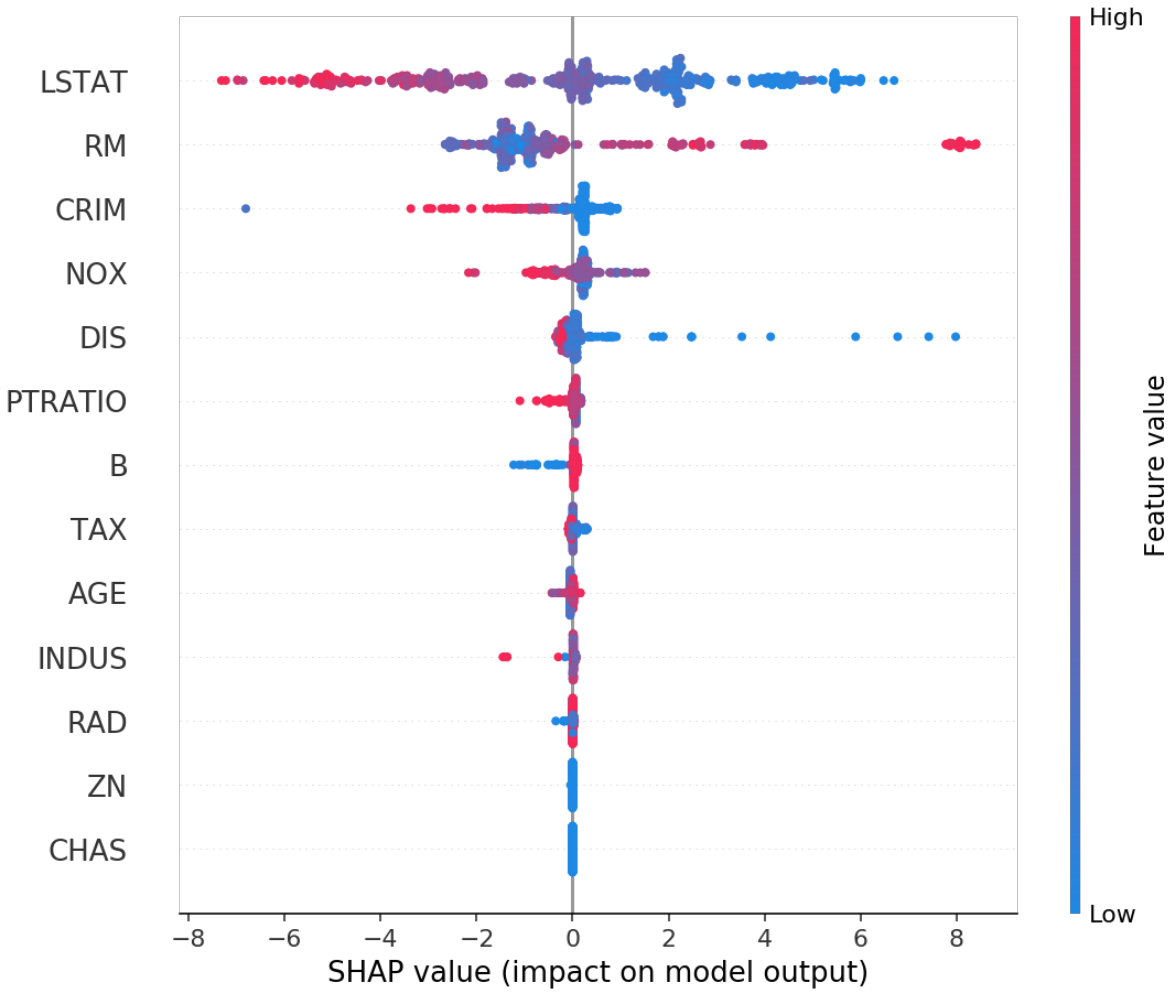


Figura 6.4: Summary Plot

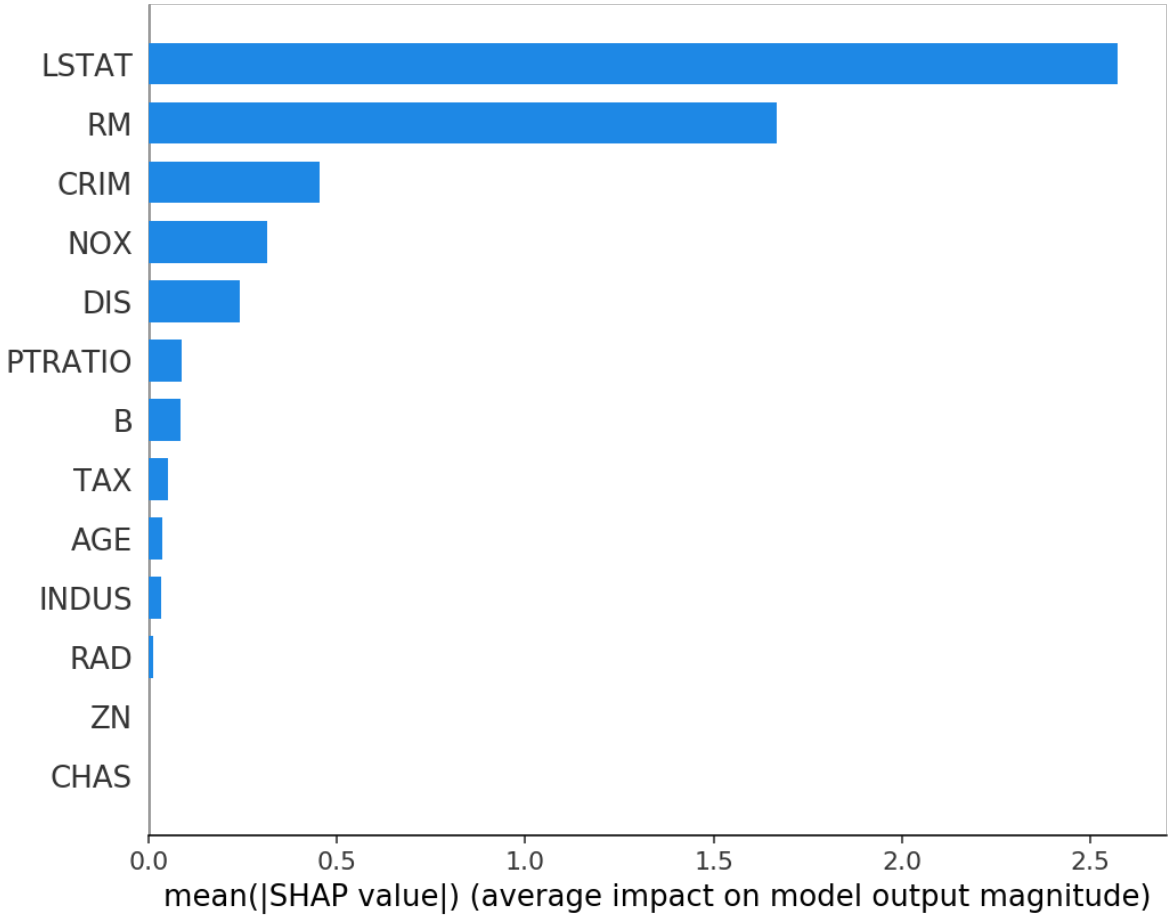


Figura 6.5: Summary Plot (Bar)

Capitolo 7

Esperimenti con MicrobiomeHD

All'interno di questo capitolo vengono descritti gli esperimenti eseguiti e i risultati ottenuti. Dove possibile viene fornito anche un confronto con gli articoli di riferimento.

Come algoritmi sono stati adoperati:

- XGBoost
- CatBoost
- Random Forest
- LigthBoost
- SVM con kerbel RBF
- Voting soft

7.1 Analisi sui singoli dataset

In questa sezione vengono riprodotti gli stessi esperimenti eseguiti nell'articolo. Come metrica di valutazione è stata utilizzata AUC in modo da avere un confronto diretto con quanto prodotto nella letteratura di riferimento. I risultati si sono verificati similari e, in alcuni casi, migliori.

7.1.1 Esperimenti di base

I primi esperimenti sono stati svolti sfruttando gli iperparametri di default messi a disposizione dai vari algoritmi e senza apportare nessuna forma di preprocessing. Sono stati rimossi i dati a cui non erano associate informazioni sullo stato di salute dato che non é possibile attribuirvi una condizione di salute normale o alterata.

	XGB	CAT	RDM	LGB	SVM	VTN	Baseline
asd_kb	72.92%	81.25%	81.67%	50.00%	60.00%	80.00%	76.00%
asd_son	53.20%	54.65%	54.96%	56.07%	54.65%	53.96%	39.00%
cdi_schubert	97.76%	98.83%	98.64%	98.46%	94.71%	98.45%	99.00%
cdi_vincent	87.20%	90.40%	87.20%	68.80%	71.20%	88.00%	91.00%
cdi_youngster	97.90%	97.90%	97.90%	99.41%	94.85%	99.08%	NaN
crc_xiang	74.50%	73.70%	65.60%	50.00%	85.00%	76.40%	78.00%
crc_zackular	47.37%	56.67%	60.71%	49.37%	59.80%	53.64%	NaN
crc_zeller	67.85%	66.57%	69.93%	60.24%	66.74%	71.10%	82.00%
crc_zhao	79.88%	85.44%	85.93%	75.57%	84.76%	82.71%	90.00%
edd_singh	90.26%	92.28%	91.91%	92.02%	85.76%	92.37%	96.00%
hiv_dinh	55.00%	68.00%	78.33%	50.00%	61.00%	59.00%	22.00%
hiv_lozupone	93.83%	96.00%	97.50%	87.33%	93.00%	94.86%	92.00%
hiv_noguerajulian	58.68%	64.30%	64.02%	59.61%	56.68%	63.30%	67.00%
ibd_alm	72.59%	79.22%	82.01%	69.92%	56.95%	76.27%	84.00%
ibd_engstrand_maxee	74.42%	71.52%	71.17%	76.62%	47.19%	74.69%	66.00%
ibd_gevers	64.71%	66.67%	65.77%	67.85%	61.45%	67.09%	71.00%
ibd_huttenhower	81.38%	78.42%	77.30%	82.22%	68.81%	82.23%	81.00%
mhe_ahang	74.01%	76.91%	78.99%	71.92%	67.68%	75.92%	80.00%
nash_chan	63.17%	66.83%	66.57%	51.64%	64.74%	60.88%	68.00%
nash_ob_baker	91.11%	93.48%	90.69%	80.22%	74.63%	89.04%	93.00%
ob_escobar	65.00%	82.50%	82.50%	50.00%	77.50%	80.00%	NaN
ob_goodrich	57.89%	59.36%	59.34%	59.78%	52.62%	59.70%	67.00%
ob_gordon	78.34%	89.19%	84.60%	81.20%	70.92%	84.63%	84.00%
ob_jumpertz	77.33%	91.81%	91.33%	64.86%	84.76%	83.81%	NaN
ob_ross	51.25%	49.26%	49.88%	47.49%	49.19%	49.52%	49.00%
ob_wu	51.84%	52.53%	51.98%	51.65%	48.28%	52.02%	NaN
ob_zeevi	51.84%	52.53%	52.75%	51.65%	48.28%	52.15%	NaN
ob_zupancic	63.63%	62.15%	63.82%	60.89%	64.11%	62.08%	44.00%
par_scheperjans	67.20%	70.22%	69.37%	56.20%	67.91%	67.27%	67.00%
ra_littman	50.30%	57.37%	55.88%	56.43%	58.44%	54.20%	62.00%
t1d_alkanani	71.32%	71.91%	71.85%	64.61%	46.96%	71.34%	71.00%
t1d_mejialeon	85.00%	72.50%	72.50%	50.00%	85.00%	77.50%	77.00%

Tabella 7.1: Risultati dei modelli addestrati sui dataset singoli e confronto con il riferimento

7.2 Analisi aggregati di malattie

In questa sezione sono stati messi alla prova dei classificatori in grado di operare sui vari dataset aggregati in base alla patologia di riferimento. Vista la riduzione nel numero di dataset è possibile rappresentare una gamma maggiore di metriche estratte durante la fase di valutazione.

Nelle figure 7.1,7.2 e 7.3 è possibile vedere i risultati ottenuti usando algoritmi addestrati sui dati grezzi, mentre nelle figure 7.4,7.5 e 7.6 sono riportati i dati ottenuti usando algoritmi addestrati su dati scalati.

É interessante notare come le performance delle SVM migliorino utilizzando lo scaling.

7.3 Analisi globale

In questa sezione vengono riportati i risultati di algoritmi addestrati sul dataset ottenuto mediante la concatenazione dei dati di tutti gli studi presi in esame. Dato che in questa fase si lavora difatto con un unico dataset è possibile procedere con delle tecniche di ricerca degli iperparametri. Anche in questo caso sono stati rieseguiti gli esperimenti applicando la normalizzazione delle feature.

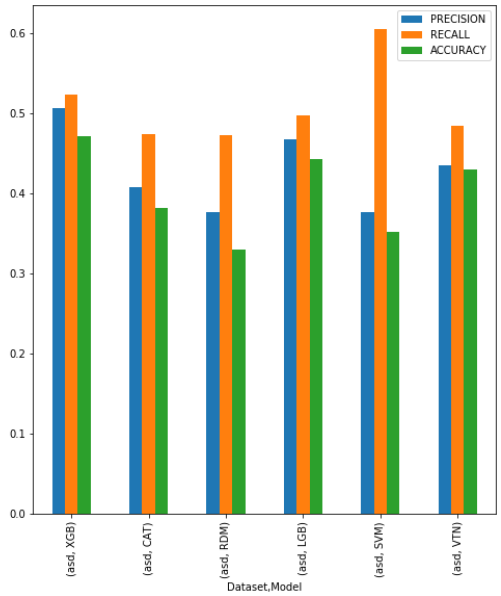
7.3.1 Ricerca degli iperparametri

Per gli algoritmi addestrati sui dati aggregati globalmente è stata effettuata una ricerca degli iperparametri mediate una RandomSearchCV. Questa tecnica è stata preferita ad una ricerca esaustiva [1] perché, solitamente, permette di ottenere risultati migliori in un tempo inferiore ricercando solo un sottoinsieme casuale delle possibili combinazioni di iperparametri. [6] Dopo avere determinato gli iperparametri ottimali sono state ripetute le analisi sia sui dati grezzi che su quelli scalati. Nella figura 7.9 sono riportati i risultati ottenuti.

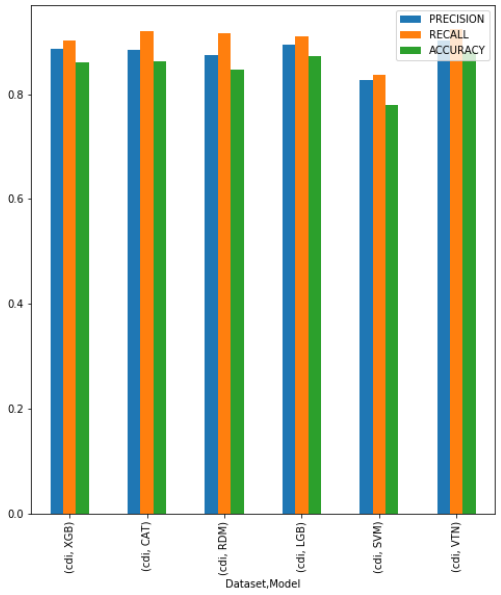
7.4 Analisi modello globale ottimizzato

Dopo la fase di valutazione sono state prodotte delle analisi con lo scopo di determinare le feature che il modello di conoscenza ha ritenuto di maggiore importanza. Di seguito vengono confrontate le feature apprese dal modello generalizzato e dai modelli specifici per le varie patologie; per brevità vengono riportate le analisi dei soli modelli XGBoost.

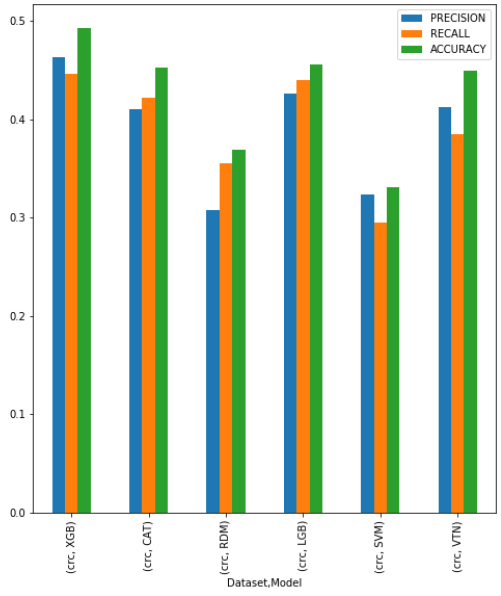
[1] GridSearch



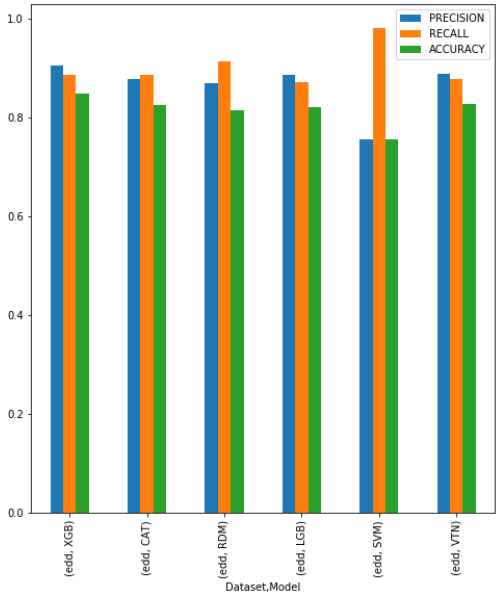
(a) Autistic Spectrum Disorder



(b) Clostridium Difficile Infection

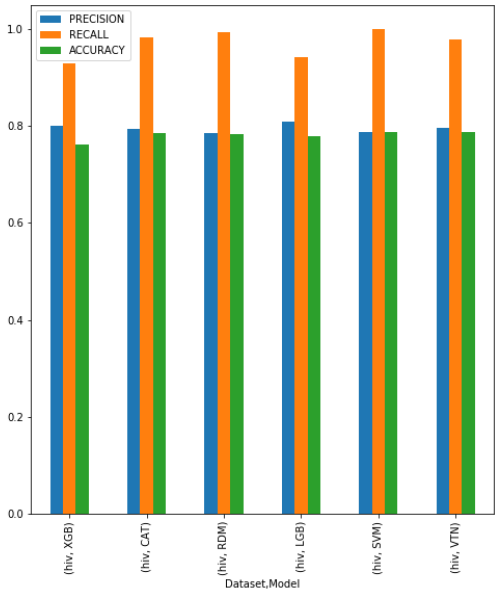


(c) ColoRectal Cancer

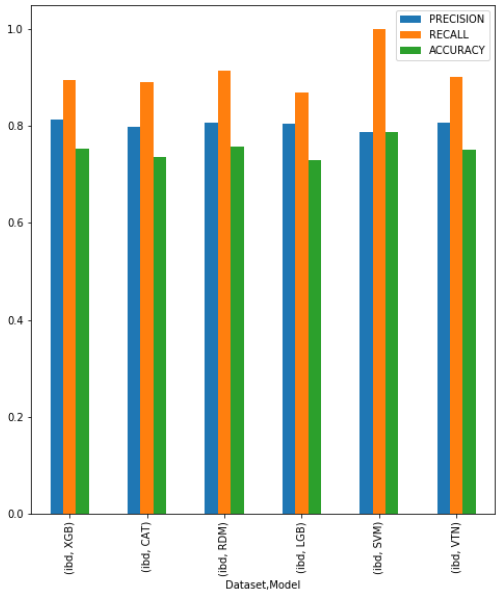


(d) Eosinophilic Digestive Disease

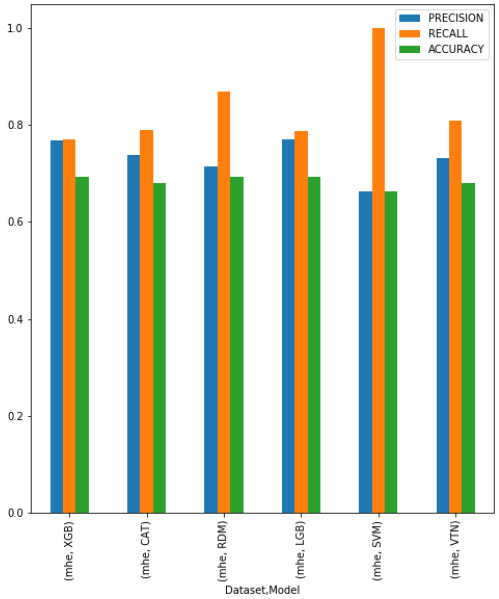
Figura 7.1: Analisi aggregato di patologie (1 di 3, dati grezzi)



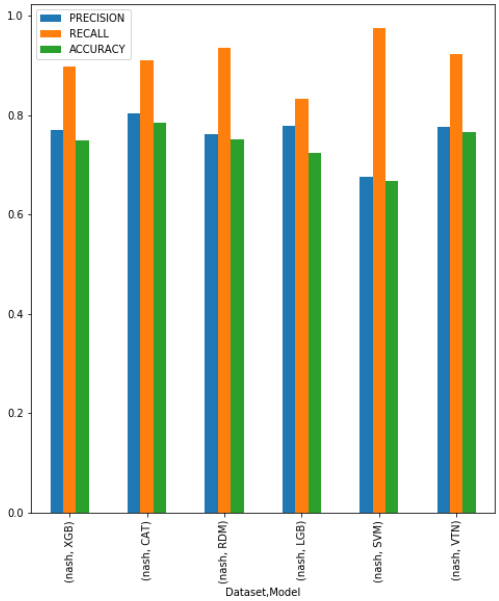
(a) HIV



(b) Inflammatory Bowel Diseases

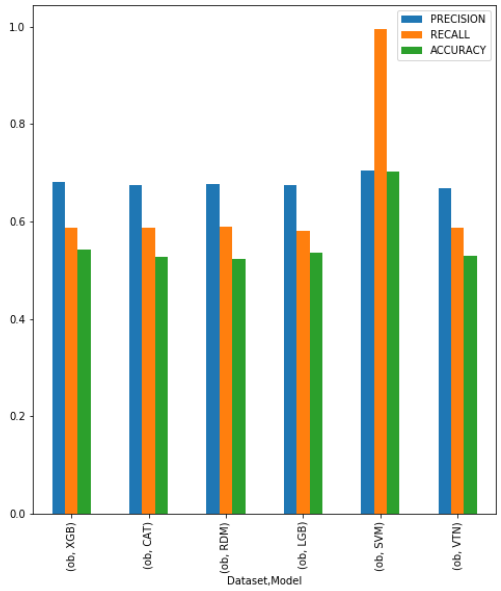


(c) Minimal Hepatic Encephalopathy

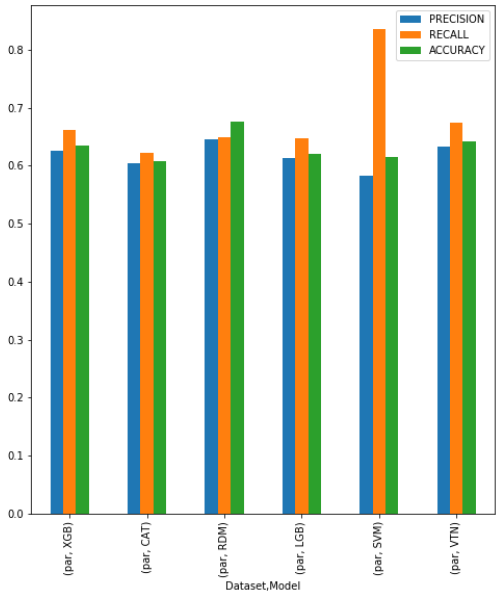


(d) NonAlcoholic SteatoHepatitis

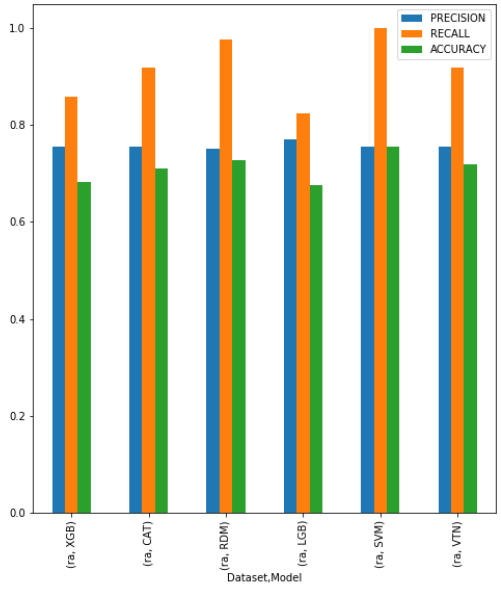
Figura 7.2: Analisi aggregato di patologie (2 di 3, dati grezzi)



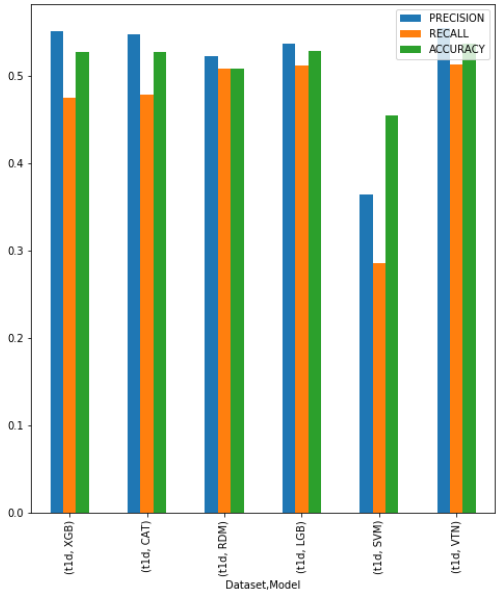
(a) Obesity



(b) Parkinson

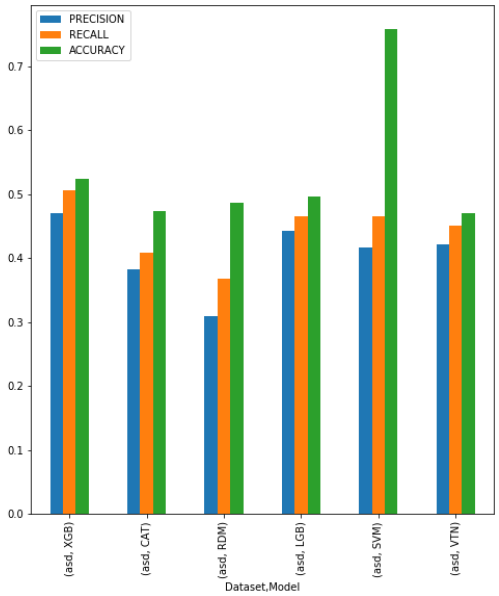


(c) Rheumatoid Arthritis

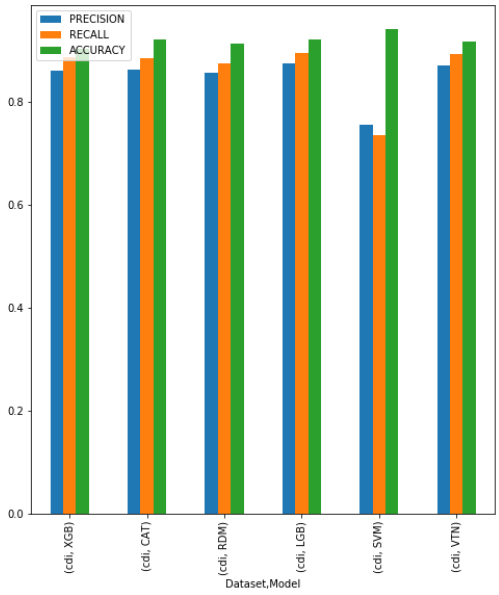


(d) Type 1 Diabetes

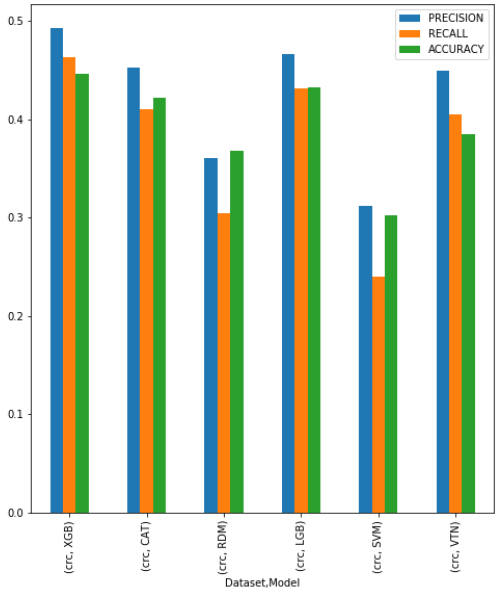
Figura 7.3: Analisi aggregato di patologie (3 di 3, dati grezzi)



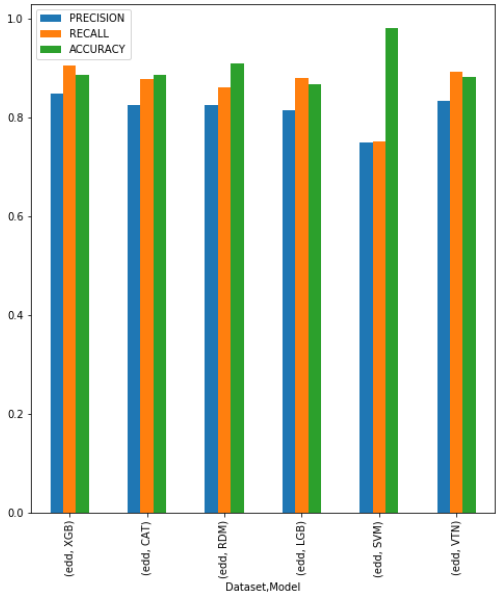
(a) Autistic Spectrum Disorder



(b) Clostridium Difficile Infection

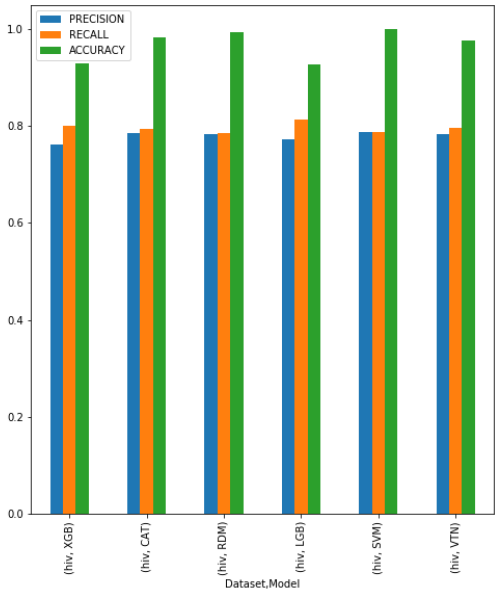


(c) ColoRectal Cancer

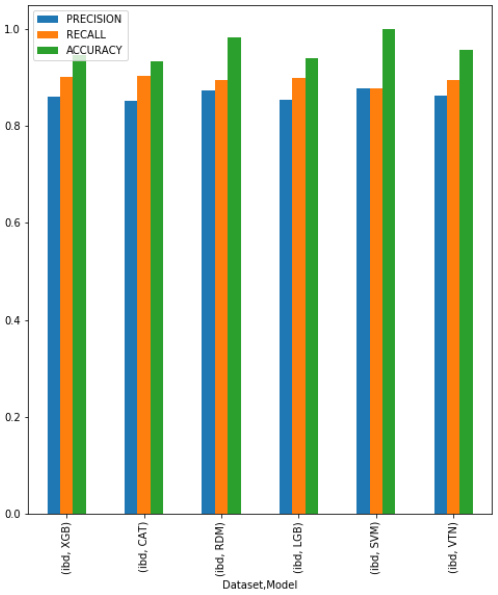


(d) Eosinophilic Digestive Disease

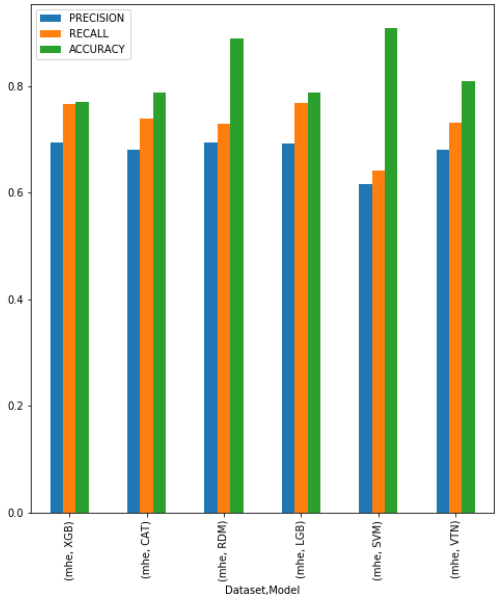
Figura 7.4: Analisi aggregato di patologie (1 di 3, con scaling)



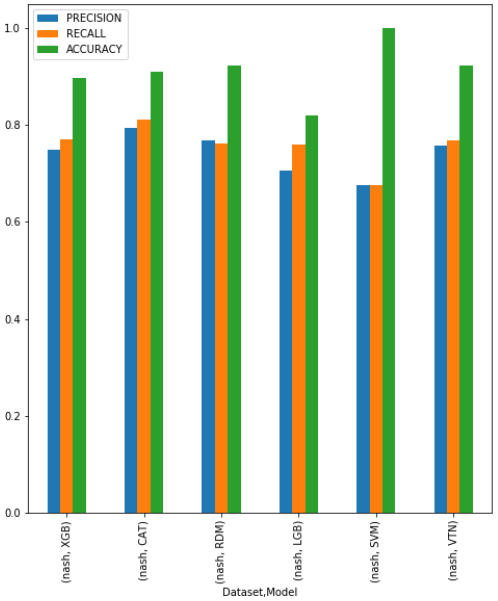
(a) HIV



(b) Inflammatory Bowel Diseases

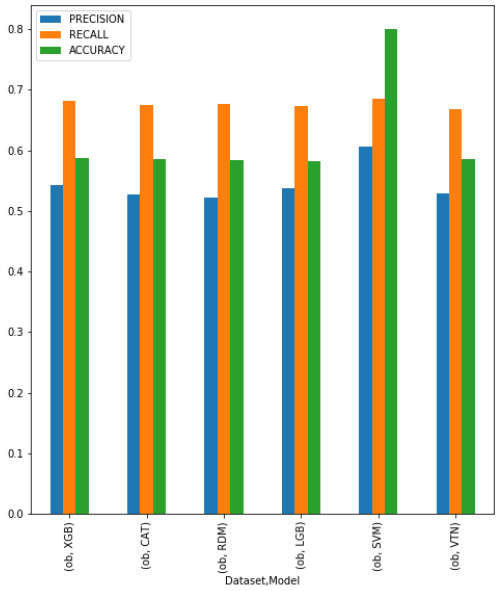


(c) Minimal Hepatic Encephalopathy

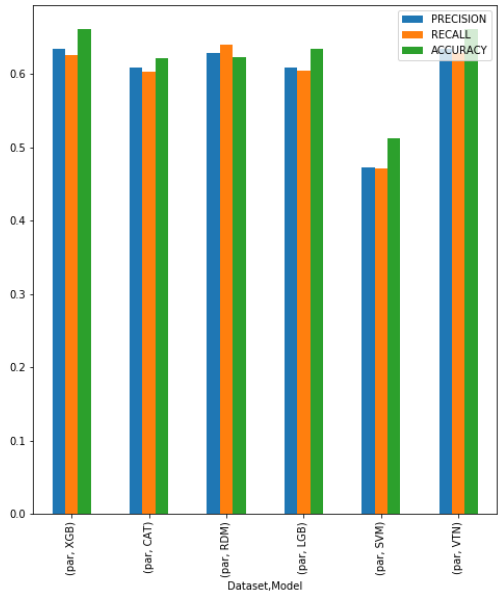


(d) NonAlcoholic SteatoHepatitis

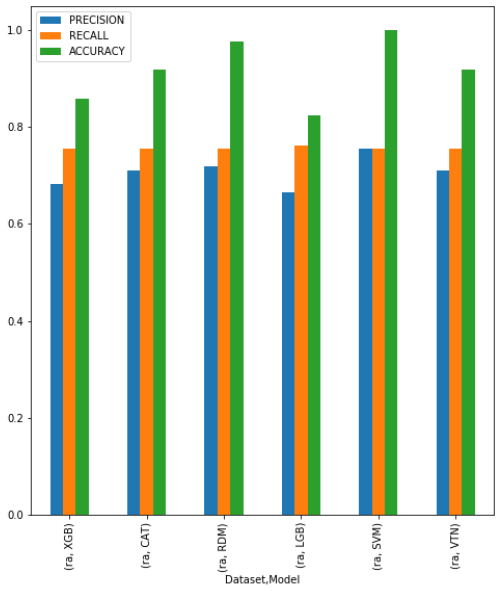
Figura 7.5: Analisi aggregato di patologie (2 di 3, con scaling)



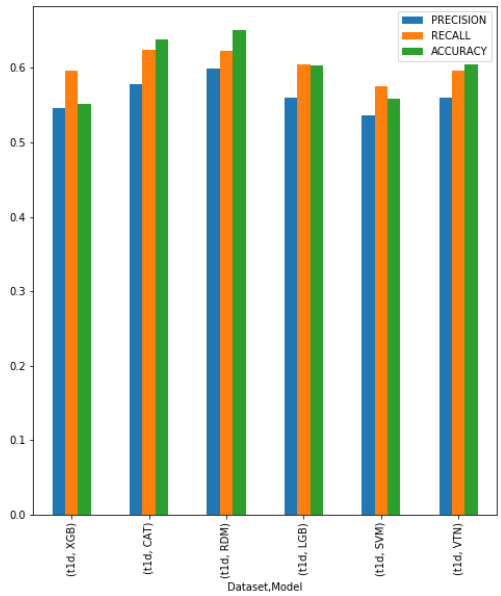
(a) Obesity



(b) Parkinson



(c) Rheumatoid Arthritis



(d) Type 1 Diabetes

Figura 7.6: Analisi aggregato di patologie (3 di 3, con scaling)

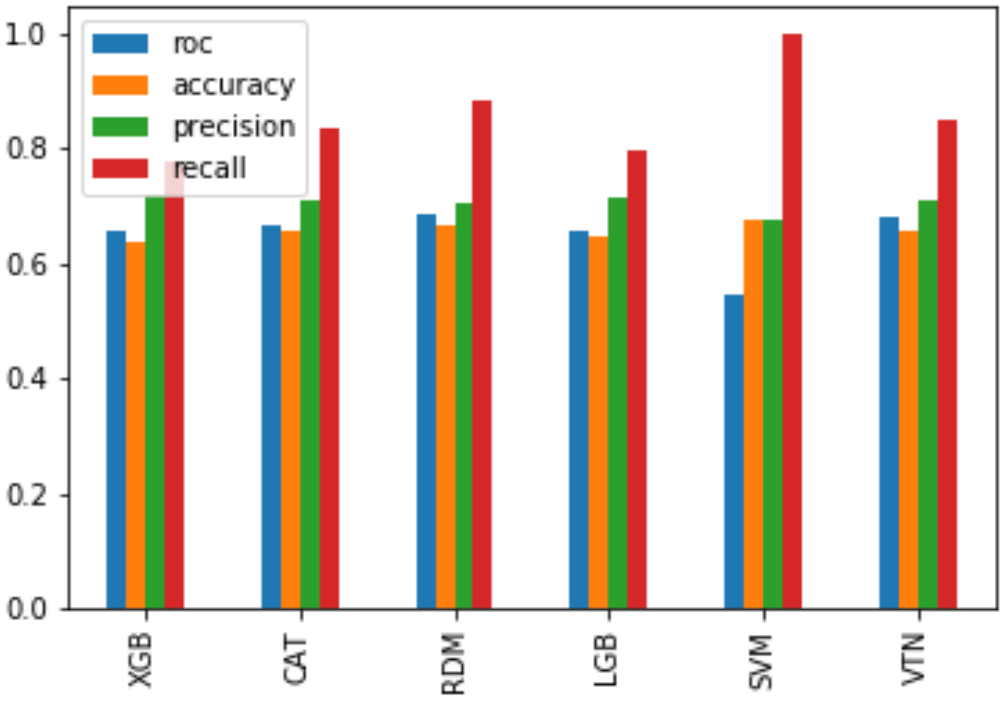


Figura 7.7: Confronto tra gli algoritmi addestrati sull'aggregato di tutti i dataset (dati grezzi)

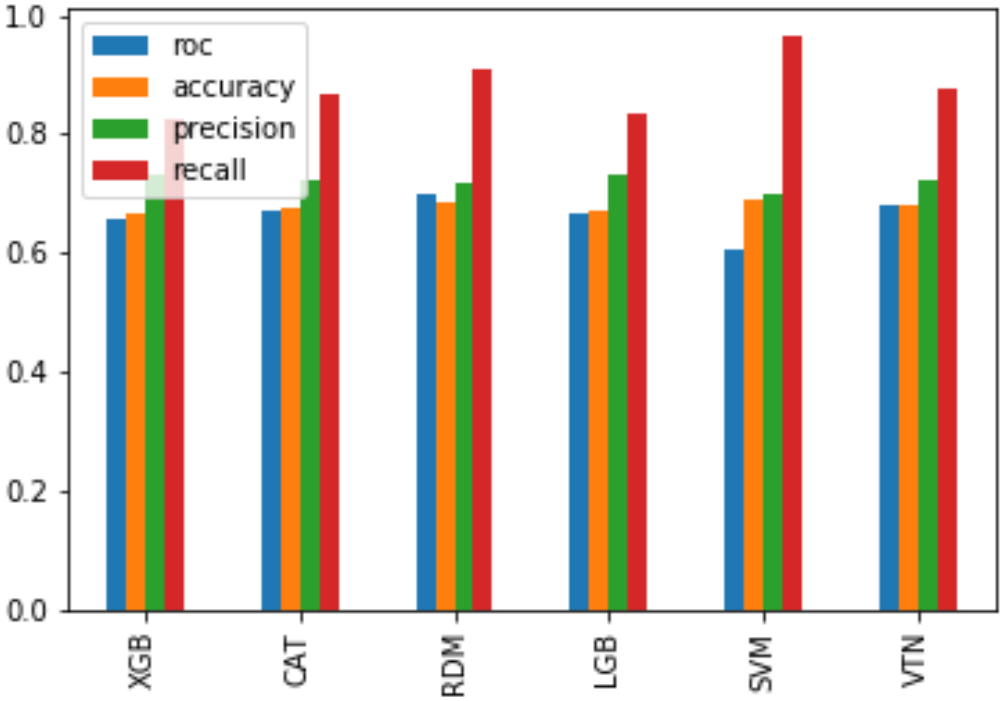
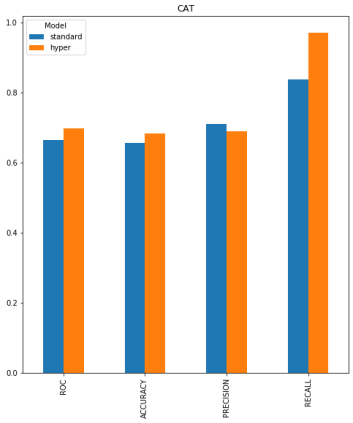
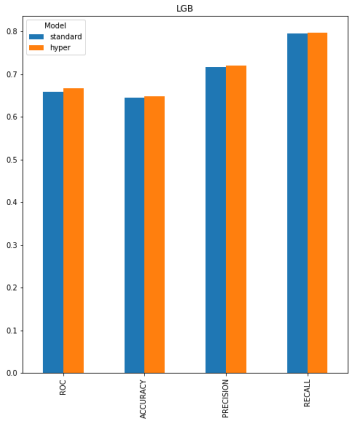


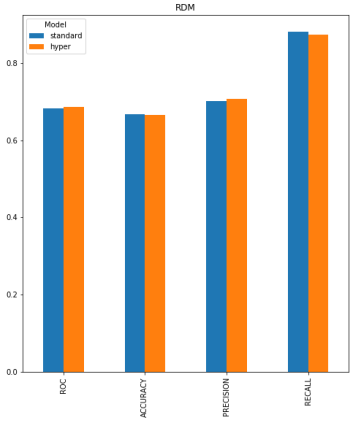
Figura 7.8: Confronto tra gli algoritmi addestrati sull'aggregato di tutti i dataset (dati scalati)



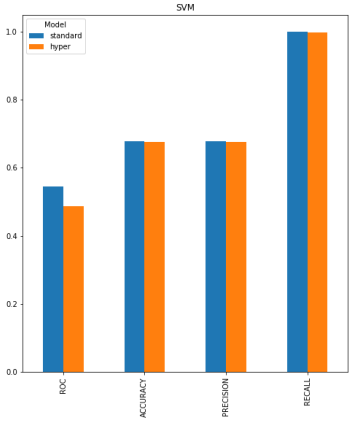
(a) CatBoost



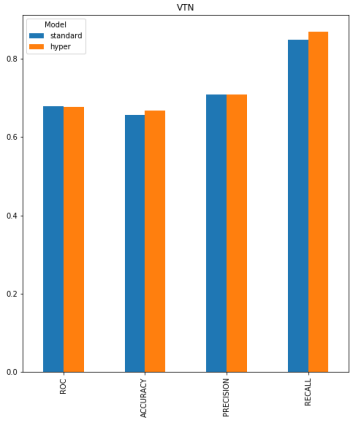
(b) LigthBoost



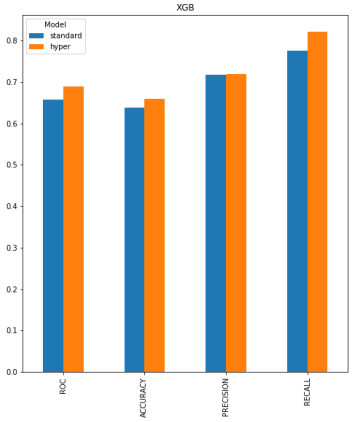
(c) Random Forest



(d) SVM

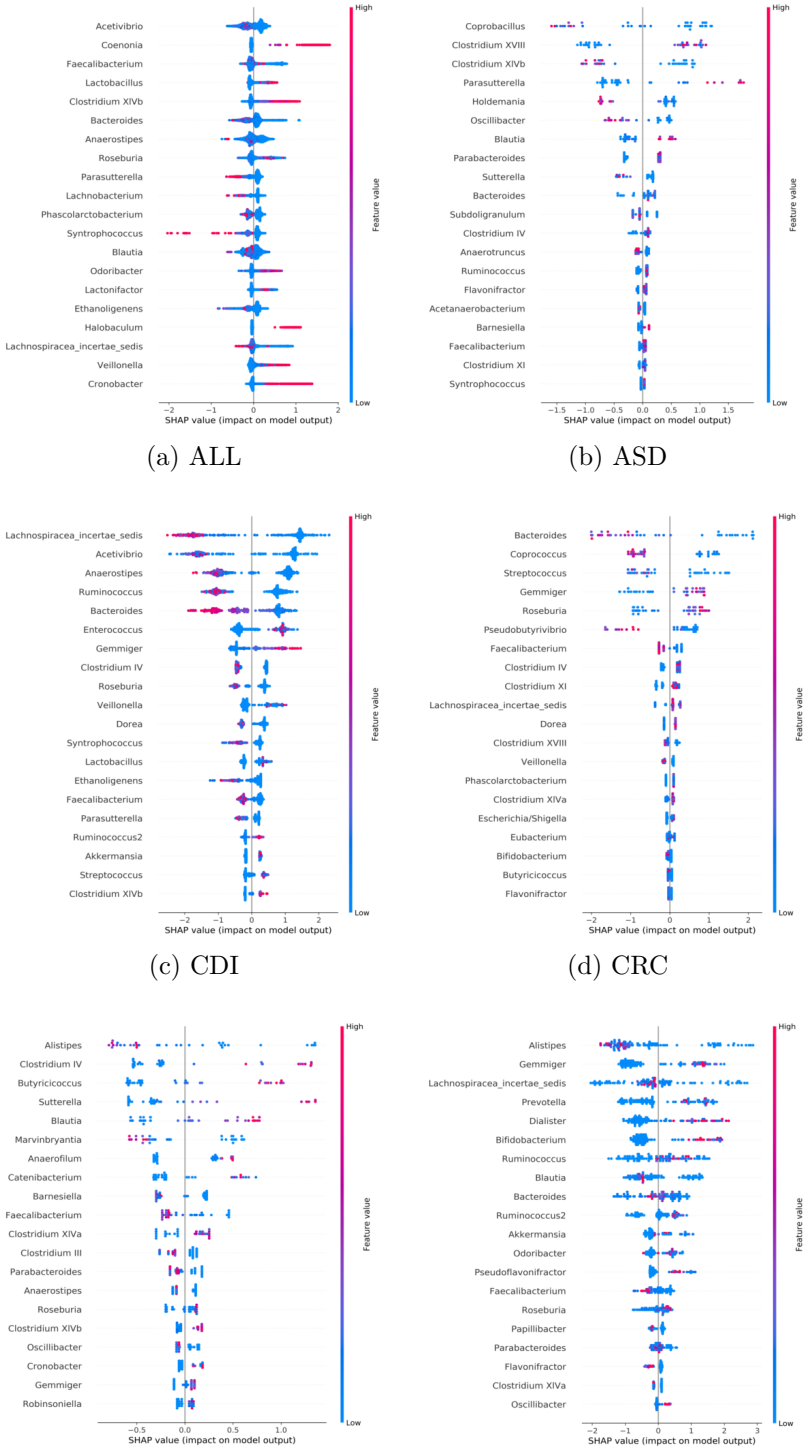


(e) Voting



(f) XGBoost

Figura 7.9: Confronto tra risultati base e ottenuti con tuning iperparametri



(a) ALL (b) ASD

(c) CDI (d) CRC

(e) HIV (f) T1D

Figura 7.10: Confronto tra i grafici SHAP ottenuti dal modello XGBoost

Capitolo 8

Esperimenti con Gene Ontology

8.1 Descrizione del problema

Come discusso in precedenza il problema può essere modellato come un task di classificazione supervisionato multilabel in cui devono essere identificate delle annotazioni non presenti ma che saranno inserite in una versione più aggiornata della matrice. Per ogni termine t viene costruito un classificatore binario con lo scopo di predire la presenza di t rispetto agli altri termini.

8.1.1 Perturbazione

Spesso non è disponibile una versione meno recente delle matrici delle annotazioni perché, per motivi di spazio, nelle banche dati vengono solitamente mantenute solo le ultime versioni aggiornate delle matrici delle annotazioni. Per ovviare a questo problema è possibile eseguire una perturbazione di una matrice a nostra disposizione per ottenere dei dati assimilabili ad una versione meno aggiornata delle annotazioni. Durante questo processo ogni annotazione presente viene eliminata con una probabilità p prefissata.

$$A_{\text{pert}}(g,t) = \begin{cases} 0 \text{ if } A(g,t) = 1 \wedge \text{random} \leq p \\ 1 \text{ if } A(g,t) = \wedge\text{random} \geq p \\ 0 \text{ if } A(g,t) = 0 \end{cases} \tag{8.1}$$

8.1.2 Approccio Cross-Domain

Dato che GO mette a disposizione un sistema di annotazioni che non dipende dalla specie è possibile addestrare un algoritmo di learning su un organismo le cui matrici hanno un numero maggiore di annotazioni e applicare il modello prodotto ad un organismo meno studiato. Questo approccio permette di arricchire le annotazioni anche di organismi poco studiati. In questo caso le predizioni sono limitate alle sole funzioni biologiche condivise

tra i due organismi. L'organismo più studiato, usato per l'addestramento, viene detto *source*, mentre l'organismo su cui vengono eseguite le predizioni viene detto *target*. Nel nostro caso l'essere umano svolgerà il ruolo di source.

8.1.3 Procedimento

1. Ottenere le matrici delle annotazioni A_{s1} e A_{t1} per sorgente e target disponibili ad un tempo T_1. In queste matrici devono essere considerate solamente le annotazioni affidabili.[1]

2. Selezionare un set di termini τ in comune tra i due organismi. In questo passaggio devono anche essere scartati i geni che presentano meno di M annotazioni.

3. Ottenere una seconda matrice delle annotazioni A_{t2} in un tempo T_2 tale che $T_2 > T_1$ per l'organismo target. Dato che questa matrice è stata ottenuta in un tempo successivo presenterà un numero maggiore di annotazioni di quella iniziale.

4. Ottenere una seconda matrice per l'organismo source in un tempo T_0 tale che $T_0 < T_1$. Nel caso questa matrice non sia disponibile è possibile ottenerla mediante il processo di perturbazione descritto in precedenza.

5. Per ogni termine addestrare un algoritmo supervisionato usando le matrici A_{s0} e A_{s1}

6. Applicare il modello prodotto ad ogni termine di A_{t1}

7. Valutare le predizioni ottenute al punto precedente confrontando il risultato con la matrice A_{t2}

8.2 True Path Rule

La regola del True Path [31] definisce il significato di ogni termine mediante una gerarchia multipla[2]. Il percorso che collega un termine a quelli superiori deve sempre essere rispettato, ovvero composto da 1. Questo comporta il fatto che se un gene viene annotato con un termine debba essere annotato anche gli ancestor di tale termine. I modelli che verranno addestrati però considerano ogni termine come indipendente; sarà quindi necessario intervenire sulla matrice prodotta in modo da garantire il rispetto di questa regola.

Un approccio per risolvere questo problema viene descritto in [13]

[1] Una annotazione viene considerata affidabile se presenta codici di annotazione diversi da quelli legati all'inferenza elettronica

[2] un termine può avere più padri

8.2.1 Parte 1

Per ogni nuova annotazione si aggiorna la sua probabilità sommandovi la media delle predizioni dei suoi ancestor[3]

$$p^H(g,t) = \frac{\frac{\sum_{t_a \in \text{ancestors}(t)} p(g,t_a)}{|\text{ancestors}(t)|} + p(g,t)}{2} \tag{8.2}$$

8.2.2 Parte 2

La seconda parte serve a correggere le annotazioni in modo iterativo partendo dalle foglie della gerarchia.

$$l(g,t) = \max\{p^H(g,t), \max_{t_c \in \text{children}(t)}\{p^H(g,t_c)\}\} \tag{8.3}$$

In questo caso per ogni termine t la probabilità della presenza di una annotazione con un gene g diviene pari al massimo della probabilità delle annotazioni associate ai suoi termini figli.

8.3 Metodi Ensemble

Anche in questo caso possono essere applicati metodi Ensemble addestrando l'algoritmo di learning su varie versioni delle matrici perturbate. In questi test verranno realizzate 5 matrici delle predizioni per ogni esperimento.

8.3.1 Averaging

La $p(g,t)$ che il gene g venga annotato con il termine t viene posta pari alla media delle probabilità ottenute dai vari modelli.

$$l(g,t) = \sum_{i=1}^{n} \frac{l_i(g,t)}{n} \tag{8.4}$$

Dopo avere aggiornato le predizioni in questo modo vengono considerate 1 le annotazioni che superano la soglia di confidenza prefissata.

8.3.2 Voting x of n

Una predizione viene considerata valida se almeno x degli n modelli l'hanno predetta con una soglia di confidenza superiore a quella fissata. É possibile notare che in questa

[3] i termine che si trovano più in alto nella gerarchia, quindi più generici

definizione $x = 1$ corrisponde all'unione di tutte le predizioni,[4] mentre $x = n$ rappresenta l'intersezione delle predizioni[5].

8.4 Risultati

[4]Di conseguenza verranno prodotte un alto numero di predizioni ma non è probabile che le metriche di valutazione ne risentano

[5]Questo comporto ad ottenere meno predizioni ma è probabile che siano più aderenti alla realtà

	N	prec_transition	MicroF1	MacroF1	RankLoss	PrecMicro	PrecMacro	RecMicro	RecMacro
SM	433	17.32%	77.86%	75.25%	0.30138	90.79%	88.00%	68.16%	68.56%
AVG	323	17.64%	77.96%	75.49%	0.30252	9121.%	88.91%	68.07%	68.47%
1_5	822	13.62%	78.03%	76.46%	0.29191	89.31%	86.08%	69.28%	71.27%
2_5	470	15.31%	78.16%	76.25%	0.29689	90.65%	87.93%	68.70%	69.92%
3_5	344	17.44%	78.02%	75.70%	0.30127	91.14%	88.79%	68.20%	68.81%
4_5	252	18.65%	77.81%	74.91%	0.30579	91.50%	89.04%	67.68%	67.61%
5_5	166	20.48%	77.39%	73.52%	0.31245	91.85%	89.37%	66.87	

Tabella 8.1: Risultati BT

	N	prec_transition	MicroF1	MacroF1	RankLoss	PrecMicro	PrecMacro	RecMicro	RecMacro
SM	165	12.72%	75.01%	71.54%	0.3342	90.57%	8615.%	64.02%	64.23%
AVG	127	14.96%	75.06%	71.49%	0.3356	90.95%	86.56%	63.90%	63.95%
1_5	354	14.12%	75.11%	72.26%	0.3274	88.89%	84.14%	65.03%	66.16%
2_5	211	15.16%	75.18%	71.97%	0.3318	90.19%	85.84%	64.45%	64.96%
3_5	140	15%	75.07%	71.53%	0.3352	90.84%	86.63%	63.97%	64.00%
4_5	105	12.38%	74.84%	70.92%	0.3397	91.11%	87.11%	63.51%	63.15%
5_5	72	9.72%	74.46%	69.66%	0.3466	91.43%	85.88%	62.81%	61.59%

Tabella 8.2: Risultati GG

	N	prec_transition	MicroF1	MacroF1	RankLoss	PrecMicro	PrecMacro	RecMicro	RecMacro
SM	7665	21.94%	59.95%	47.21%	0.4763	92.34%	83.69%	44.383%	36.35%
AVG	5684	22.83%	59.80%	47.07%	0.4806	92.83%	84.89%	44.114%	36.03%
1_5	15351	19.72%	60.50%	49.75%	0.4672	90.46%	81.07%	45.447%	39.55%
2_5	9085	21.14%	60.29%	48.66%	0.4723	92.01%	83.79%	44.842%	37.83%
3_5	6391	21.99%	60.05%	47.47%	0.4760	92.67%	84.86%	44.415%	36.48%
4_5	4492	23.08%	59.53%	45.94%	0.4837	93.10%	84.72%	43.760%	34.93%
5_5	2825	23.75%	58.94%	43.55%	0.4900	93.48%	84.12%	43.045%	32.66%

Tabella 8.3: Risultati MM

	N	prec_transition	MicroF1	MacroF1	RankLoss	PrecMicro	PrecMacro	RecMicro	RecMacro
SM	1667	6.59%	72.16	67.57%	0.3417	73.82%	71,72%	70.57%	70.44%
AVG	1256	7.88%	72.43	67.99%	0.3410	74.35%	73.05%	70.60%	70.45%
1_5	3251	6.30%	72.03	69.24%	0.3310	72.17%	70.69%	71.90%	74.56%
2_5	1959	6.68%	72.37	68.85%	0.3359	73.57%	72.28%	71.22%	72.34%
3_5	1352	7.54%	72.42	68.05%	0.3401	74.22%	72.66%	70.71%	70.82%
4_5	985	8.52%	72.33	67.02%	0.3449	74.63%	72.71%	70.17%	69.02%
5_5	675	6.81%	71.96	65.19%	0.3533	74.88%	72.13%	69.26%	66.44%

Tabella 8.4: Risultati DD

8.5 Esperimenti con gestione sbilanciamento classi

Una delle problematiche che possono emergere durante la trattazione delle matrici GO è la loro sparsità. [6]. Una situazione di questo tipo potrebbe portare l'algoritmo a prevedere ogni istanza come membro della classe maggiormente rappresentata dato che questa scelta massimizzerebbe comunque l'accuratezza, la metrica di default per i task di classificazione degli algoritmi usati in questo contesto.

All'interno del paper di riferimento non erano inseriti riferimenti agli accorgimenti presi, si è quindi deciso di procedere con delle tecniche standard applicabili in questi casi.

8.5.1 SMOTE (Synthetic Minority Over-Sampling Technique)

Uno dei possibili modi per affrontare questo problema è sfruttare tecniche di oversampling: meccanismi che permettono di aggiungere istanze alle classi sottorappresentate in modo da ripristinare il bilanciamento.

SMOTE raggiunge questo scopo generando nuove istanze partendo da quelle già presenti.

Per ognuna delle istanze della classe sottorappresentata vengono determinati K vicini della medesima classe (k-nearest-neighbors). A questo punto vengono generati dei punti artificiali sulle linee di congiunzione tra le istanze e i vicini.

8.5.2 Estensioni di SMOTE

Esistono alcune variazioni rispetto alla versione standard, ad esempio:

- BorderlineSMOTE: le istanze vengono generate considerando solamente i punti "borderline", ovvero difficilmente distinguibili dalle istanze delle altre classi
- SVMSMOTE: sfrutta le SVM per generare dei punti vicino al bordo che separa le due istanze

8.5.3 Gestione della funzione obiettivo

XGBoost permette di personalizzare la funzione obiettivo che, per i task di classificazione, è l'accuratezza.

Alternative utili in questo caso sono:

- AUC: Area Under The Curve, utilizza l'area al di sotto della curva del rapporto TPR/FPR per determinare la bontà della classificazione
- AUCPR: come sopra, ma viene utilizzata l'area al di sotto della curva ottenuta dal rapporto tra precision e recall

[6] ovvero la scarsa presenza di 1 rispetto alla dimensione

8.5.4 Risultati

Gli esperimenti sono stati eseguiti su tutti gli organismi (tranne MM), sfruttando XGB con 300 estimatori. I punti sono stati generati usando l'implementazione classica di SMOTE e utilizzando AUCPR come metrica di valutazione.

	N	prec_transition	MicroF1	MacroF1	RankLoss	PrecMicro	PrecMacro	RecMicro	RecMacro
SM	4750	5.17%	71.42%	56.46%	0.2469	75.30%	63.55%	67.92%	55.29%
AVG	1638	7.87%	74.87%	58.61%	0.2537	86.08%	76.98%	66.25%	51.03%
1_5	13376	3.58%	64.03%	51.54%	0.2220	57.08%	48.19%	72.92%	66.61%
2_5	6080	4.85%	71.09%	59.10%	0.2310	71.92%	64.08%	70.28%	60.85%
3_5	3015	6.06%	73.95%	60.72%	0.2435	81.03%	73.43%	68.02%	55.62%
4_5	1573	7.31%	74.35%	57.99%	0.2602	86.21%	77.48%	65.35%	50.10%
5_5	743	10.20%	73.27%	52.79%	0.2835	89.67%	77.94%	61.94%	43.27%

Tabella 8.5: Risultati BT con SMOTE e AUCPR

	N	prec_transition	MicroF1	MacroF1	RankLoss	PrecMicro	PrecMacro	RecMicro	RecMacro
SM	1335	14.30%	74.11%	63.41%	0.2464	81.42%	73.30%	68.00%	59.43%
AVG	795	17.86%	75.21%	63.89%	0.2487	85.78%	78.84%	66.96%	57.15%
1_5	3685	8.928%	69.24%	60.20%	0.2329	67.00%	61.08%	71.64%	67.05%
2_5	1673	12.49%	73.94%	64.80%	0.2368	78.99%	72.90%	69.50%	62.95%
3_5	924	16.34%	75.48%	65.57%	0.2428	84.72%	78.29%	68.05%	59.75%
4_5	642	19.00%	75.22%	63.27%	0.2542	87.03%	79.46%	66.23%	55.71%
5_5	411	20.92%	74.23%	59.59%	0.2713	89.08%	81.16%	63.62%	50.52%

Tabella 8.6: Risultati GG con SMOTE e AUCPR

	N	prec_transition	MicroF1	MacroF1	RankLoss	PrecMicro	PrecMacro	RecMicro	RecMacro
SM	7864	5.03%	68.67%	47.75%	0.2866	67.17%	51.43%	70.23%	53.34%
AVG	4117	7.52%	69.40%	46.44%	0.3021	70.84%	56.10%	68.01%	47.44%
1_5	20179	3.78%	65.64%	46.75%	0.2528	57.89%	40.47%	75.78%	67.27%
2_5	9203	5.16%	69.22%	51.00%	0.2714	66.22%	52.11%	72.50%	59.06%
3_5	5416	6.68%	69.84%	49.93%	0.2876	69.71%	57.35%	69.97%	52.63%
4_5	3500	7.54%	69.23%	46.53%	0.3084	71.48%	57.74%	67.12%	46.60%
5_5	2136	8.84%	68.08%	40.96%	0.3335	72.91%	55.90%	63.85%	39.44%

Tabella 8.7: Risultati DD con SMOTE e AUCPR

	Tesi		Letteratura	
Target	**N**	**P**	**N**	**P**
MM	2825	23.75%	6048	57.3%
BT	743	10.20%	234	53.8%
GG	411	20.92%	136	39%
DD	2136	8.84%	765	67.5%

Tabella 8.8: Comparazione tra risultati tesi e letteratura

8.6 Considerazioni finali

Sfortunatamente i risultati ottenuti all'interno di questi esperimenti non hanno permesso di ottenere risultati in linea con quelli dell'articolo di riferimento. La tabella 8.8 offre un confronto tra i diversi risultati ottenuti.

Capitolo 9

Implementazione degli Esperimenti con MicrobiomeHD

9.1 Linguaggio e librerie

Il progetto è stato sviluppato in Python, un linguaggio di programmazione ad oggetti molto usato nell'ambito della data science, ma anche nello sviluppo di applicazioni web. [30] [1]

Oltre alle librerie indicate nel capitolo 4 sono state usate:

- Pandas: analisi e manipolazione di dati
- NumPy: estensioni matematiche con supporto nativo per operazioni algebriche
- OS: permette di avere accesso a funzionalità del sistema operativo come, ad esempio, ottenere la lista di file in una cartella o concatenare path
- Json: libreria per la manipolazione di file nel formato json
- Matplotlib e Seaborn: realizzazione dei grafici
- BioPython: necessario per leggere i file fasta in fase di preprocessing

9.2 Elaborazione dati

9.2.1 Conversione dei dati MicrobiomeHD in dataframe pandas

I dati provenienti da MicrobiomeHD [16] sono stati convertiti in dataframe pandas prima di procedere con le successive fasi di lavoro. Questa scelta è motivata dal fatto che i dati sono suddivisi all'interno di diversi file pertanto è conveniente inserire tutto all'interno di un solo punto, in modo da facilitare le successive analisi.

```
dataset_folder = 'datasets_raw'
output_folder = 'elab_csv'

import os
from libs import gut_preprocessing as gp
import pandas as pd

datasets_name = [f for f in in sorted(os.listdir(dataset_folder) if '
    _results' in f]

for dataset_name in datasets_name:
    name = dataset_name.split("_results")[0]
    print("Working on: {}".format(name))
    metadata = gp.read_metadata(name, dataset_folder)
    otu_table = gp.read_raw_OTU_table(name, dataset_folder)
    rdp_metadata = gp.read_rdp_metadata(name, dataset_folder)
    grouped = gp.otu_groupby(otu_table, rdp_metadata, 'genus')
    recap_transpose = gp.otu_table_recap(grouped, 'genus')
    final = gp.add_health_metadata(recap_transpose, metadata)
    final.to_csv(os.path.join(output_folder, "{}_otu_metadata.csv".format(
    name)))
```

La libreria *gut_preprocessing* è stata realizzata con lo scopo di facilitare il preprocessing dei dati di Microbiome; si compone di funzioni di utilità che permettono di estrapolare le informazioni necessarie dagli specifici file che le contengono, partendo dal nome del dataset.

9.2.2 Aggregazione a livello di patologia

L'aggregazione dei dati a livello di patologia è stata portata a termine sfruttando il metodo *concat* messo a disposizione da pandas che consente di concatenare lungo un asse due dataframe.

```
import os
import pandas as pd

out_folder = '../aggregate_disease'

pat_list = set([name.split('_')[0] for name in file_list if ".csv" in name
    ])

for pato in pat_list:
    print("workin with: {}".format(pato))
    dataset_list = [dataset for dataset in file_list if "{}_".format(pato)
     in dataset]
    dataset_union = pd.DataFrame()
    for dataset in dataset_list:
```

```
        d = pd.read_csv(os.path.join(dataset_folder_input, dataset),
    index_col = 0)
        dataset_union = pd.concat([dataset_union, d], axis = 1, sort =
    True)
        dataset_union['DiseaseState'] = dataset_union['DiseaseState'].
    fillna('')
        dataset_union = dataset_union.groupby(dataset_union.columns, axis
    = 1).sum()
     dataset_union.to_csv(os.path.join(out_folder, "{}_aggregate.csv".
    format(pato)))
```

9.2.3 Aggregazione completa

La stessa procedura è stata usata per eseguire una aggregazione sull'intero insieme dei dataset.

9.3 Automazione

Visto che fin dall'inizio si è pensato di utilizzare un insieme variegato di modelli si è tenuto conto da subito della necessità di automatizzare il più possibile le indagini svolte. A questo scopo è stata prodotta la libreria *gut_model* che contiene diverse funzioni che permettono di automatizzare le analisi su una serie di dataset estraendo in modo automatico le metriche di riferimento. Di seguito è riportata la funzione di automazione che consente di eseguire le analisi in modo automatico su una lista di dataset.

```
def evalutation_H_notH_CV_extra(data_list, model_list, model_names, seed =
    42):

  result = {}
  for model, model_name in  zip(model_list, model_names):
    result[model_name] = {}
    print(model_name)
    for data_source in data_list:
      print(data_source)
      dataset = pd.read_csv(data_source, index_col = 0)
      dataset.dropna(inplace = True)
      X = dataset.drop('DiseaseState', axis = 1)
      y = dataset['DiseaseState']
      y = (y != 'H').astype('int8')
      result[model_name][data_source] = {}
      if (y.sum() != len(y)):
        result[model_name][data_source]['roc'] = cross_val_score(model, X,
    y, cv = 5, scoring='roc_auc').tolist()
        result[model_name][data_source]['accuracy'] = cross_val_score(
   model, X, y, cv = 5, scoring='accuracy').tolist()
```

```
18          result[model_name][data_source]['precision'] = cross_val_score(
    model, X, y, cv = 5, scoring='precision').tolist()
19          result[model_name][data_source]['recall'] = cross_val_score(model,
     X, y, cv = 5, scoring='recall').tolist()
20        else:
21          result[model_name][data_source]['roc'] = np.nan
22          result[model_name][data_source]['accuracy'] =  np.nan
23          result[model_name][data_source]['precision'] =  np.nan
24          result[model_name][data_source]['recall'] = np.nan
25    return result
```

La funzione in oggetto ritorna un dizionario che, per ogni modello e per ogni dataset, contiene le metriche ottenute mediante una valutazione con cross fold validation. I dati vengono salvati come liste in modo da potere calcolare valore medio e varianza

9.4 Ricerca iperparametri

Visto il numero di modelli elevato si è deciso di non usare una GridSearch ma una RandomSearch. La RandomSearch ricerca gli iperparametri testando solo una parte delle possibili combinazioni, in questo modo è stato possibile eseguire i test in un minore lasso di tempo. Per migliorare i risultati ottenuti sono stati eseguiti più passaggi di ricerca randomica, di volta in volta inserendo parametri simili ai migliori trovati.

Capitolo 10

Implementazione degli Esperimenti con Gene Ontology

10.1 Perturbazione delle matrici

La perturbazione viene ottenuta eliminando il 10% delle annotazioni dalla matrice iniziale. Questa operazione viene implementata in modo efficiente utilizzando la funzione *randr* del modulo *random* messo a disposizione dalla libreria matematica di python. Grazie alla sopracitata funzione viene creata una matrice della stessa dimensione di quella iniziale. Questa contiene una distribuzione uniforme di probabilità[1] che verrà poi moltiplicata per la matrice iniziale. Dopo questa operazione si manterranno solamente le celle che contengono un valore maggiore di 0.9[2]

```
1  return (df.multiply(np.random.rand(df.shape[0],df.shape[1])) > threshold
    ).astype('int8')
```

10.2 Predizioni

Il seguente frammento di codice permette di eseguire una predizione su un organismo partendo da una matrice perturbata. Sono necessari i seguenti argomenti:

1. identificatore organismo
2. identificatore della matrice perturbata
3. testo libero usato per identificare il risultato dell'esperimento

[1] ogni cella conterrà quindi un valore compreso tra 0 ed 1

[2] In questo modo le celle che originariamente erano 0 rimarranno tali mentre, tra gli 1, solo il 90% verrà mantenuto

Si ricorda che l'algoritmo è stato addestrato a ricostruire una matrice umana del 2009 perturbata per poi predire le annotazioni sulla matrice del 2009 integra di un altro organismo. I risultati sono valutati sulla matrice del 2013 di tale organismo.

```
As1 = pd.read_csv(os.path.join(domeniconi_path, 'hs_2009_unfolding.csv'), index_col = 0)
As0 = pd.read_csv(os.path.join(pert_path, 'HS09_{}.csv'.format(str(sys.argv[2]))), index_col = 0)

At1 = pd.read_csv(os.path.join(domeniconi_path, '{}_2009_unfolding.csv'.format(str(sys.argv[1]))), index_col  = 0)
print('fine lettura dati')

result = pd.DataFrame(index = At1.index)
result.sort_index(inplace = True)

As1.sort_index(axis=1, inplace=True)
As0.sort_index(axis=1, inplace=True)
At1.sort_index(axis=1, inplace=True)


As1.sort_index(inplace=True)
As0.sort_index(inplace=True)
At1.sort_index(inplace=True)


result.sort_index(inplace = True)

common_function = prepro.get_common_functions(As1, At1)
common_function = common_function.sort_values()

As0_common = As0[common_function]
usable_gene = prepro.get_usable_genes(As0_common)
As0_common_filtered = As0_common.loc[usable_gene]
As0_common_filtered.sort_index(inplace = True)
As0_common_filtered.sort_index(axis = 1, inplace = True)



As1_filtered = As1.loc[usable_gene]
As1_filtered.sort_index(inplace = True)
As1_filtered.sort_index(axis=1, inplace=True)

print('inizio generazione modelli')

for function in common_function:
    X = As0_common_filtered[common_function].drop(function, axis = 1)
    X.sort_index(inplace = True)
    X.sort_index(axis = 1, inplace = True)
```

```
    y = As1_filtered[function]
    y.sort_index(inplace = True)

    model = ClasifierModel()
    model.fit(X, y)
    prediction = model.predict_proba(At1[common_function].drop(function,
    axis = 1))[:, 1]
    result[function] = prediction

result.to_csv('{}_09_{}_{}.csv'.format(sys.argv[1], sys.argv[2], sys.argv
    [3]))
```

Questo procedimento deve essere applicato ad ogni matrice perturbata in modo da ottenere diverse versioni delle predizioni; in questo modo sarà possibile applicare successivamente procedimenti ensemble per ottenere risultati migliori. Nelle celle non viene inserito il risultato della classificazione ma la probabilità di appartenere alla classe 1, quella delle annotazioni presenti.

10.3 Applicazione del true path

Dopo avere eseguito le predizioni è necessario applicare ai risultai predetti la regola del true path; in questo modo si assicura che le varie annotazioni rispettino la gerarchia GO.

```
def true_path(predictions, ancestors, descendant):
  result = predictions.copy()
  result = __part1(result, ancestors)
  result = __part2(result, descendant)
  return result

def __part1(prediction, ancestors):
  result = prediction.copy()
  for f in ancestors.keys():
    result[f] = ((prediction[ancestors[f]].mean(axis = 1)) + prediction[f
    ])/2
  return result

def __part2(prediction, descendant):
  result = prediction.copy()
  for f in descendant.keys():
    result[f] = prediction[descendant[f]].values.max(axis = 1)
  return result
```

10.4 Ensemble

In questo contesto il voting non era direttamente applicabile sfruttando i metodi messi a disposizione dalle librerie; per questo è stato necessario procedere con una implementazione personalizzata.

10.4.1 Media

```
values = np.zeros((target_2009.shape[0], len(common_function)))
for df in datasets_tp_applied:
    values += df.values
values /= len(datasets_tp_applied)
values = (values > threshold).astype('int8')
avg = pd.DataFrame(values, index = target_2009.index, columns =
   common_function, dtype='int8')
```

10.4.2 Voting

Il voting viene implementato sommando le varie matrici di predizioni. I questo modo ogni cella conterrà il numero di modelli che, per essa, hanno predetto che dovrebbe essere un 1 nella versione successiva. Andando ad usare il numero di voti necessari come soglia di confidenza è possibile decidere quali celle escludere.

```
values = np.zeros((target_2009.shape[0], len(common_function)))
for df in datasets_tp_applied:
    values += (df.values > threshold).astype('int8')
voting = pd.DataFrame(values, index = target_2009.index, columns =
   common_function, dtype='int8')
```

10.5 SMOTE

```
for function in common_function:
    X = As0_common_filtered[common_function].drop(function, axis = 1)
    y = As1_filtered[function]
    sm = SMOTE(k_neighbors = 3)
    X, y = sm.fit_sample(X, y)
    model = XGBClassifier(n_estimators=300, tree_method = 'gpu_hist',
   eval_metric = 'aucpr')
    model.fit(X, y)
    prediction = model.predict_proba(At1[common_function].drop(function,
   axis = 1).values)[:, 1]
    result[function] = prediction
```

Alle righe 3 e 4 viene creata una istanza di SMOTE e utilizzata per generare una nuova coppia X, y da utilizzare per l'addestramento dell'algoritmo che, nella riga 6, verrà addestrato utilizzando AUCPR come metrica di valutazione.

10.6 Valutazioni

La seguente routine ha lo scopo di calcolare le metriche di riferimento scelte per valutare i risultati ottenuti.

```
#Con TP
ancestors_2009 = prepro.load_ancestors_2009()
descendant_2009 = prepro.load_descendant_2009()

for organism in organism_list:
    print(organism)
    organism_lower = organism.lower()

    org_09 = pd.read_csv('/home/montelli/Documents/tesi/esperimenti_ensemble_paper/data_creation/unfolding/{}_2009_unfolding.csv'.format(organism_lower), index_col = 0)
    org_09 = org_09.loc[~org_09.index.duplicated(keep='first')]
    org_13 = pd.read_csv('/home/montelli/Documents/tesi/esperimenti_ensemble_paper/data_creation/unfolding/{}_2013_unfolding.csv'.format(organism_lower), index_col = 0)
    org_13 = org_13.loc[~org_13.index.duplicated(keep='first')]
    pred_single = pd.read_csv('{}/{}_09_1_smote_auc.csv'.format(organism, organism_lower), index_col = 0)
    pred_single = pred_single.loc[~pred_single.index.duplicated(keep='first')]

    usable_functions = pred_single.columns.intersection(org_09.columns).intersection(org_13.columns)
    usable_functions.drop_duplicates()
    usable_genes = pred_single.index.intersection(org_09.index).intersection(org_13.index)
    usable_genes.drop_duplicates()

    usable_functions = usable_functions.to_list()
    usable_functions.sort()
    usable_genes = usable_genes.to_list()
    usable_genes.sort()

    org_13 = org_13.reindex(index = usable_genes, columns = usable_functions)
    org_09 = org_09.reindex(index = usable_genes, columns = usable_functions)

    usable_anc_2009 = prepro.get_usable_ancestors(ancestors_2009, usable_functions)
```

```
30      usable_des_2009 = prepro.get_usable_descendant(descendant_2009,
    usable_functions)
31
32      pred_single = pred_single.reindex(index = usable_genes, columns =
    usable_functions)
33      pred_single = models.true_path(pred_single, usable_anc_2009,
    usable_des_2009)
34
35      df = pd.DataFrame(index = ['SM', 'AVG', '1_5', '2_5', '3_5', '4_5', '5
    _5'], columns = ['N', 'prec_transition', 'MicroF1', 'MacroF1', '
    RankLoss', 'PrecMicro', 'PrecMacro', 'RecMicro', 'RecMacro'])
36
37      voting = pd.DataFrame(0, index = usable_genes, columns =
    usable_functions)
38      average = pd.DataFrame(0, index = usable_genes, columns =
    usable_functions)
39
40      for file in [f for f in os.listdir(organism) if '.csv' in f]:
41          pred = pd.read_csv('{}/{}'.format(organism, file), index_col = 0)
42          pred = pred.loc[~pred.index.duplicated(keep='first')]
43          pred = pred.reindex(index = usable_genes, columns =
    usable_functions)
44          pred = models.true_path(pred, usable_anc_2009, usable_des_2009)
45          voting += (pred > 0.8).astype('int8')
46          average += pred
47      average /= 5
48
49      #single model
50      df.loc['SM', 'MicroF1'] = f1_score(org_13, (pred_single>0.8).astype('
    int8'), average = 'micro') * 100
51      df.loc['SM', 'MacroF1'] = f1_score(org_13, (pred_single>0.8).astype('
    int8'), average = 'macro') * 100
52      df.loc['SM', 'RankLoss'] = label_ranking_loss(org_13, (pred_single >
    0.8).astype('int8'))
53      df.loc['SM', 'PrecMicro'] = precision_score(org_13, (pred_single >
    0.8).astype('int8'), average='micro') * 100
54      df.loc['SM', 'PrecMacro'] = precision_score(org_13, (pred_single >
    0.8).astype('int8'), average='macro') * 100
55      df.loc['SM', 'RecMicro'] = recall_score(org_13, (pred_single > 0.8).
    astype('int8'), average='micro') * 100
56      df.loc['SM', 'RecMacro'] = recall_score(org_13, (pred_single > 0.8).
    astype('int8'), average='macro') * 100
57
58      tp = ((pred_single > threshold) & (org_09 == 0) & (org_13 == 1)).
    astype('int8').sum().sum()
59      fp = ((pred_single > threshold) & (org_09 == 0) & (org_13 == 0)).
    astype('int8').sum().sum()
60      tn = ((pred_single <= threshold) & (org_09 == 0) & (org_13 == 0)).
    astype('int8').sum().sum()
```

```
    fn = ((pred_single <= threshold) & (org_09 == 0) & (org_13 == 1)).astype('int8').sum().sum()

    df.loc['SM', 'N'] = tp+fp
    df.loc['SM', 'prec_transition'] = (tp) / (tp + fp) * 100

    #average
    df.loc['AVG', 'MicroF1'] = f1_score(org_13, (average>0.8).astype('int8'), average = 'micro') * 100
    df.loc['AVG', 'MacroF1'] = f1_score(org_13, (average>0.8).astype('int8'), average = 'macro') * 100
    df.loc['AVG', 'RankLoss'] = label_ranking_loss(org_13, (average > 0.8).astype('int8'))
    df.loc['AVG', 'PrecMicro'] = precision_score(org_13, (average > 0.8).astype('int8'), average='micro') * 100
    df.loc['AVG', 'PrecMacro'] = precision_score(org_13, (average > 0.8).astype('int8'), average='macro') * 100
    df.loc['AVG', 'RecMicro'] = recall_score(org_13, (average > 0.8).astype('int8'), average='micro') * 100
    df.loc['AVG', 'RecMacro'] = recall_score(org_13, (average > 0.8).astype('int8'), average='macro') * 100

    tp = ((average > threshold) & (org_09 == 0) & (org_13 == 1)).astype('int8').sum().sum()
    fp = ((average > threshold) & (org_09 == 0) & (org_13 == 0)).astype('int8').sum().sum()
    tn = ((average <= threshold) & (org_09 == 0) & (org_13 == 0)).astype('int8').sum().sum()
    fn = ((average <= threshold) & (org_09 == 0) & (org_13 == 1)).astype('int8').sum().sum()

    df.loc['AVG', 'N'] = tp+fp
    df.loc['AVG', 'prec_transition'] = (tp) / (tp + fp) * 100


    for i in range(1, 6):
        df.loc['{}_5'.format(i), 'MicroF1'] = f1_score(org_13, (voting>=i).astype('int8'), average = 'micro') * 100
        df.loc['{}_5'.format(i), 'MacroF1'] = f1_score(org_13, (voting>=i).astype('int8'), average = 'macro') * 100
        df.loc['{}_5'.format(i), 'RankLoss'] = label_ranking_loss(org_13, (voting>=i).astype('int8'))
        df.loc['{}_5'.format(i), 'PrecMicro'] = precision_score(org_13, (voting>=i).astype('int8'), average='micro') * 100
        df.loc['{}_5'.format(i), 'PrecMacro'] = precision_score(org_13, (voting>=i).astype('int8'), average='macro') * 100
        df.loc['{}_5'.format(i), 'RecMicro'] = recall_score(org_13, (voting>=i).astype('int8'), average='micro') * 100
```

```
91          df.loc['{}_5'.format(i), 'RecMacro'] = recall_score(org_13,  (
    voting>=i).astype('int8'), average='macro') * 100
92
93
94          tp = ((voting>=i) & (org_09 == 0) & (org_13 == 1)).astype('int8').
    sum().sum()
95          fp = ((voting>=i) & (org_09 == 0) & (org_13 == 0)).astype('int8').
    sum().sum()
96          tn = ((voting>=i) & (org_09 == 0) & (org_13 == 0)).astype('int8').
    sum().sum()
97          fn = ((voting>=i) & (org_09 == 0) & (org_13 == 1)).astype('int8').
    sum().sum()
98
99          df.loc['{}_5'.format(i), 'N'] = tp+fp
100         df.loc['{}_5'.format(i),  'prec_transition'] = (tp) / (tp + fp)  *
     100
101
102     df.to_csv('{}_new_tp.csv'.format(organism))
```

Merita una nota a parte il metodo con cui viene calcolato il numero di transizioni e la relativa precisione. Una implementazione efficiente di questo aspetto è fondamentale. Eseguire una valutazione "naïve" elemento per elemento richiederebbe diversi giorni per la natura interpretata di python; per ottenere risultati in tempi rapidi è necessario sfruttare la *vettorizzazione*[3] messa a disposizione da Pandas, in modo da potere fare leva sulla sua implementazione C++.

[3]La vettorizzazione consiste nel definire l'operazione da compiere su intere righe o colonne piuttosto che su un singolo elemento

Capitolo 11

Conclusioni

Lo scopo della tesi è stato applicare algoritmi di machine learning a dataset provenienti della genomica e dell'healtcare e confrontare i risultati ottenuti da un singolo modello con quelli ottenuti da più modelli usati in ensable.

Nel caso di gene ontology il voting ha permesso di regolare il tradeoff tra numero di transizioni eseguite e loro precisione mentre, nel caso di microbiome, ha consentito di migliorare le metriche ottenute, in particolare nei modelli per singolo studio e per raggruppamento di patologia.

Le tecniche applicate a gene ontolgoy possono essere migliorate, in particolare, per quanto riguarda il costo computazionale. Introducendo reti neurali oppure algoritmi che permettano di eseguire predizioni multilabel in modo nativo sarebbe possibile addestrare un solo algoritmo in grado di predire tutti i termini,eliminando la necessità di addestrarne uno per termine. Il tempo necessario ad ottenere una predizione potrebbe essere ridotto sostituendo ai dataframe pandas le matrici sparse[1], un formato che permette di rappresentare i dati in un formato compresso andando a memorizzare solamente gli elementi effettivamente ad 1 piuttosto che l'intera matrice. I risultati ottenuti mediante SMOTE sono buoni pertanto anche espandere in quella direzione la modellazione del problema potrebbe essere una buona idea. Il problema dell'aggiunta di annotazioni potrebbe essere trattato anche come un task di raccomandazione in cui gli utenti sono i geni e i termini giocano il ruolo dei prodotti; un approccio di questo tipo potrebbe portare sia ad ottenere risultati migliori che ridurre i tempi di calcolo. Negli esperimenti sono state inserite metriche originariamente non presenti come la precision e recall calcolate a livello di intera matrice tramite loro medie tra i vari termini. Queste metriche consentono di determinare la qualità complessiva del risultato finale in modo da individuare comportamenti anomali del modello senza limitarsi a considerare solamente le transizioni da 0 ad 1.

[1] https://docs.scipy.org/doc/scipy/reference/sparse.html

Per quanto riguarda Microbiome gli algoritmi applicati hanno, in alcuni casi, superato i risultati pubblicati nell'articolo di riferimento [16]; in questo contesto sono state proposte anche analisi alternative quali l'aggregazione a livello di patologia. Esperimenti futuri possono riguardare analisi di alterazioni della flora intestinale tenendo conto della presenza di più patologie o l'assunzione di antibiotici; queste analisi possono essere aiutate dalla presenza di numerosi dataset prelevabili da NCBI[2]

[2]National Center for Biotechnology Information, https://www.ncbi.nlm.nih.gov/

Bibliografia

[1] *Applications for Python*. URL: https://www.python.org/about/apps/.

[2] *artificial intelligence — Definition of artificial intelligence in English by Oxford Dictionaries*. URL: https://en.oxforddictionaries.com/definition/artificial_intelligence.

[3] Michael Ashburner et al. «Gene Ontology: tool for the unification of biology». In: *Nature Genetics* 25.1 (mag. 2000), pp. 25–29. DOI: 10.1038/75556. URL: https://doi.org/10.1038/75556.

[4] Orli Bahcall. «Precision medicine». In: *Nature* 526.7573 (ott. 2015), pp. 335–335. DOI: 10.1038/526335a. URL: https://doi.org/10.1038/526335a.

[5] F. Baquero e C. Nombela. «The microbiome as a human organ». In: *Clinical Microbiology and Infection* 18 (lug. 2012), pp. 2–4. DOI: 10.1111/j.1469-0691.2012.03916.x. URL: https://doi.org/10.1111%2Fj.1469-0691.2012.03916.x.

[6] James Bergstra e Yoshua Bengio. «Random Search for Hyper-parameter Optimization». In: *J. Mach. Learn. Res.* 13 (feb. 2012), pp. 281–305. ISSN: 1532-4435. URL: http://dl.acm.org/citation.cfm?id=2188385.2188395.

[7] Evan Bolyen et al. «QIIME 2: Reproducible, interactive, scalable, and extensible microbiome data science». In: (dic. 2018). DOI: 10.7287/peerj.preprints.27295v2. URL: https://doi.org/10.7287/peerj.preprints.27295v2.

[8] Lei Chen et al. «Gene expression profiling gut microbiota in different races of humans». In: *Scientific Reports* 6.1 (mar. 2016). DOI: 10.1038/srep23075. URL: https://doi.org/10.1038/srep23075.

[9] Tianqi Chen e Carlos Guestrin. «XGBoost». In: *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining - KDD '16*. ACM Press, 2016. DOI: 10.1145/2939672.2939785. URL: https://doi.org/10.1145/2939672.2939785.

[10] Liangxi Cheng et al. «Gene Function Prediction Based on the Gene Ontology Hierarchical Structure». In: *PLoS ONE* 9.9 (set. 2014). A cura di Yi-Hsiang Hsu, e107187. DOI: 10.1371/journal.pone.0107187. URL: https://doi.org/10.1371/journal.pone.0107187.

[11] The Gene Ontology Consortium. «The Gene Ontology Resource: 20 years and still GOing strong». In: *Nucleic Acids Research* 47.D1 (nov. 2018), pp. D330–D338. ISSN: 0305-1048. DOI: 10.1093/nar/gky1055. eprint: http://oup.prod.sis.lan/nar/article-pdf/47/D1/D330/27437640/gky1055.pdf. URL: https://doi.org/10.1093/nar/gky1055.

[12] Corinna Cortes e Vladimir Vapnik. «Support-Vector Networks». In: *Machine Learning*. 1995, pp. 273–297.

[13] Giacomo Domeniconi et al. «Cross-organism learning method to discover new gene functionalities». In: *Computer Methods and Programs in Biomedicine* 126 (apr. 2016), pp. 20–34. DOI: 10.1016/j.cmpb.2015.12.002. URL: https://doi.org/10.1016/j.cmpb.2015.12.002.

[14] Anna Veronika Dorogush, Vasily Ershov e Andrey Gulin. «CatBoost: gradient boosting with categorical features support». In: *CoRR* abs/1810.11363 (2018). arXiv: 1810.11363. URL: http://arxiv.org/abs/1810.11363.

[15] Claire Duvallet et al. «Meta-analysis of gut microbiome studies identifies disease-specific and shared responses». In: *Nature Communications* 8.1 (dic. 2017). DOI: 10.1038/s41467-017-01973-8. URL: https://doi.org/10.1038/s41467-017-01973-8.

[16] Claire Duvallet et al. *MicrobiomeHD: the human gut microbiome in health and disease*. Ago. 2017. DOI: 10.5281/zenodo.840333. URL: https://doi.org/10.5281/zenodo.840333.

[17] Robert C. Edgar. «Search and clustering orders of magnitude faster than BLAST». In: *Bioinformatics* 26.19 (ago. 2010), pp. 2460–2461. DOI: 10.1093/bioinformatics/btq461. URL: https://doi.org/10.1093/bioinformatics/btq461.

[18] Andre Esteva et al. «A guide to deep learning in healthcare». In: *Nature Medicine* 25.1 (gen. 2019), pp. 24–29. DOI: 10.1038/s41591-018-0316-z. URL: https://doi.org/10.1038/s41591-018-0316-z.

[19] Nic Fleming. «How artificial intelligence is changing drug discovery». In: *Nature* 557.7707 (mag. 2018), S55–S57. DOI: 10.1038/d41586-018-05267-x. URL: https://doi.org/10.1038/d41586-018-05267-x.

[20] *Google's AI beats human champion at Go — CBC News*. 2016. URL: https://www.cbc.ca/news/technology/alphago-ai-1.3422347.

[21] Erin M. Hill-Burns et al. «Parkinson's disease and Parkinson's disease medications have distinct signatures of the gut microbiome». In: *Movement Disorders* 32.5 (feb. 2017), pp. 739–749. DOI: 10.1002/mds.26942. URL: https://doi.org/10.1002/mds.26942.

[22] Zhuye Jie et al. «The gut microbiome in atherosclerotic cardiovascular disease». In: *Nature Communications* 8.1 (ott. 2017). DOI: 10.1038/s41467-017-00900-1. URL: https://doi.org/10.1038%2Fs41467-017-00900-1.

[23] Jin Huang e C. X. Ling. «Using AUC and accuracy in evaluating learning algorithms». In: *IEEE Transactions on Knowledge and Data Engineering* 17.3 (mar. 2005), pp. 299–310. ISSN: 1041-4347. DOI: 10.1109/TKDE.2005.50.

[24] Guolin Ke et al. «LightGBM: A Highly Efficient Gradient Boosting Decision Tree». In: *Proceedings of the 31st International Conference on Neural Information Processing Systems*. NIPS'17. Long Beach, California, USA: Curran Associates Inc., 2017, pp. 3149–3157. ISBN: 978-1-5108-6096-4. URL: http://dl.acm.org/citation.cfm?id=3294996.3295074.

[25] Sergey R. Konstantinov, Ernst J. Kuipers e Maikel P. Peppelenbosch. «Functional genomic analyses of the gut microbiota for CRC screening». In: *Nature Reviews Gastroenterology & Hepatology* 10.12 (set. 2013), pp. 741–745. DOI: 10.1038/nrgastro.2013.178. URL: https://doi.org/10.1038%2Fnrgastro.2013.178.

[26] Aaron Lee. «Machine diagnosis». In: *Nature* (apr. 2019). DOI: 10.1038/d41586-019-01112-x. URL: https://doi.org/10.1038/d41586-019-01112-x.

[27] *LinkedIn Workforce Report — United States — August 2018*. URL: https://economicgraph.linkedin.com/resources/linkedin-workforce-report-august-2018.

[28] John Markoff. *A Learning Advance in Artificial Intelligence Rivals Human Abilities*. Dic. 2015. URL: https://www.nytimes.com/2015/12/11/science/an-advance-in-artificial-intelligence-rivals-human-vision-abilities.html.

[29] Marktab. *What is the Team Data Science Process?* URL: https://docs.microsoft.com/en-us/azure/machine-learning/team-data-science-process/overview.

[30] *Python Developers Survey 2018 Results*. URL: https://www.jetbrains.com/research/python-developers-survey-2018/.

[31] Junko Tanoue, Masatoshi Yoshikawa e Shunsuke Uemura. ≪The GeneAround GO viewer≫. In: *Bioinformatics* 18.12 (dic. 2002), pp. 1705–1706. ISSN: 1367-4803. DOI: 10.1093/bioinformatics/18.12.1705. eprint: http://oup.prod.sis.lan/bioinformatics/article-pdf/18/12/1705/658998/181705.pdf. URL: https://doi.org/10.1093/bioinformatics/18.12.1705.

[32] Wanli Xu et al. ≪Distinct systemic microbiome and microbial translocation are associated with plasma level of anti-CD4 autoantibody in HIV infection≫. In: *Scientific Reports* 8.1 (ago. 2018). DOI: 10.1038/s41598-018-31116-y. URL: https://doi.org/10.1038%2Fs41598-018-31116-y.

[33] P Zheng et al. ≪Gut microbiome remodeling induces depressive-like behaviors through a pathway mediated by the host's metabolism≫. In: *Molecular Psychiatry* 21.6 (apr. 2016), pp. 786–796. DOI: 10.1038/mp.2016.44. URL: https://doi.org/10.1038/mp.2016.44.

[34] Baoli Zhu, Xin Wang e Lanjuan Li. ≪Human gut microbiome: the second genome of human body≫. In: *Protein & Cell* 1.8 (ago. 2010), pp. 718–725. DOI: 10.1007/s13238-010-0093-z. URL: https://doi.org/10.1007%5C%2Fs13238-010-0093-z.