Metodi di ensemble gerarchici per la predizione strutturata della funzione delle proteine

Corso di Laurea Magistrale in Informatica

Marco ODORE

27 maggio 2018

Abstract e organizzazione della tesi

Scopo della tesi è quello di effettuare delle sperimentazioni atte a dimostrare empiricamente l'efficacia di alcuni algoritmi di ensemble gerarchici. Questi hanno la capacità di migliorare le performance di classificatori base, i quali sono stati generati da diversi algoritmi di apprendimento automatico, applicati al problema di predizione strutturata della funzione delle proteine.

In tal senso, la tesi è suddivisa in 5 parti:

1. Descrizione del problema della predizione della funzione delle proteine, tramite metodi automatici.
2. Descrizione dei diversi metodi *ensemble* per la predizione della funzione delle proteine.
3. Predizione della funzione delle proteine sulla specie *C.elegans*.
4. Discussione dei risultati.
5. Conclusioni.

Indice

Capitolo 1

Il problema della predizione della funzione delle proteine tramite metodi automatici

La predizione della funzione delle proteine costituisce uno dei problemi fondamentali nell'ambito della *biologia computazionale*[1]. Oltre ad essere di importanza fondamentale nell'ambito della *biologia molecolare*, costituisce inoltre un problema rilevante anche per la *medicina genomica*[2].

La classificazione delle proteine effettuata in maniera *automatica*[3], nasce dall'esigenza di gestire la grossa mole di nuovi dati genomici e proteomici non categorizzati, che oggi giorno risulta essere in continua crescita. A causa di questo incremento, l'etichettatura manuale risulta infatti estremamente costosa in termini di tempo e risorse diventando di fatto impraticabile.

Nel caso specifico della predizione della funzione delle proteine[1], vi sono due tassonomie principali che relazionano le diverse classi[2]:

1. La *Gene Ontology* (GO)[4]: Definisce l'universo di concetti in relazione alle funzioni dei geni e alla loro localizzazione nella cellula.

2. Il *Functional Catalogue* (FunCat)[5]: Definisce uno schema alternativo e più sintetico per la classificazione funzionale delle proteine.

[1] All'interno della tesi la parole *gene* verrà utilizzata in maniera intercambiale con la parola *proteina*, dato che associare una funzione (o più funzioni) ad una proteina, equivale ad associare la stessa al gene che la codifica.

[2] All'interno della tesi, le parole *classe*, *annotazione*, *funzione* e *termine* verranno utilizzate in maniera intercambiabile, volendo identificare lo stesso concetto.

La GO presenta una granularità fine, mentre quella di FunCat è più grossolana, ma la principale differenza è nel modo in cui sono messe in relazione le classi all'interno delle rispettive gerarchie. La GO è infatti strutturata come un *grafo diretto aciclico* (DAG), e ogni singolo arco indica diversi tipi di relazione tra le classi[6], e cioè:

- *is a*: relazione di sotto-tipo.
- *a part of*: relazione di composizione
- *regulates*: relazione di influenza/regolazione

Mentre FunCat è organizzato come un *albero*[3]. Degli esempi di DAG per la GO e albero per FunCat sono mostrati nella figura 1.1.

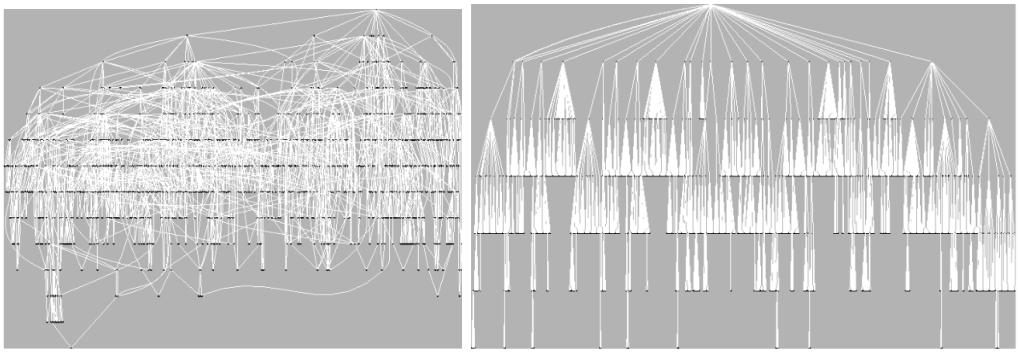

Figura 1.1: A sinistra un DAG della GO per la specie *S. cerevisiae*. A destra FunCat.

Inoltre la GO presenta una caratterizzazione ulteriore, poiché viene suddivisa in 3 diverse ontologie:

- *Biological Process:* descrive i processi ad alto livello, come insieme di diverse attività molecolari.
- *Molecular Function:* descrive le funzioni di specifici prodotti genici.
- *Cellular Component:* il luogo all'interno della cellula (nello specifico le strutture cellulari) nelle quali avviene la funzione genica.

[3]Sia il DAG che l'albero sono grafi, ma il DAG è un grafo diretto aciclico, dove un nodo può avere più genitori, a differenza dell'albero in cui ogni nodo ha al più un genitore.

FunCat possiede un albero fisso, con 1362 categorie funzionali, suddivise su 6 livelli di diversa specificità. È comparabile alle classi delle ontologie Molecular Function e Biological Process della GO.

Le annotazioni dei geni per FunCat e la GO sono governate dalla *True Path Rule* (anche conosciuta come *annotation propagation rule*)[7], una regola che garantisce la consistenza delle annotazioni. Se un gene è annotato con un determinato termine (funzione) della GO o di FunCat, allora risulta annotato anche per i predecessori (avi e genitori) di tale termine, in maniera ricorsiva. Al contrario, se un gene non è annotato per un termine, non può esserlo per i discendenti di quest'ultimo. Nella figura 1.2 sono mostrati alcuni esempi esplicativi sulla True Path Rule.

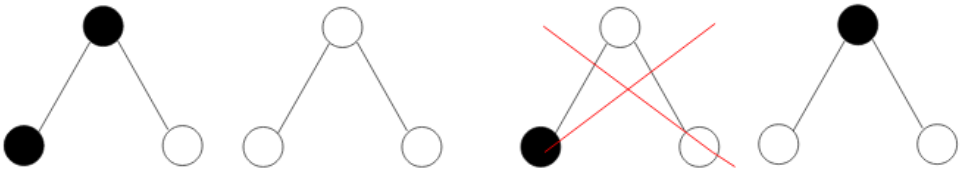

Figura 1.2: Esempi che mostrano come viene applicata la True Path Rule. I nodi in nero indicano l'annotazione, quelli in bianco l'assenza di annotazione. Il primo, il secondo e l'ultimo esempio risultano essere consistenti, mentre il terzo no. In quanto il gene risulta essere annotato per un termine, ma non per il suo genitore.

Data la granularità e specificità superiori della GO e il suo largo utilizzo nella comunità scientifica, all'interno della tesi ci si è soffermati sulla predizione delle sue funzioni.

In letteratura sono noti diversi metodi per la predizione della funzione delle proteine[8], che possiamo raggruppare in quattro macro categorie:

1. Metodi basati sulla *biosequenza*.
2. Metodi basati su *reti*.
3. Metodi *kernel per spazi di output strutturati*.
4. Metodi *ensemble gerarchici*.

1.1 Metodi basati sulle biosequenze

I metodi basati sull'allineamento della biosequenza[4] rappresentano un primo tentativo di predirre computazionalmente la funzione delle proteine. L'idea di questa tecnica si basa sul fatto che sequenze simili condividano funzioni comuni.

Esistono diversi metodi ed euristiche per la ricerca di biosequenze simile. Degli esempi sono ad esempio gli algoritmi euristici *FASTA*[9] e *BLAST*[10].

1.2 Metodi basati su reti

Questi metodi sono applicati a dati rappresentati sotto forma di grafo non direzionato $G = (V, E)$, dove i nodi $v \in V$ corrispondono ai geni e i lati $e \in E$ sono pesati in relazione al grado di co-funzionalità specificata dalla sorgente dati.

Tramite delle *relazioni di prossimità* tra i nodi connessi, questi metodi sono capaci di trasferire annotazioni da nodi precedentemente annotati a quelli che non lo sono.

Esistono diversi *algoritmi di propagazione delle etichette*(o annotazioni), come ad esempio:

- Metodi che valutano il flusso funzionale dei grafi. [11]
- Metodi semi-supervisionati come le reti di *HopField*[12].
- Metodi basati su *Markov* [13].
- Metodi basati sui *Campi Gaussiani Aleatori* [14].
- Metodi *Guilty-by-association* [15].
- Metodi basati su funzioni obiettivo con kernel. [16]

1.3 Metodi Kernel per spazi di output strutturati

I metodi *kernel* sono metodi molto utilizzati per problemi di classificazione. Tramite una mappatura *non lineare* dello spazio di partenza delle feature è infatti

[4]Esistono diversi tipi di sequenze, ad esempio quelle degli amminoacidi o dei nucleotidi.

possibile ottenere un predittore non lineare, sfruttando algoritmi nati per la separazione lineare. L'idea base è quella di sfruttare quindi un predittore lineare, del tipo:

$$f(x) = w^T \phi(x)$$

Dove x rappresenta il vettore istanza in input, e w è un vettore di pesi. ϕ è la funzione di mapping, che trasforma lo spazio di partenza R^{d5}, in un nuovo spazio R^k, con $k >> d$

$$\phi : R^d \to R^k.$$

Dato poi che il vettore w, ottenuto dall'apprendimento di algoritmi come il *percettrone*[17], può essere rappresentato come una somma pesata del tipo:

$$w = \sum_{i \in S} y_i \phi(x_i)$$

dove S è l'insieme delle istanze di training e le y_i sono le etichette, possiamo trasformare il predittore come

$$f(x) = \sum_{i \in S} y_i \phi(x_i)^T \phi(x)$$

L'idea del metodo consiste infine di rendere computazionalmente gestibile il prodotto $\phi(x_i)^T \phi(x)$, sostituendolo con una funzione kernel:

$$\phi(x_i)^T \phi(x) = K(x_i, x)$$

trasformando quindi il predittore lineare in una funzione del tipo:

$$f(x) = \sum_{i \in S} y_i K(x_i, x)$$

La criticità di questo approccio è che risulta non applicabile a problemi con output strutturato, come nel caso del problema della predizione della funzione delle proteine. Per ovviare a questa situazione, si può utilizzare una funzione kernel congiunta del tipo:

$$K : (x \times y) \times (x \times y) \to R$$

che tiene conto sia degli input (x) che degli output (y), computando la compatibilità di una data coppia di input-output. Il predittore finale si occuperà infine di selezionare l'etichetta con la maggior compatibilità.

Esempi di metodi di questo tipo sono:

- *Percettrone strutturato* [18]
- Algoritmi di *margine massimo* per output strutturati [19]

56

1.4 Metodi ensemble gerarchici

Sono metodi che tengono in considerazione esplicitamente le relazioni gerarchiche tra i termini funzionali. Sono definiti metodi *ensemble*[7] perché vengono addestrati indipendentemente diversi predittori (uno per classe funzionale della gerarchia), le cui predizioni vengono poi *combinate* in qualche maniera in una seconda fase. Questa fase, definita di ensemble o di correzione, si rende necessaria in quanto ogni predittore non è a conoscenza della struttura gerarchica dell'output (essendo addestrato indipendentemente) finendo per generare risultati errati e inconsistenti. Un esempio generale di ensemble di predittori, lo troviamo in figura 1.3.

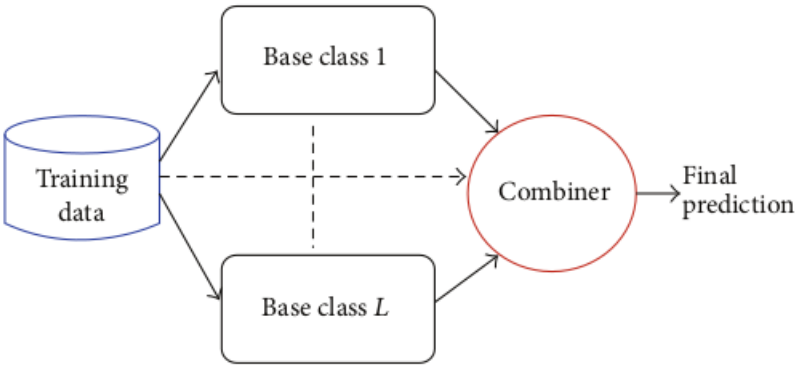

Figura 1.3: Un esempio sul funzionamento generale di un metodo ensemble. Ci sono due *learner* che lavorano indipendentemente per la classificazione di due classi (base class 1 e 2) e infine c'è l'algoritmo di ensemble (Combiner) che combina i risultati per migliorarli.

Esistono diversi tipi di metodi ensemble, ad esempio:

- Metodi *Top-Down* [20]
- Metodi *Bayesiani*[21]
- Metodi di *Riconciliazione* [22]
- Metodi basati sulla *True Path Rule* [20]
- Metodi basati sugli *alberi di decisione* [23]

In questa tesi ci si è soffermati allo studio di alcuni metodi di Ensemble Gerarchici (Top-Down e basati sulla True Path Rule).

[7]Gli ensemble di classificatori sono insiemi di predittori che lavorano assieme per risolvere un problema di classificazione.

Capitolo 2

Metodi di predizione ensemble gerarchici

I *Metodi di Ensemble Gerarchici*[8] sono metodi caratterizzati da due step principali:

1. Predizione flat delle diverse classi dell'ontologia, in maniera indipendente.
2. Combinazione e correzione delle predizioni sfruttando la relazione gerarchica tra i termini della GO.

Il secondo step rappresenta la componente *ensemble* del metodo, in quanto sfrutta gli score ottenuti dai diversi predittori generati in fase di learning, modificandoli e tendendo in considerazione la struttura dell'output. Questo permette di migliorare l'accuratezza e di rendere consistenti le predizioni fatte precedentemente. Un esempio in cui vengono mostrate le due fasi dei metodi ensemble gerarchici, lo troviamo in figura 2.1.

2.1 Predizione flat

La predizione *flat* consiste nel trattare ogni annotazione del grafo come un problema di classificazione indipendente e slegato dagli altri termini. Questa tipologia di predizione non tiene quindi in considerazione la struttura dell'ontologia su cui viene applicata.

Questo porta ad avere delle predizioni *inconsistenti* (che non rispettano la *True Path Rule*), ma portando un vantaggio in termini di semplicità e applicazione.

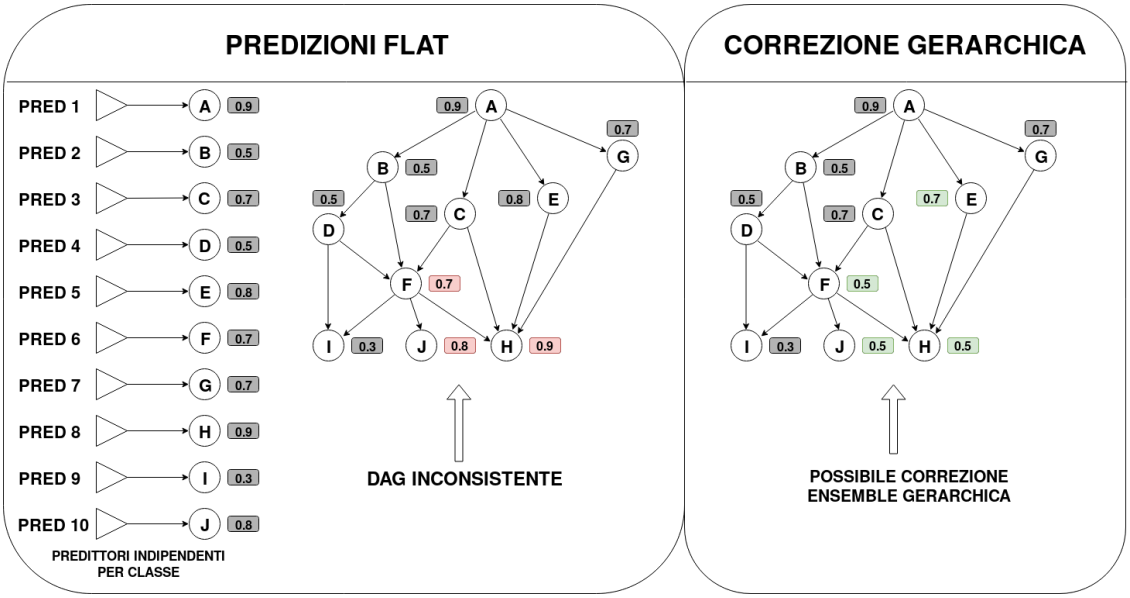

Figura 2.1: Esempio in cui vengono mostrati i due step dei metodi ensemble gerarchici. Nel primo step vengono generati gli score flat, che per alcuni nodi/termini risultano inconsistenti (F, J, H), secondo la *True Path Rule*, che verrà spiegata nel dettaglio in seguito. Nello step gerarchico ensemble vengono corretti alcuni score in modo da rendere il risultato globale consistente.

Consistenza & True Path Rule

Un insieme di predizioni $\hat{y} =< \hat{y}_1, \hat{y}_2, \ldots, \hat{y}_{|N|} >$, dove $|N|$ è la cardinalità dei termini della gerarchia, è definito *consistente*, se rispetta la *True Path Rule*, e cioè:

$$y \quad consistente \quad \leftrightarrow \forall i \in N, j \in par(i) \rightarrow y_j \geq y_i$$

Dove $par(i)$ indica l'insieme dei termini genitori del nodo i nella gerarchia.

Essendo i problemi slegati dal grafo e gestiti indipendentemente, questi possono essere trattati con diversi algoritmi di apprendimento automatico, scegliendo in base alla situazione e necessità i metodi più idonei.

2.2 Metodi ensemble

Tra le diverse tecniche ensemble utilizzate per la correzione delle predizioni flat, in questa tesi descriviamo quelle che sfruttano l'approccio *Top Down* e la *True Path Rule*. Per quest'ultima verrà descritto un nuovo metodo, che implementa in modo efficiente la *isotonic regression* per il passo top-down dell'algoritmo True Path Rule per DAG.

2.2.1 Metodo Top-Down gerarchico (HTD-DAG)

L'approccio Top-Down[24] consiste nel modificare le predizioni base dei predittori dall'alto verso il basso, e cioè dalle classi più generali a quelle più specifiche.

La correzione avviene ricorsivamente, percorrendo il grafo per *livelli*. Più precisamente, dato il grafo $G = (N, E)$, gli score flat $f(x) = \hat{y}$ sono corretti gerarchicamente a $\bar{y}$, applicando la seguente regola:

$$\bar{y}_i := \begin{cases} \hat{y}_i & if \;\; i \in root(G) \\ min_{j \in par(i)} \bar{y}_j & if \;\; min_{j \in par(i)} \bar{y}_j < \hat{y}_i \\ \hat{y}_i & altrimenti \end{cases}$$

Dove $par(i)$ specifica i genitori del nodo i.

Per garantire la correttezza e consistenza delle correzioni, i *livelli* del grafo sono definiti come *cammino massimo dalla radice*. Più formalmente, dobbiamo definire una funzione ψ che, applicata ad un nodo $i \in N$, restituisce il livello associato al cammino massimo, cioè:

$$\psi(i) = max_{p(i,r)} \; l(p(r, i))$$

dove la funzione $p(r, i)$ calcola il cammino dalla radice r al nodo i e la funzione l restituisce il livello associato ad un cammino.
Come algoritmi per il calcolo del cammino massimo si possono utilizzare ad esempio quello di *Bellman-Ford* o i metodi basati sull'ordinamento topologico[25].

Nello pseudocodice 1 è mostrato il funzionamento dell'algoritmo HTD-DAG.
La consistenza delle predizioni per il metodo HTD-DAG è dimostrata dal teorema 1[1].

Teorema 1. *Dato un grafo diretto aciclico $G = (N, E)$, la funzione ψ che assegna a un nodo il suo livello in relazione al suo cammino massimo dalla radice, e l'insieme delle predizioni $\hat{y} =< \hat{y}_1, \hat{y}_2, \ldots, \hat{y}_{|N|} >$, la correzione dell'algoritmo HTD-DAG assicura che il set di predizioni ensemble $\bar{y} =< \bar{y}_1, \bar{y}_2, \ldots, \bar{y}_{|N|} >$ soddisfi la seguente proprietà: $\forall i \in N, j \in par(i) \rightarrow y_j \geq y_i$.*

2.2.2 Metodo True Path Rule (TPR-DAG)

La correzione basata sulla True Path Rule[24], integra la correzione HTD, introducendo uno step ulteriore, e cioè quello che corregge le predizioni flat dalle classi

[1] Per la dimostrazione del teorema si rimanda al paper [24]

Pseudocodice 1 HTD-DAG

```
1: procedure HTD-DAG
2:     dist := computeMaxDistances(G)
3:     // Correzione Top-Down
4:     ȳ_r := ŷ_r // nodo radice
5:     for each d from 1 to max(dist) do
6:         N_d := {i|dist(i) = d} // nodi a distanza d
7:         for each i ∈ N_d do
8:             x := min_{j∈par(i)} ȳ
9:             if x < ŷ_i then
10:                ȳ_i := x
11:            else
12:                ȳ_i := ŷ_i
```

più specifiche a quelle più generali, in maniera quindi *bottom-up*. Lo step top-down è invece effettuato alla stessa maniera del metodo HTD.

Lo step bottom-up permette di trasferire informazione dalle predizioni che sono considerate *positive*. Più specificatamente, data una predizione $\hat{y}_i$ per il nodo $i \in N$, questa può essere aggiornata come:

$$\bar{y}_i = \frac{1}{1 + |\phi_i|}(\hat{y}_i + \sum_{j \in \phi_i} \bar{y}_j)$$

Dove ϕ_i rappresenta l'insieme dei figli del nodo i che sono considerati *positivi* in relazione alla predizione. In tal senso, esistono diverse strategie per la selezione dei figli positivi di un nodo i, come ad esempio:

- Selezione con soglia costante: la predizione viene considerata positiva se supera una soglia.
- Selezione con soglia adattiva: la soglia è selezionata per massimizzare una metrica.
- Selezione priva di soglia: vengono considerati positivi quei nodi che incrementano il valore della predizione del genitore.

A seguito dello step bottom-up, deve poi essere eseguito lo step top-down della strategia HTD, in quanto la propagazione delle predizioni positive dal basso verso l'alto non garantisce la *consistenza* delle predizioni necessarie alla TPR. Nello pseudocodice 2 sono specificati tutti i passaggi del metodo TPR-DAG.

Pseudocodice 2 TPR-DAG

```
1: procedure TPR-DAG
2:     dist := computeMaxDistances(G)
3:     // Step Bottom-Up
4:     for each d from max(dist) to 0 do
5:         N_d := {i|dist(i) = d} // nodi a distanza d
6:         for each i ∈ N_d do
7:             φ_i := getPositiveChildren(i) // seleziona i figli con un metodo
8:             ȳ_i = 1/(1+|φ_i|) (ŷ_i + Σ_{j∈φ_i} ȳ_j)
9:     // Step Top-Down
10:    ȳ_r := ŷ_r // nodo radice
11:    for each d from 1 to max(dist) do
12:        N_d := {i|dist(i) = d} // nodi a distanza d
13:        for each i ∈ N_d do
14:            x := min_{j∈par(i)} ȳ
15:            if x < ŷ_i then
16:                ȳ_i := x
17:            else
18:                ȳ_i := ŷ_i
```

Per l'algoritmo TPR-DAG valgono i seguenti teoremi[2]:

Teorema 2. *Dato un grafo diretto aciclico $G = (N, E)$ e l'insieme delle predizioni flat $\hat{y} =< \hat{y}_1, \hat{y}_2, \ldots, \hat{y}_{|N|} >$, la correzione dell'algoritmo TPR-DAG assicura che il set di predizioni ensemble $\bar{y} =< \bar{y}_1, \bar{y}_2, \ldots, \bar{y}_{|N|} >$ soddisfi la seguente proprietà: $\forall i \in N, j \in anc(i) \rightarrow y_j \geq y_i$, dove $anc(i)$ è l'insieme di tutti i predecessori del nodo i nella gerarchia (compresi i genitori).*

Teorema 3. *L'algoritmo ensemble TPR-DAG con la selezione dei positivi ϕ_i, tale che $\phi_i := \{j \in child(i) | \bar{y}_j > \hat{y}_i\}$, ottiene sempre una Recall[3] uguale o maggiore dell'algoritmo ensemble HTD-DAG.*

Non è garantita invece che la *Precision*[4] dell'algoritmo TPR-DAG sia sempre migliore o uguale a quella dell'algoritmo HTD-DAG.

[2]Per la dimostrazione dei teoremi si rimanda al paper [24]
[3]Per i dettagli della metrica si rimanda al capitolo 3.4.
[4]Per i dettagli della metrica si rimanda al capitolo 3.4.

2.2.3 Metodo True Path Rule con Isotonic Regression (ISO-TPR)

Per il passo Top-Down dell'algoritmo TPR-DAG è possibile sfruttare gli algoritmi per la risoluzione dei problemi di *Isotonic Regression* (IR) (chiamati anche di *Monotonic Regression*). In questa sezione discuteremo di uno di questi metodi, noto *come Pool-Adjacent-Violators*, capace di risolvere il problema in maniera efficiente.

Pool-Adjacent-Violators (PAV)

L'algoritmo PAV è utilizzato per la risoluzione dei problemi di IR, in cui si esegue il fitting di una linea (senza vincolo di forma) ad una sequenza di osservazioni, rispettando dei vincoli di *ordinamento totale*. Questi vincoli fanno in modo cioè che i punti della linea, output della regressione, siano crescenti. Più formalmente, PAV risolve problemi di minimo del tipo:

$$\min_x \sum_{i=1}^{n} w_i(x_i - a_i)^2$$

$$\text{such that } x_1 \leq x_2 \leq ... \leq x_n$$

Dove le a_i, sono le nostre n osservazioni e le x_i i valori ricercati. I valori w_i sono degli eventuali pesi associati all'equazione (generalmente settati a 1).

Un esempio di isotonic regression con ordinamento totale messo a confronto con una regressione lineare[27], lo troviamo nella figura 2.2.

Essendo un problema quadratico (quindi convesso), la soluzione è unica. Per la sua risoluzione, PAV ha una complessità computazionale pari a $O(n)$.

Generalizzazione dell'algoritmo PAV (GPAV)

Generalizzando la isotonic regression, e cioè non avendo necessariamente un vincolo di ordinamento totale, il problema può essere riformulato sotto forma di grafo aciclico diretto (DAG). Più formalmente, dato un DAG, $G(N, E)$, con il set di nodi $N = \{1, 2, ..., n\}$, si deve trovare il vettore $x^* \in R^n$ tale che:

$$min \sum_{i=1}^{n} w_i(x_i - a_i)^2$$

$$\text{such that } x_i \leq x_j \ \forall (i, j) \in E$$

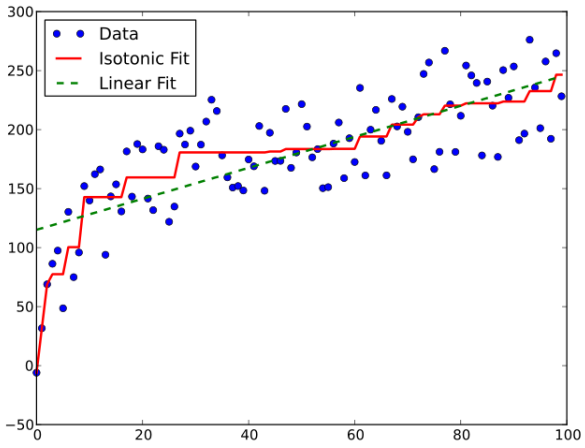

Figura 2.2: Un esempio di IR (linea rossa) a confronto con una regressione lineare (linea tratteggiata).

Dove diciamo che x_i è *adiacente* alla componente x_j se $(i, j) \in E$.

Tale problema è risolvibile dalla versione generalizzata di PAV (GPAV) con un complessità computazionale pari a $O(n^2)$[26].

L'algoritmo GPAV genera uno split del set di nodi N, in blocchi disgiunti. Il valore dei nodi all'interno di questi blocchi è il medesimo e per questa motivazione ognuno dei blocchi è rappresentato da un singolo elemento, chiamato *head*. Formalizzando, il blocco rappresentato dal nodo (head) $i \in N$, viene denotato dal simbolo B_i. Il set di blocchi è invece rappresentato dall'insieme H, per cui valgono le seguenti proprietà:

$$H \subseteq N, \quad \forall i \in H$$

$$\cup_{i \in H} B_i = N$$

$$B_i \cap B_j = 0, \quad \forall i, j \in H,$$

Ogni componente di x, associato ad un blocco $B_i \in H$, ha il medesimo valore, calcolato come segue:

$$x_j = \frac{\sum_{k \in B_i} w_k a_k}{W_i}, \quad \forall j \in B_i$$

Dove W_i, peso del blocco B_i, è calcolato come:

$$W_i = \sum_{k \in B_i} w_k$$

Per il caso specifico in cui $w_k = 1, \forall k \in B_i$, si ottiene

$$x_j = \frac{\sum_{k \in B_i} a_k}{|B_i|}$$

Dove $|B_i|$ è la cardinalità del blocco B_i.

Dato poi un nodo $i \in N$, l'insieme dei suoi predecessori immediati j, e cioè, tali che, $\{j \in N : (j,i) \in E\}$, viene denotato con i^-. Il blocco B_i è detto *adiacente* al blocco B_j, o immediato predecessore, se esiste un $k \in B_i$ e un $l \in B_j$ tali che $k \in l^-$. Sia infine B_i^- il set di nodi immediatamente adiacenti a B_i.

L'algoritmo GPAV assegna nella fase iniziale $B_i = \{i\}$ e $B_i^- = i^-, \quad \forall i \in N$. Successivamente tali blocchi si possono ridurre, in quanto un blocco ne può *assorbire* uno adiacente se questo rispetta determinate condizioni. Nello pseudocodice 3 sono specificati i passaggi per la procedura di assorbimento di un blocco B_i da parte di un blocco B_j.

Pseudocodice 3 Absorb

1: **procedure** ABSORB(j, i)
2: $H = H \backslash \{i\}$
3: $B_j^- = B_i^- \cup B_j^- \backslash \{i\}$
4: $x_j = \frac{W_j x_j + W_i x_i}{W_j + W_i}$
5: $B_j = B_j \cup B_i$
6: $W_j = W_j + W_i$
7: **for** (each $k \in H \;\; s.t. \;\; i \in B_k^-$) **do**
8: $B_k^- = B_k^- \backslash \{i\} \cup \{j\}$

Assumendo che i nodi siano stati pre ordinati topologicamente[5], GPAV esegue le operazioni specificate nello pseudocodice 4

Pseudocodice 4 GPAV

```
1:  procedure GPAV
2:      H = N
3:      for (each i ∈ N) do
4:          B_i = {i}
5:          B_i^- = i^-
6:          x_i = a_i
7:          W_i = w_i
8:      for k = 1, 2, ..., n do
9:          // finché esiste un predecessore di B_k che viola la monotonicità
10:         while {i ∈ B_k^- : x_i ≥ x_k} ≠ 0 do
11:             // Trova l'elemento che viola maggiormente il vincolo
12:             Find j ∈ B_k : U_j = max{U_i : i ∈ B_k^-}
13:             Absorb(i, j) // j viene assorbito da B_i
14:     //Aggiornamento per soluzione finale
15:     for each k ∈ H do
16:         for each i ∈ B_k do
17:             x_i^* = x_i
```

Riassumendo, l'algoritmo effettua degli assorbimenti di blocchi adiacenti, finché questi violano i vincoli del problema quadratico, generando di fatto una partizione dei nodi, in cui le parti condividono lo stesso valore.

ISO-TPR

Sostituendo quindi il secondo step del metodo TPR-DAG, e cioè quello top down, con GPAV, otteniamo l'algoritmo ISO-TPR. Il funzionamento di tale algoritmo è riassunto nello pseudocodice 5.

[5]Esiste una soluzione esatta per determinati ordinamenti topologici del grafo[28].

Pseudocodice 5 ISO-TPR

```
1: procedure ISO-TPR
2:     dist := computeMaxDistances(G)
3:     // Step Bottom-Up
4:     for each d from max(dist) to 0 do
5:         N_d := {i|dist(i) = d} // nodi a distanza d
6:         for each i ∈ N_d do
7:             φ_i := getPositiveChildren(i) // seleziona i figli con un metodo
8:             ȳ_i = 1/(1+|φ_i|) (ŷ_i + Σ_{j∈φ_i} ȳ_j)
9:     // Step Top-Down
10:    H = N
11:    for (each i ∈ N) do
12:        B_i = {i}
13:        B_i^- = i^-
14:        x_i = ŷ_i
15:        W_i = w_i
16:    for k = 1, 2, ..., n do
17:        // finché esiste un predecessore di B_k che viola la monotonicità
18:        while {i ∈ B_k^- : x_i ≥ x_k} ≠ 0 do
19:            // Trova l'elemento che viola maggiormente il vincolo
20:            Find j ∈ B_k : U_j = max{U_i : i ∈ B_k^-}
21:            Absorb(i, j) // j viene assorbito da B_i
22:    //Aggiornamento per soluzione finale
23:    for each k ∈ H do
24:        for each i ∈ B_k do
25:            ȳ_i = x_i
```

Capitolo 3

Predizione della funzione delle proteine della specie C.elegans

Per questa tesi sono stati eseguiti degli esperimenti al fine di verificare empiricamente, tramite un problema reale, se ed in quali circostanze le predizioni flat possono essere migliorate da metodi di ensemble gerarchici. Nello specifico si è eseguito la sperimentazione sul genoma della specie *Caenorhabditis elegans*, un organismo modello molto semplice, utilizzato frequentemente in biologia[29].

3.1 Dataset

L'insieme delle istanze utilizzato come input dai diversi algoritmi di apprendimento automatico per il nostro problema, è dato dall'insieme dei geni della specie oggetto della sperimentazione, il quale è rappresentato da una matrice simmetrica generata dal network di interazione *proteina-proteina* estratto dal database *STRING* (Search Tool for the Retrieval of Interacting Genes/Proteins)[1][30]. Le interazioni tra le proteine aiutano a descrivere e a identificare le loro funzioni: ad esempio, la struttura tridimensionale di una proteina acquisisce significato solo nel contesto di una più grande combinazione di proteine. Il database STRING integra diversi tipi *canali di evidenza*[2] proteina-proteina, in base alla specie di riferimento. Un network di esempio è dato dalla figura 3.1.

[1] La versione del network di interazioni proteina-proteina è la v10.5 del 20 Dicembre 2017.
[2] `https://string-db.org/help/getting_started/#evidence`

Figura 3.1: Una porzione del network proteina-proteina per la specie C.elegans. I nodi rappresentano le proteine, e gli archi colorati le differenti relazioni tra queste.

Nel caso della nostra matrice, il valore della relazione proteina-proteina è dato da uno score combinato[3] di differenti sorgenti, e cioè:

- Le interazioni proteiche verificate sperimentalmente, presenti in database primari.
- Le informazioni del pathway[4] ottenute da database curati.
- I collegamenti semantici e/o statistici fra le proteine, ottenuti dall'analisi effettuata tramite text-mining di sommari MedLine e da grosse collezioni documentali.

[3] `https://string-db.org/help/faq/#how-are-the-scores-computed`

[4] La via metabolica (spesso chiamata pathway metabolico o più semplicemente pathway) è l'insieme delle reazioni chimiche coinvolte in uno o più processi di anabolismo o catabolismo all'interno di una cellula.

- Le interazioni ottenute tramite diversi algoritmi sulle informazioni genetiche.
- Le interazioni dei geni ortologhi, e cioè geni che sono presenti in specie diverse ma correlate, che codificano per proteine con strutture e funzioni simili.

Questi score combinati proteina-proteina rappresentano di fatto le *feature* (o *caratteristiche*) delle istanze, e sono espressi con un valore intero in $[0, 1000]$, dove 1000 indica il massimo livello interazione e 0 la sua assenza.

Nel caso specifico della specie C.elegans, la matrice STRING ha dimensione 15752×15752.

3.2 Classi/Annotazioni

Il numero di classi per la specie C.elegans, varia di molto in base all'ontologia di riferimento della GO. Il numero delle classi (termini) aventi almeno 1 annotazione sperimentale per una proteina di C. Elegans ed i relativi archi, sono mostrati nella tabella 3.1.

ontologia	numero di termini	numero di archi
BP	4068	8066
MF	1163	1567
CC	578	1082

Tabella 3.1: Statistiche relative ai DAG delle annotazioni della specie C.elegans.

Come si può vedere dalla precedente tabella, l'ontologia più problematica e onerosa risulta essere la BP, che oltre ad avere un numero considerevole di classi, possiede un DAG molto più complesso rispetto alle altre due gerarchie. Questo vuol dire che sia l'intero processo di predizione flat che il processo di correzione gerarchica richiederanno più tempo rispetto alle ontologie MF e CC.

Al fine di garantire che nella cross-validation non vengano generati fold privi di esempi annotati, in fase sperimentale sono state selezionate solo quelle classi che hanno almeno 10 annotazioni tra le istanze. Nella tabella 3.2 sono mostrate il numero di classi per ontologia, a seguito della riduzione, con annesse alcune statistiche.

Onto	numero di termini	media	d.std.	massimo	minimo
BP	1335	71,33	151,68	2597	10
MF	186	61,23	191,84	1806	10
CC	221	131,9	302,25	1924	10

Tabella 3.2: La colonna *numero di termini* indica il numero di termini ottenuti dopo la selezione, *media* la media delle annotazioni per classe, per l'ontologia di riferimento, *d.std.* la deviazione standard delle annotazioni per l'ontologia di riferimento, *massimo* e *minimo* rispettivamente il massimo e minimo numero di annotazioni.

3.3 Algoritmi di apprendimento automatico utilizzati per le predizioni flat

Per quanto riguarda gli algoritmi di apprendmento automatico da utilizzare per le predizioni flat, si sono considerati i seguenti metodi:

1. *K-Nearest Neighbors*[31]
2. *Logit Boost*[32]
3. *Linear Discriminant Analysis*[33]
4. *eXtreme Gradient Boosting* [34]
5. *C5.0* (Alberi di decisione)[35]
6. *Random Forest*[36]
7. *Multilayer Perceptron*[37]
8. *Support Vector Machine lineare*[38]
9. *Bagged CART* (Bagged ensemble di alberi di decisione) [39]
10. *AdaBoost.M1*[40]
11. *Naive Bayes*[41]
12. *Glmnet*[42]

Dato l'elevato numero di algoritmi selezionati per la sperimentazione, si è deciso di non effettuare il tuning dei parametri, questo per evitare di allungare ulteriormente i tempi dell'intero processo di valutazione e generazione degli score flat.

Per ogni metodo si sono utilizzati valori prefissati dei parametri degli algoritmi

di apprendimento. Tali valori sono riassunti nello script A.1 in appendice.

Per quanto riguarda l'implementazione degli algoritmi di apprendimento si è utilizzato il package *Caret*[43] di R. In appendice è presente uno script (A.2) di esempio di training e test dei metodi flat con cross-validation.

Tutti i test e l'intera sperimentazione sono stati eseguiti su una macchina che dispone di:

- 12 Processori Intel(R) Xeon(R) CPU E5-2630 v2 @ 2.60GHz.
- 128 GB di Memoria RAM.

3.4 Valutazione delle performance

Come metriche relative alla valutazione della qualità dei predittori sono state utilizzate l'*Area Under Precision Recall Curve* (AUPRC) e l'*Area Under Receiver Operating Characteristic Curve* (AUROC).

La AUROC è una misura di *robustezza* del classificatore, e mette in relazione le misure di *Recall* e *False Positive Rate*(FPR), al variare di una *soglia* applicata all'output del modello[5]. La Recall è calcolata come:

$$Recall = \frac{TruePositive}{TruePositive + FalseNegative}$$

mentre la FPR come:

$$FPR = \frac{FalsePositive}{TrueNegative + FalsePositive}$$

Entrambe sono definite in $[0, 1]$. La Recall ci indica quanti elementi sono stati classificati correttamente come *rilevanti* ($TruePositive$), appartenenti cioè alla classe che si sta osservando, sul totale delle istanze appartenenti alla classe ($TruePositive + FalseNegative$). La FPR invece ci da una stima della frequenza (*rate*) di istanze che sbagliamo a classificare come appartenenti alla classe ($FalsePositive$) sul totale delle istanze non appartenenti alla classe ($FalsePositive + TrueNegative$). Facendo un plot dei valori di queste metriche al variare della soglia è possibile generare una curva, la cui area sottesa rappresenta la misura ricercata. Più il valore di quest'area si avvicina a 1, più il classificatore è considerato

[5]La soglia funge da discriminante per l'output del predittore e ci permette di decidere se classificare l'istanza come appartenente alla classe di riferimento o meno. Ex. if *output*>=soglia then 1 else 0.

affidabile. Al contrario, più il risultato è vicino a 0.5 più il predittore è vicino ad un predittore casuale.[6]. Un esempio di plot per i risultati di un predittore sono mostrati nella figura 3.2.

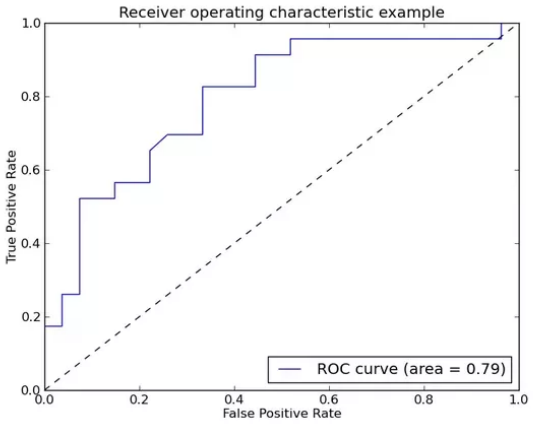

Figura 3.2: Un grafico generato dai diversi valori di Recall (True Positive Rate) e di False Positive Rate.

La metrica di AUPRC si rende invece necessaria in quanto il problema della predizione della funzione delle proteine risulta generalmente *sbilanciato*. Per problemi sbilanciati ci ri riferisce a quei problemi che presentano un numero di istanze annotate di molto inferiore al numero di quelle che non lo sono. La AUPRC mette in relazione la variazione di *Precision* al variare della *Recall*. La Precision è una misura calcolata come:

$$Precision = \frac{TruePositive}{TruePositive + FalsePositive}$$

Anche questa definita in [0, 1]. Tale metrica esprime la frazione delle predizioni positive corretta rispetto all'insieme delle predizioni positive.

La variazione di Recall e Precision è ottenuta sempre modificando una soglia applicata sull'output del classificatore, generando così diversi punti del grafico. L'area al di sotto del grafico (sempre compresa in $[0, 1]$) ci da poi un'indicazione della performance del predittore. Infatti più il valore di tale area è grande, più il predittore è da considerarsi affidabile. Un esempio di plot delle curve per due algoritmi/predittori è mostrato in figura 3.3.

[6]Valori vicini a 0.5 indicano che il predittore non è tanto diverso da un predittore ottenuto con il lancio di una moneta, che predice ad esempio positivo con testa, negativo con croce. È quindi da considerarsi simile ad un predittore casuale.

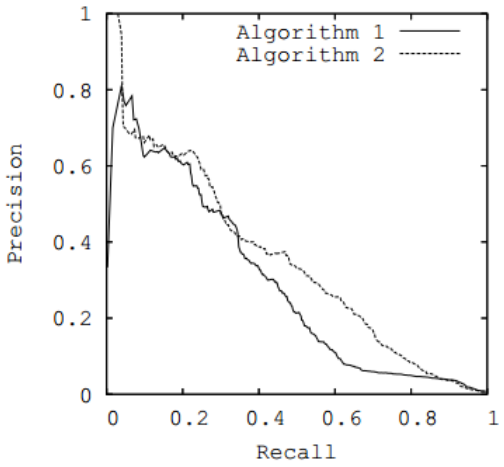

Figura 3.3: Due curve di due diversi algoritmi, generate dai diversi valori di Precision e Recall.

3.5 Stime preliminari dei tempi di calcolo e delle performance

Al fine di identificare delle configurazioni per gli esperimenti, capaci di terminare in tempi ragionevoli l'apprendimento e la valutazione delle performance, si sono eseguite delle stime preliminari sulla complessità temporale empirica dei modelli stessi.

La stima è stata effettuata su di un campione casuale di 10 classi (mostrate nella tabella 3.3) su ciascuno dei tre domini della Gene Ontology[7], eseguendo una *cross-validation*[44] a 10 fold.

Per evitare di generare fold privi di esempi positivi, questi sono stati generati in maniera stratificata.[8].

3.5.1 Risultati delle stime iniziali

Le stime iniziali per l'intero processo di valutazione del learning, hanno evidenziato tempi di calcolo molto lunghi, nonostante sia stato escluso il tuning dei parametri

[7]Biological Process, Molecular Function and Cellular Component.

[8]Per generare i fold in maniera stratificata, si è sfruttata la funzione di R, *do.stratified.cv.data.single.class[45]*, presente nella libreria HEMDAG.

Biological Process		Molecular Function		Cellular Component	
Classe	**annotazioni**	**Classe**	**annotazioni**	**Classe**	**annotazioni**
GO:0015031	44	GO:0005261	33	GO:0005929	80
GO:0031325	216	GO:0008237	12	GO:0030424	105
GO:0045176	10	GO:0016818	56	GO:0043226	995
GO:1901565	48	GO:0060089	53	GO:0098687	13
GO:0009792	278	GO:0004857	13	GO:0005829	56
GO:1901046	26	GO:0046872	15	GO:0098562	14
GO:1903507	71	GO:0097159	198	GO:0098802	10
GO:0048814	10	GO:0019887	12	GO:0044422	588
GO:0048489	19	GO:0017048	12	GO:0043234	201
GO:0006508	53	GO:0001664	12	GO:0000932	13

Tabella 3.3: Le classi selezionate nella fase preliminare, per ontologia

di ogni algoritmo.

Come mostrato nelle tabelle 3.4, 3.5 e 3.6, i tempi, che riguardano l'intero processo di cross-validation per alcuni algoritmi, sono sull'ordine delle ore per classe. Mantenendo questo set-up, prendendo ad esempio in considerazione solo le classi della BP ontology con almeno 10 annotazioni (1335), utilizzando l'algoritmo K-NN, servirebbero tra i 16 e i 17 mesi di computazione con i sistemi di calcolo disponibili per questo esperimento[9].

Dati i tempi molto lunghi riscontrati per i primi 6 algoritmi utilizzati[10], si è preferito non proseguire ulteriormente con questa configurazione per le stime dei tempi di calcolo dei metodi rimanenti.

[9] $9.16h \times 1335 = 1228.9h = \ 510gg = \ 17months$

[10] C5.0, glmnet, knn, mlp, svmLinear, xgbLinear

Algo	TempoMassimo	TempoMinimo	TempoMedio	DevSt.Tempo
C5.0	5.29	3.53	4.07	0.52
glmnet	2.94	2.12	2.4	0.23
knn	21.66	6.55	9.16	4.86
mlp	9.4	5.67	6.51	1.06
svmLinear	3.8	2.62	3.06	0.37
xgbLinear	2.79	2.11	2.29	0.18

Tabella 3.4: Alcune statistiche dei vari algoritmi per l'ontologia BP. I tempi sono da intendersi in ore e per classe, per un campione di 10 classi.

Algo	TempoMassimo	TempoMinimo	TempoMedio	DevSt.Tempo
C5.0	3.75	3.17	3.44	0.18
glmnet	2.46	2.16	2.33	0.11
knn	7.44	6.39	6.59	0.29
mlp	7.9	5.05	5.68	0.81
svmLinear	2.92	2.42	2.64	0.16
xgbLinear	2.5	2.06	2.24	0.12

Tabella 3.5: Alcune statistiche dei vari algoritmi per l'ontologia MF. I tempi sono da intendersi in ore e per classe, per un campione di 10 classi.

Algo	TempoMassimo	TempoMinimo	TempoMedio	DevSt.Tempo
C5.0	5.29	3.53	4.07	0.52
glmnet	2.94	2.12	2.4	0.23
knn	21.66	6.55	9.16	4.86
mlp	9.4	5.67	6.51	1.06
svmLinear	3.8	2.62	3.06	0.37
xgbLinear	2.79	2.11	2.29	0.18

Tabella 3.6: Alcune statistiche dei vari algoritmi per l'ontologia CC. I tempi sono da intendersi in ore e per classe, per un campione di 10 classi.

Nei boxplot nella figura 3.4, è possibile vedere come i tempi di calcolo varino (anche di molto) in base all'algoritmo utilizzato.

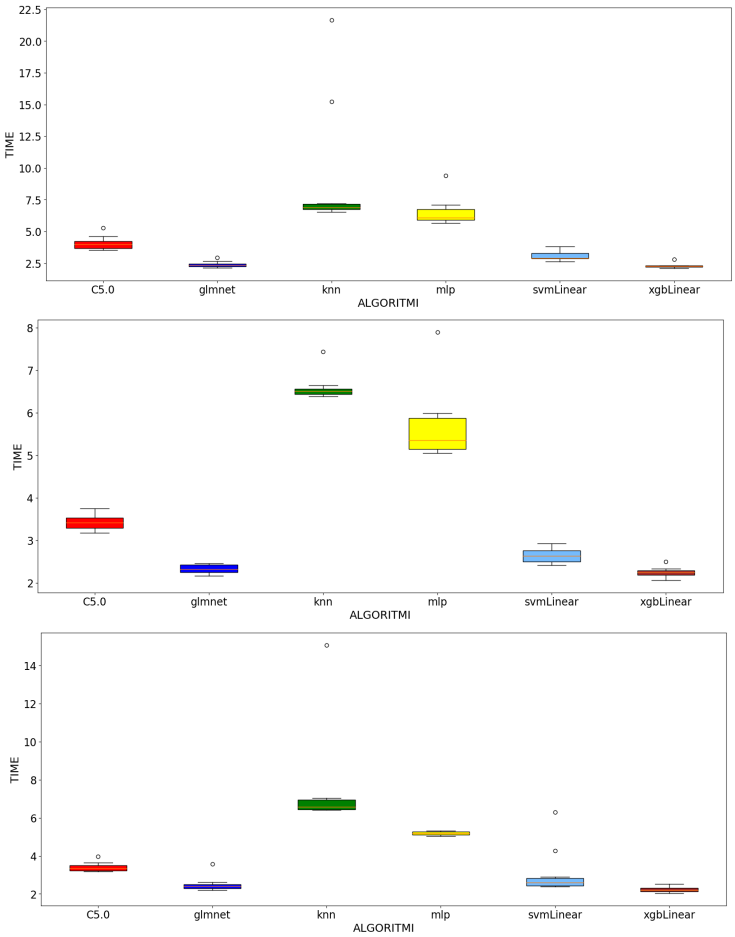

Figura 3.4: I box plot dei tempi di esecuzione, con cross-validation a 10 fold per le ontologie BP, MF e CC. I tempi sono da intendersi in ore e per classe, per un campione di 10 classi.

Considerato quindi l'elevato numero di classi per ontologia, si è deciso di effettuare diverse operazioni volte a ridurre la complessità del problema, come la riduzione del numero di fold nella cross-validaton e delle feature da utilizzare nel processo di learning.

3.6 Riduzione della complessità del problema

Verificati i tempi proibitivi nella fase iniziale esplorativa, si è deciso di effettuare due principali operazioni per ridurre la complessità dell'esperimento, e cioè:

- Introdurre la *cross-validation a 5 fold* anziché a 10 fold per la stima delle performance.
- Ridurre la complessità del learning con due metodi alternativi: *Selezione delle feature* e *PCA*.

La riduzione della complessità non solo è utile a ridurre i tempi di calcolo dell'intero esperimento, ma ha lo scopo inoltre di ridurre il rischio di *overfitting* dei modelli. Nel nostro caso specifico infatti, il numero di istanze del dataset risulta essere limitato in relazione al numero di feature (15752 istanze x 15752 feature) e parametri dei diversi modelli[46].

3.6.1 Selezione delle Feature

Per la selezione delle feature si è deciso di utilizzare una selezione basata sulla *Correlazione di Pearson*[50], che ci permette di capire quanto una feature/caratteristica è *correlata* con la classe di riferimento.

Il calcolo della correlazione avviene tramite la seguente formula:

$$corr_{X,Y} = \frac{COV(X,Y)}{\sigma_X \sigma_Y}$$

Dove X è ad esempio la variabile aleatoria che rappresenta l'annotazione delle istanze in relazione alla classe e Y la variabile aleatoria che rappresenta il valore della feature (al variare delle istanze). $COV(X,Y)$ indica il valore di covarianza tra le variabili aleatorie e σ la deviazione standard.

In questa maniera è possibile realizzare un *ranking* (ordinamento) delle feature, in base al loro potenziale informativo. Un' alta correlazione è infatti *spesso*[11] associata ad un grossa capacità predittiva della caratteristica.

Un altro motivo che ha portato all'utilizzo di questo tipo di selezione è la ridotta complessità temporale associata al calcolo della correlazione. Infatti, al fine di effettuare una selezione *unbiased*[47][12], tale selezione delle feature deve essere effettuata per ogni training set generato nella cross-validation. È chiaro quindi che se tale operazione diventa onerosa dal punto di vista computazionale[13], il vantaggio che ne deriva si riduce.

[11] Non sempre un'alta correlazione implica una causalità, ma anzi può essere casuale.

[12] Effettuare una selezione delle feature a monte della cross-validation, porterebbe a dei problemi di *data leakage*, in quanto si trasmetterebbe l'informazione presente nei test set dei diversi fold, nei training set, portando a delle stime delle performance con *bias*.

[13] TempoFeatureSelection × NumeroClassi × NumeroFoldCrossValidation

3.6.2 Analisi delle componenti principali

La *PCA* è una tecnica che ci permette di individuare il sotto-spazio *k-dimensionale* di R^d [14] in cui è massimizzata la varianza della componente principale[46]. Quindi fissato un k, il procedimento ci permette di estrarre le k feature (in questo caso chiamate *componenti*) del nuovo spazio. In questa maniera si può passare da uno spazio ad elevata dimensionalità (d feature), ad uno con dimensionalità ridotta (k componenti), con $k << d$.

Concretamente il metodo genera le nuove k componenti come combinazione lineare delle d feature di partenza, in maniera tale da massimizzare la *varianza spiegata*. Tutto ciò è reso possibile grazie ad un metodo di scomposizione matriciale, chiamato *Singular Value Decomposition[46]*, che ci permette di individuare l'operatore lineare P_k, che risolve questo problema di minimo:

$$min_{P_k \in P} ||X - P_k X||_F^2$$

Dove X è la matrice che rappresenta le nostre istanze, e l'operatore $||.||_F^2$ è la *norma di Frobenius*, calcolata su di una matrice come:

$$||X||_F^2 = \sum_{i=1}^{m} \sum_{j=1}^{m} x_{ij}^2$$

Un piccolo esempio di PCA lo possiamo vedere in figura 3.5[49].

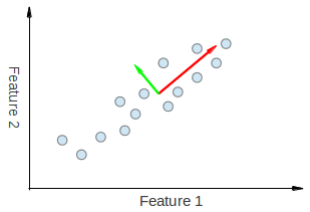

Figura 3.5: Le feature iniziali sono la Feature 1 e la Feature 2. Le componenti ottenute dalla PCA sono il vettore in rosso e quello in verde. Se volessimo ridurre la dimensionalità di questo piccolo dataset, potremmo proiettare i punti sulla componente rossa (quella che massimizza la varianza) eliminando la componente verde.

Le componenti risultate dall'analisi, si possono poi ordinare per varianza, la quale rappresenta il *ranking* delle componenti del nuovo spazio vettoriale, e quindi il criterio di selezione per la riduzione della dimensionalità del problema.

[14] Spazio di partenza in cui si trova il nostro dataset, con d feature.

3.6.3 Stime dei tempi di calcolo e performance con selezione delle feature

Per la selezione delle feature si è deciso di stimare i tempi di calcolo e le performance per 3 differenti livelli di selezione delle feature con cross-validation a 5 fold:

- Le prime 1000 feature nel ranking
- Le prime 500 feature nel ranking
- Le prime 100 feature nel ranking

Come si può vedere dalla tabella B.1 in appendice, i tempi di calcolo ottenuti, sempre selezionando 10 classi per ognuna delle 3 ontologie (BP, MF, CC), per le selezioni delle feature a 1000 e 500 feature, risultano particolarmente alti, soprattutto per alcuni algoritmi (*Adaboost, SvmLinear, LogitBoost e Treebag*).

Per questa motivazione si è deciso di optare per la selezione delle prime 100 feature nel ranking ottenuto dal calcolo della correlazione di Pearson. Nei boxplot in figura 3.6, è possibile vedere come i tempi di calcolo si siano ridotti di molto a seguito della selezione delle prime 100 feature e con cross-validation a 5 fold.

Considerato il tempo medio per classe stimato e il numero totale delle classi, gli algoritmi più problematici a livello di tempo di calcolo risultano essere *K-NN* (6 giorni di computazione), *Adaboost* (3 gg di computazione) e *svmLinear* (meno di 3 giorni di computazione). Mentre a livello di performance non mostrano buoni risultati, vicino ad un predittore casuale, gli algoritmi *glmnet*, *C5.0*, *K-NN*. Questo è probabilmente dovuto al fatto che si sono utilizzati i parametri di default delle librerie di caret senza effettuare una ricerca dei valori ottimali per lo specifico problema[15]. *Naive Bayes* mostra risultati scadenti per la metrica AUPRC, ma non per la AUROC, dove anzi ha risultati molto buoni. Nella figura 3.7 sono messe a confronto le performance stimate degli algoritmi, relative alla metrica AUROC, tramite boxplot, per le ontologie BP, MF e CC. Nella figura 3.8 troviamo invece i boxplot per le stime della AUPRC.

[15]Lo scopo della tesi è di dimostrare empiricamente l'efficacia dei metodi ensemble a prescindere dei risultati flat.

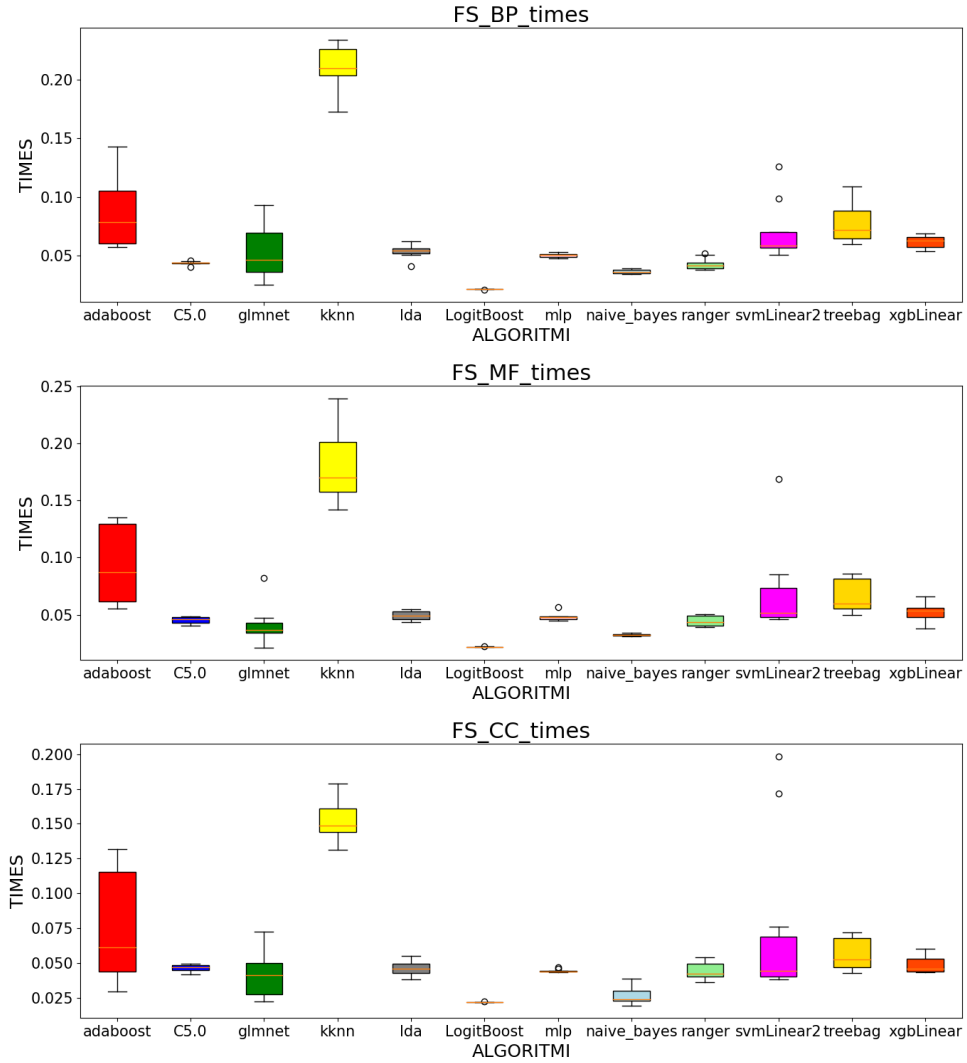

Figura 3.6: Boxplot dei tempi di calcolo dei diversi algoritmi per classe, su un campione di 10 classi, con selezione del prime 100 feature (correlazione di Pearson) e cross-validation a 5 fold, ontologie BP, MF e CC. I tempi sono da intendersi in ore.

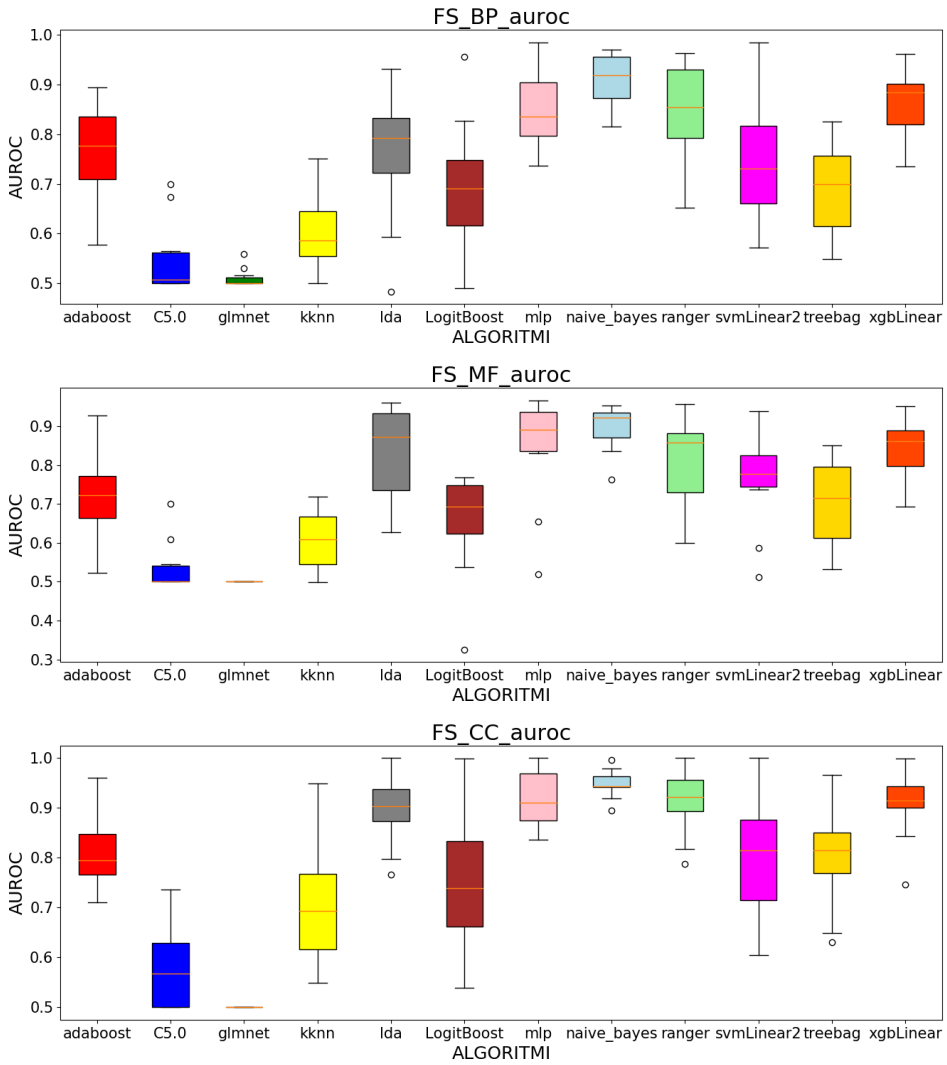

Figura 3.7: Boxplot delle performance di AUROC per classe dei diversi algoritmi, su un campione di 10 classi, con selezione del prime 100 feature (correlazione di Pearson) e cross-validation a 5 fold, per le ontologie BP, MF, CC. Sono evidenti le performance scadenti di *C5.0*, *glmnet* e *K-NN*.

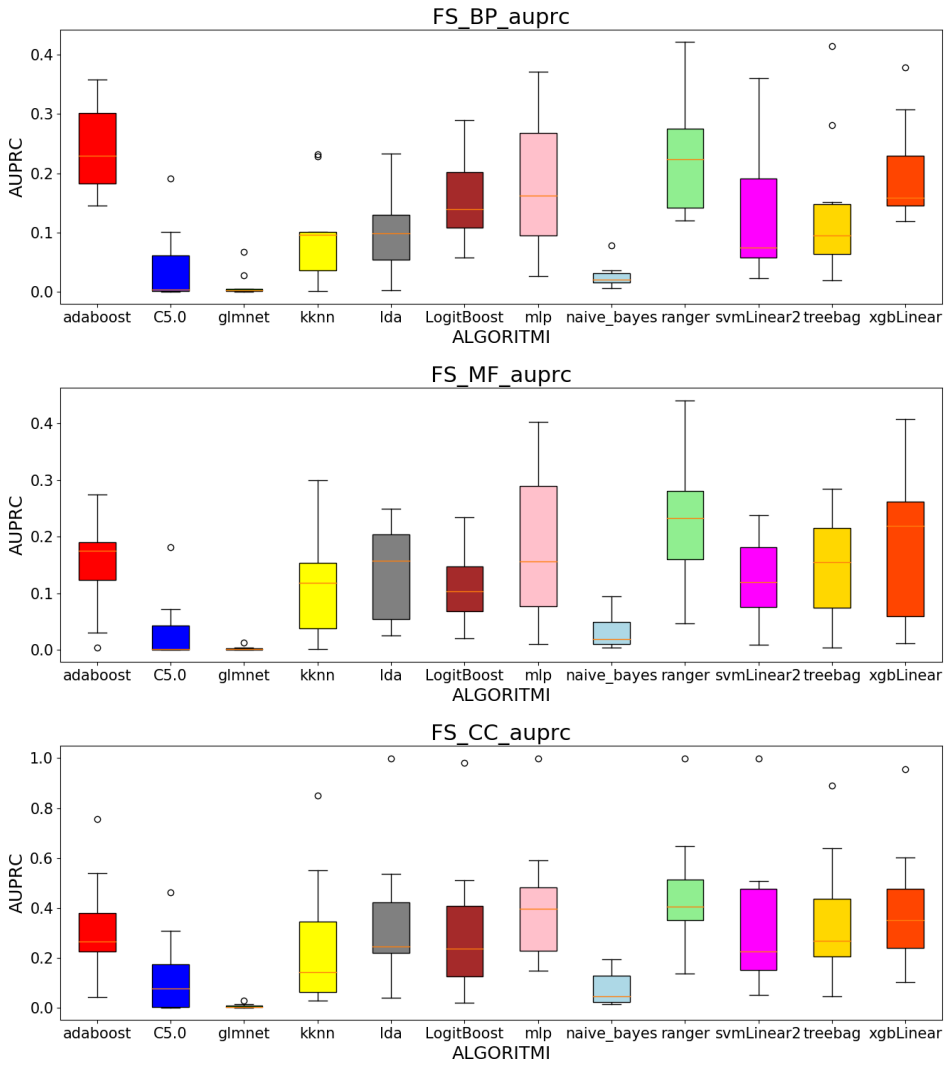

Figura 3.8: Boxplot delle performance di AUPRC per classe dei diversi algoritmi, su un campione di 10 classi, con selezione del prime 100 feature (correlazione di Pearson) e cross-validation a 5 fold, per le ontologie BP, MF, CC. Sono evidenti le performance scadenti di *C5.0*, *glmnet* e *K-NN*, ma anche per l'algoritmo *Naive Bayes*.

3.6.4 Stime dei tempi di calcolo e performance con PCA

L'operazione di generazione delle nuove componenti tramite PCA è stata effettuata per comodità sull'intera matrice delle istanze e non in maniera *unbiased* come per la selezione delle feature, in quanto il *data leakage* può essere trascurato, con una certa approssimazione, essendo l'analisi non supervisionata[16]. Dall'analisi dei risultati della PCA, mostrata nella figura 3.9, emerge che gran parte della varianza può essere spiegata dalle prime componenti ottenute, le quali racchiudono gran parte dell'informazione.

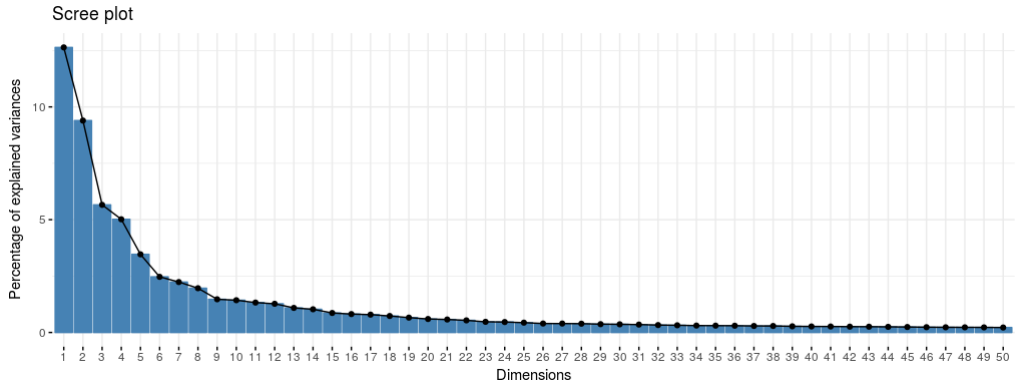

Figura 3.9: La varianza spiegata delle prime componenti. Come si può vedere dal grafico, l'informazione/varianza decresce piuttosto rapidamente.

Per completezza si sono provati a stimare i tempi di calcolo e le performance di alcuni algoritmi, con 3 diversi livelli di varianza spiegata (con cross-validation a 10 fold), e cioè:

- 90% di varianza spiegata: 1000 componenti
- 70% di varianza spiegata: 100 componenti
- 50% di varianza spiegata: 15 componenti

Come si può vedere dalla tabella B.2 in appendice , nella quale sono mostrate le stime, basate sempre sulla selezione di 10 classi per ontologia, le tempistiche diventano particolarmente lunghe sia per quanto riguarda la selezione con 1000 componenti, che con quella a 100 componenti.

Per questo motivo, si è deciso di selezionare le prime 15 componenti, che racchiudono circa il 50% della varianza spiegata.

[16]Le classi sono escluse dall'analisi delle componenti.

Effettuando le stime con questo tipo di selezione, introducendo inoltre una cross-validation a 5 fold, selezionando 10 classi randomicamente, i tempi si abbassano drasticamente, anche rispetto alla selezione di 100 feature con Pearson. Le performance non sono invece altrettanto positive, e anzi mostrano un netto calo generale, sia a livello di AUROC che AUPRC. Le stime sono mostrate nella tabella B.3.

Nelle figure 3.10 e 3.11 sono mostrati i boxplot relativi alle performance di AUROC e AUPRC per gli algoritmi dell'esperimento.

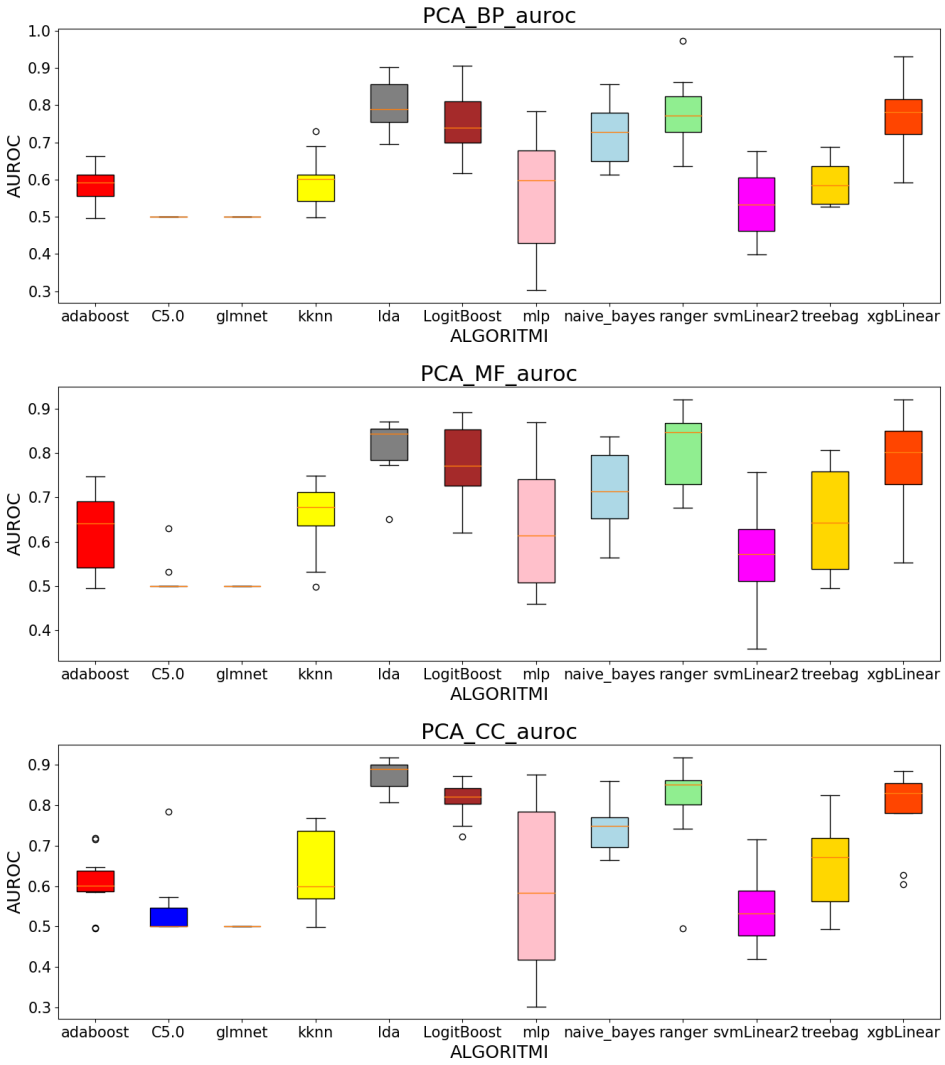

Figura 3.10: Boxplot delle performance di AUROC per classe dei diversi algoritmi, su un campione di 10 classi, con selezione delle prime 15 componenti e cross-validation a 5 fold, per le ontologie BP, MF, CC.

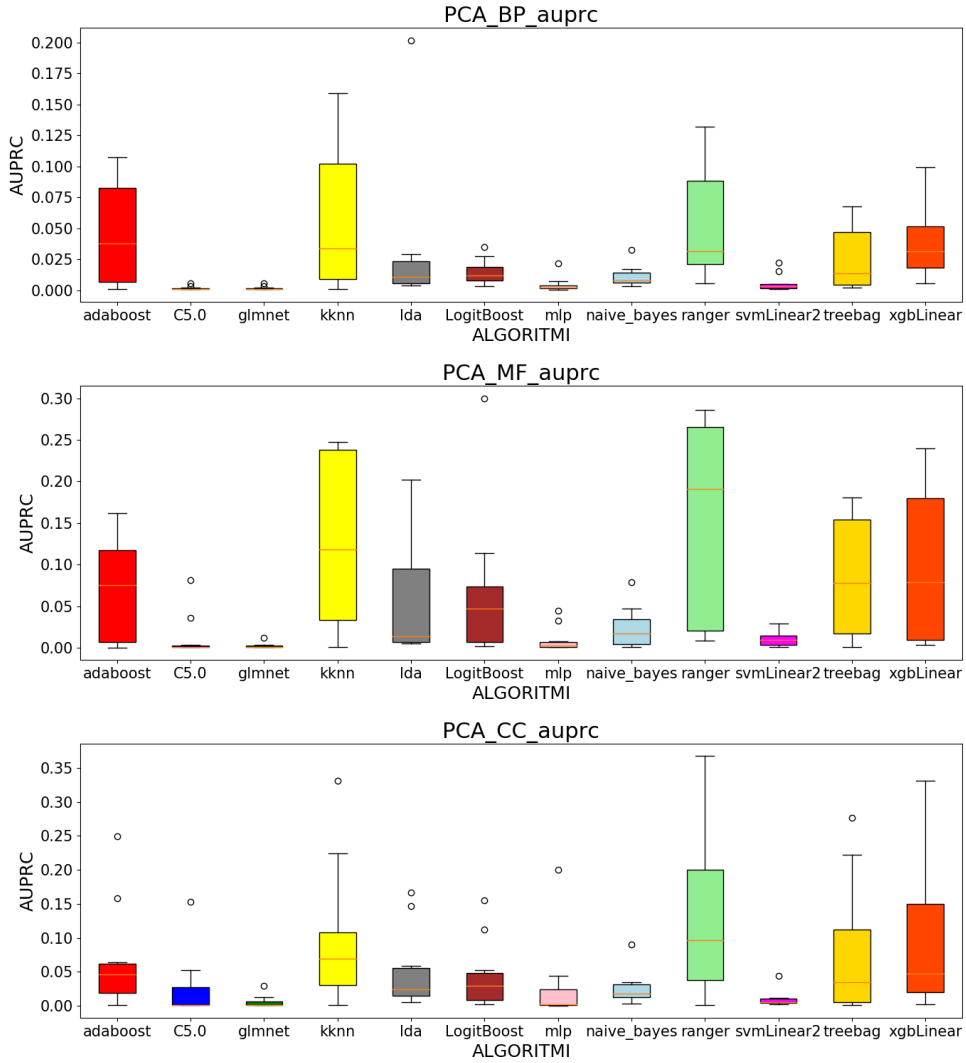

Figura 3.11: Boxplot delle performance di AUPRC per classe dei diversi algoritmi, su un campione di 10 classi, con selezione delle prime 15 componenti e cross-validation a 5 fold, per le ontologie BP, MF, CC.

3.7 Organizzazione degli esperimenti

Si sono divisi gli esperimenti in due fasi principali:

1. Predizioni flat
2. Applicazione del metodo gerarchico TPR-DAG, con GPAV per lo step top-down, sulle predizioni flat.

Data l'analisi precedente dei tempi di calcolo, si è deciso di generare le predizioni flat di tutte le classi delle tre ontologie[17], con due configurazioni principali:

- cross-validation 5 fold + Selezione delle prime 100 feature con Correlazione di Pearson.
- cross-validation 5 fold + Selezione delle prime 15 componenti della PCA.

A seguito della generazione di tutti i risultati, questi verranno quindi corretti tramite il metodo ensemble selezionato. Si valuterà infine la variazione di performance in relazione sia ad ogni singolo algoritmo utilizzato, sia in relazione al metodo di riduzione della dimensionalità utilizzato (selezione delle feature, PCA).

3.7.1 Predizioni flat

La generazione di tutti gli score flat, per entrambi i metodi di riduzione della complessità, ha evidenziato delle performance generalmente migliori per la selezione delle feature effettuta con la correlazione di Pearson. Le performance migliori sono probabilmente da ricondurre al fatto che il tipo di riduzione per la correlazione è *supervisionato* a differenza della PCA.

Nella figura 3.12 sono messi a confronto i risultati degli algoritmi a coppie, con feature selection e con PCA, per la metrica AUROC, tramite dei boxplot. Nella figura 3.13troviamo invece i risultati per la AUPRC, sempre tramite boxplot.

Le differenze maggiori sono evidenti soprattutto per la metrica AUPRC, dove difficilmente la PCA (con selezione della varianza al 50%) riesce a essere competitiva.

[17] Come si è specificato in preccedenza, si sono escluse dal grafo delle ontologie quelle classi che hanno meno di 10 annotazioni.

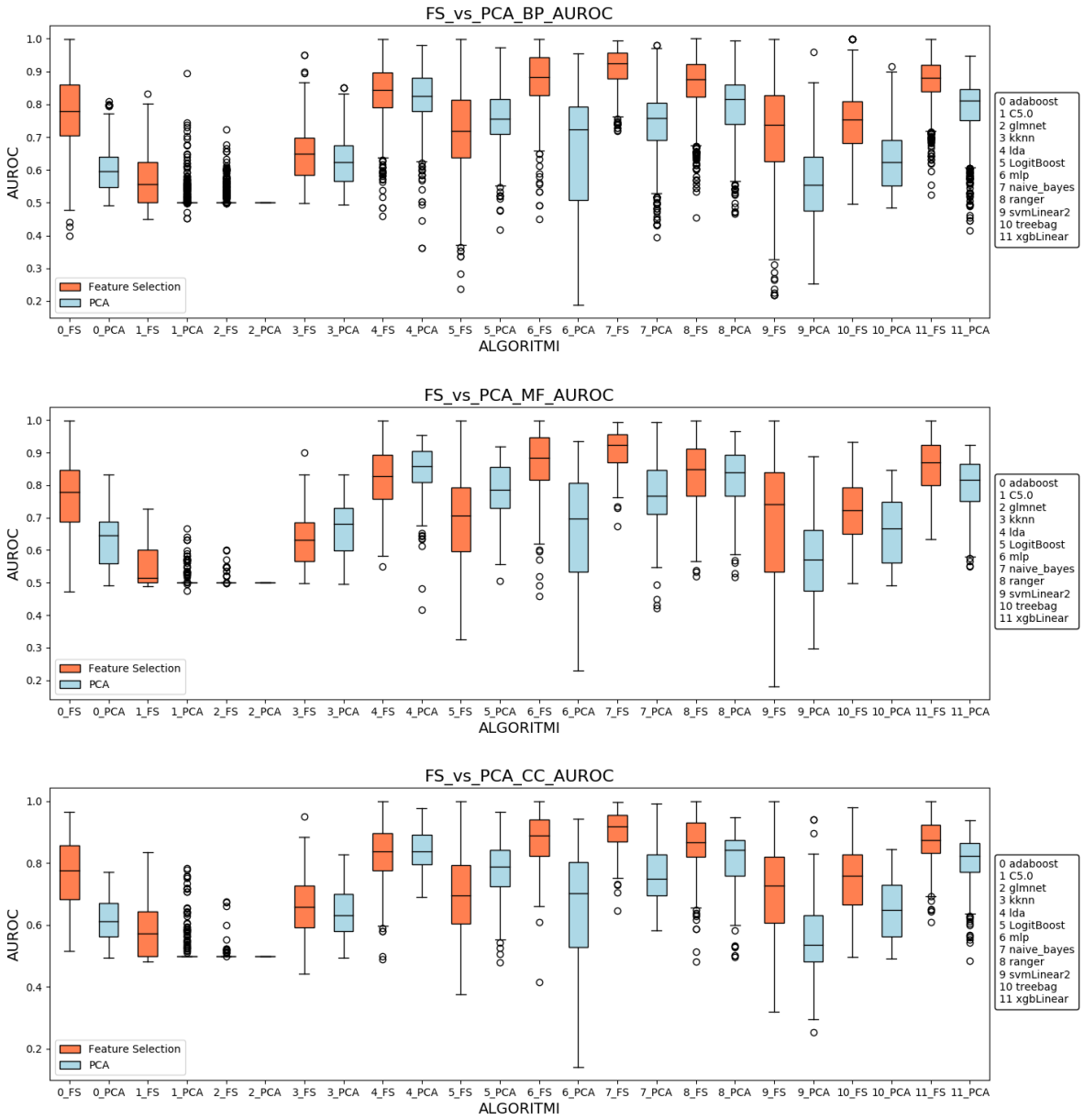

Figura 3.12: I risultati della AUROC per algoritmo, alternando feature selection a PCA. In ordine abbiamo: Adaboost, C5.0, glmnet, knn, lda, Logit Boost, mlp, naive bayes, Random Forest, Svm , tree bag, xgbLinear.

Test di Kolmogorov e Smirnov

Per verificare inoltre se la differenza in performance tra selezione delle feature e PCA è statisticamente significativa, sono stati effettuati dei test di ipotesi sui

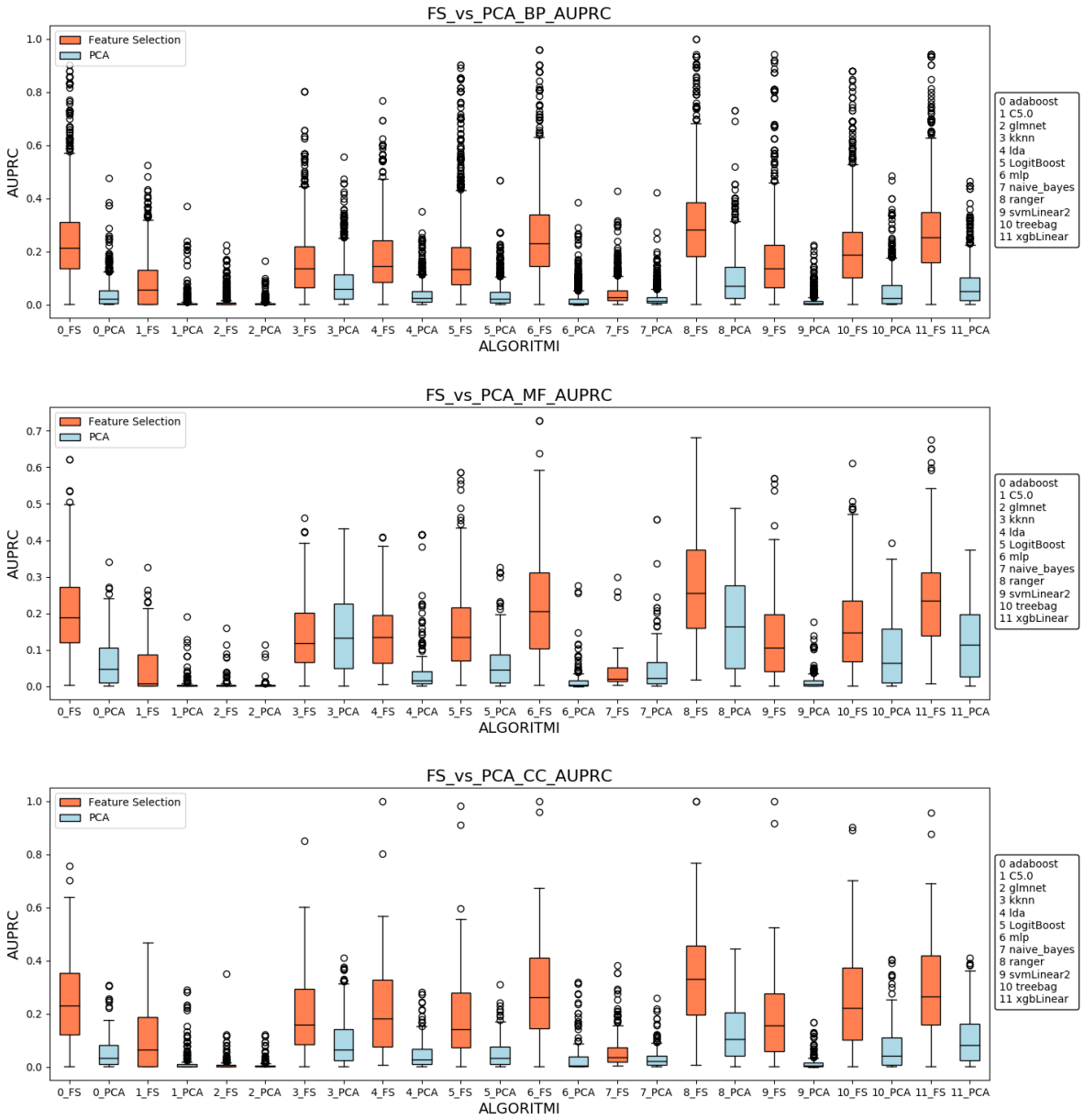

Figura 3.13: I risultati della AUPRC per algoritmo, alternando feature selection a PCA. In ordine abbiamo: Adaboost, C5.0, glmnet, knn, lda, Logit Boost, mlp, naive bayes, Random Forest, Svm , tree bag, xgbLinear

campioni dei risultati ottenuti dai due metodi di riduzione della dimensionalità del problema.

Come test di ipotesi è stato utilizzato quello di Kolmogorov e Smirnov[50], un

test non parametrico e a due code, che ci permette di verificare la probabilità che due campioni provengono dalla stessa popolazione senza fare assunzioni sul tipo di distribuzione.

Dato il livello di significatività del test $a = 0.05$ e come ipotesi nulla H_0 l'appartenenza dei due campioni alla medesima popolazione, sono quindi stati effettuati dei test a due a due sui campioni ottenuti variando il metodo di riduzione della dimensionalità del problema, fissato un algoritmo. Come si può vedere dalle tabelle 3.7 e 3.8, che mostrano rispettivamente i p-value ottenuti per la metrica AUROC e per la metrica AUPRC, i casi in cui l'ipotesi nulla non viene scartata ($p - value >= a$), fanno riferimento prevalentemente all'algoritmo *glmnet*. Questo è dovuto probabilmente ai risultati dell'algoritmo che sono quasi sempre vicini a quelli di un predittore casuale. Un altro algoritmo che genera risultati statisticamente indistinguibili al variare del metodo di riduzione della dimensionalità è *K-NN*, per l'ontologia MF. In tutti gli altri casi si può dire, con una confidenza del 95%, che gli algoritmi producono popolazioni (e quindi risultati) differenti. Consequenzialmente si può affermare che in generale il metodo di selezione delle feature genera performance migliori.

Onto\Algo	**adaboost**	**C5.0**	**glmnet**	**knn**	**lda**	**LogitBoost**
BP	0	0	0	0	0	0
MF	0	0	0,65118	7E-05	0,00337	0
CC	0	0	0,67381	0,02319	0,03078	0
Onto\Algo	**mlp**	**naiveBayes**	**ranger**	**svm**	**treebag**	**xgbLinear**
BP	0	0	0	0	0	0
MF	0	0	0,02632	0	5E-05	0
CC	0	0	0	0	0	0

Tabella 3.7: P-value ottenuti dal test di kolmogorov per la metrica AUROC. Fissato l'algoritmo (colonna) si può identificare il p-value relativo ad una specifica ontologia (riga). Il p-value fa riferimento ai due campioni delle performance (AUROC) ottenuti dallo stesso algoritmo, variando il tipo di riduzione della dimensionalità (Selezione delle feature e PCA).

3.7.2 Predizioni strutturate

Onto\Algo	**adaboost**	**C5.0**	**glmnet**	**knn**	**lda**	**LogitBoost**
BP	0	0	0	0	0	0
MF	0	0	0,73899	0,13797	0	0
CC	0	0	0,94171	0	0	0
Onto\Algo	**mlp**	**naiveBayes**	**ranger**	**svm**	**treebag**	**xgbLinear**
BP	0	0	0	0	0	0
MF	0	0,0023	0	0	0	0
CC	0	0	0	0	0	0

Tabella 3.8: P-value ottenuti dal test di kolmogorov per la metrica AUPRC. Fissato l'algoritmo (colonna) si può identificare il p-value relativo ad una specifica ontologia (riga). Il p-value fa riferimento ai due campioni delle performance (AUPRC) ottenuti dallo stesso algoritmo, variando il tipo di riduzione della dimensionalità (Selezione delle feature e PCA).

Capitolo 4

Discussione dei risultati

Capitolo 5

Conclusioni

Appendices

Appendice A

Codice

```
algorithm <- "ranger"
defGrid <- data.frame(mtry=trunc(sqrt(nfeature)), splitrule="gini",
    min.node.size=1);
# ftp://cran.r-project.org/pub/R/web/packages/ranger/ranger.pdf

algorithm <- "kknn"
defGrid <- data.frame(kmax=19, distance=2, kernel="optimal");
# https://cran.r-project.org/web/packages/kknn/kknn.pdf

algorithm <- "mlp"
defGrid <- data.frame(size=5);
# https://cran.r-project.org/web/packages/RSNNS/RSNNS.pdf

algorithm <- "xgbLinear"
defGrid <- data.frame(nrounds=15, lambda=1, alpha=0, eta=0.3);
# https://cran.r-project.org/web/packages/xgboost/xgboost.pdf (pag
    38)

algorithm <- "LogitBoost"
defGrid <- data.frame(nIter=nfeature);
# https://cran.r-project.org/web/packages/caTools/caTools.pdf (pag
    10)

algorithm <- "treebag"
defGrid <- data.frame(parameter="none");
# https://cran.r-project.org/web/packages/ipred/ipred.pdf
# NOTE: no tuning parameter

algorithm <- "lda"
defGrid <- data.frame(parameter="none");
# https://cran.r-project.org/web/packages/MASS/MASS.pdf
# NOTE: no tuning parameter

```

```
algorithm <- "C5.0"
defGrid <- data.frame(trials=1, model="tree", winnow=FALSE);
# https://cran.r-project.org/web/packages/C50/C50.pdf (pag 2)

algorithm <- "glmnet"
defGrid <- data.frame(alpha=1, lambda=100);
# https://cran.r-project.org/web/packages/glmnet/glmnet.pdf

algorithm <- "naive_bayes"
defGrid <- data.frame(laplace=0, usekernel=FALSE, adjust=1);
# https://cran.r-project.org/web/packages/naivebayes/naivebayes.pdf

algorithm <- "svmLinear2"
defGrid <- data.frame(cost=1);
# https://cran.r-project.org/web/packages/e1071/e1071.pdf

algorithm <- "adaboost"
defGrid <- data.frame(nIter=trunc(sqrt(nfeature)), method="Adaboost.
    M1");
# https://cran.r-project.org/web/packages/fastAdaboost/fastAdaboost.
    pdf
```

Listing A.1: Le configurazioni di default utilizzate per i diversi algoritmi di apprendimento

```
## EXAMPLE TRAINING AND TEST ###

## annotation vector of the current GO class
y <- ann[,i];
indices <- 1:length(y);
positives <- which(y==1);
#stratified partitioning
folds <- do.stratified.cv.data.single.class(indices, positives,
                                            kk=kk, seed=seed);
testIndex <- mapply(c, folds$fold.positives,
                    folds$fold.negatives, SIMPLIFY=FALSE);
model <- vector(mode="list", length=kk);
charpos <- "annotated";
charneg <- "not_annotated";
y[which(y==1)] <- charpos;
y[which(y==0)] <- charneg;
y <- as.factor(y);
## setting caret parameter
fitControl <- trainControl(method="none", classProbs=TRUE,
                           returnData=TRUE, sampling=NULL,
                           seeds=seed,
                           summaryFunction=summaryFunction);
#cross-validation
for(k in 1:kk){
  ## training set
  W.training <- W[-testIndex[[k]], ];
  # model tuning
  model[[k]] <- train(
    x=as.data.frame(W.training),
    y=y[-testIndex[[k]]],
    method=algorithm,
    trControl=fitControl,
    tuneGrid=defGrid,
    tuneLength=1,
    metric=metric
  );
  ## test
  W.test <- W[testIndex[[k]], ];
  ## test model
  model.prob <- predict(model[[k]], newdata=as.data.frame(W.test),
    type="prob");
  ## true labels
  obs <- y[testIndex[[k]]];
  ## computing predicted labels at a given cutoff
  pred <- factor(ifelse(model.prob[[charpos]] >= cutoff, charpos,
    charneg), levels=levels(y));
}
```

Listing A.2: Script di esempio per il train e test dei metodi flat con cross-validation

Appendice B

Tabelle

Algoritmo	**Onto**	**feature**	**AUPRC**	**AUROC**	**TempoMedio**	**TempoTotale**
adaboost	BP	1000	0,2279	0,8429	1,44	1919,73
adaboost	BP	500	0,1969	0,8207	0,50	666,61
adaboost	BP	100	0,1759	0,715	0,05	64,97
adaboost	CC	1000	0,2002	0,8007	1,73	381,74
adaboost	CC	500	0,2032	0,8037	0,47	102,99
adaboost	CC	100	0,1566	0,6931	0,05	11,49
adaboost	MF	1000	0,2121	0,8094	1,30	242,17
adaboost	MF	500	0,1912	0,7661	0,47	88,04
adaboost	MF	100	0,1227	0,6183	0,06	10,29
C5.0	BP	1000	0,27	0,8732	0,06	78,32
C5.0	BP	500	0,2694	0,8686	0,03	43,61
C5.0	BP	100	0,2336	0,7731	0,01	14,24
C5.0	CC	1000	0,2049	0,8612	0,09	19,30
C5.0	CC	500	0,2137	0,8452	0,03	7,51
C5.0	CC	100	0,2067	0,7798	0,01	2,65
C5.0	MF	1000	0,2686	0,8536	0,06	10,42
C5.0	MF	500	0,2767	0,8338	0,03	5,70
C5.0	MF	100	0,2299	0,7299	0,01	1,98
glmnet	BP	1000	0,0083	0,7448	0,08	104,13
glmnet	BP	500	0,0112	0,8176	0,05	67,64
glmnet	BP	100	0,0288	0,8863	0,03	42,72
glmnet	CC	1000	0,011	0,7373	0,09	20,18
glmnet	CC	500	0,0137	0,8113	0,05	10,02
glmnet	CC	100	0,0311	0,9124	0,03	7,07
glmnet	MF	1000	0,0044	0,7148	0,06	10,42

glmnet	MF	500	0,0067	0,7951	0,06	11,28
glmnet	MF	100	0,0252	0,8743	0,04	7,44
kknn	BP	1000	0,0709	0,5618	1,01	1343,01
kknn	BP	500	0,0828	0,5727	0,49	659,49
kknn	BP	100	0,0874	0,564	0,14	190,46
kknn	CC	1000	0,0694	0,5565	1,22	270,36
kknn	CC	500	0,0727	0,5617	0,72	158,38
kknn	CC	100	0,085	0,5666	0,06	12,97
kknn	MF	1000	0,0217	0,5228	0,90	166,66
kknn	MF	500	0,0265	0,5245	0,52	95,98
kknn	MF	100	0,0289	0,5283	0,08	15,00
lda	BP	1000	0,0137	0,5167	0,04	48,95
lda	BP	500	0,0137	0,5167	0,02	22,25
lda	BP	100	0,0137	0,5167	0,01	8,01
lda	CC	1000	0,006	0,5045	0,04	9,72
lda	CC	500	0,006	0,5045	0,01	3,24
lda	CC	100	0,006	0,5045	0,01	1,47
lda	MF	1000	0,0026	0,5005	0,04	6,57
lda	MF	500	0,0026	0,5005	0,01	2,60
lda	MF	100	0,0026	0,5005	0,01	1,12
LogitBoost	BP	1000	0,1018	0,6046	0,66	882,88
LogitBoost	BP	500	0,1092	0,6031	0,17	229,62
LogitBoost	BP	100	0,1748	0,668	0,01	16,91
LogitBoost	CC	1000	0,1095	0,593	1,24	275,07
LogitBoost	CC	500	0,1142	0,6058	0,19	41,99
LogitBoost	CC	100	0,15	0,6633	0,01	3,09
LogitBoost	MF	1000	0,1875	0,6279	0,65	120,65
LogitBoost	MF	500	0,1954	0,6333	0,17	31,00
LogitBoost	MF	100	0,1683	0,6482	0,01	2,36
mlp	BP	1000	0,1612	0,783	0,08	104,13
mlp	BP	500	0,1729	0,812	0,04	49,84
mlp	BP	100	0,167	0,8486	0,02	20,47
mlp	CC	1000	0,1468	0,7221	0,08	17,68
mlp	CC	500	0,1591	0,78	0,04	9,13
mlp	CC	100	0,1695	0,8237	0,01	2,95
mlp	MF	1000	0,181	0,7663	0,07	13,76
mlp	MF	500	0,1712	0,7862	0,04	7,19
mlp	MF	100	0,1584	0,7698	0,01	2,48

naive_bayes	BP	1000	0,1924	0,7537	0,01	12,46
naive_bayes	BP	500	0,1939	0,7467	0,01	11,57
naive_bayes	BP	100	0,1825	0,7393	0,01	8,01
naive_bayes	CC	1000	0,1705	0,7111	0,01	2,06
naive_bayes	CC	500	0,1634	0,7103	0,01	1,92
naive_bayes	CC	100	0,1893	0,7226	0,01	1,62
naive_bayes	MF	1000	0,1766	0,7057	0,01	1,74
naive_bayes	MF	500	0,1725	0,7069	0,01	1,74
naive_bayes	MF	100	0,1815	0,7023	0,01	1,49
ranger	BP	1000	0,1947	0,7029	0,02	25,81
ranger	BP	500	0,1673	0,681	0,01	18,69
ranger	BP	100	0,1479	0,7174	0,01	13,35
ranger	CC	1000	0,1715	0,7141	0,02	4,71
ranger	CC	500	0,1676	0,6938	0,02	5,45
ranger	CC	100	0,1559	0,719	0,01	1,77
ranger	MF	1000	0,2376	0,709	0,01	2,23
ranger	MF	500	0,2633	0,6915	0,01	2,23
ranger	MF	100	0,1717	0,6487	0,01	1,61
svmLinear	BP	1000	0,292	0,8868	0,09	121,04
svmLinear	BP	500	0,281	0,8765	0,06	85,44
svmLinear	BP	100	0,2623	0,8881	0,02	25,81
svmLinear	CC	1000	0,2277	0,8911	0,38	83,39
svmLinear	CC	500	0,2314	0,8961	0,17	36,83
svmLinear	CC	100	0,2419	0,9082	0,04	8,40
svmLinear	MF	1000	0,2398	0,7886	0,02	4,46
svmLinear	MF	500	0,2428	0,8112	0,02	3,60
svmLinear	MF	100	0,2305	0,8192	0,05	8,43
treebag	BP	1000	0,2688	0,9047	0,17	220,72
treebag	BP	500	0,2951	0,8985	0,08	110,36
treebag	BP	100	0,2794	0,8452	0,02	28,48
treebag	CC	1000	0,2464	0,8899	0,16	35,80
treebag	CC	500	0,2596	0,8751	0,08	17,24
treebag	CC	100	0,2583	0,8557	0,02	4,86
treebag	MF	1000	0,2967	0,8607	0,13	24,43
treebag	MF	500	0,2978	0,8483	0,07	12,52
treebag	MF	100	0,2697	0,7833	0,02	3,84
xgbLinear	BP	1000	0,2575	0,9007	0,01	15,13
xgbLinear	BP	500	0,2661	0,8911	0,01	13,35

xgbLinear	BP	100	0,2664	0,8489	0,01	10,68
xgbLinear	CC	1000	0,2407	0,8998	0,05	11,49
xgbLinear	CC	500	0,2378	0,8879	0,01	2,36
xgbLinear	CC	100	0,2502	0,8685	0,01	2,21
xgbLinear	MF	1000	0,282	0,8549	0,01	1,36
xgbLinear	MF	500	0,2742	0,8507	0,01	1,12
xgbLinear	MF	100	0,2671	0,8045	0,01	1,61

Tabella B.1: Le stime per classe (ad eccezione di Tempo Totale, che considera tutte le classi) ottenute dalla selezione delle feature a 5 fold con 3 differenti configurazioni per le feature (100, 500, 1000) su un campione di 10 classi per ontologia. I tempi sono espressi in ore.

Algoritmo	**Onto**	**Var**	**#comp**	**AUPRC**	**AUROC**	**TempoMed**	**TempoTot**
adaboost	BP	90	1000	0,1442	0,8245	3,642	4862,07
adaboost	BP	70	100	0,1695	0,8016	0,485	647,475
adaboost	BP	50	15	0,0857	0,7363	0,212	283,02
adaboost	CC	90	1000	0,1887	0,808	4,348	960,908
adaboost	CC	70	100	0,2059	0,7873	0,759	167,739
adaboost	CC	50	15	0,1152	0,7227	0,306	67,626
adaboost	MF	90	1000	0,1859	0,7783	2,999	557,814
adaboost	MF	70	100	0,1962	0,7844	0,763	141,918
adaboost	MF	50	15	0,1533	0,7213	0,133	24,738
C5.0	BP	90	1000	0,0378	0,545	0,398	531,33
C5.0	BP	70	100	0,0367	0,5328	0,034	45,39
C5.0	BP	50	15	0,0062	0,5098	0,007	9,345
C5.0	CC	90	1000	0,0578	0,5672	0,53	117,13
C5.0	CC	70	100	0,0625	0,571	0,044	9,724
C5.0	CC	50	15	0,0305	0,5356	0,01	2,21
C5.0	MF	90	1000	0,0325	0,5365	0,608	113,088
C5.0	MF	70	100	0,0472	0,5469	0,024	4,464
C5.0	MF	50	15	0,0088	0,5202	0,006	1,116
glmnet	BP	90	1000	0,0049	0,5	0,145	193,575
glmnet	BP	70	100	0,0049	0,5	0,126	168,21
glmnet	BP	50	15	0,0049	0,5	0,005	6,675
glmnet	CC	90	1000	0,0132	0,5	0,461	101,881
glmnet	CC	70	100	0,0132	0,5	0,424	93,704
glmnet	CC	50	15	0,0132	0,5	0,01	2,21
glmnet	MF	90	1000	0,0026	0,5	0,159	29,574

glmnet	MF	70	100	0,0026	0,5	0,459	85,374
glmnet	MF	50	15	0,0026	0,5	0,014	2,604
knn	BP	90	1000	0,1701	0,6747	0,573	764,955
knn	BP	70	100	0,1443	0,6831	0,049	65,415
knn	BP	50	15	0,076	0,6482	0,004	5,34
knn	CC	90	1000	0,1608	0,6947	0,714	157,794
knn	CC	70	100	0,1314	0,6656	0,061	13,481
knn	CC	50	15	0,0835	0,6307	0,007	1,547
knn	MF	90	1000	0,1929	0,7106	0,858	159,588
knn	MF	70	100	0,1874	0,7182	0,039	7,254
knn	MF	50	15	0,1292	0,6952	0,003	0,558
lda	BP	90	1000	0,1988	0,8459	0,113	150,855
lda	BP	70	100	0,1381	0,8859	0,004	5,34
lda	BP	50	15	0,0337	0,8168	0,002	2,67
lda	CC	90	1000	0,2697	0,8493	0,167	36,907
lda	CC	70	100	0,1803	0,8725	0,007	1,547
lda	CC	50	15	0,0638	0,8064	0,003	0,663
lda	MF	90	1000	0,2238	0,883	0,099	18,414
lda	MF	70	100	0,1245	0,8594	0,003	0,558
lda	MF	50	15	0,0706	0,8061	0,001	0,186
mlp	BP	90	1000	0,0607	0,7629	0,183	244,305
mlp	BP	70	100	0,0626	0,7026	0,026	34,71
mlp	BP	50	15	0,0208	0,666	0,017	22,695
mlp	CC	90	1000	0,1301	0,7017	0,246	54,366
mlp	CC	70	100	0,1045	0,6157	0,032	7,072
mlp	CC	50	15	0,049	0,6289	0,02	4,42
mlp	MF	90	1000	0,027	0,7138	0,354	65,844
mlp	MF	70	100	0,0154	0,6251	0,024	4,464
mlp	MF	50	15	0,0139	0,6594	0,014	2,604
svmLinear	BP	90	1000	0,2228	0,8118	0,106	141,51
svmLinear	BP	70	100	0,1333	0,7555	0,017	22,695
svmLinear	BP	50	15	0,0136	0,5497	0,007	9,345
svmLinear	CC	90	1000	0,262	0,8075	0,23	50,83
svmLinear	CC	70	100	0,1595	0,6955	0,027	5,967
svmLinear	CC	50	15	0,0229	0,5534	0,011	2,431
svmLinear	MF	90	1000	0,2567	0,8545	0,162	30,132
svmLinear	MF	70	100	0,1286	0,7354	0,011	2,046
svmLinear	MF	50	15	0,0329	0,6031	0,005	0,93

treebag	BP	90	1000	0,0894	0,664	1,281	1710,135
treebag	BP	70	100	0,1033	0,6849	0,12	160,2
treebag	BP	50	15	0,0572	0,6412	0,015	20,025
treebag	CC	90	1000	0,1175	0,664	2,032	449,072
treebag	CC	70	100	0,1577	0,6879	0,173	38,233
treebag	CC	50	15	0,089	0,6378	0,022	4,862
treebag	MF	90	1000	0,1426	0,6601	1,041	193,626
treebag	MF	70	100	0,1655	0,6715	0,093	17,298
treebag	MF	50	15	0,1071	0,6448	0,011	2,046
xgbLinear	BP	90	1000	0,1855	0,832	0,048	64,08
xgbLinear	BP	70	100	0,1866	0,821	0,02	26,7
xgbLinear	BP	50	15	0,0991	0,7935	0,015	20,025
xgbLinear	CC	90	1000	0,2078	0,8172	0,065	14,365
xgbLinear	CC	70	100	0,2073	0,7934	0,029	6,409
xgbLinear	CC	50	15	0,1168	0,75	0,021	4,641
xgbLinear	MF	90	1000	0,1982	0,8123	0,04	7,44
xgbLinear	MF	70	100	0,1987	0,7825	0,009	1,674
xgbLinear	MF	50	15	0,1297	0,7722	0,009	1,674

Tabella B.2: Le stime per classe (ad eccezione di Tempo Totale, che considera tutte le classi) ottenute con cross-validation a 10 fold, con diversi livelli di selezione della varianza (90%, 70% e 50%) su un campione di 10 classi per ontologia. I tempi sono espressi in ore.

Algoritmo	Onto	AUPRC	AUROC	TempoMedio	TempoTotale
adaboost	BP	0,0447	0,5861	0,0028	3,6718
adaboost	CC	0,066	0,6071	0,0028	0,6093
adaboost	MF	0,0707	0,6205	0,0027	0,4996
C5.0	BP	0,0017	0,5	0,0029	3,9018
C5.0	CC	0,0257	0,5419	0,0026	0,5707
C5.0	MF	0,0128	0,5161	0,0031	0,5705
glmnet	BP	0,0017	0,5	0,0025	3,3391
glmnet	CC	0,0059	0,5	0,0031	0,6764
glmnet	MF	0,0026	0,5	0,005	0,9216
kknn	BP	0,0554	0,5949	0,0109	14,49
kknn	CC	0,0973	0,6333	0,0093	2,0456
kknn	MF	0,1305	0,6555	0,0098	1,8138
lda	BP	0,0317	0,8018	0,0007	0,9604
lda	CC	0,0513	0,8743	0,0008	0,1819
lda	MF	0,0528	0,8124	0,0009	0,1756
LogitBoost	BP	0,0147	0,7486	0,001	1,2772
LogitBoost	CC	0,0441	0,8127	0,001	0,2107
LogitBoost	MF	0,0677	0,7781	0,001	0,178
mlp	BP	0,0045	0,5683	0,0061	8,1541
mlp	CC	0,0298	0,5963	0,0056	1,2439
mlp	MF	0,0099	0,6305	0,0056	1,0506
naive_bayes	BP	0,011	0,7224	0,0007	0,9106
naive_bayes	CC	0,0259	0,7417	0,0007	0,1505
naive_bayes	MF	0,0243	0,7163	0,0007	0,1271
ranger	BP	0,052	0,7826	0,0021	2,7632
ranger	CC	0,1288	0,8092	0,0024	0,522
ranger	MF	0,1522	0,8124	0,0021	0,3854
svmLinear2	BP	0,0058	0,537	0,0027	3,6499
svmLinear2	CC	0,01	0,5413	0,0042	0,9206
svmLinear2	MF	0,0113	0,5697	0,0032	0,6006
treebag	BP	0,0246	0,5897	0,0042	5,5443
treebag	CC	0,0773	0,6521	0,0044	0,974
treebag	MF	0,086	0,6488	0,0043	0,7982
xgbLinear	BP	0,0365	0,7677	0,0029	3,89
xgbLinear	CC	0,0966	0,7919	0,0034	0,7607
xgbLinear	MF	0,0976	0,7863	0,0029	0,5453

Tabella B.3: Le stime per classe (ad eccezione di Tempo Totale, che considera tutte le classi) per PCA con selezione di 15 componenti (50% di varianza spiegata) e cross-validation 5 fold , su un campione di 10 classi per ontologia. I tempi sono espressi in ore.

Bibliografia

[1] Jiang, Y., et al., *An expanded evaluation of protein function prediction methods shows an improvement in accuracy*, Genome Biology 17, anno 2016.

[2] M. Leung, A. Delong, B. Alipanahi, B. Frey, *Machine Learning in Genomic Medicine: A Review of Computational Problems and Data Sets*, Proceeding of the IEEE 104(1), anno 2016.

[3] Giorgio Valentini, *Machine learning methods for gene/protein function prediction*, `https://homes.di.unimi.it/valenti/SlideCorsi/Bioinformatica1617/IntroGFP.pdf`, ultimo accesso Aprile 2018.

[4] *Gene Ontology* `http://www.geneontology.org/page/ontology-documentation`, ultimo accesso Aprile 2018.

[5] *Functional Catalogue* `https://www.helmholtz-muenchen.de/ibis/resourcesservices/genomics/funcat-the-functional-catalogue/index.html`, ultimo accesso Aprile 2018.

[6] *Gene Ontology DAG Relations* `http://geneontology.org/page/ontology-relations`, ultimo accesso Maggio 2018.

[7] Giorgio Valentini, *True Path Rule Hierarchical Ensembles for Genome-Wide Gene Function Prediction*, in IEEE/ACM Transactions on Computational Biology and Bioinformatics, anno 2010.

[8] G. Valentini, *Hierarchical Ensemble Methods for Protein Function Prediction*, ISRN Bioinformatics, anno 2014.

[9] Lipman, DJ; Pearson, WR ,*Rapid and sensitive protein similarity searches*, Science. 227 (4693): 1435–41, anno 1985.

[10] Altschul, Stephen; Gish, Warren; Miller, Webb; Myers, Eugene; Lipman, David , *Basic local alignment search tool*, Journal of Molecular Biology. 215 (3), 403–410, anno 1990.

[11] Vazquez A, Flammini A, Maritan A, Vespignani A. *Global protein function prediction from protein-protein interaction networks.* Nat Biotechnol. 2003 Jun;21(6):697-700. Epub 2003 May 12. PubMed PMID: 12740586.

[12] Bertoni A., Frasca M., Valentini G. (2011) *COSNet: A Cost Sensitive Neural Network for Semi-supervised Learning in Graphs.* In: Gunopulos D., Hofmann T., Malerba D., Vazirgiannis M. (eds) Machine Learning and Knowledge Discovery in Databases. ECML PKDD 2011. Lecture Notes in Computer Science, vol 6911. Springer, Berlin, Heidelberg.

[13] Azran, Arik, *The Rendezvous Algorithm: Multiclass Semi-supervised Learning with Markov Random Walks*, Proceedings of the 24th International Conference on Machine Learning, ICML '07, ACM, New York, NY, USA, anno 2007.

[14] Mostafavi S, Morris Q. *Fast integration of heterogeneous data sources for predicting gene function with limited annotation.*, Bioinformatics; 26(14):1759-1765, anno 2010.

[15] Oliver S., *Guilt-by-association goes global.*, Nature, 2000 Feb 10;403(6770):601-3, PubMed PMID: 10688178.

[16] Valentini G, Armano G, Frasca M, Lin J, Mesiti M, Re M. *RANKS: a flexible tool for node label ranking and classification in biological networks.*, Bioinformatics. 2016 Sep 15;32(18):2872-4. doi: 10.1093/bioinformatics/btw235. Epub 2016 Jun 2. PubMed PMID: 27256314.

[17] Rosenblatt, Frank, *The Perceptron–a perceiving and recognizing automaton.* Report 85-460-1, Cornell Aeronautical Laboratory, anno 1957.

[18] Tsochantaridis, T. Joachims, T. Hofmann, and Y. Altun, *Large margin methods for structured and interdependent output variables*, Journal of Machine Learning Research, vol.6,pp. 1453–1484, anno 2005.

[19] K.Astikainen, L.Holm, E.Pitkanen, S.Szedmak, and J.Rousu, *Towards structured output prediction of enzyme function*, BMC Proceedings, vol. 2, supplement 4, article S2, anno 2008.

[20] G. Valentini, *True path rule hierarchical ensembles for genome-wide gene function prediction*, IEEE/ACM Transac- tions on Computational Biology and Bioinformatics, vol.8, no. 3, pp. 832–847, anno 2011.

[21] Z. Barutcuoglu, R. E. Schapire, and O. G.Troyanskaya, *Hierarchical multi-label prediction of gene function*, Bioinformatics, vol.22,no.7,pp.830–836, anno 2006.

[22] G. Obozinski, G. Lanckriet, C. Grant, M. I. Jordan, and W. S. Noble, *Consistent probabilistic outputs for protein function prediction*, Genome Biology, vol.9, no.1, articleS6, anno 2008.

[23] J. R. Quinlan, *Induction of decision trees*, Machine Learning, vol.1,no.1,pp.81–106, anno 1986.

[24] M. Notaro, M. Schubach, P. N. Robinson e Giorgio Valentini, *Prediction of Human Phenotype Ontology terms by means of hierarchical ensemble methods*, BMC Bioinformatics, vol 18, numero 1, pagine 449, 12 Ottobre, anno 2017.

[25] Cormen T, Leiserson C, Rivest R, RL S. *Introduction to Algorithms*. Boston: MIT Press; anno 2009.

[26] Burdakov O., Sysoev O., Grimvall A., Hussian M. *An O(n2) Algorithm for Isotonic Regression.*, Large-Scale Nonlinear Optimization. Nonconvex Optimization and Its Applications, vol 83. Springer, anno 2006.

[27] *Isotonic Regression*, `https://en.wikipedia.org/wiki/Isotonic_regression`, Wikipedia, ultimo accesso maggio 2018.

[28] Oleg Burdakov, Anders Grimvall and Oleg Sysoev, *Data Preording in Generalized PAV algorithm for monotonic regression*, Journal of Computational Mathematics, Vol. 24, No. 6, 771–790, anno 2006, .

[29] Hodgkin, Jonathan and Horvitz, H. Robert and Jasny, Barbara R. and Kimble, Judith, *C. elegans: Sequence to Biology*, American Association for the Advancement of Science, `http://science.sciencemag.org/content/282/5396/2011`, vol. 282, anno 1998.

[30] Szklarczyk D, Franceschini A, Wyder S, *STRING v10: protein–protein interaction networks, integrated over the tree of life*, Nucleic Acids Research, anno 2015.

[31] Hechenbichler K. and Schliep K.P., *Weighted k-Nearest-Neighbor Techniques and Ordinal Classification*, Discussion Paper 399, SFB 386, Ludwig-Maximilians University Munich `http://www.stat.uni-muenchen.de/sfb386/papers/dsp/paper399.ps`, anno 2004.

[32] Dettling and Buhlmann, *Boosting for Tumor Classification of Gene Expression Data*, disponibile sulla pagina `http://stat.ethz.ch/~dettling/boosting.html`, anno 2002.

[33] Hastie T., Tibshirani R., Friedman J., *The Elements of Statistical Learning*, Section 4.3, p.106-119, anno 2008.

[34] Chen, Tianqi and Guestrin, Carlos, *XGBoost: A Scalable Tree Boosting System*, Proceedings of the 22Nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, KDD '16, editore ACM, anno 2016.

[35] Quinlan, J. R. *C4.5: Programs for Machine Learning* Morgan Kaufmann Publishers, anno 1993.

[36] Xu, R., Nettleton, D. e Nordman, D.J., *Case-specific random forests*, Journal of Computational and Graphical Statistics, anno 2014.

[37] Dreyfus, Stuart E. *Artificial neural networks, back propagation, and the Kelley-Bryson gradient procedure*. Journal of Guidance, Control, and Dynamics. 13 (5): 926–928, anno 1990.

[38] Boser, Bernhard E.; Guyon, Isabelle M.; Vapnik, Vladimir N., *A training algorithm for optimal margin classifiers*. Proceedings of the fifth annual workshop on Computational learning theory – COLT, p. 144, anno 1992.

[39] Leo Breiman , *Bagging Predictors*, Machine Learning 24, 123-40 anno1996.

[40] Freund, Y. and Schapire, R.E. *Experiments with a new boosting algorithm*, In Proceedings of the Thirteenth International Conference on Machine Learning, pp. 148–156, Morgan Kaufmann, anno 1996.

[41] Langley, P., W. Iba, & K. Thompson, *An analysis of Bayesian classifiers*, da Proceedings, Tenth National Conference on Artificial Intelligence (pp. 223–228). Menlo Park, CA: AAAI Press, anno 1992.

[42] Friedman, J., Hastie, T. and Tibshirani, R. *Regularization Paths for Generalized Linear Models via Coordinate Descent*, `https://web.stanford.edu/~hastie/Papers/glmnet.pdf` Journal of Statistical Software, Vol. 33(1), 1-22 Feb anno 2010.

[43] R Core Team, *R: A Language and Environment for Statistical Computing*, `https://www.R-project.org/`, R Foundation for Statistical Computing, anno 2018.

[44] Nicolò Cesa-Bianchi, *Dispense del corso di Metodi Statistici per l'Apprendimento: Cross-validation*, `http://homes.dsi.unimi.it/~cesabian/MSA/Note/4-cv.pdf`, ultimo accesso maggio 2018.

[45] *HEMDAG*, `https://www.rdocumentation.org/packages/HEMDAG/versions/1.1.1/topics/stratified.cross.validation`, ultimo accesso Gennaio 2018.

[46] Nicolò Cesa-Bianchi, *Dispense del corso di Metodi Statistici per l'Apprendimento: Riduzione dimensionalità.*, `http://homes.dsi.unimi.it/~cesabian/MSA/Note/17-svd.pdf`, ultimo accesso maggio 2018.

[47] Christophe Ambroise, Geoffrey J. McLachlan, *Selection bias in gene extraction on the basis of microarray gene-expression data*, `http://www.pnas.org/content/99/10/6562`, PNAS May 14, anno 2002.

[48] Sheldon M. Ross, *Introduzione alla statistica, 2ª ed.*, Maggioli Editore, anno 2014.

[49] *OPENCV* `https://docs.opencv.org/3.1.0/d1/dee/tutorial_introduction_to_pca.html`, ultimo accesso maggio 2018.

[50] Ross, S.M., *Introduction to Probability and Statistics for Engineers and Scientists*, Elsevier Science, anno 2004.