UNIVERSITÀ DEGLI STUDI DI NAPOLI
"FEDERICO II"

Scuola Politecnica e delle Scienze di Base

Area Didattica di Scienze Matematiche Fisiche e Naturali

Dipartimento di Fisica "Ettore Pancini"

Laurea Triennale in Fisica

Tecniche di intelligenza computazionale per feature selection

Relatori:
Dott.ssa Autilia Vitiello

Candidato:
Davide Zaccarella
Matr. $N85001455$

Anno Accademico 2020/2021

Indice

Introduzione

La feature selection rappresenta uno dei concetti fondamentali nel machine learning ed ha un impatto notevole sulle performance di un modello.
Essa consiste nella selezione di quelle variabili che contribuiscono maggiormente alla predizione di una variabile o dell'output di cui si è interessati.
La possibilità di individuare tra le features quelle rilevanti, irrilevanti e ridondanti rappresenta un grande vantaggio per qualsiasi applicazione.
Difatti, la presenza di features inutili ai fini del problema può portare a diverse difficoltà nell'algoritmo, quali ad esempio l'elevato costo computazionale.
Oltre quindi a ridurre certamente i tempi di calcolo, talvolta, il metodo può portare anche a risultati dall'accuracy migliorata.
In questo lavoro di tesi, la feature selection viene affrontata come problema di ottimizzazione risolto attraverso l'applicazione di un algoritmo genetico. Il metodo implementato è di tipo wrapper, pertanto, dipendente dall'algoritmo di classificazione. Nel mio lavoro di tesi, il Multi-layer Perceptron è scelto come algoritmo di classificazione. L'approccio implementato è testato usando vari dataset e valutando le prestazioni sia in termini di tempi di esecuzione che di possibili miglioramenti dell'accuratezza. Inoltre, la sessione sperimentale coinvolge la comparazione tra il modello di classificazione appreso usando tutte le feature contro lo stesso modello appreso usando solo l'insieme di feature selezionate dall'approccio di feature selection implementato.
Tutti gli algoritmi sono stati sviluppati in Python, sfruttando il modulo DEAP per l'algoritmo genetico ed il modulo sklearn per la classificazione con le reti neurali.
Nel Capitolo 1 sono evidenziate le caratteristiche del metodo di feature selection e viene descritto il principio di funzionamento delle reti neurali.
Nel Capitolo 2 viene approfondito l'Algoritmo Genetico.
Nel Capitolo 3 è mostrata l'implementazione dell'Algoritmo Genetico per la feature selection.
Nel Capitolo 4 sono brevemente presentati i datasets su cui è stata effettuata la sperimentazione e ne vengono illustrati i risultati.

Capitolo 1

Feature selection

1.1 Che cos'è il machine learning

Il Machine Learning è un sottoinsieme dell'intelligenza artificiale che si occupa di creare sistemi che apprendono o migliorano le performance,in base ai dati che utilizzano.
Le diverse modalità di apprendimento di un algoritmo definiscono la differenza principale tra machine learning supervisionato,non supervisionato e per rinforzo.
Per il primo, vengono forniti al modello possibili input e rispettivi output predefiniti, così da realizzare degli esempi guida da cui generalizzare le regole migliori per la predizione.
Per il secondo, invece, è il modello stesso a cercare una struttura negli input forniti, privi di etichette e di corrispondenti output predefiniti.
Nell'apprendimento per rinforzo il modello migliora le sue performance in base alle interazioni con l'ambiente. Tuttavia a differenza del machine learning supervisionato il feedback di tali interazioni non è legato alla corretta etichetta o valore della predizione, bensì viene misurato da una "funzione di premio", che risulta da massimizzare con un approccio trial and error.
A seconda dell'output desiderato è possibile categorizzare i compiti dell'apprendimento e sceglierne la modalità più adatta.

1.2 Problema di classificazione

Per i problemi di classificazione, il machine learning è supervisionato grazie ad un training set di dati, cioè un insieme di dati che ha la funzione di allenare il modello ad individuare le regole per la predizione degli output, classificati in due (classificazione binaria) o più etichette. I passi fondamentali sono:

- scelta del training set al fine di rappresentare al meglio il contesto reale che si vuole classificare;
- rapresentazione della funzione di decisione nello spazio delle feature;
- scelta della struttura del classificatore e del conseguente algoritmo applicativo;
- prova dell'algoritmo sul training set e taratura dei parametri di controllo tramite validazione;
- valutazione dell'algoritmo su un nuovo set di istanze (test set) per stimarne l'accuratezza;

Nel caso in cui, a priori, non sono noti i label della classificazione il problema è risolto dall'unsupervised learning e prende il particolare nome di clustering.

1.3 Preparazione dei dati

Come si mostra in Figura 1.1, l'apprendimento automatico in qualsiasi delle sue forme necessita di una fase preliminare in cui i dati vengono prima estratti da un dataset (data retrieval), quindi preparati(data preparation), attraverso tecniche specifiche, ad essere gli input del modello.

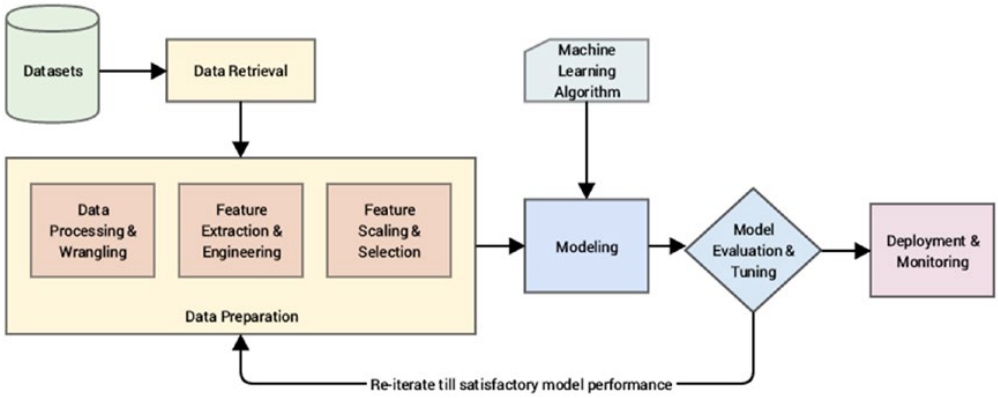

Figura 1.1: Schema generale[4]

Nel dettaglio, la preparazione dei dati può essere suddivisa in varie categorie fondamentali.

- Data Processing & Wrangling:

  cleansing: pulizia dei dati (ad esempio in caso di valori mancanti);

  wrangling: conversione e mappatura ad un formato richiesto dal modello;

  editing: rilevamento e gestione di errori ed outlier;

  reduction: riduzione del numero di dati in base all'appartenenza ad un'ipotetica distribuzione statistica;

- Feature extraction & Engineering: estrazione delle variabili a seconda dei differenti tipi di dati (numerical, categorical, image, text data)

- Feature selection & scaling: selezione delle variabili e bilanciamento dei valori assunti da queste per poter comparare anche variabili con range di valori molto diversi;

1.4 Feature selection

1.4.1 Definizione e metodo

Una volta estratte le variabili dai datasets, soprattutto per quelli di grandi dimensioni, risulta fondamentale identificarne quali siano rilevanti, quali non aggiungano ulteriori informazioni (ridondanti) e quali non influiscano (irrilevanti) nel modello predittivo. Dunque, la feature selection può definirsi come la selezione di M variabili a partire da un insieme di taglia N, con M<N, tale che il valore di una funzione specifica sia ottimizzato.
Una definizione più forte la identifica come quel processo di selezione volto a trovare il più piccolo sottoinsieme di variabili che sono necessarie e sufficienti per la predizione. Da tali definizioni si intuisce il vantaggio più importante offerto dalla feature selection, ovvero la riduzione della dimensionalità dello spazio delle variabili di input del modello, e quindi la conseguente riduzione nel tempo di computazione dei risultati finali della predizione.
La bontà della selezione è stabilita a partire da due criteri: che l'accuracy della predizione non diminuisca significativamente e che la "class distribution" selezionata sia quanto più simile a quella con tutte le variabili.
A partire da un dataset di N variabili, in cui si associa, nel risultato finale, ad ognuna il valore logico alto se rilevante e quello basso se ridondante o irrilevante, scegliere il migliore sottoinsieme tra le 2^N possibili combinazioni può essere particolarmente costoso a livello computazionale. Per questo motivo esistono metodi

coadiuvati da un criterio di stop. In generale, lo schema di un qualsiasi metodo di feature selection è il seguente:

- procedura di generazione del sottoinsieme candidato;
- funzione di valutazione del sottoinsieme;
- criterio di stop;
- procedura di validazione;

Più precisamente, la procedura di validazione può anche essere categorizzata al di fuori del metodo di feature selection ma è comunque una pratica indispensabile e necessaria. Essa viene effettuata su datasets artificiali oppure reali. Nel primo caso, il metodo è applicato su un training set creato con variabili rilevanti note, in aggiunta ad altre ridondanti oppure irrilevanti, quindi solo l'uguaglianza tra l'output della selezione e le variabili note in partenza definisce il metodo come valido. Nell'altro caso si usano datasets di cui non sono note le variabili rilevanti e quindi il sottoinsieme selezionato viene testato in base alla propria accuracy in un numero signficativo di classificatori. Tuttavia, la possibilità che si presentino bias vieta alcune combinazioni metodo-classificatore ed inoltre è necessario un numero significativo di datasets.

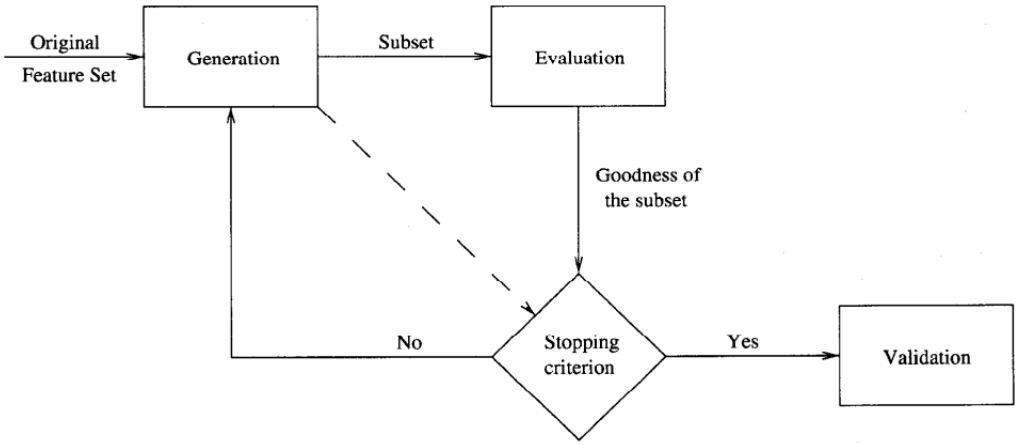

Figura 1.2: Feature selection e validation
[2]

Senza un criterio d'arresto, la feature selection proseguirebbe indefinitamente. Tale criterio può essere relativo alla procedura di generazione e quindi imporre l'arresto ad un numero predefinito di variabili oppure ad un numero predefinito di

iterazioni. Tuttavia, il loop può estinguersi anche in relazione alla funzione di valutazione: se l'aggiunta o la rimozione di una variabile non produce un sottoinsieme con accuracy migliore oppure se viene ottenuto un sottoinsieme che massimizza una certa funzione.

1.4.2 Procedura di generazione

Per generare il sottoinsieme di variabili candidato esistono diversi approcci.
La generazione di tipo completo valuta tutte le possibili combinazioni delle variabili e dunque si muove in uno spazio 2^N-dimensionale. Tale ricerca può comunque essere limitata da un parametro di stop predefinito, mentre la possibilità di ottenere un valore ottimale è garantita dalla capacità di backtracking di tale procedura, attraverso apposite tecniche (findbest) che individuano il sottoinsieme ottimale.
La generazione di tipo euristico prevede che ad ogni iterazione le rimanenti variabili ancora da selezionare (rimuovere) vengano selezionate (rimosse). L'ordine dello spazio di ricerca è al più pari a O(N^2) e questo riduce notevolmente il tempo di tale procedura di generazione rispetto alle altre.

1.4.3 Funzione di valutazione

Un sottoinsieme è definito ottimale dal valore che assume la funzione di valutazione, quindi a differenti funzioni possono corrispondere diversi sottoinsiemi ottimali. Una prima distinzione per i metodi di selezione risiede nel fatto che siano indipendenti dall'algoritmo che userà il sottoinsieme selezionato (filter), oppure che tale algoritmo sia la funzione di valutazione (wrapper).
La funzione di valutazione è divisa in 5 categorie in base alle misure che performa:

- misure di distanza, particolarmente utili per un problema di classificazione in due classi. Date due variabili X e Y, se X induce una differenza maggiore rispetto a Y nella probabilità condizionata tra le due classi, allora X è selezionata. Se tale differenza è nulla, X e Y sono variabili indistinguibili

- misure di informazione, determinano il guadagno in termini di informazione, ovvero una feature viene scelta rispetto ad un'altra se porta ad una riduzione dell'entropia

- misure di dipendenza, con le quali una variabile X viene scelta rispetto ad una Y se la correlazione tra X e la classe C é maggiore di quella tra Y e C. Talvolta, si può usare questo tipo di funzione di valutazione per valutare la correlazione di una variabile con le rimanenti per discriminarne la ridondanza.

Tali misure hanno generalità totale di utilizzo ed un tempo di computazione basso. Tuttavia da queste non può essere nota l'accuracy dato che sono indipendenti dall'algoritmo di predizione. Al contrario, le

- misure di "classifier error rate" (per i metodi wrapper) quantificano direttamente il tasso di errori nella predizione. Il loro tempo di computazione è elevato e conseguentemente risulta elevata l'accuracy rispetto ai tipi di misure elencati prima. La generalità viene meno dato che le misure sono proprie di un determinato classificatore.

1.5 Reti neurali

1.5.1 Modello del neurone

Tra i classificatori più utilizzati ci sono le reti neurali. Esse sfruttano come principio il modello di funzionamento del neurone: multipli segnali di input arrivano ai dendriti, sono poi convogliati all'interno del corpo della cellula e se il segnale accumulato supera una certa soglia allora viene generato ed inviato un segnale di output. L'algoritmo del percettrone prevede l'apprendimento automatico dei pesi il cui prodotto con gli input rappresenta il valore da comparare con la threshold per discriminare se un neurone "spara" il segnale di output oppure no, ovvero negli usi pratici, l'appartenenza ad una classe.

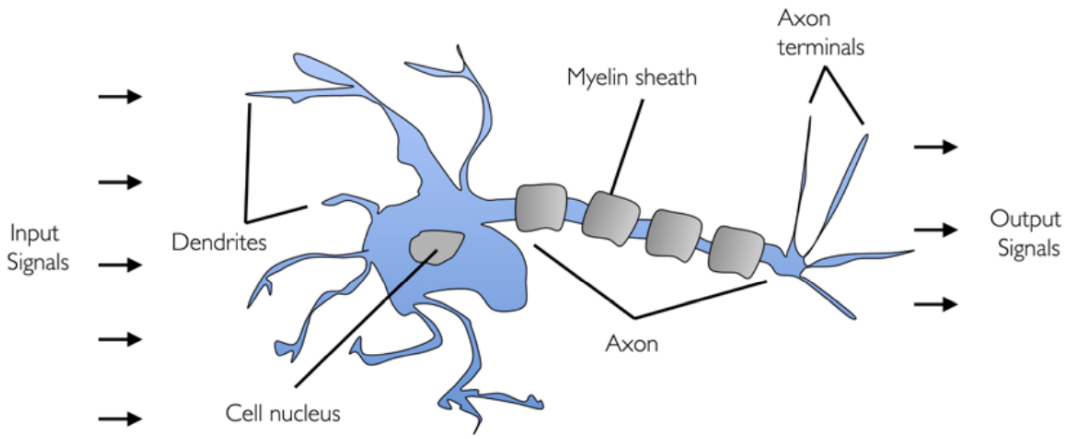

Figura 1.3: Schema di funzionamento del neurone[3]

1.5.2 Multi-Layer Perceptron

I modelli di percettrone possono essere composti da una struttura semplice ad unico layer oppure da una più complessa, definita Multi-Layer Perceptron (MLP), da almeno tre layer di nodi, come mostrato in Figura 1.4: input, hidden e output layer, in cui ogni nodo è un neurone.
L'MLP si differenzia dal percettrone lineare per la presenza di più layer, per l'uso di una funzione di attivazione non lineare nell'aggiornamento dei pesi, e quindi per la classificazione anche in caso di classi non linearmente separabili.

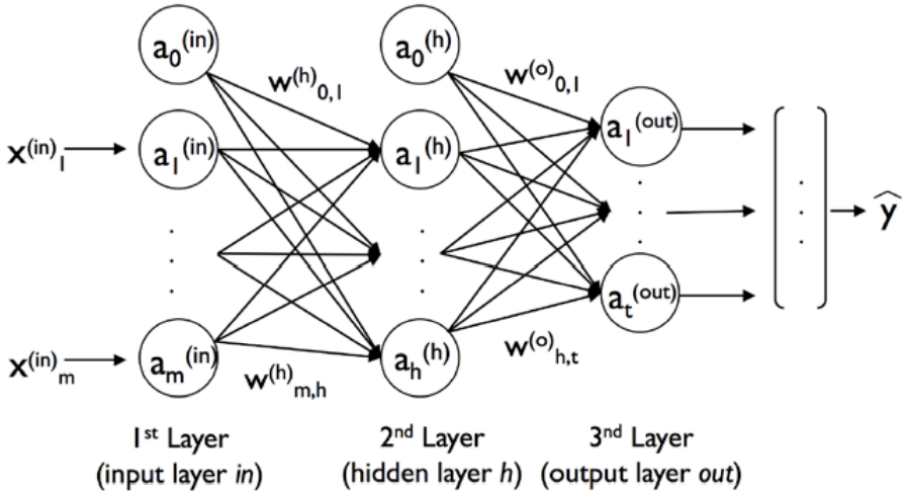

Figura 1.4: Multi-Layer Perceptron[3]

Ciascuna unità del layer *l* è connessa ad una del layer *l+1* attraverso un coefficiente di peso. Denotiamo quindi, con $W^{(h)}$ la matrice dei pesi che connette gli input all'hidden layer e con $W^{(o)}$ la matrice dei pesi tra hidden e output layer.
L'output viene comparato ad un vettore in rappresentazione one-hot cioè che presenta il valore 1 alla m-sima componente dove *m* è il label della classe, e 0 per le restanti componenti. Tale scelta permette di risolvere problemi di classificazione con un numero di classi generalizzato.
Il processo di calcolo dell'output finale di un MLP può essere scomposto in 3 tappe principali:

1. forward propagation, dagli input fino a generare un output;
2. in base all'output generato, si calcola l'errore da minimizzare utilizzando la funzione di costo
3. backpropagation dell'errore

Esse vengono ripetute per un certo numero di epoche per generare l'output su cui applicare la threshold e quindi ottenere il label della classe in rappresentazione one-hot.

Forward propagation

Questa tappa è caratterizzata dal fatto che ogni layer serve come input di quello successivo.
In particolare per la prima unità dell'hidden layer $z_1 = a_0^{in} * w_0^h + ... + a_m^{in} * w_n^h$ va applicata la funzione di attivazione Φ, che nel caso dell'MLP è la sigmoide, mostrata in Figura 1.5.
Ripetendo tale operazione per tutte le unità dell'hidden layer, in una forma più compatta risulta: $Z^{(h)} = A^{(in)}W^{(h)}$, dove Z^h è una matrice *n x d*, dove n rappresenta il numero di input e d il numero di nodi dell'hidden layer; $A^{(in)}$ è la matrice degli n input con m features, mentre W^h è la matrice dei pesi *m x d*.
L'attivazione sarà quindi una matrice *n x d*: $A^{(h)} = \Phi(Z^{(h)})$ ed è necessaria per computare la matrice di output $Z^{(o)} = A^{(h)}W^{(o)}$ che è di dimensione *n x t*, dove t rappresenta il numero di unità nell'output layer. Infine, su tale matrice va nuovamente applicata la funzione di attivazione: $A^{(o)} = \Phi(Z^{(o)})$

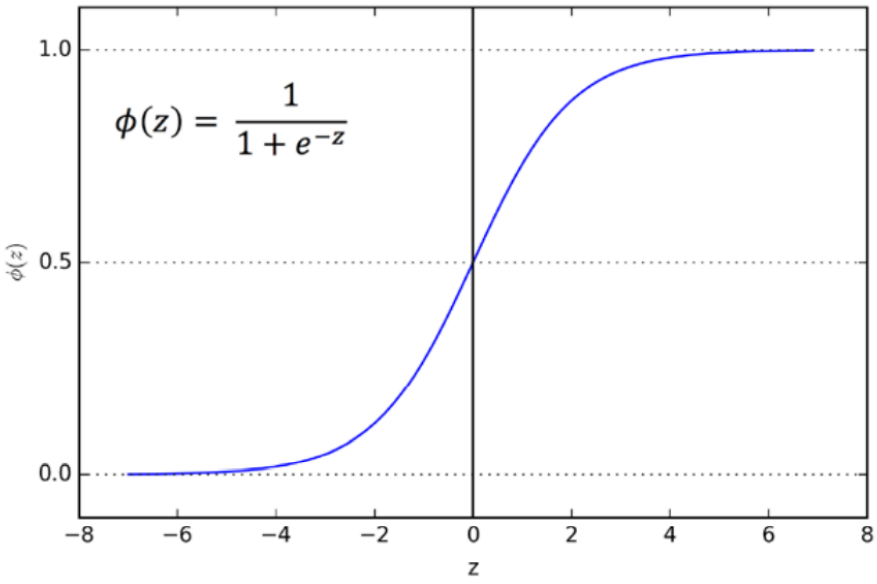

Figura 1.5: Funzione sigmoide

Minimizzazione funzione di costo

I pesi vengono aggiornati tramite steps nella direzione opposta a quella del gradiente della funzione di costo J(**W**), fino a trovare i pesi ottimali che la minimizzano. La funzione di costo J(**W**) è la seguente:

$$J(\boldsymbol{W}) = -\left[\sum_{i=1}^{n}\sum_{j=1}^{t} y_j^{[i]} \log\left(a_j^{[i]}\right) + \left(1 - y_j^{[i]}\right)\log\left(1 - a_j^{[i]}\right)\right] + \frac{\lambda}{2}\sum_{l=1}^{L-1}\sum_{i=1}^{u_l}\sum_{j=1}^{u_{l+1}}\left(w_{j,i}^{(l)}\right)^2 \tag{1.1}$$

dove $a_j^{[i]}$ è l'attivazione per gli n-input alle t-unità di output, mentre $y_j^{[i]}$ rappresenta il vettore dei label di classe per gli n-input scritto in rappresentazione one-hot con t componenti. Il termine finale viene chiamato "termine di regolarizzazione", utile per fronteggiare il problema dell'overfitting. Esso racchiude la somma di tutti i pesi di un l-esimo layer di u_l unità, e dipende dal parametro di regolarizzazione λ (forza di regolarizzazione) che è un iperparametro del metodo: maggiore il suo valore e maggiore sarà il penalty pagato dai valori estremi dei pesi.
A questo punto per minimizzare la funzione occorre calcolare le derivate parziali di J(W) rispetto ad ogni peso di ciascun layer, dove W è il tensore composto dalle matrici $W^{(h)}$ e $W^{(o)}$.
Calcolarne le derivate parziali utilizzando la chain-rule per le funzioni composte risulta molto complesso da gestire e dall'elevato costo computazionale, quindi è preferita la tecnica della backpropagation.

Backpropagation

Il vantaggio di tale tecnica sta nel fatto di procedere in maniera inversa e dunque moltiplicare il vettore d'errore finale per la matrice dei pesi degli output, così da riottenere un vettore, facilmente gestibile rispetto ai numerosi jacobiani richiesti dalla chain rule. A partire da tale vettore, lo schema prosegue iterativamente all'indietro.
Il primo passo è calcolare la matrice d'errore dell'output layer: $\delta^{(out)} = a^{(out)} - y$ dove y raprresenta il vero label della classe.
L'errore dell'hidden layer:

$$\boldsymbol{\delta}^{(h)} = \boldsymbol{\delta}^{(\text{out})}\left(\boldsymbol{W}^{(\text{out})}\right)^T \odot \frac{\partial\phi\left(z^{(h)}\right)}{\partial z^{(h)}} \tag{1.2}$$

Più nel dettaglio, $\left(W^{(out)}\right)^T$ è la trasposta(di dimensione *t x h*, dove t è il numero di unità dell'output layer e h il numero di hidden layer) della matrice dei pesi tra hidden e output layer. Essa, moltiplicata con la matrice d'errore $\delta^{(out)}$, *n x t*-dimensionale restituisce una matrice $\delta^{(h)}$ *n x h* a cui si applica con il simbolo $\odot$

la moltiplicazione di Hadamard con la derivata della sigmoide. Quindi una volta ottenuti i termini δ possiamo scrivere:

$$\frac{\partial}{\partial w_{i,j}^{(\text{out})}} J(\boldsymbol{W}) = a_j^{(\text{h})} \delta_i^{(\text{out})} \tag{1.3}$$

$$\frac{\partial}{\partial w_{i,j}^{(h)}} J(\boldsymbol{W}) = a_j^{(in)} \delta_i^{(h)} \tag{1.4}$$

Bisogna perciò sommare le derivate parziali di ciascun nodo in ogni layer ($\Delta_{i,j}^{(l)}$). E tale somma va effettuata per ciascun sample del training set. In maniera compatta:

$$\Delta^{(h)} = \Delta^{(h)} + \left(\boldsymbol{A}^{(in)}\right)^T \delta^{(h)} \tag{1.5}$$

$$\Delta^{(\text{out})} = \Delta^{(\text{out})} + \left(\boldsymbol{A}^{(h)}\right)^T \delta^{(\text{out})} \tag{1.6}$$

A questi può aggiungersi il termine di regolarizzazione:

$$\Delta^{(l)} := \Delta^{(l)} + \lambda^{(l)} \tag{1.7}$$

Una volta computati i gradienti di ciascun layer, si possono quindi aggiornare i pesi nella direzione opposta a quella del gradiente, per ogni layer l:

$$\boldsymbol{W}^{(l)} := \boldsymbol{W}^{(l)} - \eta \Delta^{(l)} \tag{1.8}$$

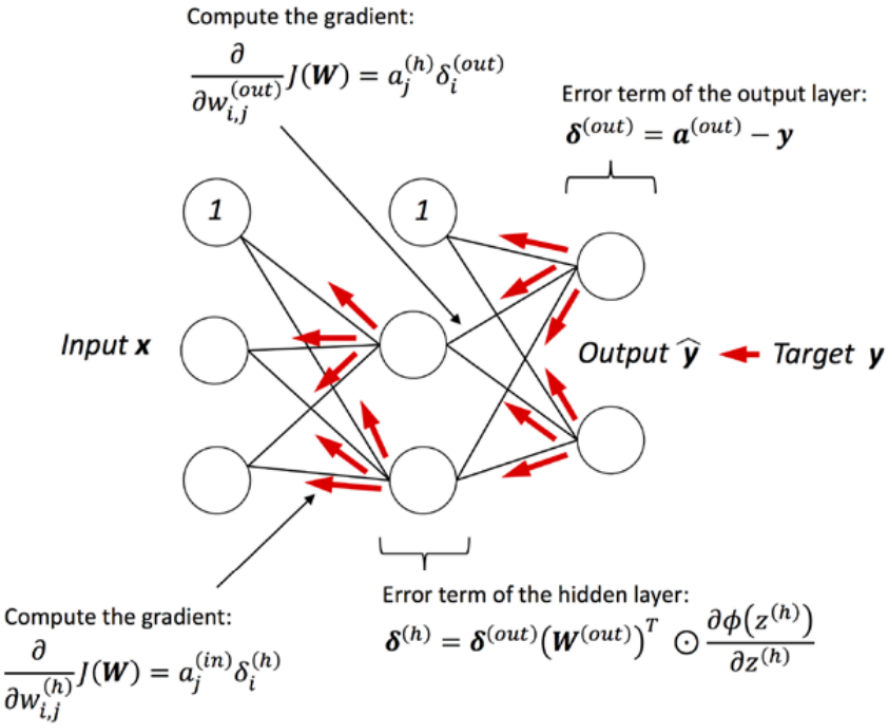

Figura 1.6: Gli step della backpropagation[3]

Il parametro η in (1.8) determina lo step size di modifica dei pesi ed il valore da selezionare è un compromesso tra la capacità di scappare dai minimi locali (alto learning rate) e l'over-shooting dei minimi globali dovuto ad un learning rate eccessivamente elevato.

Nelle applicazioni pratiche dell'MLP, tuttavia, la discesa del gradiente risulta svantaggiosa perchè il suo costo computazionale cresce con la quantità di dati in input, dato che per compiere ogni step viene rivalutato l'intero training dataset. In alternativa,la tecnica della discesa stocastica del gradiente mini-batch prevede il calcolo del gradiente per un sottoinsieme k alla volta degli n training samples (1<k<n) per performare l'aggiornamento dei pesi. In questo modo la convergenza viene raggiunta in tempi più brevi, pur aggiungendo "noise" dovuto al fatto che solo una parte del dataset viene valutata. Talvolta, tale rumore di fondo può risultare utile per sfuggire dai minimi locali che rappresentano il vero antagonista di tale modello, e quindi va tenuto in considerazione nella scelta del learning rate.

Capitolo 2

Algoritmi genetici

2.1 Problema di ottimizzazione

Le tecniche di intelligenza computazionale si applicano anche ad una classe di problemi che richiedono come soluzioni quelle che massimizzano o minimizzano una certa funzione, definita "funzione obiettivo".
L'insieme delle possibili soluzioni S è un sottoinsieme di $\mathbb{R}^n$, mentre la funzione obiettivo f è una funzione di n variabili reali $f(x_1, x_2, ..., x_n)$. Di solito l'insieme S è descritto da un numero finito di diseguaglianze (che prendono il nome di vincoli) del tipo $g(x) \leq b$. Formalmente, date m funzioni scalari $g_i : \mathbb{R}^n \to \mathbb{R}$, $i = 1, .., m$ ed m scalari $b_i \in \mathbb{R}$ si esprime S nella forma:

$$S = \{x \in \mathbb{R}^n \mid g_1(x) \leq b_1, g_2(x) \leq b_2, \ldots, g_m(x) \leq b_m\} \tag{2.1}$$

Perciò il problema di ottimizzazione può anche formalizzarsi:

$$\begin{cases} \min f(x) \\ g_i(x) \leq b_i, \quad i = 1, \ldots, m \end{cases} \tag{2.2}$$

All'interno dei problemi di ottimizzazione si possono riconoscere le seguente classi.

- Problemi di ottimizzazione continua
  Le variabili possono assumere tutti i valori reali ($x \in \mathbb{R}^n$) e si parla di problemi:
  - vincolati, se $S \subset \mathbb{R}^n$
  - non vincolati se $S = \mathbb{R}^n$

- Problemi di ottimizzazione discreta
  Le variabili sono vincolate ad essere interi($x \in \mathbb{Z}^n$); distinguiamo due classi:
  - programmazione a numeri interi se $S \subseteq \mathbb{Z}^n$
  - ottimizzazione booleana se $S \subseteq 0, 1^n$
- Problemi misti
  Solo alcune variabili vincolate ad essere interi

2.2 Algoritmo evolutivo: l'idea generale

L'algoritmo evolutivo si basa sull'idea della competizione tra individui di una popolazione in un ambiente dalle risorse limitate che quindi causa la selezione naturale del più "adatto" tra gli individui, secondo una funzione di qualità. La popolazione può essere creata in maniera casuale ed una prima applicazione della funzione di qualità ne classifica i migliori individui, che sono i candidati più probabili a dare vita alla nuova generazione. Questa viene creata a partire dai processi di ricombinazione e di mutazione. Il primo è un operatore applicato a due o più individui (i genitori) che restituisce uno o più individui figli. La mutazione, invece, interessa un individuo al quale si opera una variazione. A partire da questi due operatori quindi viene generata la offspring, ovvero l'insieme di candidati nuovi che competono con quelli vecchi, in base al valore di fitness espresso da una nuova applicazione della funzione di qualità. Questo processo può essere iterato fino a quando un individuo (soluzione) con un sufficiente valore di fitness viene trovato oppure fino ad un numero predeterminato di generazioni. Possiamo perciò dedurre due forze principali dei sistemi evoluzionari: gli operatori di variazione (ricombinazione e mutazione) che introducono la diversità nella popolazione, e la selezione che tende a migliorare la qualità delle soluzioni. È importante notare il carattere fortemente stocastico delle componenti dei sistemi evoluzionari: la selezione stessa non avviene in maniera deterministica, quindi ci sono chance anche per gli individui a basso valore di fitness di sopravvivere e diventare genitori, nella ricombinazione quali pezzi dei genitori verranno ricombinati sono scelti randomicamente, e allo stesso modo viene scelta la parte da sostituire nella mutazione.
Gli algoritmi evolutivi, dunque, condividono la struttura sopra descritta e si differenziano nelle applicazioni pratiche per le diverse rappresentazioni della candidata soluzione: una stringa per gli algoritmi genetici (GA), un vettore di valori reali per le strategie d'evoluzione (ES), alberi in genetic programming (GP). Le ragioni dell'uso di differenti rappresentazioni per i tipi diversi di algoritmo sono di tipo storico, mentre la scelta del tipo di algoritmo evolutivo è relativa ai problemi da

risolvere: ad esempio, dato un problema di n variabili da selezionare, risulta più intuitiva la rappresentazione in stringhe di bit di lunghezza n dove il contenuto dell'i-esimo bit è 1 oppure 0 a seconda che la variabile sia selezionata. In questo caso è conveniente scegliere un GA.

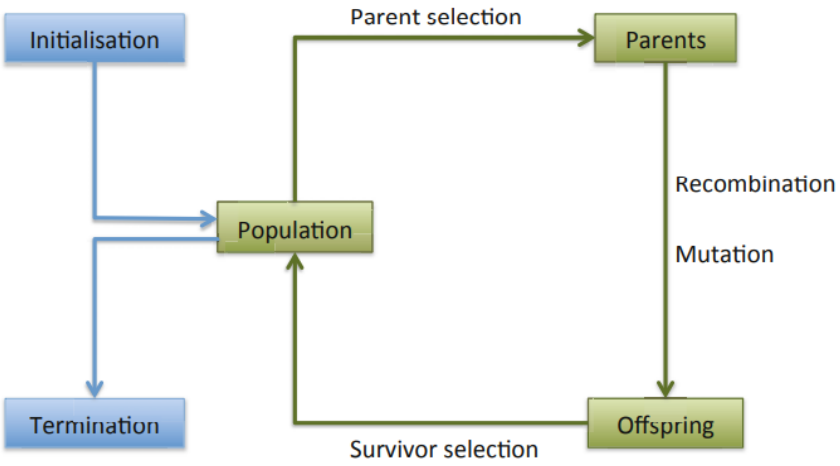

Figura 2.1: Flowchart dell'algoritmo evolutivo[1]

2.3 Principali componenti dell'algoritmo evolutivo

2.3.1 Rappresentazione

Per rappresentazione s'intende la definizione degli individui. Tuttavia può anche assumere il significato di mappatura dallo spazio del fenotipo allo spazio del genotipo: per fenotipo s'intende qualsiasi delle possibili soluzioni nel contesto del problema, mentre il genotipo è l'individuo presente nell'EA, a volte anche denominato come cromosoma. Perciò la fase di problem solving consiste di una fase preliminare di decisione riguardo quale sia la migliore definizione delle possibili soluzioni per poter essere manipolate nella fase successiva, cioè la fase di ricerca dell'algoritmo evolutivo. Questa viene svolta solo nello spazio del genotipo e restituisce un punto di esso, da cui, tramite una mappa inversa detta "decodifica" (nel caso di rappresentazione invertibile), è possibile risalire al fenotipo della soluzione.

Nella terminologia usuale, il genotipo ha un certo numero di variabili o geni, i cui valori sono anche detti alleli.

2.3.2 Evaluation Function

L'evaluation function definisce i requisiti d'adattamento per la popolazione, dato che assegna una misura di qualità ai genotipi. Tipicamente ne effettua prima una decodifica e poi restituisce una misura nello spazio del fenotipo. Per esempio, se la richiesta del problema è trovare un intero x per la massimizzazione di x^2, il valore di fitness del genotipo 10010 può essere definito portando 10010 in intero (fenotipo) e facendone il quadrato.
Con fitness o evaluation function ci si riferisce alla stessa funzione, anche nel caso in cui il problema sia di minimizzazione, nonostante la parola "fitness" suggerisca una massimizzazione come task principale dell'algoritmo. Dato che comunque da un punto di vista matematico non vi è alcun affanno nel passare da una massmimizzazione ad una minimizzazione, evaluation o fitness si possono considerare sinonimi.

2.3.3 Popolazione

La popolazione rappresenta il set di genotipi del sistema evoluzionario. È un'entità dinamica, che varia e si adatta nel trascorrere delle generazioni. La sua inizializzazione può essere fatta in maniera immediata specificando la population size, ovvero il numero di individui generati random, ma esistono anche strutture più complicate che aggiungono misure di distanza o relazioni di vicinanza che riflettono sulla possibilità di variazioni di individui "vicini". La population size resta costante così da incrementare sempre di più la competizione tra individui in un ambiente dalle risorse limitate. Sull'intera popolazione agiscono gli operatori di selezione che influenzano il carattere dell'offspring successiva. Non esiste comunque una vera e propria misura della diversità della popolazione e tipicamente ci si riferisce ad essa come il numero delle diverse fitness, ovvero una prima stima del numero di diversi fenotipi, perciò delle diverse soluzioni possibili. Questa comunque non è una stima esatta, in quanto non necessariamente a stessi valori di fitness corrispondono stessi fenotipi.

2.3.4 Meccanismo della parent selection

La parent selection distingue all'interno della popolazione gli individui migliori da cui generare la prole. L'offspring così generata spinge l'intero sistema a migliorare la qualità d'adattamento. Il meccanismo di scegliere gli individui genitori

è regolato stocasticamente, perciò seleziona con maggiori probabilità i genotipi ad alta fitness, ma non sono nulle le possibilità di un genitore a bassa fitness. In effetti, un meccanismo che escluderebbe a priori valori bassi di fitness porterebbe l'algoritmo rapidamente a convergere e rimanere bloccato in ottimi locali.

2.3.5 Operatori di variazione

Il ruolo di questi operatori è quello di creare nuovi individui a partire dai vecchi, ovvero creare nuove possibili soluzioni nello spazio dei fenotipi. Il modo in cui questi operatori agiscono li distingue in unari e n-ari.

Mutazione

Essendo un operatore unario, la mutazione agisce su un unico individuo per restituirne un "mutante", che presenta caratteristiche diverse. È un operatore di tipo stocastico quindi il mutante dipende da una serie di scelte random. La grande importanza della mutazione risiede nel fatto che contribuisce a rendere lo spazio di ricerca connesso: tramite una mutazione è possibile esplorare un nuovo punto dello spazio dei genotipi. Idealmente, quindi, in un certo tempo vengono esplorate tutte le possibili configurazioni. Tale proprietà rappresenta l'ipotesi su cui si fondano i teoremi che garantiscono che gli algoritmi evolutivi siano in grado di trovare l'ottimo globale di un certo problema. La stessa proprietà può essere soddisfatta in maniera più agevole permettendo a tutti gli alleli di essere mutati con probabilità diversa da zero, così da far esplorare tramite l'operatore di variazione tutte le configurazioni possibili in un tempo inferiore.

Ricombinazione

La ricombinazione o crossing-over mescola in maniera stocastica le informazioni di due genotipi genitori per restituire uno o più genotipi. L'idea alla base è quella di combinare due caratteristiche diverse ma desiderabili nella offspring generata, rinvigorita proprio da tale mescolamento. La ricombinazione è fortemente dipendente dalla rappresentazione, per cui in un algoritmo genetico in cui gli individui sono stringhe di bit, un metodo di crossing over può invertire uno dei bit.

2.3.6 Environmental selection

Dopo la creazione dell'offspring, dato che la popolazione mantiene una taglia costante, è necessario un meccanismo di selezione degli individui adatti a popolare la generazione successiva. Tale selezione prende il nome di "environmental selection" ed è di tipo deterministico, basandosi su un ranking fatto o in base ai valori

di fitness di offspring e genitori oppure in base all'età scartando dunque i genitori, rimpiazzati interamente dall'offspring.

2.3.7 Condizione di terminazione

Per un problema la cui soluzione ottimale risulta nota a priori, la presenza del genotipo corrispondente sarebbe la condizione di terminazione dell'algoritmo. Tuttavia, se si è consapevoli che l'algoritmo presenta rumore o semplificazioni necessarie, la condizione di terminazione può essere posta entro un certo range di approssimazione sulla soluzione ottimale. Nelle applicazioni pratiche, una tale condizione è garantita raramente, soprattutto per il carattere stocastico dell'EA. Quindi, tipicamente, per terminare la ricerca del candidato ottimale basta il sopraggiungere di una di queste condizioni: un massimo in termini di tempo o generazioni, un massimo di valutazioni della fitness dei candidati, oppure la condizione per cui i miglioramenti della fitness rimangono al di sotto di una certa threshold per un preciso numero di generazioni, oppure se la diversità della popolazione resta inferiore ad una soglia preimpostata.

2.4 L'algoritmo genetico

Nell'algoritmo genetico tradizionale la rappresentazione dell'individuo è di tipo binario. Le caratteristiche generali sono quelle descritte per un generico algoritmo evolutivo, la principale distinzione con altri tipi di EA sta però nella rappresentazione ed inoltre nell'applicazione di operatori di variazione particolari.
Una delle pecularità della rappresentazione binaria è la differente significatività dei bit, e quindi l'effetto di una mutazione sul bit è fortemente variabile in base alla posizione stessa del bit mutato.
Il bit-flip risulta tra le mutazioni più comuni per il GA. Esso permette a ciascun bit mutato di complementare il suo valore originale(da 0 a 1, oppure da 1 a 0), con una probabilità p_m. Dunque, il numero di possibili bit interessati dalla mutazione, dato un individuo di L bits risulta $L * p_m$. Nella maggior parte dei casi, p_m viene scelta per avere un gene per generazione ed un gene per offspring mutato. Tuttavia il particolare valore dipende in realtà dal tipo di problema: a seconda che l'applicazione richiede che vi siano tanti individui ad alti valori di fitness oppure che ne venga trovato solo uno con alta fitness, si preferisce un p_m basso oppure alto. Nel primo caso, infatti, la mutazione agisce meno e quindi l'algoritmo è meno propenso a scartare soluzioni a basso fitness. Nel secondo caso, invece si preferiscono i vantaggi di una maggiore copertura dello spazio di ricerca rispetto alla possibilità che vengano scartate buone soluzioni a causa della mutazione.

Figura 2.2: Esempio di bitflip[1]

Nei GA, tipicamente, la ricombinazione avviene a partire da due genitori che creano due figli. Esistono tre forme standard di quest'operatore.
Il one-point crossover seleziona randomicamente un numero l (compreso tra 1 e $L - 1$ dove L è il numero di bit); all'l-simo bit avviene il taglio dei due cromosomi genitori; a questo punto, l'operatore accoppia la prima parte del primo genitore con la seconda dell'altro genitore per creare il figlio 1, e simmetricamente per creare il figlio 2.

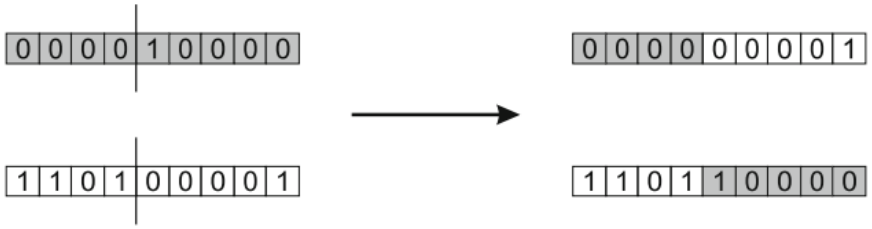

Figura 2.3: Esempio di one point crossover[1]

L'operatore può essere generalizzato a n punti di crossover.

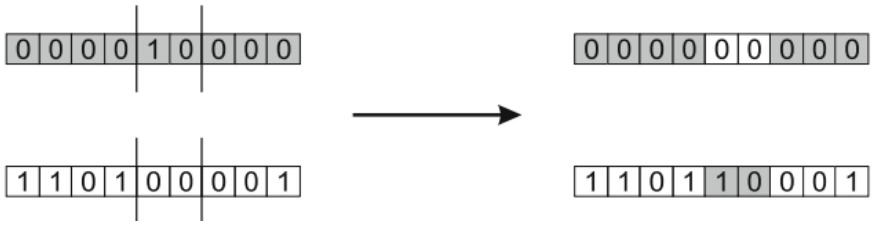

Figura 2.4: Esempio di n-point crossover, con n=2[1]

A differenza dei due operatori visti prima, l'uniform crossing over agisce trattando indipendente ciascun gene di entrambi i genitori ed assegnandone una probabilità. Se tale probabalità risulta inferiore ad un parametro p (solitamente 0,5) preimpostato, allora il gene viene ereditato, altrimenti viene ereditato il gene dell'altro genitore.

Figura 2.5: Esempio di uniform crossover con parametro p=0,5 e con il primo genitore avente valori di probabilità: [0.3, 0.6, 0.1, 0.4, 0.8, 0.7, 0.3, 0.5, 0.3][1]

Non vi è una particolare preferenza nella scelta dell'operatore di ricombinazione ma vanno notate le differenze: l'n-point crossover ha un bias intrinseco che tende a tenere insieme geni che sono allocati vicini; in particolare se n è dispari esiste un forte bias di tipo posizionale che rende poco probabili le combinazioni di geni localizzati agli estremi opposti del cromosoma.
L'uniform crossover invece ha una forte tendenza a trasmettere il 50% dei geni di ciascun genitore, esibendo perciò un forte bias distribuzionale. La scelta tra i due operatori dipende dalla definizione dell'individuo: per attributi che hanno un peso diverso a seconda della posizione è consigliabile usare l'n-point crossover per evitare variazioni troppo forti legati alla diversa significatività dei bits, al contrario per attributi generati random il bias di tipo distribuzionale ha meno effetto di quello posizionale.
La gestione della popolazione è di due tipi: generazionale e secondo il modello "steady-state". Nel primo, da una popolazione di μ individui, tutti entrano nella mating pool da cui tramite mutazioni e crossover viene creata l'offspring di taglia λ. Quest'ultima sostituirà interamente la popolazione iniziale nella successiva generazione.
Il modello steady-state invece prevede che la popolazione non venga cambiata interamente, ma si verifica una sostituzione dei vecchi individui con alcuni dell'offspring in una proporzione pari a $\frac{\lambda}{\mu}$. È importante notare che i metodi per la gestione della popolazione sono indipendenti dal tipo di rappresentazione usata per gli individui.
All'interno dei meccanismi generali di gestione entrano comunque metodi basati sulle misure della fitness, in particolare per la parent selection e per la survivor selection, rispettivamente per individuare i candidati migliori a fare da genitori e per selezionarne i partecipanti della generazione successiva.

Parent selection

Per la parent selection si utilizza un approccio fitness proportional (FPS, acronimo di Fitness Proportional Selection), in cui ciascun individuo ha una probabilità di essere scelto come genitore pari a

$$P_{FPS}(i) = f_i / \sum_{j=1}^{\mu} f_j \tag{2.3}$$

ovvero il valore di fitness dell'i-esimo individuo viene diviso per la somma dei valori di fitness dell'intera popolazione.
I limiti di quest'approccio sono vari:

- l'algoritmo può risultare sensibile alla presenza di individui dominanti con alti valori di fitness. Questo risulta in una convergenza prematura che limita lo spazio di ricerca
- la "pressione di selezione" è molto bassa, ovvero i valori di fitness si mantengono pressochè uniformi al passare delle generazioni, fornendo bassa diversità della popolazione
- l'algoritmo mostra una dipendenza al variare delle traslazioni della fitness, come si mostra in figura

Individual	Fitness for f	Sel. prob. for f	Fitness for $f+10$	Sel. prob. for $f+10$	Fitness for $f+100$	Sel. prob. for $f+100$
A	1	0.1	11	0.275	101	0.326
B	4	0.4	14	0.35	104	0.335
C	5	0.5	15	0.375	105	0.339
Sum	10	1.0	40	1.0	310	1.0

Figura 2.6: Si noti come al variare della fitness, siano mutati i valori delle probabilità, mentre la posizione dell'ottimo globale rimane fissa.
[1]

Per evitare gli ultimi due effetti, si preferisce il "windowing", una procedura che prevede che i valori di fitness vengano diminuiti di un certo parametro dipendente dal tempo β^t che in genere è posto uguale al valore minimo di fitness della popolazione al tempo t. In questo modo quindi diminuisce fortemente il carattere uniforme dei valori di fitness, favorendo una "pressure selection" maggiore.
Tra i metodi più comuni di parent selection c'è la roulette wheel selection. Concettualmente corrisponde a selezionare l'individuo genitore tramite un giro di una one-armed (cioè seleziona un individuo alla volta) roulette, i cui bin sono tanto

larghi quanto grande è il valore di probabilità associato. Tale metodo utilizza una lista di valori $[a_1, a_2, ..., a_\mu]$ tale che $a_i = \Sigma_1^i P_{sel}(i)$ e quindi con $a_\mu = 1$. Randomicamente viene generato un numero r compreso tra 0 e 1, e l'individuo a cui corrisponde un valore di funzione cumulativa $a_i > r$ viene selezionato.
Tuttavia quando è richiesto un sampling di un numero λ individui risulta più sensato ricorrere al SUS (Stochastic Universal Sampling) che consiste in una multi-armed roulette wheel, più adatta rispetto a fare un numero λ di giri su una one-armed roulette wheel.

```
BEGIN
  /*  Given the cumulative probability distribution a */
  /*  and assuming we wish to select λ members of the mating pool */
  set current_member = i = 1;
  Pick a random value r uniformly from [0, 1/λ];
  WHILE ( current_member ≤ λ ) DO
    WHILE (  r ≤ a[i] ) DO
      set mating_pool[current_member] = parents[i];
      set r = r + 1/λ;
      set current_member = current_member + 1;
    OD
    set i = i + 1;
  OD
END
```

Figura 2.7: Pseudocodice del SUS: da notare la condizione $r = r+1/\lambda$ garantisce che ogni genitore venga selezionato al più una volta
[1]

I due metodi appena visti si basano sulla conoscenza dell'intera popolazione e sull'assunzione che la misura della fitness sia effettivamente significativa in maniera isolata per ogni individuo. Talvolta, in casi di popolazioni molto numerose, oppure in algoritmi in cui l'effettivo valore della fitness risulta significativo solo se relazionato ad altri individui, risulta conveniente utilizzare la tournament selection. Essa prevede che all'interno di un numero k di individui, venga selezionato il più adatto. Può essere con o senza replacement: nel secondo caso i k-1 individui scartati dal primo tournament non potranno più essere selezionati. Il metodo è usato perchè molto facile da implementare. Pur soffrendo degli stessi problemi della roulette wheel, ha il grande pregio di poter gestire facilmente la "selection pressure", aumentando (o diminuendo) k per aumentarla (o diminuirla), il parametro di tournament.

Survivor selection

In linea di principio la selezione degli individui che faranno parte della generazione successiva può essere effettuata dagli stessi metodi visti per la parent selection, ma nel corso di anni si sono sviluppate tecniche apposite che sono comunemente usate. La principale discriminante della survivor selection sta nella strategia di replacement basata sull'età oppure sulla fitness.
La prima è la più usata e, dal momento che il numero di offspring è pari a quello dei genitori ($\lambda = \mu$), consiste nella sostituzione totale della popolazione con l'offspring ad ogni generazione (generational survivor selection). Altre varianti presentano la possibilità di avere una popolazione con sovrapposizioni tra offspring ed individui iniziali ($\lambda < \mu$), oppure selezionano in maniera random l'individuo da rimpiazzare. Quest'ultimo caso è stato approndidamente studiato da Smith e Vavak che ne hanno dimostrato la maggiore tendenza a rimpiazzare i migliori individui rispetto ai metodi convenzionali age-based, e quindi ne viene sconsigliato l'utilizzo.
In qualsiasi metodo age-based, la strategia di rimpiazzo non dipende dai valori di fitness per cui è possibile che tra una generazione e l'altra la fitness del miglior individuo (e/o la fitness media della popolazione) diminuisca. Seppure possa sembrare controintuitivo visto il problema di ottimizzazione che vogliamo risolvere, questa situazione non è necessariamente d'ostacolo ed anzi, se non si verifica in un numero elevato di generazioni, può avvantaggiare l'uscita da qualche ottimo locale. Difatti, l'aumento di fitness che desideriamo ottenere, completato l'algoritmo, è garantito da una sufficiente "pressure selection" e dall'uso di bilanciati operatori di variazione.
Per le strategie fitness-based si distinguono due principali varianti di selezione dei μ prossimi individui a partire dai $\mu + \lambda$ della popolazione iniziale e dell'offspring. Worst-replacement scarta i λ peggiori individui. Tale tecnica tuttavia, pur migliorando i valori di fitness, può portare ad una convergenza prematura. Il suo utilizzo è perlopiù limitato ai casi di popolazioni molto grandi in cui tale difetto viene "mascherato" per via dell'ingente numero di individui che ne rallenta la convergenza.
L'elitismo prevede che il miglior individuo venga sempre conservato nella prossima generazione e solitamente tale tecnica è applicata congiunta a tecniche SUS oppure age-based.

Pressure selection

Qualitativamente ci si può riferire all'aumento della pressure selection come un meccanismo ad imbuto sempre più stretto sugli individui del GA che privilegia

quelli con fitness elevata. Varie misure sono state proposte per quantificare tale processo, e la più significativa è il takeover time τ, definito come il numero di generazioni necessarie ad ottenere una popolazione costituita da copie di migliori individui, a partire da uno iniziale. Quantitativamente: $\tau^* = \frac{ln\lambda}{ln(\frac{\lambda}{\mu})}$

Smith derivò in particolare che la media e la varianza di τ^* possono prevedere media e varianza del tempo che occorre a localizzare l'ottimo globale, non solo per i GA ma anche per varie categorie di EA. Tuttavia, non è presente nessuna correlazione particolare tra il takeover time ed il vero obiettivo degli EA, ovvero la misura della qualità della soluzione trovata, quindi non risulta nessuna rilevante differenza di performance in termini di τ diversi.

Capitolo 3

Progettazione ed implementazione

3.1 Progettazione di un GA per feature selection

Per l'implementazione del GA che risolve il problema d'ottimizzazione della feature selection è stato utilizzato il linguaggio di programmazione ad alto livello, Python. Esso ha come particolare vantaggio la possibilità di richiamare facilmente l'uso di framework, ovvero un insieme di file contenenti costanti, funzioni e classi che rendono più facile la programmazione.

3.1.1 Struttura della soluzione

L'algoritmo di ottimizzazione dovrà restituire come soluzione l'individuo che presenta il massimo valore di fitness dell'intero processo evolutivo. Quindi, l'output finale sarà ancora una stringa, di lunghezza N pari al numero di variabili, fatta di 0 e 1 nelle posizioni *ottimali*: i valori 1 occuperanno nella stringa le posizioni relative alle variabili "migliori", cioè quelle che producono la predizione migliore tra tutte quelle effettuate durante l'evoluzione; i valori 0 segnaleranno quindi le corrispondenti variabili irrilevanti o ridondanti per il problema di classificazione. Per le applicazioni successive, ad esempio nel confronto tra prestazioni con e senza feature selection su un test set, per accedere più facilmente ai dati delle sole variabili sfruttate è comunque conveniente sfruttare la relazione che a partire dalla stringa-soluzione del problema di ottimizzazione restituisca gli indici a cui corrispondono i valori 1.

3.1.2 L'evaluation function

È fondamentale chiarire nel dettaglio com'è strutturata l'evaluation function, perchè questa definisce come viene calcolata la fitness di ogni individuo.

Il problema che si vuole risolvere riguarda la selezione delle variabili "migliori" per la classificazione. Quindi è chiaro che bisogna modellizzare il problema come di massimizzazione e la fitness come l'accuracy della classificazione.
L'individuo è una stringa formata da 0 e 1 randomicamente, di lunghezza data dal numero di variabili presenti nel dataset. Il valore 1 al posto i-esimo indica che l'i-esima variabile è considerata, il valore 0 il contrario. L'idea sta quindi nel considerare per la classificazione solo le variabili corrispondenti agli 1 presenti nella stringa. Questo significa che è necessario un comando per "sbloccare", a partire dal dataset, soltanto i dati relativi a quelle variabili. La struttura completa dell'evaluation function per un individuo è la seguente:

- creazione della lista delle variabili selezionate dall'individuo;
- accesso ai dati delle sole variabili selezionate;
- accesso ai dati delle variabili target;
- splitting dei dati in un set per l'addestramento ed in uno per il test: i dati delle variabili selezionate e di quelle target vengono per il 70% raccolti in un set utile al solo addestramento del classificatore, il restante verrà utilizzato per la predizione;
- addestramento del MultiLayerPerceptron sul training set;
- predizione del MultiLayerPerceptron sul test set;
- calcolo dell'accuracy sulla predizione del test set;
- output finale: l'evaluate function restituisce come valore di fitness il valore dell'accuracy sulla predizione effettuata;

3.1.3 Operatori utilizzati

Per questo tipo di problema sono state utilizzati i seguenti operatori ed adottate le seguenti scelte:

- la parent selection viene effettuata tramite il metodo di SelTournament, con tournamentsize k=3. La scelta di tale metodo è giustificata dai vantaggi che offre per questo tipo di problema: il valore dell'accuracy nel caso di feature selection assume infatti maggiore rilevanza se confrontato con quello di altri individui, piuttosto che valutato singolarmente, dal momento che non si conosce a priori alcuna indicazione sull'ottimo globale del problema qui trattato;

- l'operatore di mutazione scelto è il mutUniformInt(0,1,indpb) che cambia il valore di un attributo da 0 a 1 con una probabilità indipendente *indpb*;

- l'operatore di crossing-over scelto è il cxTwoPoint ovvero un n-point crossover con numero di tagli fissato a 2 su ciascun cromosoma padre. La regione delimitata dai due tagli viene scambiata tra i due cromosomi padri, creando così due nuovi individui; per la probabilità associata CXPB si usa un valore standard (pari a 0,9) per l'algoritmo genetico tradizionale;

- la survivor selection è di tipo generazionale, come nei SGA (Simple Genetic Algorithm), ovvero l'offspring creata a partire dagli operatori di variazione sostituisce interamente la popolazione ad ogni generazione; essendo una strategia indipendente dalla fitness, quest'ultima può variare valore medio, massimo e minimo durante le diverse generazioni. Tale cambiamento può essere utile per evitare l'intrappolamento negli ottimi locali;

- la condizione di terminazione è una scelta standard basata sul numero di generazioni;

3.2 Implementazione usando il modulo DEAP

Tra i framework messi a disposizione da Python è stato utilizzato DEAP (Distributed Evolutionary Algorithms in Python) che agevola l'implementazione dei GA. In effetti, i vantaggi risiedono nella:

- grande flessibilità che offre per l'uso degli operatori di variazione, che essendo parametrizzati sono facilmente "manovrabili" dall'utente. Questo implica la possibilità di esplorare più punti all'interno dello spazio di ricerca dell'algoritmo;

- facilità con cui è possibile inizializzare le strutture dati;

- presenza di meccanismi che inizializzano valori a partire da una distribuzione impostata;

- atomicità del framework in strutture specializzate;

In riferimento all'ultimo punto, il modulo *base* mette a disposizione specifiche parti dell'algoritmo.
La creazione delle classi è di competenza del modulo creator, tramite il metodo create, con la seguente sintassi:
deap.creator.create(name, base[attribute,...])
Il primo argomento è il nome della classe, e come secondo gli attributi che la classe eredita.
A partire dalla classe Fitness, messa a disposizione del metodo base, si può creare una classe che ne eredita le proprietà ed aggiunge un attributo definito "weights". La classe Fitness rappresenta a tutti gli effetti la qualità della soluzione dell'algoritmo, la cui strategia di ricerca è parametrizzata dalla tupla "weights" (1.0,) nel caso di problema unidimensionale di massimizzazione. Da notare come l'utilizzo della tupla renda agevole la gestione di problemi multidimensionali di tipo diverso. La sintassi è la seguente: *creator.create("Fitness", base.Fitness, weights)*.

Anche la creazione dell'individuo viene fatta a partire dal metodo create. In questo caso si specifica che l'individuo è del tipo lista ed ha un attributo "fitness", della classe Fitness precedentemente creata. In sintassi,
creator.create("Individual", list, fitness=creator.Fitness

Il valore della fitness è facilmente accessibile richiamandone il valore dell'attributo dell'individuo: *Individual.fitness.values*

Tra gli strumenti messi a disposizione di DEAP, merita menzione il toolbox, di cui si sfruttano i seguenti metodi:

- register(): permette di salvare qualsiasi funzione (secondo argomento) sottoforma di un alias (primo argomento), specificandone gli attributi della funzione registrata: *deap.base.Toolbox.register (alias, function, argument)*. Nel GA è utilizzato per registrare:
  - la funzione per generare gli attributi:
    toolbox.register("attrbool", random.randint, 0, 1)
    La funzione di cui ne registriamo l'alias "attrbool" genera randomicamente 0 oppure 1;
  - la funzione per generare gli individui:
    toolbox.register("individual", tools.initRepeat, creator.Individual, toolbox.attrbool, Nfeat)
    Viene dato l'alias alla funzione initRepeat, che applica la funzione attrbool alla classe Individual per un numero di volte pari a Nfeat, cioè il

numero delle variabili: si sta costruendo così l'individuo delle variabili selezionate randomicamente.

- la funzione per generare la popolazione:
  toolbox.register("population", tools.initRepeat, list, toolbox.individual)
  Viene applicata la funzione "individual" all'interno di una lista, che quindi conterrà tutti gli individui generati.
- tutte le funzioni di mutazione, crossing-over e survivor selection;
- la funzione di valutazione che restituisce il valore di fitness di ogni individuo:
  toolbox.register("evaluate", EvaluateFeatures)
  Come visto nella sezione precedente, l'evaluate function è divisa in tappe, delle quali ora si mostrano i dettagli implementativi:
  - *listvar = [i for i in range(len(individual)) if individual[i] == 1]*
    Questo comando restituisce la lista degli indici corrispondente agli "1" presenti nella stringa "individual", ovvero la lista delle variabili selezionate;
  - *X=df.iloc[,listvar].values*
    Per tale istruzione si fa uso del modulo pandas: prima tramite il comando "read" il dataset viene riconosciuto per poi rendersi accessibile tramite il comando iloc, applicato al dataset(df, nel codice), fornendo indice di riga e di colonna. L'inserimento di listvar come secondo argomento del metodo iloc[] fa sì che vengano restituiti tutti e soli i dati con indice di colonna pari all'indice delle variabili selezionate; i valori così "sbloccati" saranno l'input del classificatore;
  - *y=df.iloc[,targetcolumn].values*
    In questo modo si definiscono i valori target della classificazione, grazie all'utilizzo del metodo iloc, in corrispondenza dell'indice o degli indici(se è un problema multi-classe) di colonna relativo alla(e) variabile(i) di cui si vuole fare la predizione;
  - Per la procedura di splitting dei dati in test e training set si è utilizzato il metodo train_test_split da scikit-learn.model_selection;
  - *MLP.fit(train_input,train_target)*
    Con quest'istruzione viene effettuato l'addestramento; il classificatore utilizzato fa parte del modulo scikit-learn.neural_network;

    - *prediction_test=MLP.predict(test_input,test_target)*
      In questo modo viene effettuata la predizione sul test;

    - *test_accuracy=accuracy_score(predictions_test, target_test)*
      Il metodo accuracy_score fa parte del modulo scikit-learnmetrics e restituisce l'accuracy della predizione effettuata;

    - *return test_accuracy*
      La funzione quindi ritorna come output finale l'accuracy calcolata nell'istruzione precedente;

- unregister(): permette la cancellazione dell'elemento registrato, semplicemente scrivendone l'alias;

- clone(): duplica l'elemento passato come argomento;

- map(): applica la funzione (primo argomento) sull'oggetto iterabile (secondo argomento):
  fitnesses = list(map(toolbox.evaluate, pop))
  È stato utilizzato per valutare la fitness su ciascun individuo della popolazione. Tramite il comando list poi sono stati raccolti i valori di fitness in una lista;

I metodi utilizzati per gli operatori di variazione e selezione, pur essendo registrati tramite il modulo toolbox, appartengono al modulo tools. Quest'ultimo contiene anche strumenti utili a raccogliere informazioni e statistiche del processo evolutivo: Statistics e Logbook.
Statistics prende come argomento la funzione da applicare per computare i valori su cui compiere l'analisi statistica:
stats = tools.Statistics(key=lambda ind: ind.fitness.values)
In particolare, come mostrato nel codice di sopra, le statitiche possono essere effettuate a partire da un attributo degli individui, come il valore della fitness. Se ne calcolerà poi la media, la deviazione standard, massimo e minimo, una volta registrate con toolbox.register.

Il logbook permette di raccogliere al suo interno (come dizionario) le statistiche di ogni generazione.
*logbook.record(gen=g,evals=len(pop),**record)*
gen,avg=logbook.select("gen","avg")
È stato usato per raccogliere le statistiche di ogni individuo della popolazione e per estrarre, tramite il metodo select, la media della fitness ad ogni generazione.

Capitolo 4

Esperimenti e risultati

L'obiettivo della sperimentazione è trovare per ciascun dataset il set di variabili rilevanti (problema di ottimizzazione) e confrontare le prestazioni del classificatore (problema di classificazione) per tutte le variabili e per quelle selezionate. A priori, non è necessariamente vero che l'accuracy della classificazione con le variabili selezionate venga migliorata, tuttavia, per tutti i dataset, risulta ridotto il tempo d'addestramento del classificatore ed il tempo di computazione della predizione.
Ciascun dataset è diviso per un 70% in training set e per la restante percentuale nel test set. La feature selection è effettuata grazie al GA sul training set. Quest'ultimo viene utilizzato anche per l'addestramento dell'MLP, di cui ne vengono computati i tempi. Uno degli obiettivi di questa tesi è difatti dimostrare la riduzione dei tempi d'addestramento una volta avvenuta la feature selection rispetto al caso in cui tutte le variabili siano date in input al classificatore.
Dal test set invece ricaviamo il confronto tra l'accuracy della predizione nel caso in cui venga effettuata feature selection oppure nel caso contrario. Inoltre, se ne computano anche i tempi in entrambi i casi, per dimostrare un'effettiva riduzione grazie alla feature selection.
Essendo l'algoritmo di natura non deterministica, viste le componenti stocastiche che lo caratterizzano, sono state effettuate 15 run del GA. Tra i 15 valori di fitness ottimali ricavate dai run, si è scelta la mediana. Alla mediana delle fitness corrisponde un certo set di variabili, ovvero la soluzione del metodo di feature selection. Talvolta tale corrispondenza non è univoca e quindi alla mediana delle fitness corrispondono più set di variabili scelte. In questo caso, si seleziona il set con il numero di variabili inferiori per perseguire l'obiettivo principale del metodo, cioè ridurre la dimensionalità dello spazio di ricerca dell'algoritmo. Una volta scelto il set ottimale, sul test vengono confrontate le accuracy ed i tempi di predizione relativi ai dati del set di variabili selezionate rispetto ai dati di tutte le variabili disponibili.

4.1 Datasets

4.1.1 Dataset 1: Congressional Voting Records

È un dataset che raccoglie i risultati di 16 diverse votazioni per 435 rappresentanti del Congresso, e per ognuno specifica la classe d'appartenza, democratico oppure repubblicano. L'obiettivo della classificazione è quindi predirre in base all'esito delle votazioni l'appartenenza ad una delle due classi. I 16 esiti sono le features, di tipo booleano (a favore o contrario), delle quali vorremmo selezionare solo le rilevanti che massimizzano l'accuracy della predizione.

Numero di istanze	Numero di features	Tipo di feature
435	16	Categoriche (Booleane)

Tabella 4.1: Dataset 1

4.1.2 Dataset 2: Ionosphere

In questo dataset sono raccolti tutti i dati acquisiti da un array di 16 antenne radar ad alta frequenza. Il target di tali radar sono gli elettroni all'interno della ionosfera e la classificazione avviene a partire dalla qualità del segnale di ritorno dalle antenne: "buono" se mostra qualche evidenza della struttura della ionosfera, "pessimo" se il flusso di elettroni oltrepassa la ionosfera senza fornire informazioni. Le features riassumono le caratteristiche del flusso di elettroni e delle antenne e sono 34, di tipo continuo.

Numero di istanze	Numero di features	Tipo di feature
351	34	Continue

Tabella 4.2: Dataset 2

4.1.3 Dataset 3: Magic Telescope

L'obiettivo della classificazione è distinguere le deposizioni di energia del rivelatore MAGIC come provenienti da raggi gamma oppure da raggi cosmici. Le features utilizzate sono parametri che descrivono le caratteristiche dell'ellisse che viene visualizzato sulla camera del rivelatore, e sono tutte di tipo continuo. del

flusso di elettroni e delle antenne e sono 34, di tipo continuo.

Numero di istanze	Numero di features	Tipo di feature
19020	10	Continue

Tabella 4.3: Dataset3

4.1.4 Dataset 4: Heart failure clinical records

In base alle caratteristiche del paziente e del trattamento, si vuole predire la morte o la sopravvivenza durante il periodo d'osservazione. Le features sono di tipo numerico (intero, ad esempio per l'età o per la percentuale delle dosi del trattamento) oppure di tipo booleano (per valutare la presenza o meno di patologie pregresse).

Numero di istanze	Numero di features	Tipo di feature
299	13	Interi, reali, booleane

Tabella 4.4: Dataset 4

4.1.5 Dataset 5: BHP flooding attack on OBS

La classificazione, a differenza degli altri dataset, è di tipo multiplo in base ai 4 differenti esiti di un attacco di tipo BHP (Burst Header Packet) sui network OBS(Optical Burst Switching). Le features sono parametri interi che descrivono ad esempio, i tempi di un eventuale ritardo, la banda utilizzata ed altre caratteristiche.

Numero di istanze	Numero di features	Tipo di feature
1075	22	Interi

Tabella 4.5: Dataset 5

4.1.6 Dataset 6: Bank Marketing

Si vuole prevedere il successo o meno di una campagna di marketing su una serie di utenti, possibili clienti di una banca. Le caratteristiche riguardano i parametri della pubblicità effettuata (ad esempio la durata dell'interazione con il cliente, il tipo di interazione oppure il numero di esse) ed il cliente (tipo di lavoro, condizione economica, ecc..)e sono quindi di tipo booleano, categorico e numerico.

Numero di istanze	Numero di features	Tipo di feature
45221	17	Categoriche, reali, booleane

Tabella 4.6: Dataset 6

4.1.7 Dataset 7: Leaf

Si vuole prevedere la classificazione in 40 diverse specie di foglie in base alle diverse caratteristiche fisiche. Le features utilizzate sono 15 e sono tutte di tipo numerico, frutto di misure accurate sulla forma e sulle proprietà emerse dagli istogrammi raccolti dalle immagini delle foglie studiate.

Numero di istanze	Numero di features	Tipo di feature
340	15	Reali

Tabella 4.7: Dataset 7

4.2 Risultati

Per ciascun dataset, è utile visualizzare tramite boxplot verticali i valori di fitness ottenuti dai 15 run del GA. Il boxplot consiste in un rettangolo diviso in due parti, da cui escono due segmenti (detti "whiskers"). Il rettangolo è diviso a partire dalla mediana $q_{\frac{1}{2}}$ in due parti che corrispondono al primo quartile $q_{\frac{1}{4}}$ ed al terzo quartile $q_{\frac{3}{4}}$. i whiskers sono invece delimitati dal valore massimo e dal valore minimo.

4.2.1 Dataset 1

Numero di istanze	Numero di features
435	16

Tabella 4.8: Istanze e feature, dataset 1

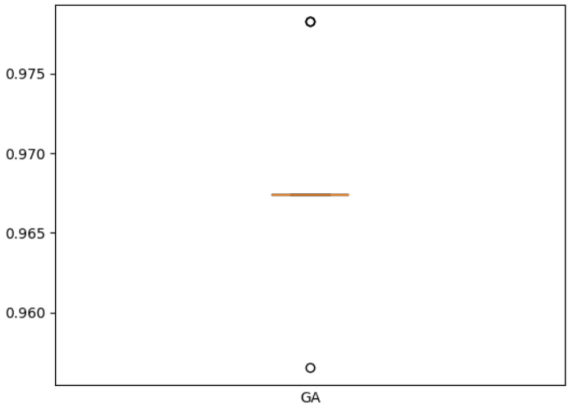

Figura 4.1: Boxplot dei 15 valori di fitness per il dataset 1

Nel caso in questione, si nota che tra i 15 risultano solo 3 valori diversi: la mediana, valore medio e minimo; per questo motivo il boxplot collassa nella linea (evidenziata di arancione) che rappresenta la mediana, mentre i due puntini sono i restanti massimo e minimo valore della fitness.

Numero di features selezionate: 7

Feature Selection	Train accuracy	Test accuracy	Fit time(s)	Predict time(s)
Sì	0,967	0,962	0,67	0,000796
No	0,991	0,939	0,59	0,000997

Tabella 4.9: Risultati dataset 1

Definiamo ora il "gain" percentuale dalla differenza dei parametri calcolati nel caso d'utilizzo della feature selection e nel caso opposto. Si noti che il gain di tempo d'addestramento e della predizione sono sempre positivi, perchè la feature selection riduce sempre i valori di questi parametri e dunque si eviterà di specificarne il segno. Al contrario, per l'accuracy il gain potrà essere positivo se la feature selection migliora la predizione, negativo nel caso contrario.

% Test accuracy gain	% Fit time gain	% Predict time gain
+2,4%	20,1%	11,9%

Tabella 4.10: Gain percentuale dataset 1

In conclusione, per il dataset 1, come evidenziato dal boxplot, la mediana dei 15 run del GA è il valore di fitness pari a 0,967 sul training set. A questa corrisponde un set di 7 features.
A partire dalle 16 variabili del dataset, quindi, sono state selezionate 7 variabili rilevanti che migliorano l'accuracy del 2,4% e riducono i tempi di addestramento e di predizione rispettivamente del 20% e dell'11%.

4.2.2 Dataset 2

Numero di istanze	Numero di features
351	34

Tabella 4.11: Istanze e feature dataset 2

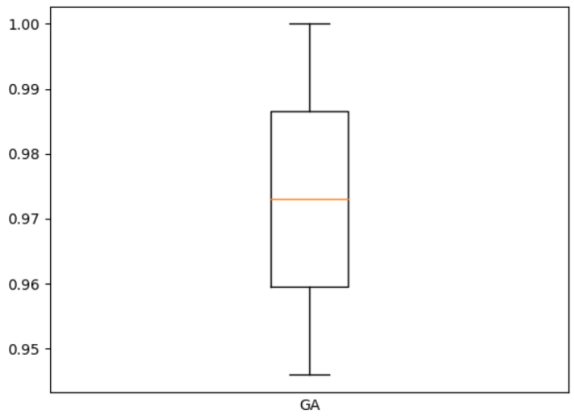

Figura 4.2: Boxplot dei 15 valori di fitness per il dataset 2

Numero di features selezionate: 11

Features Selection	Train accuracy	Test accuracy	Fit time(s)	Predict time(s)
Sì	0,973	0,925	0,55	0,000463
No	0,996	0,906	0,67	0,000544

Tabella 4.12: Risultati dataset 2

% Test accuracy gain	% Fit time gain	% Predict time gain
+2,1%	17,8%	14,7%

Tabella 4.13: Gain percentuale dataset 2

Si ricava quindi che a partire da 34 variabili, 11 ne sono selezionate, per un miglioramento dell'accuracy sul test del 2,1%, con tempo di addestramento e di predizione ridotti del 17,8% e del 14,7%.

4.2.3 Dataset 3

Numero di istanze	Numero di features
19020	10

Tabella 4.14: Istanze e feature dataset 3

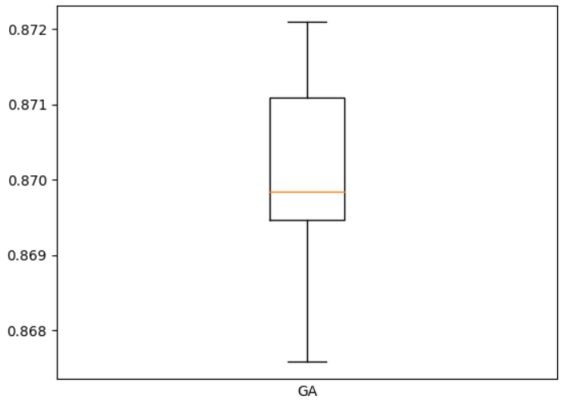

Figura 4.3: Boxplot dei 15 valori di fitness per il dataset 3

Numero di features selezionate: 9

Features Selection	Train accuracy	Test accuracy	Fit time(s)	Predict time(s)
Sì	0,870	0,874	24,6	0,014
No	0,884	0,873	26,3	0,011

Tabella 4.15: Risultati dataset 3

% Test accuracy gain	% Fit time gain	% Predict time gain
+0,1%	15,7%	6,6%

Tabella 4.16: Gain percentuale dataset 3

A partire da 10 variabili, quindi ne sono state selezionate 9 che pur avendo portato ad un miglioramento non molto significativo dell'accuracy (+0,1%), hanno notevolmente ridotto il fit time del 15,7%, che in caso di grande numero di istanze risulta dell'ordine della decina di secondi e quindi molto rilevante, ed il predict time del 6,6%.

4.2.4 Dataset 4

Numero di istanze	Numero di features
299	13

Tabella 4.17: Istanze e feature dataset 4

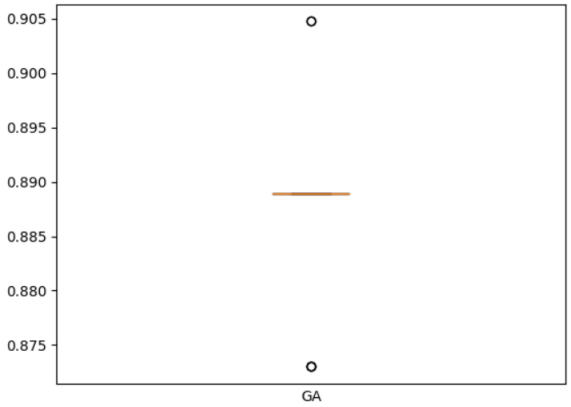

Figura 4.4: Boxplot dei 15 valori di fitness per il dataset 4 da cui si nota che esistono solo 3 valori differenti, il minimo, il massimo e la mediana

Numero di features selezionate: 4

Features Selection	Train accuracy	Test accuracy	Fit time(s)	Predict time(s)
Sì	0,889	0,767	0,48	0,000997
No	0,880	0,756	0,49	0,000998

Tabella 4.18: Risultati dataset 4

% Test accuracy gain	% Fit time gain	% Predict time gain
+1,51%	3,61%	0,02%

Tabella 4.19: Gain percentuale dataset 4

È interessante notare come tutti i valori di gain siano abbastanza modesti, ma la feature selection abbia ridotto il numero delle variabili da 13 a 4, il che suggerisce in effetti che ben 9 siano irrilevanti o ridondanti. Valori di accuracy così simili

infatti rendono l'idea di come l'MLP nel caso di tutte le variabili considerate "converga" velocemente a non considerarne quelle ridondanti o irrilevanti.La presenza di variabili da trascurare facilmente detectable dal classificatore rendela features selection, pur sicuramente vantaggiosa, dai vantaggi meno evidenti rispetto ad altri datasets.

4.2.5 Dataset 5

Numero di istanze	Numero di features
1075	22

Tabella 4.20: Istanze e feature dataset 5

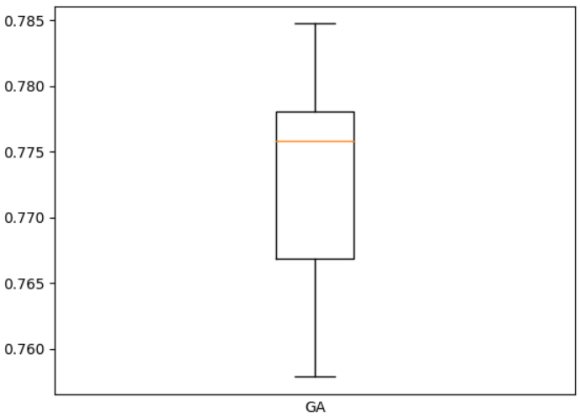

Figura 4.5: Boxplot dei 15 valori di fitness per il dataset 5

Numero di features selezionate: 11

Features Selection	Train accuracy	Test accuracy	Fit time(s)	Predict time(s)
Sì	0,776	0,783	1,19	0,000996
No	0,810	0,756	1,37	0,00104

Tabella 4.21: Risultati dataset 5

% Test accuracy gain	% Fit time gain	% Predict time gain
+1,2%	13%	4,2%

Tabella 4.22: Gain percentuale dataset 5

A partire da 22 variabili, il metodo ne ha selezionato 11, migliorando l'accuracy sul test, riducendo tempo di fit e di predict rispettivamente del 13% e del 4,2%.

4.2.6 Dataset 6

Numero di istanze	Numero di features
45221	17

Tabella 4.23: Istanze e feature dataset 6

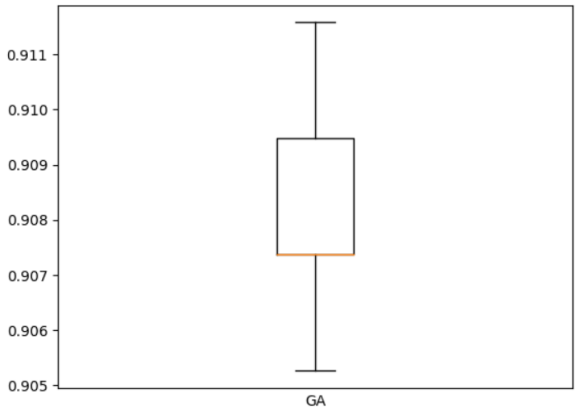

Figura 4.6: Boxplot dei 15 valori di fitness per il dataset 6

Numero di features selezionate: 6

Features Selection	Train accuracy	Test accuracy	Fit time(s)	Predict time(s)
Sì	0,907	0,891	5,01	0,00196
No	0,938	0,756	5,82	0,00299

Tabella 4.24: Risultati dataset 6

% Test accuracy gain	% Fit time gain	% Predict time gain
+0,2%	15%	34%

Tabella 4.25: Gain percentuale dataset 6

Applicato il metodo, le variabili passano da 17 a 6, mostrando un lieve miglioramento dell'accuracy ma una consistente riduzione nei tempi di fit e predict del 15% e del 34%.

4.2.7 Dataset 7

Numero di istanze	Numero di features
340	15

Tabella 4.26: Istanze e feature dataset 7

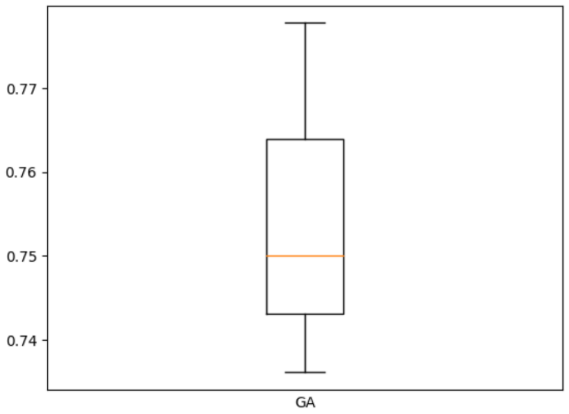

Figura 4.7: Boxplot dei 15 valori di fitness per il dataset 7

Numero di features selezionate: 6

Features Selection	Train accuracy	Test accuracy	Fit time(s)	Predict time(s)
Sì	0,71	0,63	0,65	0,00099
No	0,59	0,59	0,69	0,00100

Tabella 4.27: Risultati dataset 7

% Test accuracy gain	% Fit time gain	% Predict time gain
+6,7%	4,5%	0,2%

Tabella 4.28: Gain percentuale dataset 7

Per questo dataset è evidente il vantaggio della feature selection nella test accuracy, che migliora del 6,7%. Le variabili rilevanti per la classificazione sono dunque 6 a dispetto delle 15 iniziali. La riduzione dei tempi si evidenzia maggiormente nel fit (4,5%) piuttosto che nella predizione (0,2%).

Conclusioni

La feature selection ha mostrato per tutti i datasets la riduzione dei tempi d'addestramento e di predizione, come ci si aspettava data la riduzione della dimnensionalità che offre questo metodo. Si è notato però anche un miglioramento dell'accuracy sul test per ciascuno dei datasets presi in esame, che non era garantito a priori (vedi Tabella 4.29).

Dataset	% Accuracy Gain	% Fit time gain	% Predict time gain
1	+2,4%	20,1%	12%
2	+2,1%	18%	15%
3	+0,1%	16%	6,6%
4	+1,51%	3,61%	0,02%
5	+1,2%	13%	4,2%
6	+0,2%	15%	34%
7	+6,7%	4,5%	0,2%

Tabella 4.29: Tabella riassuntiva dei datasets

Alcuni casi particolari come il dataset 4 mostrano come, pur essendo un metodo che sicuramente offre dei vantaggi, questi possono essere meno percettibili quando sono presenti variabili fortemente irrilevanti ai fini della classificazione. In questa situazione, il metodo resta comunque di grande importanza per il rilevamento di quelle correlate alla variabile target.
A parte due eccezioni (dataset 4 e dataset 7), il fit time risulta ridotto dell'ordine del 15% ed acquista particolare rilievo nel caso dei datasets con istanze dell'ordine delle decine di migliaia, dove il tempo d'addestramento è dell'ordine della decina di secondi. Per i datasets 4 e 7 la riduzione è più modesta e si attesta attorno al 4%.
Inoltre, si è notato il ruolo delle variabili non selezionate nel parametro di train accuracy gain: 5 datasets su 7 mostrano una diminuizione dell'accuracy sul training set in seguito alla feature selection, pur guadagnandone in accuracy sul test. In questo caso, quindi, la presenza delle variabili non scelte addestra la rete in maniera troppo selettiva sul training set e meno adatta ai nuovi dati che le vengono

presentati nel test. Per i datasets 4 e 7 invece il metodo porta ad un miglioramento anche dell'accuracy del train, comparabile per ordine di grandezza a quella che si ritrova con il test: in questo caso, quindi, il "noise" generato dalle variabili non rilevanti è tale da complicare l'addestramento fornendo quindi "esempi" distorti per l'apprendimento della rete neurale.
In conclusione, la feature selection ha fornito un miglioramento dell'accuracy sul test, ha ridotto i tempi dell'addestramento e della predizione per tutti i datasets; inoltre, per i datasets 4 e 7 ha migliorato la qualità dell'apprendimento, eliminando effetti di distorsione di variabili non rilevanti, mentre per i restanti datasets si è dimostrata uno strumento utile ad arginare l'overfitting causato dall'elevato numero di variabili.

Ringraziamenti

Il primo ringraziamento va alla Professoressa Autilia Vitiello, per la grandissima disponibilità mostrata. La ringrazio per la chiarezza dei consigli e per il contributo fondamentale per questa tesi.

Ringrazio tutti gli amici, mio fratello e soprattutto i miei genitori.

Cari mamma e papà, a voi dedico questo lavoro. Grazie

Bibliografia

[1] J.E. Smith A.E. Eiben. *Introduction to Evolutionary Computing*. Springer.

[2] H. Liu M. Dash. Feature selection for classification. *Intelligent Data Analysis 1*, 1997.

[3] Vahid Mirjalili Sebastian Raschka. *Python Machine Learning*. Packt.

[4] Dipanjan SarkarRaghav BaliTushar Sharma. *Practical Machine Learning with Python*. Apress.