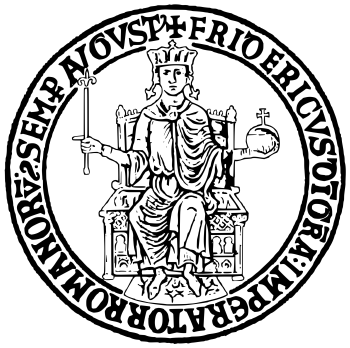
UNIVERSITÀ DEGLI STUDI DI NAPOLI
"FEDERICO II"



Scuola politecnica e delle scienze di base
Area Didattica di Scienze Matematiche Fisiche e Naturali

Dipartimento di Fisica "Ettore Pancini"

Laurea Triennale in Fisica

Algoritmi genetici per l'Instance Selection nel Machine Learning

Relatore:
Dott.ssa Autilia Vitiello

Candidato:
Edoardo Imparato
Matr. N85001450

Anno Accademico 2023/2024

Indice

Introduzione

L'Instance Selection rappresenta un importante argomento di studio nel Machine Learning a causa della sua influenza nelle performance di un modello. Essa consiste nel selezionare esclusivamente i campioni più significativi all'interno di un Dataset.
L'eliminazione di campioni meno significativi comporta una diminuzione del tempo di addestramento del modello rappresentando un grande vantaggio per qualsiasi applicazione.
In questo lavoro di tesi l'Instance Selection sarà trattato come un problema di ottimizzazione e risolto tramite algoritmo genetico. Essendo un problema di ottimizzazione la soluzione è strettamente legata alla funzione di valutazione. La funzione di valutazione presa in considerazione in questo lavoro di tesi sarà l'accuracy di due modelli il Multi Layer Perceptro(MLP) e il K-nearest Neighbors(KNN). La scelta di tali modelli deriva dalla volontà di mostrare i guadagni nei tempi di fit e nei tempi di test che l'Instance Selection comporta. La MLP ha una fase di training la cui durata è strettamente legata alla dimensione del dataset di training mentre nella KNN, per sua struttura, è la durata del test ad essere dipendente dalla dimensione del dataset di training.
Tutti gli algoritmi sono stati sviluppati in python, utilizzando il modulo DEAP per l'algoritmo genetico e il modulo Scikit-learn per i modelli di classificazione.
La tesi è strutturata in 4 capitoli. Nel primo capitolo verrà introdotto il machine learning e in particolare la sua applicazione nei problemi di classificazione, verrano introdotti i due modelli sopra citati e presentate le procedure di preprocessing dei dati, introducendo così l'instance selection. Nel secondo capitolo si parlerà dei problemi di ottimizzazione, descrivendo poi la struttura e le proprietà degli algoritmi evolutivi, nello specifico quello genetico, introdotti come possibili approcci risolutivi a tali problemi. Nel terzo capitolo sarà descritto l'algoritmo genetico implementato per questa tesi. Nel quarto e ultimo capitolo verrano presentati e discussi i risultati delle sperimentazioni.

Capitolo 1

Machine Learning

1.1 Cos'è il Machine learning

Il machine learning è una branca dell'intelligenza artificiale nella quale sono raggruppate quelle procedure automatiche che hanno come scopo quello di creare un modello capace di portare a termine un compito o risolvere un problema a seguito di un addestramento coadiuvato da un set di dati. Il grande vantaggio di questo metodo è che il compito di selezione dei singoli parametri che risolvono il problema, ad esempio quello della classificazione, non sarà più compito del programmatore ma potrà essere trasferito al modello, che a seguito della fase di training li "imparerà" autonomamente, simulando quello che vuole essere l'apprendimento umano [3].
A seconda di come è strutturato il nostro set di dati esistono 3 tipi di approcci principali al machine learning [4]:

- Machine learning supervisionato in cui il modello viene allenato usando un set di dati labellati, ovvero dove sono indicati i vari output associati agli input
- Machine learning non supervisionato in cui viene usato un set di dati non labellati
- Machine learning per rinforzo dove il modello viene allenato tramite l'interazione con l'ambiente. Tuttavia a differenza del machine learning supervisionato il feedback di tali interazione non è dettato dalla corretta predizione dei label dei dati, ma da una "funzione premio" da massimizzare.

1.2 Classificazione

La classificazione dei dati è una delle applicazioni più diffuse e utili del machine learning. Consiste nel prendere un set di dati e suddividerli in due (classificazione binaria) o più categorie. Per risolvere il problema della classificazione l'approccio

di machine learning più utilizzato è quello supervisionato. La classificazione dei dati avviene tramite i seguenti passaggi principali:

- selezione e preparazione del set di dati più adatto alla rappresentazione dell'ambiente in cui opera;
- scelta del modello per la classificazione;
- adestramento di tale modello su un set di training;
- confronto delle predizioni del modello allenato su un set di test.

Da qui verranno discussi alcuni dei più comuni modelli di classificatori.

1.2.1 Reti neurali

I classificatori più comuni sono le reti neurali. Esse sfruttano il modello di funzionamento dei neuroni nel nostro cervello: gli input vengono passati tramite i dendriti al nucleo della cellula dove avverà l'elaborazione, il segnale cosi generato se supera la soglia di attivazione del nostro neurone verrà inviato come output. Il percettrone [5] è l'esempio più semplice che sfrutta tale modello, esso è formato da un solo neurone che prende i vari input associandogli dei pesi, inizializzati in modo randomico e tarati in una fase di training, e 'spara' un output qualora il segnale elaborato superi il threshold, che in senso pratico indica l'appertenza ad una classe.

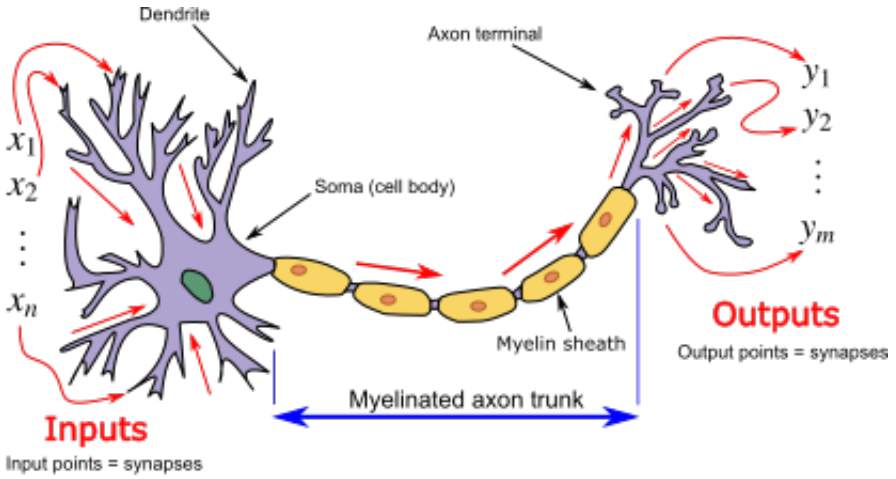


Figura 1.1: Schema di funzionamento del neurone (fonte:google immagini)

La multilayer perceptron (MLP) è la rete neurale più utilizzata. La MLP è formata da un numero potenzialmente infinito di layer, minimo tre, formati da più nodi, che singolarmente rappresentano un percettrone. Uno dei vantaggi dell'uso della MLP è nell'utilizzo di una funzione di attivazione non lineare. La funzione di

attivazione può essere qualsiasi tipo di funzione non lineare ad eccezione di quelle polinomiali, le più utilizzate al momento sono:
la funzione sigmoide

$$f(s) = \frac{1}{1 + e^{-s}} \tag{1.1}$$

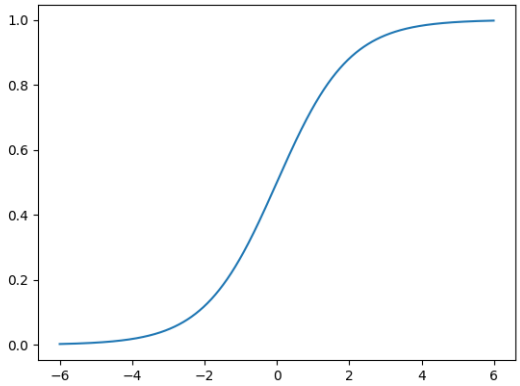


Figura 1.2: funzione sigmoide

la funzione sigmoide bipolare

$$f(s) = \frac{1 - e^{-as}}{1 + e^{-as}} \tag{1.2}$$

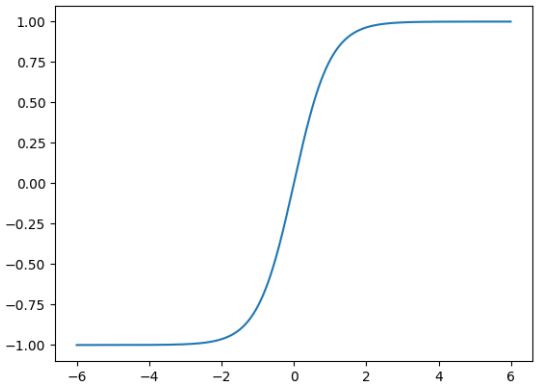


Figura 1.3: funzione sigmoide bipolare con a=2

Possiamo osservare come per valore piccoli di $|s|$ abbiamo un andamento lineare delle funzioni di attivazione che satura per vaolri alti di $|s|$. Un altro vantaggio dell'utilizzo di una MLP è quindi la possibilità di risolvere problemi di classificazione anche non linearmente separabili.

I layer della MLP sono suddivisi in: un input e un output layer, che si interfacciano con l'ambiente, e un numero potenzialmente infinito di hidden layer, interni alla rete, che interagiscono esclusivamente con altri layer.

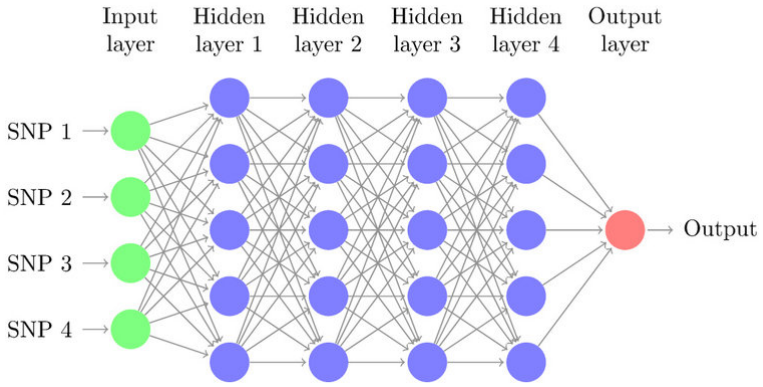


Figura 1.4: Struttura MLP (fonte:Researchgate.com)

Ogni nodo di un layer è collegato e scambia segnali con tutti i nodi del layer successivo e precedente. I segnali possono passare attraverso il sistema in modo unidirezionale dall'input all'output definendo un'architettura feed-forward, in cui non c'è nessun ciclo e i segnali di output non alterano il modello. Più interessanti sono invece le architetture feed-back, dove i segnali possono muoversi in entrambe le direzioni. Le architetture feed-back cambiano dinamicamente i loro parametri finchè il sistema non raggiunge un punto di equilibrio. Dal punto di vista matematico possiamo considerare i collegamenti tramite dei pesi che ne identificano l'importanza o meno ai fini della risoluzione del problema, questi pesi vengono tarati tramite un training supervisionato, ripetuto per varie epoche, suddiviso in:

- forward propagation del segnale fino a generare un output;
- calcolo dell'errore tramite il confronto dell'output del modello con il target;
- aggiornamento dei pesi.

Forward propagation

All'interno di una MLP ogni layer è input del layer successivo, in particolare l'input layer ha come segnale di ingresso $s_i = w_0^{(h)} * a_o + w_0^{(h)} * a_1 + ... + w_o^{(h)} * a_n$ o in forma

compatta $S^{(i)} = A * W^{(i)}$, dove A è la matrice di input di dimensione $n * m$, con n il numero di istanze e m il numero di feature, $W^{(i)}$ è la matrice dei pesi di dimensione $m * d$ dove d rappresenta il numero di nodi del layer, $S^{(i)}$ è la matrice dei segnali $n * d$ a cui dovrà essere applicata la funzione di attivazione sigmoide Φ, ottenendo la matrice dell'attivazione $A^i = \Phi(S^{(i)})$ di dimensione $n * d$. Il funzionamento dei successivi layer è il medesimo con l'eccezione di $A = S^{(i-1)}$.

Errore e aggiornamento dei pesi

Il segnale una volta uscito dal layer di output viene confrontato con il target usando la funzione:

$$J(\omega) = \frac{\sum_i (y_i - \Phi(s_i^{(o)}))^2}{2} \tag{1.3}$$

definita la funzione di costo, dove s_i è l'output della MLP e y_i rappresenta il target. Essendo la funzione convessa è possibile utilizzare un semplice quanto potente meccanismo, la discesa del gradiente (fig. 1.5)

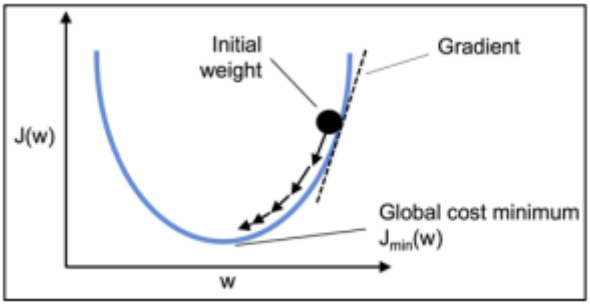


Figura 1.5: Batch gradient descent (fonte:Python machine learning Second Edition)

Una volta calcolato $\nabla J(\omega)$ della nostra funzione possiamo calcolare la variazione dei pesi:

$$\omega = \omega + \Delta\omega \tag{1.4}$$

Dove definiamo $\Delta\omega$ come:

$$\Delta\omega = -\eta \nabla J(\omega) \tag{1.5}$$

dove il termine η rappresenta il learning rate.
Per scrivere il gradiente della funzione di costo dobbiamo svolgere le derivate parziali rispetto i pesi:

$$\frac{\partial J(\omega)}{\partial \omega} = \sum_i (y_i - \Phi(s_i^{(o)}))s_i \tag{1.6}$$

sfruttando la relazione $s_i = \sum_i \omega_i * s_i^{(h)}$.
Usando la 1.6 la 1.5 diventa:

$$\Delta J(\omega) = -\eta \frac{\partial J(\omega)}{\partial \omega} = -\eta \sum_i (y_i - \Phi(s_i^{(o)}))s_i^{(h)} \tag{1.7}$$

Questo metodo di discesa del gradiente dove i pesi vengono aggiornati dopo il passaggio di tutte le istanze nella MLP si chiama *batch gradient descent* [6]. In presenza di dataset molto grandi non è la scelta ottimale e si preferisce un approccio in cui i pesi vengono aggiornati dopo ogni output del modello, questo approccio prende il nome di *stocastic gradient descent*. Nonostante possa essere considerata un'approssimazione del precedente approccio, grazie alla maggior frequenza di aggiornamento dei pesi converge più velocemente. Nel *stocastic gradient descent* poichè i pesi vengono aggiornati ad ogni iterazione viene generato del rumore e il $\Delta\omega$ assume valori maggiori, ciò è un vantaggio in quanto questo rumore permette di fuggire da eventuali minimi locali.

1.2.2 K-nearest neighbors

Il K-nearest neighbors è un altro esempio di classificatore. L'idea sul quale si basa è quella che la classe di appartenenza di una determinata istanza è simile a quella del suo vicino. Nello specifico il controllo non avviene esclusivamente con un solo vicino ma generalmente con k istanze vicine, il modello assegnerà la classe più ricorrente nei suoi vicini dando, in caso di parità, più importanza al vicino con distanza minore. Per il funzionamento dell'algoritmo è necessario introdurre una distanza. La distanza può essere misurata in vari modi:

- distanza Euclidea: rappresentata dalla distanza delle due istanze viste come vettori di uno spazio vettoriale n-dimensionale. Prese due istanze $p = (p_0, p_1, ..., p_n)$ e $q = (q_0, q_1, ..., q_n)$ dove le p_i e le q_i rappresentano le feature delle istanze, la loro distaza Euclidea è definita:

$$d(p, q) = \sqrt{\sum_i (p_i - q_i)^2} \tag{1.8}$$

- distanza di Manhattan: consideriamo le istanze come vettori n-dimensionali la cui distanza e rappresentata dalla somma delle proiezioni sugli n assi del segmento che li congiunge:

$$d(p, q) = \sum_i p_i - q_i \tag{1.9}$$

- similarità del coseno: tale relazione deriva dal prodotto scalare tra due vettori. Abbiamo:

$$p \cdot q = ||p||||q||cos(\Theta) \tag{1.10}$$

$$cos(\Theta) = \frac{p \cdot q}{||p||||q||} \tag{1.11}$$

$cos(\Theta)$ indica la similarità tra le due istanze variando da -1 a 1, -1 in caso di istanze contrapposte, 1 nel caso di istanze identiche e assumendo il valore 0 nel caso di due istanze indipendenti.

- Correlazione: la distanza di correlazione di due variabili casuali è definita come il rapporto della covarianza delle due variabile con la differenza tra le loro deviazioni standard:

$$d(p,q) = \frac{cov(p,q)}{\sigma_p - \sigma_q} = \sqrt{n}\frac{\sum_i (p_i - \mu_p)(q_i - \mu_q)}{\sqrt{\sum_i (p_i - \mu_p)^2} - \sqrt{\sum_i (q_i - \mu_q)^2}} \tag{1.12}$$

La KNN non avendo parametri da tarare non ha una fase di training come quella della MLP. Il training della KNN consiste invece nel costruire l'ambiente di variabili dove in fase di test verrà inserito l'input da classificare, permettendo cosi di calcolarne i vicini e quindi la classe

1.3 Preparazione dei dati

Abbiamo quindi visto come tutti i classificatori necessitano di un dataset di training anche nel caso di modelli particolari come ad esempio la KNN. Risulta necessario che il dataset venga diviso in un set di training e in un set di test, ciò però non può avvenire subito è neccessario un controllo a priori (preprocessing) dei dati affinchè il modello possa sfruttarli al meglio(fig. 1.6).

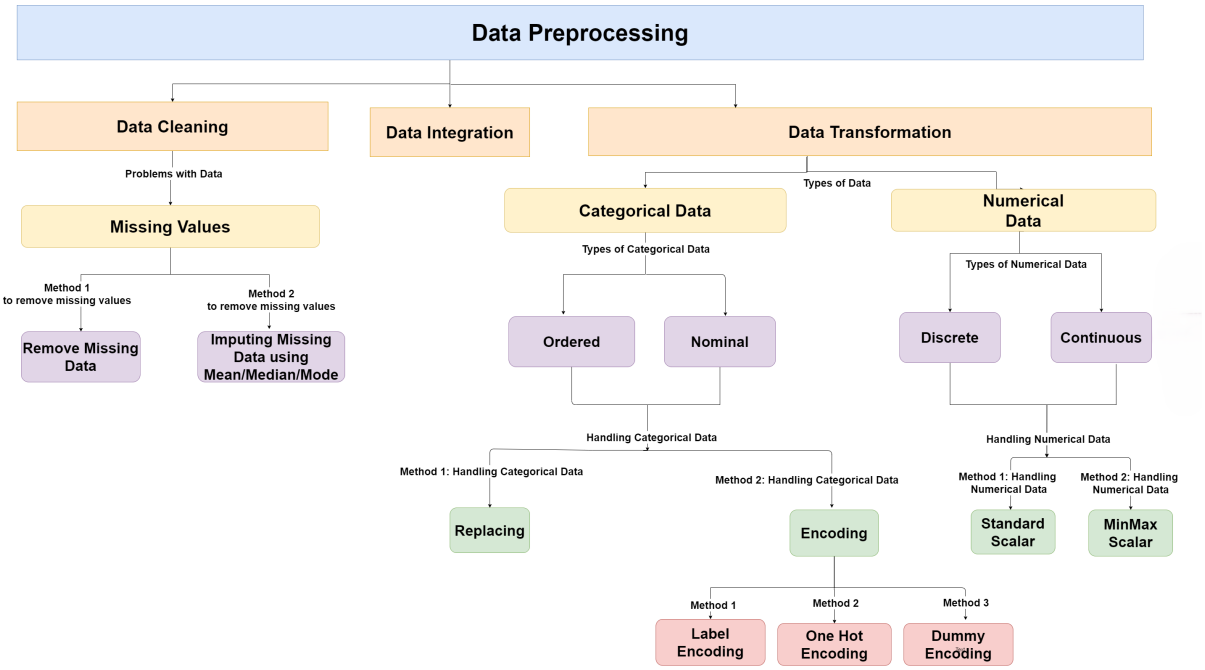


Figura 1.6: Data preprocessing (fonte:ai-ml-analytics.com)

- Data Cleaning: Il primo controllo da fare consiste nel controllare la presenza di valori mancanti all'interno del dataset, non è raro che in fase di raccolta dati per motivi ambientali, hardware o software le strumentazioni di presa dati possano avere problemi di funzionamento causando la presenza di "buchi" in determinate intanze. Le soluzioni principali sono due:
  - eliminare completamente l'istanza incompleta
  - "riempire" il buco, assegnandogli un valore a seguito di uno studio della distribuzione statistica di quella determinata feature
- Data Integration: Non sempre un solo dataset è sufficiente per assicurare la miglior copertura del problema da classificare. Spesso quindi vengono uniti, a seguito sempre di un'analisi sulla compatibilità della struttura dei dataset e delle condizioni e delle metodologie di prese dati, più dataset tra di loro.
- Data Transformation: L'ultimo controllo è quello che interessa il formato dei dati. I modelli di classificatori descritti in precedenza per poter funzionare necessitano che le varie feature siano rappresentate in forma numerica. Ciò non è assicurato a priori, infatti alcuni dataset hanno rappresentazioni nominali di alcune feature. Ad esempio alcune banche nel valutare se concedere o meno

un prestito ad un cliente utilizzano dataset dove il fattore di rischio è espresso con termini quali: "basso,medio,alto". La conversione dei dati può essere applicata anche quando le feature sono già rappresentate numericamente con l'obiettivo, in questo caso, di ottenere dati più manipolabili dall'algoritmo, ad esempio normalizzandoli o standardizzandoli.

1.4 Instance selection

Non sempre l'utilizzo diretto del dataset dopo il preprocessing risulta essere la scelta più efficiente. Molti dataset cercando di coprire il più possibile il campo di studio del problema, presentano una dimensione che può variare dalle migliaia ai milioni di istanze. Tali dimensioni comportano lo svantaggio che i dataset possano presentare delle istanze "eccezionali", che producono del rumore. L'instance selection è un metodo di preprocessing che ha come obiettivo l'eliminazione del rumore e la conseguente riduzione delle istanze del dataset. L'obiettivo teorico dell'instance selection è quindi l'ottenimento del più piccolo sottoinsieme di istanze del dataset senza eccessive perdite di accuracy rispetto all'insieme completo [1]. Il procedimento per l'Instance selection è mostrato in fig. 1.7.

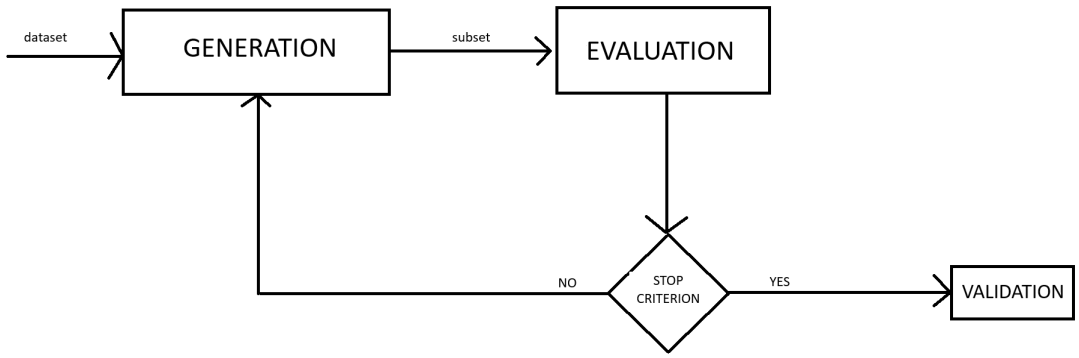


Figura 1.7: Instance selection

Generation

Con generation si intende la creazione del sottoinsieme di istanze del dataset, questo può avvenire in diversi modi:

- La generazione di tutti i possibili sottoinsiemi del dataset, questo approccio nonostante sia facilmente implementabile e assicuri di trovare il migliore sottoinsieme possibile per quanto riguarda l'accuracy, comporta un'occupazione di memoria e costo temporale enorme a partire già da valori piccoli di N,

essendo il numero di sottoinsiemi generabile da un dataset di N istanze di andamento esponenziale pari a 2^N.

- L'approccio euristico, più efficiente temporalmente e nell'occupazione della memoria, consiste nel generare inizialmente il sottoinsieme in modo randomico e modificarlo a seguito di ogni iterazione.

Evaluation e Stop

La bontà dei sottoinsiemi così generati viene valutata da una funzione definita appunto di valutazione. Nell'approccio euristico lo studio della variazione della funzione di valutazione permette all'algoritmo di capire come modificare il sottoinsieme prima di passare all'iterazione successiva. Il sottoinsieme ottimale è quindi strettamente legato alla funzione di valutazione, al cambiare di essa dovrà neccessariamente cambiare anche il sottoinsieme ottimale associato. Fondamentale per l'instance selection è anche il criterio di stop, senza il quale il ciclo continuerebbe all'infinito senza dare mai un risultato. I criteri di stop possono essere molteplici e sono strettamente collegati al risultato che si vuole ottenere, un possibile criterio di stop è il raggiungimento di una determinata dimensione del sottoinsieme, questo può comportare una significativa riduzione dell'accuracy qualora la rischiesta sia eccessiva. Un altro criterio è il raggiungimento di una determinata accuracy, rischiando d'altro canto un ciclo infinito qualora tale accuracy non sia raggiungibile con i dati a disposizione. Lo stop può avvenire anche a seguito di un numero fissato di iterazioni, criterio scelto in questo lavoro di tesi, che rappresenta un buon compromesso in termini di accuracy e riduzione del dataset nonostante il rischio di bloccarsi in punti stazionari locali.

Validation

La fase di validation potrebbe essere anche considerata esterna al processo di instance selection, essa però viene comunemente usata come ulteriore controllo. La validazione si avvale di una funzione (fig. 1.8), che può essere diversa dalla funzione di valutazione (filter) oppure uguale (wrapper), che agisce sia sul dataset completo che sul sottoinisieme ottimale ottenuto precedentemente. Confrontando i due output abbiamo l'ulteriore controllo sulla bontà dell'algoritmo.

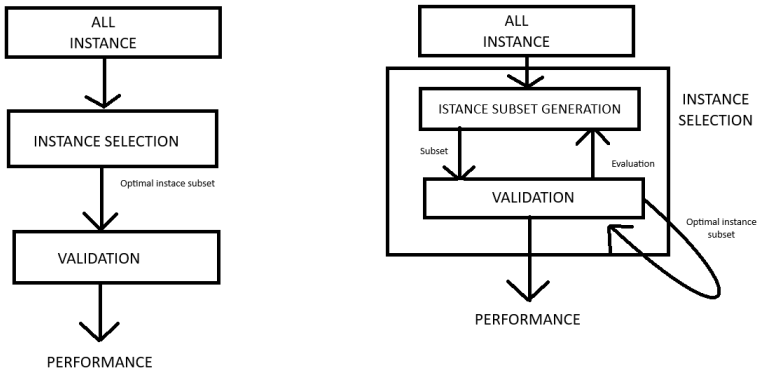


Figura 1.8: Filter vs wrapper

1.5 Approcci risolutivi

La letteratura scientifica per l'instance selection è molto ampia e, nel 1999 da Pradhan e Wu e successivamente nel 2004 da Jankowski e Grochowski, fu fatto un censimento dei vari metodi proposti. Furono notati tre gruppi principali in cui suddividere gli algoritmi: noise filters, condensation algorithms e prototype searching algorithms.

ENN

L'Edited Nearest Neighbor (ENN), sviluppato da Wilson D. Randall nel 1972, è un algoritmo di noise filters. L'algoritmo crea inizialmente un subset di istanze S identico al set completo T, ogni istanza verrà poi eliminata qualora la sua classe non coincida con quella della maggioranza dei suoi k vicini (con $k = 3$ tipicamente). L'ENN comporta quindi l'eliminazione di istanze rumorose aumentando così i gap tra le varie classi.

CNN

Il Condensed Nearest Neighbor Rule, ideato nel 1968, è uno dei primi algoritmi applicati all'instance selection. L'algoritmo inizializza il subset S selezionando un'istanza del set T in modo randomico, ogni istanza del set T verrà poi classificata tramite il subset S. Se la classificazione di una determinata istanza del set T è concorde con la classe dell'instanza, allora l'algoritmo passerà all'instanza successiva viceversa l'instanza che non viene classificata correttamente dal subset S verrà aggiunta ad esso e l'algoritmo continuerà finchè tutte le istanze di T saranno classificate correttamente.

RNN

Il Reduced Nearest Neighbor Rule (1972), è un algoritmo che partendo da un subset $S = T$ elimina tutte quelle istanze di S che una volta rimosse non comportano una classificazione errata delle istanze di T. Nonostanze questo algoritmo sia computazionalmente più costoso del CNN, il subset S risultante è un sottoinisieme di quello della CNN, ottenendo così un guadagno computazionale e di memoria in fase di classificazione.

Prototypes

Il Prototypes (1974) è un algoritmo dall'approccio molto particolare, esso infatti non seleziona semplicemente le istanze di T ma le modifica completamente. L'algoritmo seleziona due istanze vicine e le "unisce" sostituendole con una sola istanza, ottenuta dalla media pesata delle due, questo processo viene iterato finchè non si inizia ad osservare una perdita significativa dell'accuracy.

Algoritmi Genetici

Gli algoritmi genetici rappresentano un altro tipo di approccio molto efficace per l'instance selection [2], questi fanno parte degli algoritmi evolutivi, una particolare categoria di algoritmi che sfrutta le maccaniche principali dei processi evolutivi. Questi algoritmi generano una popolazione di individui, ognugno di essi rappresenta un determinato subset S, che verranno sottoposti a un processo di evoluzione, basato principalmente su operatori genetici di crossover e mutazione, con lo scopo di ottenere individui migliori e quindi subset S più performanti.
Gli algoritmi genetici sono stati l'approccio scelto per questa tesi e saranno discussi in dettaglio nel prossimo capitolo.

Capitolo 2

Algoritmi genetici

2.1 Problema di ottimizzazione

Un problema di ottimizzazione in matematica ed in informatica consiste nel trovare la migliore soluzione tra tutte le possibili soluzioni accettabili. L'ottimizzazione quindi presuppone la presenza di soluzioni alternative da confrontare sulla base di un criterio o obiettivo scelto. Nella maggior parte dei casi il criterio o l'obiettivo è la minimizzazione di una funzione di costo o la massimizzazione di una funzione beneficio. In termini astratti per definire un problema di ottimizzazione bisogna definire un isieme delle soluzioni X e una funzione da ottimizzare $f(x)$ definita per ogni $x \in X$, di conseguenza il problema di ottimizzazione può avere due rappresentazioni:

$$\min_{x \in X} f(x) \qquad \max_{x \in X} f(x) \tag{2.1}$$

L'insieme X viene espresso tramite m vincoli:

$$g^{(m)}(x) \leq b \quad e \ o \quad k^{(m)}(x) = a \tag{2.2}$$

I problemi di ottimizzazione a seconda della classificazione delle soluzioni sono divisi in:

- Problemi di ottimizzazioni continui: dove le variabili assumono varoli continui, nello specifivo se:
  - $X = \mathbb{R}^n$ sono non vincolati
  - $X \subset \mathbb{R}^n$ sono vincolati
- Problemi di ottimizzazioni combinatori: dove le variabili assumono valori interi, distinguendo due classi:
  - problema a numeri interi $X \subseteq \mathbb{Z}^n$
  - problemi binari $X \subseteq 0, 1^n$

- Problemi misti: una combinazione dei precedenti.

Per risolvere il problema di ottimizzazione bisogna definire un metodo di risoluzione. Un metodo di risoluzione può essere *esatto* se permette di trovare la migliore soluzione possibile, o *euristico* se orientato alla ricerca di una soluzione valida ma non necessariamente ottima.

2.2 Algoritmo evolutivo

L'algoritmo evolutivo (EA) emerge come uno dei metodi più accessibili per affrontare problemi di ottimizzazione, esso trae ispirazione dal modello di evoluzione proposto da Darwin [7]. In un ambiente caratterizzato da una capacità limitata di individui, ognuno con l'impulso intrinseco alla riproduzione, il processo di selezione naturale si rivela inevitabile. La selezione naturale, in un contesto in cui gli individui competono per risorse limitate, favorisce coloro che meglio si adattano alle caratteristiche dell'ambiente, fenomeno noto come "sopravvivenza del più adatto". La selezione naturale costituisce uno dei pilastri dell'evoluzione, accompagnata dalla mutazione del fenotipo. Il fenotipo, rappresentato dalle caratteristiche comportamentali e fisiche di un individuo, determina il suo grado di adattamento all'ambiente (fitness). Ogni individuo manifesta un insieme di fenotipi che influenzano la sua fitness; quelli con un alto livello di fitness hanno maggiori probabilità di sopravvivere e riprodursi, trasmettendo parte dei propri fenotipi alla generazione successiva. Al contrario, gli individui con una bassa fitness tendono a scomparire, senza lasciare traccia dei loro fenotipi nella popolazione. Darwin ipotizzò che i fenotipi ereditabili potessero subire mutazioni randomiche, parziali o totali, tra una generazione e l'altra. Una volta formata la nuova generazione, il processo di evoluzione riprende. Nella pratica, il problema da risolvere costituisce l'ambiente stesso, mentre ogni individuo della popolazione rappresenta una possibile soluzione e la sua fitness riflette la qualità di tale soluzione [8]. Il processo di evoluzione continua fino al soddisfacimento di un criterio di arresto, come il raggiungimento di un certo numero di generazioni o la presenza di un individuo con un determinato valore di fitness. Gli algoritmi evolutivi sono tutti caratterizzati dal processo descritto precedentmente, essi si distinguono esclusivamente per la forma della loro popolazione: vettori binari per l'algoritmo genetico(GA), vettori reali per le strategie di evoluzione(ES), e alberi per il genetic programming(GP). La scelta dell'algoritmo evolutivo dipende strettamente dal tipo di problema che si vuole rappresentare, ad esempio se si vuole risolvere un problema con n variabili di cui bisogna selezionarne solo una parte, allora la miglior rappresentazione è quella dell'algortimo genetico dove l'individuo è rappresentato da un vettore di 1 e 0, dove con 1 nella i-esima posizione si intende la selezione di quella variabile.

2.3 Elementi dell'algoritmo evolutivo

Il primo passo per la creazione di un algoritmo evolutivo come accennato in precedenza è quello della rappresentazione della popolazione [9]. Tale processo serve per ottenere la mappatura più adatta tra lo spazio del fenotipo, le effettive variabili del problema, e gli elementi degli individui nell'algoritmo evolutivo (genotipo). Tale processo quindi consiste in una fase preliminare dell'algoritmo per ottenere la rappresentazione dell'individuo più manipolabile dalla fase di ricerca dell'individuo migliore. Questa avvenendo nello spazio del genotipo darà come risultato un individuo che dovrà essere posto a decodifica per ottenere i fenotipi e quindi la soluzione annessa. Nella terminologia usuale gli elementi che compongono un individuo nello spazio del genotipo sono chiamati geni. Una volta scelta la rappresentazione più adatta si genera la popolazione. La popolazione può essere inizializzata in modo semplice, scegliendo una population size e inizializzando in modo randomico tutti gli individui, ma esistono anche strutture più complicate che introducono concetti di distanza in cui la generazione degli individui è condizionata dai suoi vicini. La population size rimane costante in modo da accentuare la competizione tra gli individui.

2.3.1 Funzione di valutazione

Per la ricerca della soluzione e dell'individuo più adatto è necessario calcolarne il suo fitness tramite una funzione di valutazione. Essa agisce tipicamente, prima decodificando l'individuo e poi restituendone un misura nello spazio del fenotipo. Ad esempio se la richiesta del problema è quella di trovare un intero x tale da minimizzare la funzione $\frac{1}{x}$, la nostra funzione dato l'elemento 001, lo convertirà nel suo corrispettivo decimale e ne calcolerà il suo valore $\frac{1}{x}$. Risulta immediato che la nostra funzione di valutazione rappresenta nella maggior parte dei casi il problema stesso dell'algoritmo evolutivo. I termini fitness o funzione di valutazione si riferisco sia ai casi in cui si ha un problema di massimizzazione che di minimizzazione.

2.3.2 Parent Selection

Con Parent Selection si intende il meccanismo con il quale si identificano gli individui più adatti a riprodursi in una popolazione. La nuova generazione(offspring) così generata presenterà tendenzialmente una miglior capacità di adattamento ereditata dalla vecchia generazione. La parent selection è un processo stocastico, quindi associa una maggiore probabilità di riprodursi ad individui con una fitness maggiore, questo vuol dire che individui a bassa fitness hanno possibilità non nulle di riprodursi, ciò impedisce all'algoritmo di convergere troppo velocemente e fermarsi in punti stazionari locali.

2.3.3 Operatori di Variazione

Gli operatori di variazione agiscono sugli individui di una popolazione per generarne di nuovi. Si possono distinguere due tipi di operatori, quelli che agiscono su un gene alla volta o quelli che agiscono su più geni in contemporanea:

- Mutazione: gli operatori di mutazione sono operatori che agiscono su un gene alla volta. Questi partendo da un individuo della popolazione restituiscono un individuo mutato, ovvero con uno o più geni mutati. La Mutazione è un operatore stocastico, esso modifica in modo randomico un gene partendo da un certo set di parametri come: la probabilità di mutazione, la frequenza di mutazione e lo step di mutazione. La mutazione permette al nostro algoritmo di ampliare il suo campo di ricerca permettendogli di raggiungere nuovi punti. Idealmente tramite la mutazione è possibile esplorare tutto lo spazio genotipico rendendo possibile trovare soluzioni globali al problema.
- Ricombinazione: gli operatori di ricombinazione o crossover sono operatori che agiscono su più geni in contemporanea. Questi partendo da due individui mescola il loro genotipo restituendo due o più individui. Questo processo vuole simulare la riproduzione naturale permettendo così idealmente la creazione di individui con geni più performanti e con fitness migliori.

2.3.4 Enviroment Selection

L' offspring generata a seguito di questi processi presenterà una population size maggiore di quella di partenza, sarà quindi necessario sfoltire la popolazione. Ciò avviene in modo deterministico basandosi su vari fattori, si può decidere infatti di selezionare gli individui da conservare in base al loro valore di fitness, conservando solo gli inividui più adatti, oppure in base all'età dell'individuo eliminando completamente i genitori e lasciando solo i figli.

2.3.5 Condizione di terminazione

L'algoritmo evolutivo in assenza di condizione di stop potrebbe continuare all'infinito. Tale condizione, qualora fosse già conosciuta la soluzione ottimale del problema coinciderebbe, con il raggiungimento del genotipo corrispondente oppure, considerando eventuali rumori e approssimazioni del problema, ad un suo intorno. Tale condizione però rappresenta una situazione ideale che di rado si presenta, infatti i criteri di stop più utilizzati negli algoritmi evolutivi generalmente sono: il raggiungimento di un determinato tempo di esecuzione o di numero di generazioni, il raggiungimento di un massimo nella fitness degli individui, la condizione per cui i miglioramenti delle fitness degli individui rimangono sotto una certa soglia di threshold per un certo numero di generazioni, oppure la diversità della popolazione

rimane inferiore ad una certa soglia preimpostata sempre per un certo numero di generazioni.

2.4 Algoritmo genetico

L'algoritmo genetico sfrutta la più semplice e la più vecchia delle rappresentazioni, quella binaria. Il genotipo è formato da un vettore di 0 e 1. In base al problema che vogliamo risolvere bisogna scegliere la dimensione del vettore e come esso codifichi il fenotipo. Fondamentale è controllare che tale rapprentazione riesca a mappare tutti i fenotipi accettabili del problema. Nel caso di problemi associati a scelte Booleane la rappresentazione binaria mappa facilmente tutte le possibili combinazioni essendo binario anche il problema. La rappresentazione binaria può essere applicata anche in caso di problemi non binari, ad esempio un vettore di 80 elementi può essere diviso in gruppi da 8-bit che codificano ognuno un numero decimale. L'uso della rappresentazione binaria in questi problemi però è un errore, si otterebbero migliori risultati utilizzando una rappresentazione a numeri reali. Il principale problema della rappresentazione binaria per problemi non binari è che i bit in questo caso hanno significati diversi, ciò influisce negativamente sulle mutazioni. Dato un individuo di partenza e data una mutazione, le possibili mutazioni dovrebbero avere tutte la stessa probabilità, in questo caso ciò non viene rispettato. Se per esempio l'obiettivo è variare un numero reale, dato il numero 6 vogliamo avere la stessa probabilità di mutarlo in un numero 7 o 5, in binario ciò non è vero perchè per passare da 0110(6) a 0101(5) servono due mutazioni flipbit, mentre per passare a 0111(7) serve una sola mutazione. L'operatore di mutazione più utilizzato nella rappresentazione binaria è il flipbit, esso considera ogni gene separatemente invertendo 0 con 1 e vicerversa con una probabilità p_m. Il numero di bit cambiati non è fissato ma dipende dalla lunghezza del vettore, data un lunghezza L il numero medio di bit mutati sarà $L \cdot p_m$.

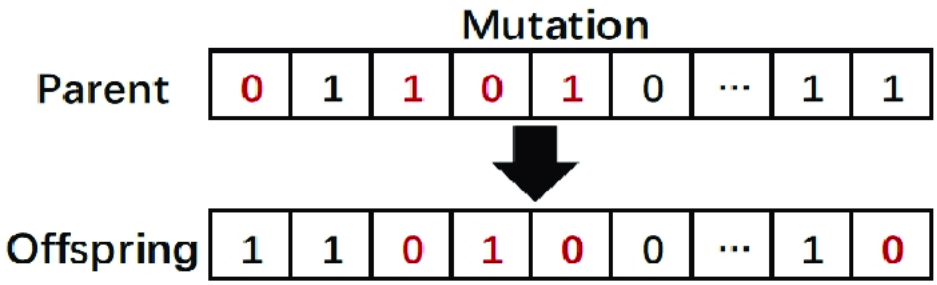


Figura 2.1: Bitflip (fonte:google immagini)

La ricombinazione nella rappresentazione binaria agisce in tre modi, tutti e tre i modi partono da due genitori per generare due individui figli.

One-point crossover

Il One-point crossover è il primo operatore di ricombinazione implementato, consiste nel generare un numero randomico $r \in [1, l-2]$, dove l rappresenta la lunghezza dell'individuo. I geni dei genitori vengono spezzati in due frammenti $[0, r]$ e $]r, l-1]$ che vengono invertiti tra di loro generando i due figli. La scelta dell'intervallo di generazione di r tra $[1, l-2]$ non è casuale, con questa scelta si impedisce ad r di assumere valori problematici come 0 ed $l-1$ ovvero il primo e l'ultimo elemento a cui corrispondono crossover irrilevanti, infatti a 0 corrisponde il semplice scambio di tutti i geni e a $l-1$ lo scambio di nessun gene.

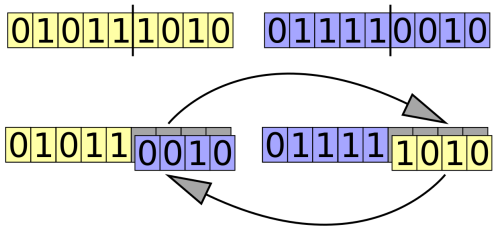


Figura 2.2: One-point crossover (fonte:wikipedia.org)

n-point crossover

L'n-point crossover è la generazione del one-point crossover, dove l'individuo viene spezzato in $n+1$ frammenti, ottenunti generando randomicamente n numeri diversi nell'intervallo $[1, l-2]$

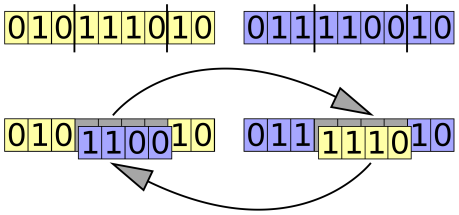


Figura 2.3: n-point crossover con n=2 (fonte:wikipedia.org)

Uniform crossover

L'uniform crossover a differenza dei precedenti crossover, vede i due genitori in maniera indipendente e sceglie in modo randomico se l'i-esimo gene dell'Offspring sarà preso dal primo o dal secondo genitore. Questo avviene generando un vettore

della stessa lunghezza l degli individui della popolazione formato da numeri reali generati randomicamente in un range $[0, 1]$. Scelto un valore p_c, generalmente del valore di 0.5, se l'elemento i-esimo del vettore precedentemente generato sarà $> p_c$ allora il gene verra scelto dall'individuo 1, se invece sarà $< p_c$ sarà scelto dall'indivduo 2. Il secondo individuo dell'offspring viene generato facendo scelte complementari a quelle precedenti.

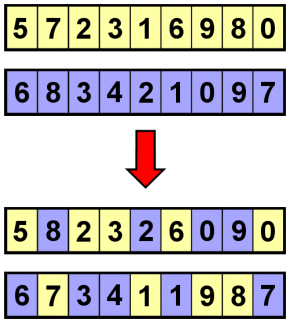


Figura 2.4: Uniform crossover (fonte:wikipedia.org)

L'n-point crossover, segmentando il genetipo, tende a conservare i geni vicini portando a un bias posizionale, tale bias è accentuato nel momento in cui n è dispari, risultando in una tendenza dell'operatore nel separare il primo e ultimo gene dell'individuo. L'uniform crossover nonostante non presenti bias posizionali, ha la tendenza a far ereditare il 50% dei geni di ogni genitore, impedendo la trasmissione di un largo set di geni più performanti da parte di un solo genitore, generando così un bias di tipo distribuzionale. Per la presenza di questi bias non c'è un metodo di crossover migliore a priori, la scelta sarà strettamente collegata al problema da affrontare, al tipo di decodifica del genotipo e i suoi pattern conosciuti. Come accennato in precedenza non tutti gli individui della specie si riproduranno, essi verranno scelti durante la fase di parent selection.

Fitness Proportional Selection

La fitness proportional selection assegna la probabilità di riproduzione di un individuo in base alla sua fitness.

$$P_i = \frac{f_i}{\sum_{j=1}^{n} f_j} \tag{2.3}$$

Dove il termine $\sum_{i}^{n} f_i$ è il temine di normalizzazione. I problemi principali di questo metodo sono:

- La presenza di fitness molto alti rispetto agli altri individui porta a una loro predominazione e ad una convergenza prematura dell'algortimo;

- Quando i valori di fitness sono molto vicini tra di loro non c'è una significativa predominanza di un gruppo di individui portando a una lenta modifica della popolazione;
- La selezione agisce in modo diverso qualora il fitness fosse trasposto.

Per risolvere gli ultimi due problemi viene applicato il metodo del windowing. Le differenze tra le fitness vengono mantenute sottraendo un termine β^t, dipendente dal tempo, alla fitness f_i. Il termine β^t nell'approccio più semplice può assumere il peggior valore di fitness in un determinato istante di tempo

$$\beta^t = \min f_i \tag{2.4}$$

Rank Selection

La rank selection non utilizza direttamente la fitness dell'individuo ma tramite essa crea un ranking nella popolazione ed utilizza il rank dell'individuo per assegnargli una probabilità di selezione, in tal modo preserva la pressure selection dell'ambiente. Il rank dell'individuo va da un range di $n-1$(fitness migliore) a 0(fitness peggiore) dove n è la dimensione della popolazione. La conversione tra rank e probabilità di selezione può essere lineare:

$$P_{li} = \frac{2-s}{n} + \frac{2i(s-1)}{n(n-1)} \tag{2.5}$$

Dove s è un parametro appartenente all'intervallo $]1, 2]$. La conversione lineare applica un pressure selection limitata e qualora servisse una pressure selection maggiore si usa una coversione esponenziale.

$$P_{expi} = \frac{1-e^{-i}}{c} \tag{2.6}$$

Dove il termine c rappresenta il fattore di normalizzazione.

Tournament Selection

I metodi precedenti sfruttano la conoscenza dei fitness di tutta la popolazione, in caso di popolazioni dalle grandi dimensioni ciò richiederebbe un grande sforzo computazionale e di tempo. La tournament selection risolve questo problema confrontando tra di loro non tutti gli individui in contemporanea ma un numero k, che rappresenta la torunament size, restituendo il migliore tra di loro.

Uniform Parent Selection

In alcuni approcci la probabilità di essere selezionati per la riproduzione è fissata per tutti gli individui. Questo approccio, nonostante possa sembrare controintuitivo in quanto non favorisce l'ereditarietà di geni "migliori", se affiancata da

una survival selection fortemente basata sul valore dei fitness, permette una maggior varietà nelle combinazioni di geni raggiungendo nuovi punti nello spazio del genotipo.

Survival Selection

Conseguenza della riproduzione degli individui è la crescita della popolazione, che rende necessario l'eliminazione degli individui superflui fino al raggiungimento della dimensione iniziale della popolazione. Tale processo prende il nome di survival selection, può avvenire tramite gli stessi metodi della parent selection oppure tramite strategie pensate esclusivamente per questo.

- Not fitness based. Metodi in cui il fitness della popolazione è ininfluente, un esempio è l'Age-Based Replacement, dove un determinato individuo viene selezionato in base al numero di generazioni in cui ha vissuto nella popolazione. Possiamo distinguere due casi prendendo un numero μ di genitori e un offspring λ, se $\mu = \lambda$ allora i genitori saranno completamente sostituiti dal'offspring rimanendo quindi nella popolazione una sola generazione, oppure $\mu > \lambda$ in questo caso affinchè la dimensione della popolazione rimanga invariata, basterà sostituire i λ individui più vecchi della popolazione. A differenza del primo metodo nel secondo gli individui eliminati dalla popolazione non saranno necessariamente i genitori della nuova prola.
- Fitness based. Metodi in cui la selezione dipende dal valore del fitness dell'individuo. Ad esempio se si ha un offspring di λ individui si elimineranno i peggiori λ individui della popolazione, oppure si creerà un torneo round-robin in cui k individui vengono fatti sfidare con q avversari, sempre della popolazione, e gli verrà assegnato un punto per ogni vittoria, alla fine il vincitore sarà selezionato. Fino ad ora abbiamo fatto esempi in cui l'offspring viene sempre conservata e gli viene costruito un posto nella popolazione eliminandone alcuni individui, è possibile però inserire l'offspring nella survival selction creando una nuova popolazione di $n + \lambda$ individui da cui poi verranno selzionati gli n individui della popolazione successiva, in tal modo si potrebbe rischiare, anche se con probabilità basse, il collasso dell'algoritmo in un minimo locale.

Capitolo 3

Progettazione e Implementazione

3.1 Progettazione dell'algoritmo genetico

3.1.1 La popolazione

Per definire un algoritmo genetico è necessario partire dalla scelta degli individui della popolazione. Nell'instance selection l'obiettivo è l'ottenimento del subset migliore per la classificazione, quindi gli individui verranno decodificati da una scelta booleana, ovvero di selezionare o meno una determinata istanza. In considerazione di ciò, la scelta di una rappresentazione binaria degli individui è la più adatta. Definiamo quindi l'individuo come un vettore di N elementi, dal valore 0 o 1, dove N coincide con la lunghezza del dataset da ridurre. I set associati ai nostri individui saranno quelli ottenuti selezionando soltanto le istanze nelle posizioni corrispondenti ai geni di valore 1. L'algoritmo restituirà il miglior individuo, ovvero quello con la fitness più alta, ottenuto in 25 generazioni.

3.1.2 Funzione di valutazione

Fondamentale è definire la funzione di valutazione, o in questo caso le funzioni di valutazione che definiscono la modalità di calcolo della fitness di ogni individuo. L'instance selection consiste nel selezionare le istanze migliori per la classificazione. Appare chiaro quindi che bisogna modellizzare il problema come di massimizzazione e che la nostra fitness debba coincidere con l'accuracy del modello usato. L'individuo è inizializzato come un vettore di 0 e 1 in modo randomico, con l'unica condizione che la lunghezza del vettore corrisponda a quella del numero di istanze del dataset. L'1 nella posizione $i-$esima verrà decodificato come la selezione dell'istanza $i-$esima nel dataset. La nostra funzione di valutazione convertirà il nostro individuo in un determinato subset che verrà usato per addestrare il modello (MLP o KNN). Dal test dell'accuracy del modello così allenato otterremo la fitness legata all'individuo. I passaggi della funzione sono:

- selezione delle istanze del train e validation set tramite l'individuo;

- accesso ai target dell'istanze selezionate;
- addestramento dei modelli sul train set ridotto;
- predizione del modello sul validation set ridotto;
- calcolo dell'accuracy della predizione del modello;
- restituzione del valore di fitness, ovvero l'accuracy del modello.

3.1.3 Operatori

Per questo algoritmo sono stati adotatti i seguenti operatori:

- la parent selection è stata effettuata tramite il metodo SelTorunament, utilizzando un tournsize di 3, valore di default per tale metodo. Il vantaggio di tale metodo risiede nel confronto diretto tra le accuracy degli individui. Non conoscendo a priori il valore ottimale che l'accuracy debba assumere, la rilevanza che un determinato valore possa avere è definita solo tramite il confronto diretto tra individui;
- la mutazione è effettuata tramite il metodo mutFlipBit, che cambia il valore dei geni da 0 a 1 e viceversa con una certa probabilità indipendente tra un gene e l'altro;
- per il crossover s'è scelto l'operatore cxTwoPoint, un operato $n-$point crossover con $n = 2$. L'operatore prenderà la sequenza di geni, delimitata dai due punti, di entrambi gli individui genitori invertendole tra di loro. Generando così due nuovi individui;
- la survival selection utilizzata è di tipo generazionale. L'offspring generata dagli operatori di generazione sostituisce completamente la generazione precendente. Essendo una strategia indipendente dal fitness essa può modificare la distribuzione statistica della popolazione. Tale conseguenza può essere utile per sfuggire da massimi locali;
- Il criterio di stop dell'algoritmo è stato attribuito alla generazione. L'algoritmo termina una volta raggiunta la venticinquesima generazione.

3.2 Implementazione

3.2.1 Python

La scelta del linguaggio di programmazione per l'implementazione è ricaduta su Python, un linguaggio di programmazione ad alto livello, con una grande versatilità. La scelta di Python è stata influenzata dalla vasta disponibilità di framework,

pachetti di file con funzioni e classi già definite, per il Data analysis, il Machine learning e la programmazione scientifica in generale. I framework principali utilizzati sono:

- **DEAP**, framework specifico per l'implementazione di algoritmi genetici.
- **Scikit-learn**, framework per il data analysis e il machine learning contenente algoritmi di classificazione, clastering, regressione.

Affiancati da altri framework dall'utilizzo più generico come:

- **Numpy**, per la programmazione scientifica;
- **Pandas**, per la manipolazione e l'analisi dati;
- **Matplotlib**, per la rappresentazione grafica dei dati.

3.2.2 Popolazione

Per l'implementazione della popolazione sono necessari i seguenti passaggi:

- creazione della classe fitness;
- creazione della classe individuo;
- definizione di una funzione per l'inizializzazione dei geni dell'individuo;
- definizione di una funzione per l'inizializzazione della popolazione.

Per tali passaggi si è sfruttato il modulo *creator*, messo a disposizione dal framework **DEAP**, che permette di creare le classi utili all'algoritmo. La classe fitness del nostro algoritmo erediterà dalla classe *Fitness* del modulo *base*, utile principalmente per l'attributo *weights*. L'assegnazone del valore 1 o -1 a tale attributo definisce il problema come di ottimizzazione o minimizzazione.

3.2.3 Operatori

Per implementare gli operatori si è sfruttato il metodo *toolbox*, sempre del modulo *base*. *Toolbox* è un metodo che permette di organizzare tutti gli operatori che si intendono utilizzare, il vantaggio è nella possibilità di registrare gli operatori memorizzando anche i parametri con cui li si vuole utilizzare. Ad esempio: *toolbox.register('mutate', tools.mutFlipBit, indpb=0.1)*.

3.2.4 La funzione di valutazione

Per le funzioni di valutazioni sono stati necessari due framework, **Scikit-learn** e **Pandas**. Come visto in precedenza la funzione di valutazione necessita di gestire e modificare i dataset, questo è possibile avvalendosi di **Pandas** e le sua classe

Dataframe, in cui sono importati tutti i dataset della tesi. I classificatori verranno importati invece dal framework **Scikit-learn**. In questo modo non solo si evita l'implementazione dei modelli, ma si possono sfruttare anche i metodi *fit* e *predict*, che permettono di addestrare e testare il modello evitando di implementare il relativo codice.
Essendo la survival selection indipendente dal valore della fitness, non è assicurata la permanenza nella popolazione degli individui più efficienti. Per risolvere tale problema è stata utilizzata la classe *HallOfFame* implementata anch'essa dal framework **DEAP**. Tale classe tramite il metodo *update*, "salva" i migliori n individui della popolazione, sostituendoli qualora comparissero nella popolazione individui migliori, in questo lavoro n è stato posto uguale a 1.

Capitolo 4

Analisi dei Risultati

L'obiettivo delle sperimentazioni è trovare per ciascun dataset il subset di istanze ottimale e confrontare le performance dei classificatori sul subset e sul dataset completo. L'applicazione dell'instance selection non assicura, che con l'addestramento tramite il subset, si ottenga un miglioramento del valore di accuracy. Per tutti i dataset risulta tuttavia una riduzione del tempo di training per la MLP e nel tempo di predizione per la KNN. Ogni dataset è splittato secondo la figura 4.1.

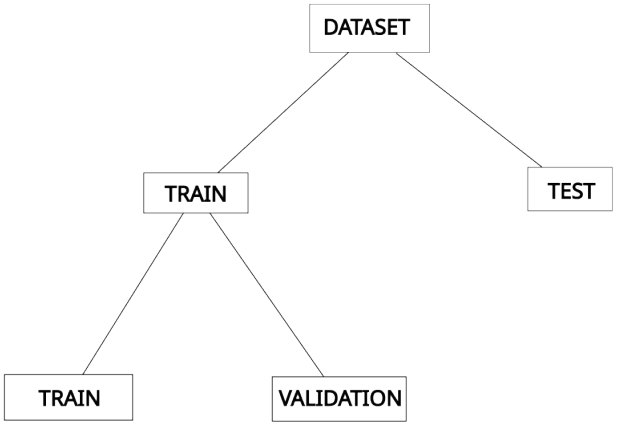


Figura 4.1: Dataset splitting

Il dataset viene inizialmente diviso in **TRAIN** e **TEST** set, dove il **TRAIN** rappresenta il 75% del dataset completo mentre il **TEST** il restante 25%. La divisione continua con l'ulteriore splitting del set di training in un altro set defini-

to per comodità anch'esso **TRAIN** e in un set **VALIDATION**, rispettivamente l'80% e il 20% del dataset orignale **TRAIN**. Questo ulteriore splitting è reso necessario dalla funzione **evaluate**, analizzata nel capitolo 3, poichè sfruttando i classificatori, anch'essa necessiterà di set per il training (**TRAIN**) e per il test (**VALIDATION**), che dovranno essere diversi da quelli poi utilizzati per le verifiche finali.

4.1 Datasets

4.1.1 Dataset 1: Breast Cancer Wisconsin Diagnostic

È un dataset che contiene 30 diverse caratteristiche di una massa tumorale, estratta dal seno tramite agobiopsia. Le caratteristiche, le nostre feature, sono estratte da un'immagine digitalizzata della massa

Dataset	Istanze	Tipo di feature
Breast Cancer Wisconsin Diagnostic	569	Continui, Categorici

Tabella 4.1: Dataset 1

4.1.2 Dataset 2: Hearth Attack

Cerca di predire la probabilità che avvenga un attacco cardiaco al paziente, categorizzato come 0(basso) e 1(alto), usando 13 feature come ad esempio: sesso, età, pressione a riposo ecc.

Dataset	Istanze	Tipo di feature
Hearth Attack Analysis	303	Continui, Categorici

Tabella 4.2: Dataset 2

4.1.3 Dataset 3: Glass Identification

Ha l'obiettivo di identificare la provenienza dei frammenti di vetro, ad esempio: vetro di un tavolo, stoviglia, finestre, fari o finestrini della macchina. Tale dataset viene utilizzato soprattutto nelle investigazioni, per classificare i frammenti di vetro delle prove.

Dataset	Istanze	Tipo di feature
Glass Identification	214	Continui, Categorici

Tabella 4.3: Dataset 3

4.1.4 Dataset 4: Ionosphere

Contiene dati presi da un phased array di 16 antenne ad alta frequenza con una potenza totale di trasmissione dell'ordine di 6,4 kilowatt. Il sistema invia dei segnali a elettroni liberi nella ionosfera, categorizzando i segnali di ritorno in "buoni" e "cattivi". I ritorni radar "buoni" sono quelli che mostrano l'evidenza di un qualche tipo di struttura nella ionosfera. I ritorni "cattivi" sono quelli che non vengono riflessi dalla ionosfera attraversandola.

Dataset	Istanze	Tipo di feature
Ionosphere	351	Continui,Categorici

Tabella 4.4: Dataset 4

4.1.5 Dataset 5: Iris

È uno dei primi dataset utilizzati per valutare i metodi di classificazione. Il dataset contine 3 classi da 50 istanze ciascuna.

Dataset	Istanze	Tipo di Feature
Iris	150	Continui, Categorici

Tabella 4.5: Dataset 5

4.1.6 Dataset 6: Hearth Failure

Contiene 12 feature per la predizione del rischio di mortalità tra pazienti con problemi cardiovascolari.

Dataset	Istanze	Tipo di Feature
Hearth Failure	299	Interi, Continui, Categorici

Tabella 4.6: Dataset 6

4.1.7 Dataset 7: Cellular Price

In questo dataset sono raccolte tutte le caratteristiche dei cellulari in commercio, categorizzati per fascia di prezzo, e precisamente 0(bassa), 1(media), 2(medio-alta), 3(alta).

Dataset	Istanze	Tipo di Feature
Cellular Price	2000	Interi, Continui, Categorici

Tabella 4.7: Dataset 7

4.2 Procedura raccolta dati

L'algoritmo genetico restituisce il migliore individuo comparso nelle 25 generazioni per cui è eseguito. L'individuo così ottenuto non è detto che corrisponda ad un ottimo globale, in quanto l'algoritmo in una delle generazioni potrebbe essere rimasto intrapollato in un ottimo locale senza riuscire a sfuggirne. Questo problema è influenzato dagli operatori evolutivi, che permettono all'algoritmo di muoversi nello spazio delle soluzioni arrivando a nuovi punti. Tali operatori sono legati a degli iper-parametri. In questa tesi tali parametri definiscono la probabilità che l'operatore associato agisca sull'individuo, la scelta di tali parametri solitamente avviene a seguito di una fase di "tuning". In questo lavoro si è però preferito utilizzare i valori standard tipici dei SGA(Simple Genetic Algorithm). Essi sono:

- **CXPB=50%**: la probabilità di riproduzione di due individui;
- **MUTPB=20%**: la probabilità di mutazione di un individuo.

L'individuo una volta "uscito" dall'algoritmo genetico passa alla fase di testing della qualità della soluzione. Qui avviene un processo simile a quello della funzione di valutazione:

- creazione del dataset relativo all'individuo;
- addestramento del modello con il dataset ridotto;
- addestramento di un altro modello utilizzando il dataset non modificato;
- predizione tramite modello addestrato con il dataset ridotto;
- predizione tramite modello addestrato con il dataset non ridotto;
- calcolo dell'accuracy dei due modelli;
- i proccessi di addestramento e predizione dei modelli vengono cronometrati e salvati.

Vista la natura stocastica dell'algoritmo i dati di seguito riportati sono stati ottenuti dalla media di 20 run del GA.

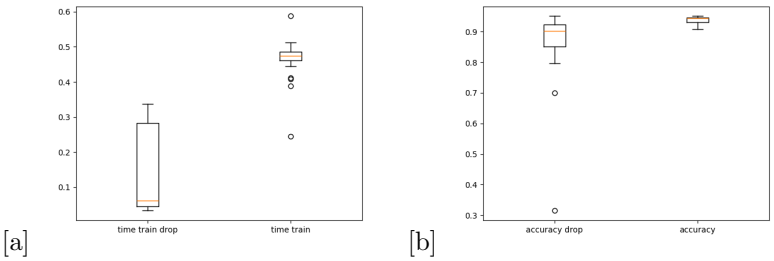
4.3 Risultati

4.3.1 Dataset 1

MLP

Instance Selection	Istanze	Tempo Fit	Accuracy
SI	170	0.1384	85%
NO	340	0.4622	93%

Tabella 4.8: MLP Dataset 1



[a] [b]

Figura 4.2: (a) Tempo Test: dataset modificato vs dataset completo (b) Accuracy: dataset modificato vs dataset completo

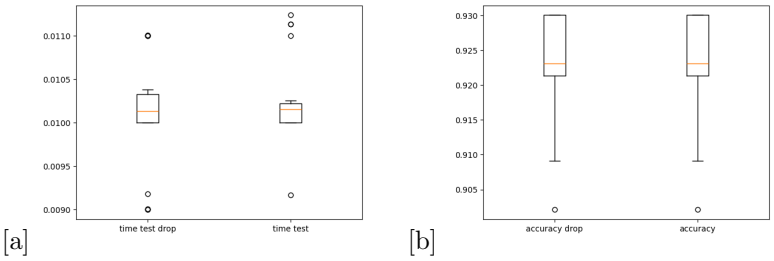
Delle 340 instanze del **TRAIN** ne sono state selezionate solo il 50%. Definiamo ora il **Gain**, il guadagno percentuale della differenza in tempo e in accuracy dell'instance selection rispetto al caso standard.

Time Gain	Accuracy Gain
70%	-8%

Nel caso della KNN

Instance Selection	Istanze	Tempo Test	Accuracy
SI	171	0.0101	92%
NO	340	0.0103	92%

Tabella 4.9: KNN Dataset 1



[a] [b]

Figura 4.3: (a) Tempo fit: dataset modificato vs dataset completo (b) Accuracy: dataset modificato vs dataset completo

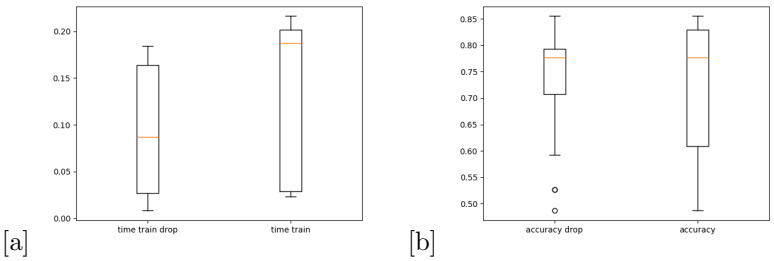
Time Gain	Accuracy Gain
1.5%	0%

4.3.2 Dataset 2

MLP

Instance Selection	Istanze	Tempo Fit	Accuracy
SI	93	0.1011	73%
NO	181	0.1237	71%

Tabella 4.10: MLP Dataset 2



[a] [b]

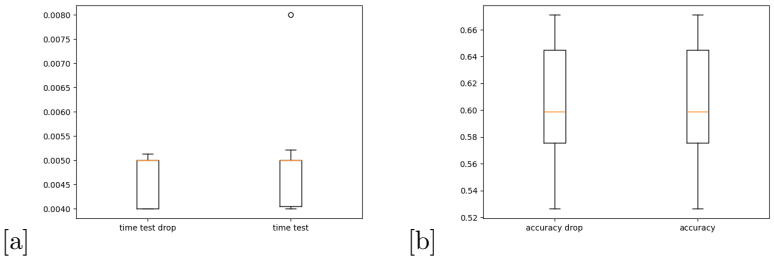
Figura 4.4: (a) Tempo Test: dataset modificato vs dataset completo (b) Accuracy: dataset modificato vs dataset completo

Time Gain	Accuracy Gain
18%	+2%

Nel caso della KNN

Instance Selection	Istanze	Tempo Test	Accuracy
SI	90	0.0046	60%
NO	181	0.0048	60%

Tabella 4.11: KNN Dataset 2



[a] [b]

Figura 4.5: (a) Tempo fit: dataset modificato vs dataset completo (b) Accuracy: dataset modificato vs dataset completo

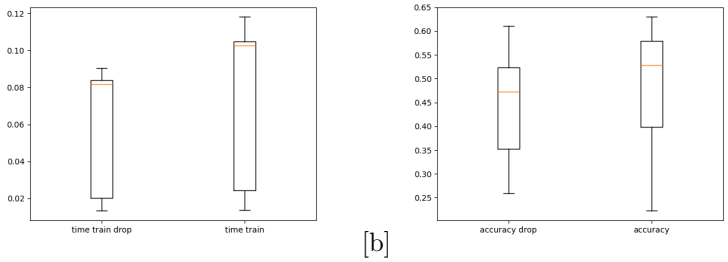
Time Gain	Accuracy Gain
4%	0%

4.3.3 Dataset 3

MLP

Instance Selection	Istanze	Tempo Fit	Accuracy
SI	61	0.0607	45%
NO	128	0.0793	49%

Tabella 4.12: MLP Dataset 3



[a] [b]

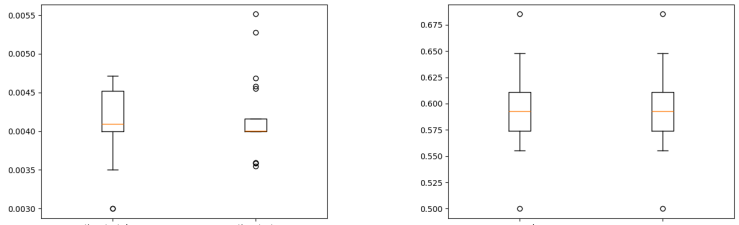
Figura 4.6: (a) Tempo Test: dataset modificato vs dataset completo (b) Accuracy: dataset modificato vs dataset completo

Time Gain	Accuracy Gain
24%	-4%

Nel caso della KNN

Instance Selection	Istanze	Tempo Test	Accuracy
SI	63	0.0040	60%
NO	128	0.0042	60%

Tabella 4.13: KNN Dataset 3



[a] [b]

Figura 4.7: (a) Tempo fit: dataset modificato vs dataset completo (b) Accuracy: dataset modificato vs dataset completo

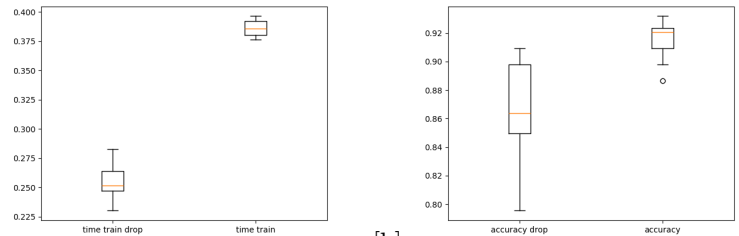
Time Gain	Accuracy Gain
5%	0%

4.3.4 Dataset 4

MLP

Instance Selection	Istanze	Tempo Fit	Accuracy
SI	105	0.2543	86%
NO	210	0.3857	91%

Tabella 4.14: MLP Dataset 4



[a] [b]

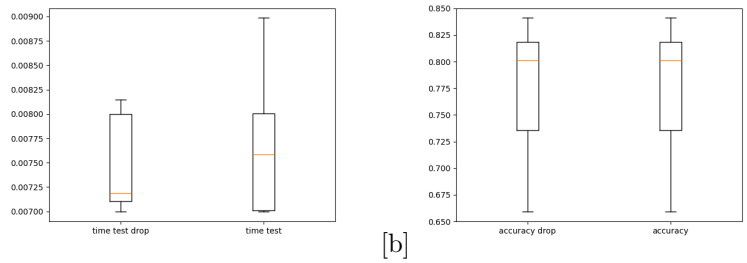
Figura 4.8: (a) Tempo Test: dataset modificato vs dataset completo (b) Accuracy: dataset modificato vs dataset completo

Time Gain	Accuracy Gain
34%	-5%

Nel caso della KNN

Instance Selection	Istanze	Tempo Test	Accuracy
SI	63	0.0075	77%
NO	128	0.0076	77%

Tabella 4.15: KNN Dataset 4



[a] [b]

Figura 4.9: (a) Tempo fit: dataset modificato vs dataset completo (b) Accuracy: dataset modificato vs dataset completo

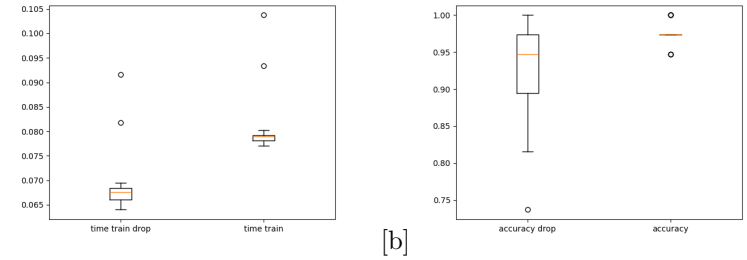
Time Gain	Accuracy Gain
1%	0%

4.3.5 Dataset 5

MLP

Instance Selection	Istanze	Tempo Fit	Accuracy
SI	45	0.0690	93%
NO	89	0.0806	97%

Tabella 4.16: MLP Dataset 5



[a] [b]

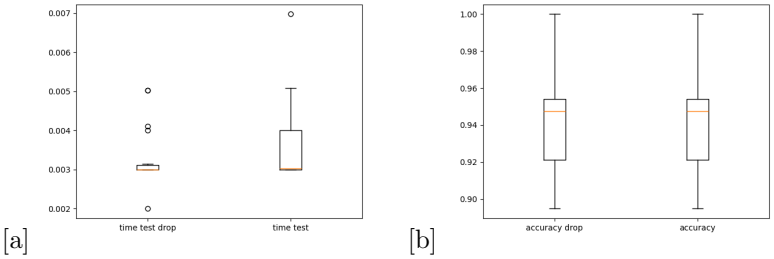
Figura 4.10: (a) Tempo Test: dataset modificato vs dataset completo (b) Accuracy: dataset modificato vs dataset completo

Time Gain	Accuracy Gain
24%	-4%

Nel caso della KNN

Instance Selection	Istanze	Tempo Test	Accuracy
SI	44	0.0032	94%
NO	89	0.0035	94%

Tabella 4.17: KNN Dataset 5



[a] [b]

Figura 4.11: (a) Tempo fit: dataset modificato vs dataset completo (b) Accuracy: dataset modificato vs dataset completo

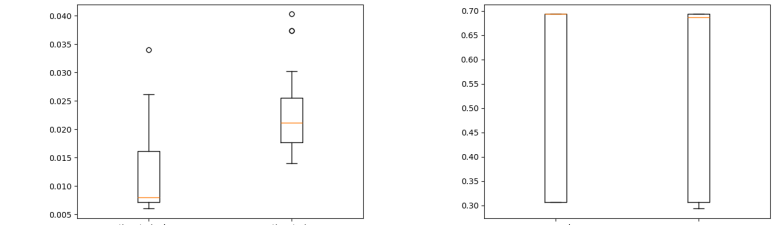
Time Gain	Accuracy Gain
9%	0%

4.3.6 Dataset 6

MLP

Instance Selection	Istanze	Tempo Fit	Accuracy
SI	91	0.0118	54%
NO	179	0.0231	55%

Tabella 4.18: MLP Dataset 6



[a] [b]

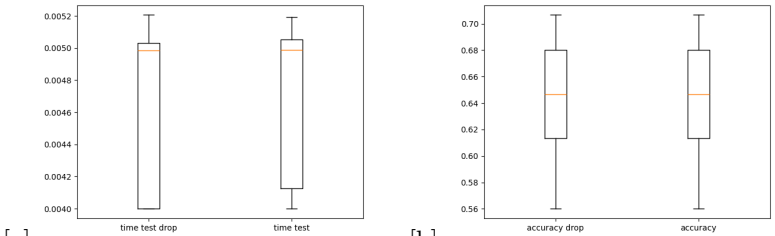
Figura 4.12: (a) Tempo Test: dataset modificato vs dataset completo (b) Accuracy: dataset modificato vs dataset completo

Time Gain	Accuracy Gain
49%	-1%

Nel caso della KNN

Instance Selection	Istanze	Tempo Test	Accuracy
SI	91	0.0046	64%
NO	179	0.0047	64%

Tabella 4.19: KNN Dataset 6



[a] [b]

Figura 4.13: (a) Tempo fit: dataset modificato vs dataset completo (b) Accuracy: dataset modificato vs dataset completo

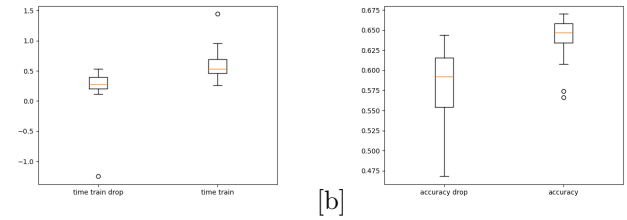
Time Gain	Accuracy Gain
2%	0%

4.3.7 Dataset 7

MLP

Instance Selection	Istanze	Tempo Fit	Accuracy
SI	606	0.2267	58%
NO	1200	0.6112	64%

Tabella 4.20: MLP Dataset 7



[a] [b]

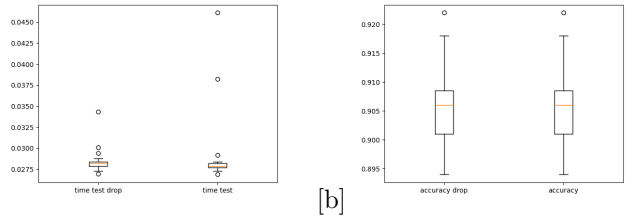
Figura 4.14: (a) Tempo Test: dataset modificato vs dataset completo (b) Accuracy: dataset modificato vs dataset completo

Time Gain	Accuracy Gain
63%	-6%

Nel caso della KNN

Instance Selection	Istanze	Tempo Test	Accuracy
SI	595	0.0284	90%
NO	1200	0.0293	90%

Tabella 4.21: KNN Dataset 7



[a] [b]

Figura 4.15: (a) Tempo fit: dataset modificato vs dataset completo (b) Accuracy: dataset modificato vs dataset completo

Time Gain	Accuracy Gain
5%	0%

Conclusioni

Si è potuto osservare come l'instance selection abbia confermato la sua validità nella diminuzione dei tempi di fit per la MLP e di test della KNN, come si ipotizzava dalla riduzione delle dimensioni del dataset utilizzato.

Dataset		Time Gain	Accuracy Gain
Breast Cancer	MLP	70%	-8%
	KNN	1.5%	0%
Hearth Attack	MLP	18%	+2%
	KNN	4%	0%
Glass	MLP	24%	-4%
	KNN	5%	0%
Ionosphere	MLP	34%	-5%
	KNN	1%	0%
Iris	MLP	24%	-4%
	KNN	9%	0%
Hearth Failure	MLP	49%	-1%
	KNN	2%	0%
Cellular Price	MLP	63%	-6%
	KNN	5%	0%

Tabella 4.22: Tabella riassuntiva

In particolare si ha che:

- La relazione tra il tempo e il set di training della MLP risulta più forte di quella tra tempo e set di test della KNN;
- La riduzione del dataset è stata mediamente del 50%, sia per la MLP che per la KNN;
- Il dataset 2 è l'unico a presentare, nel caso della MLP, un gain dell'accuracy. Questo fa supporre che sia l'unico in cui sono presenti istanze effettivamente rumorose.

Possiamo concludere confermando l'Instance Selection come valido mezzo per l'eliminazione di istanze rumorose, qualora presenti, e per la riduzione dei tempi di run

dei modelli. Per la MLP tale riduzione risulta mediamente del 40%, comportando una diminuzione seppur accettabile dell'accuracy. Per la KNN invece la riduzione temporale risulta più mitigata lasciando però l'accuracy invariata. La scelta di tale processo rimane dunque da valutare in base alle necessità di ogni singolo caso.

Ringraziamenti

Ringrazio la mia relatrice la professoressa Autilia Vitiello per la sua disponibilità, professionalità nel darmi consigli fondamentali per la stesura di questa tesi e per la gentilezza dimostrata.
Ringrazio i miei amici di una vita, sempre presenti e pronti a tirarmi su dopo una lunga sessione di studio, un esame non superato e a darmi la carica per affrontarne uno nuovo.
Ringrazio i miei colleghi di sempre: Luca il mio compagno di laboratorio di una vita con il quale ho affrontato le domande più profonde dell'universo "Se vai a comprare una lavatrice .....", Rodolfo, Giovanni, Samuel, Alessandra, Alessandro, Simone, compagni insostituibili d'esami, di studio, di idee folli, di idee miliardarie "non ancora realizzate", con i quali ho condiviso e vissuto le gioie e le avversità di questo mio percorso. Ringrazio Fabio alleato fondamentale nell'assalto alla Fisica Quantistica.
Ringrazio tutta la mia famiglia, che ha sempre creduto in me e che con il loro affetto mi ha sempre sostenuto e incoraggiato. Ringrazio chi vorrei fosse presente ma purtroppo non c'è più, da lassù non hanno mai smesso di starmi vicino. Sono sicuro che anche oggi stiano festeggiando con noi.
Un ringraziamento speciale va ai miei genitori e a mia sorella sempre li pronti a supportarmi e sopportarmi in tutte le mie passioni e ossessioni. Il vostro amore e la vostra fiducia sono la forza che mi guida anche nei momenti più duri. Senza di voi non sarei niente di quello che sono oggi.
A voi è dedicata questa tesi.

Bibliografia

[1] Chen, Z. Y., Tsai, C. F., Eberle, W., Lin, W. C., & Ke, S. W. (2015). Instance selection by genetic-based biological algorithm. Soft Computing, 19, 1269-1282.

[2] Cano, J. R., Herrera, F., & Lozano, M. (2006). On the combination of evolutionary algorithms and stratified strategies for training set selection in data mining. Applied Soft Computing, 6(3), 323-332.

[3] Zhang, Y. (Ed.). (2010). New advances in machine learning. BoD–Books on Demand.

[4] Alpaydin, E. (2020). Introduction to machine learning. MIT press.

[5] Maglogiannis, I. G. (Ed.). (2007). Emerging artificial intelligence applications in computer engineering: real word ai systems with applications in ehealth, hci, information retrieval and pervasive technologies (Vol. 160). Ios Press.

[6] Raschka S., & Mirjalili V. (2017). Python machine learning Second Edition. Packt publishing ltd.

[7] Mitchell, M. (1995, September). Genetic algorithms: An overview. In Complex. (Vol. 1, No. 1, pp. 31-39).

[8] Forrest, S. (1996). Genetic algorithms. ACM computing surveys (CSUR), 28(1), 77-80.

[9] Eiben, A. E., & Smith, J. E. (2015). Introduction to evolutionary computing. Springer-Verlag Berlin Heidelberg.