ALMA MATER STUDIORUM BOLOGNA
Corso di Laurea in Fisica
Scuola di Scienze

ALGORITMI DI MACHINE LEARNING PER CLASSIFICARE IL COMPORTAMENTO DI DROSOPHILE IN PRESENZA DI MUTAZIONI GENETICHE

Relatore: Chia.mo Prof. Armando Bazzani
Correlatore: Dott.ssa Rachele Luzi

Tesi di Laurea di:
Gianmarco Sabbatini

Anno Accademico 2017-2018

Ai miei genitori

SOMMARIO

Nel mio lavoro di tesi ho effettuato uno studio teorico e pratico su alcuni algoritmi di machine learning. In seguito a una prima parte esplicativa e compilativa sulla struttura teorica di tali algoritmi, mi sono occupato della classificazione, tramite algoritmi di machine learning, di drosophile con il patrimonio genetico intatto rispetto a drosophile che hanno subito mutazioni genetiche. I dati che ho elaborato sono stati forniti da un esperimento svolto all'Università di Padova.

CAPITOLO 1

INTRODUZIONE

Il machine learning costituisce una particolare branca dell'informatica, più in particolare dell'intelligenza artificiale. Definirne in maniera semplice le caratteristiche e le applicazioni non è sempre possibile, visto che questo ramo è molto vasto e prevede differenti modalità, tecniche e strumenti per essere realizzato. Inoltre, le differenti tecniche di apprendimento e sviluppo degli algoritmi danno vita ad altrettante possibilità di utilizzo che ne allargano il campo di applicazione rendendo difficile una definizione specifica. Si può tuttavia osservare che con il termine machine learning si identificano differenti meccanismi che permettono a una "macchina intelligente" di migliorare le proprie capacità e prestazioni nel tempo. La macchina, quindi, sarà in grado di imparare a svolgere determinati compiti migliorando, tramite l'esperienza, le proprie capacità, le proprie risposte e funzioni. Alla base dell'apprendimento automatico ci sono una serie di differenti algoritmi che, partendo da nozioni primitive, elaboreranno un risultato.
Anche se oggi parlare di machine learning, di intelligenza artificiale, di computer e macchine intelligenti sembra quasi la normalità, per arrivare ai risultati odierni la strada è stata molto complessa, perché divisa tra sperimentazioni e scetticismo. I primi esperimenti per la realizzazione di macchine intelligenti risalgono agli inizi degli anni Cinquanta del Novecento, quando alcuni matematici e statistici iniziarono a pensare di utilizzare i metodi probabilistici per realizzare macchine che potessero prendere decisioni proprio tenendo conto delle probabilità di accadimento di un evento. Il primo grande nome legato al machine learning è sicuramente quello di Alan Turing, che ipotizzò la necessità di realizzare algoritmi specifici per realizzare macchine in grado di apprendere [8]. In quegli stessi anni, anche gli studi sull'intelligenza artificiale, sui sistemi esperti e sulle reti neurali vedevano momenti di grossa crescita alternati da periodi di abbandono, causati soprattutto

dalle molte difficoltà riscontrate nelle possibilità di realizzazione dei diversi sistemi intelligenti, nella mancanza di sussidi economici e dallo scetticismo che circondava spesso chi provava a lavorarci. A partire dagli anni Ottanta, una serie di interessanti risultati ha portato alla rinascita di questo settore della ricerca: una rinascita che è stata resa possibile da nuovi investimenti nel settore. Alla fine degli anni Novanta il machine learning trova nuova linfa vitale in una serie di innovative tecniche legate ad elementi statistici e probabilistici: si trattava di un importante passo che permise quello sviluppo che ha portato oggi l'apprendimento automatico ad essere un ramo della ricerca riconosciuto e altamente richiesto.
Il mio lavoro sfrutta una specifica classe di algoritmi di machine learning chiamati classificatori. Hanno come obiettivo allenandosi su un dataset, quello di imparare, appunto, a classificare per cercare di attribuire ad un dato mai visto una determinata classe di appartenenza.
Lo scopo della tesi è quello di estrarre una feature che ci permetta di classificare drosophile sane da quelle che hanno subito mutazioni genetiche. Le analisi sono state effettuate su un dataset fornito dall'Università di Padova. Lo studio è rilevante per i fisiologi perchè mette in relazione malattie genetiche umane, come l'Alzheimer, e il comportamento delle drosophile modificate. I dati a disposizione sono di carattere dinamico in quanto sono stati estrapolati da registrazioni che hanno rilevato il movimento di tali insetti.
La tesi è strutturata nel seguente modo:

- nel secondo capitolo si introdurranno le principali caratteristiche del machine learning e si spiegherà in termini teorici il funzionamento di alcuni classificatori;
- nel terzo capitolo si mostrerà l'analisi statistica del dataset;
- nel quarto capitolo verranno esposti e commentati i risultati.

CAPITOLO 2

ALGORITMI DI MACHINE LEARNING

A coniare per primo il termine fu Arthur Lee Samuel, scienziato americano pioniere nel campo dell'Intelligenza Artificiale, nel 1959 anche se, ad oggi, la definizione più accreditata dalla comunità scientifica è quella fornita da un altro americano, Tom Michael Mitchell, direttore del dipartimento Machine Learning della Carnegie Mellon University:

> «si dice che un programma apprende dall'esperienza E con riferimento a alcune classi di compiti T e con misurazione della performance P, se le sue performance nel compito T, come misurato da P, migliorano con l'esperienza E».

Quindi non si tratta di una tecnica ben precisa, bensì di una famiglia di algoritmi che usano i più differenti approcci di analisi dell'input (dalla statistica bayesiana, alla geometria analitica) per riuscire a ricavare informazioni sull'output. Una prima caratteristica che fornisce una distinzione tra i vari algoritmi è data dal tipo di apprendimento effettuato: a seconda della modalità di addestramento si può infatti distinguere l'apprendimento supervisionato da quello non supervisionato. Nel primo caso si addestra il modello dando in input dei dati etichettati, quindi oltre al valore della feature scelta, si avrà in input anche una label con la classe di appartenenza del dato. Al contrario ad algoritmi che sfruttano l'apprendimento non supervisionato si danno datasets senza labels ed è quindi il programma che deve trovare le classi in cui dividere i dati tramite delle features. Esiste un terzo tipo di apprendimento che è quello per rinforzo; è guidato e punta a creare sistemi che si muovono in un ambiente dinamico e mutevole. L'apprendimento avviene tramite un processo di "ricompensa" detta appunto rinforzo tramite il quale l'algoritmo riesce a imparare e capire il task per cui è stato creato. A seconda poi

del diverso tipo di compito che eseguono gli algoritmi prendono vari nomi. I classificatori sono quelli che avendo in input dei dati e le classi a cui appartengono, riescono a sviluppare il modo di capire a che classe appartiene un dato che non avevano mai visto. Un altro tipo di algoritmi sono quelli di clustering, che riescono a suddividere dei dati presi da input senza conoscere a priori le classi di appartenenza. Esistono poi algoritmi di regressione che cercano di prevedere il valore di un dato. Avendone in input altri dello stesso fenomeno, questi cercano di capire il loro andamento prevedendo il valore di un dato sconosciuto per interpolazione. In questo lavoro verrà sempre fatto riferimento ad algoritmi di classificazione, che sono anche quelli usati nella stesura del programma.

2.1 Funzione costo

Introduciamo in questa sezione la *funzione costo* o *loss function*, importante per la trattazione successiva. Dato X spazio di tutti i possibili input e $Y = -1, 1$ lo spazio di tutti i possibili output, vorremmo trovare la funzione $f : X \to \mathbb{R}$ che meglio mappa $\vec{x}$ in y. Per motivi vari dovuti al rumore, a informazioni incomplete o alla natura stocastica dell'evento preso in considerazione, però, alcune $\vec{x}$ potrebbero generare una y errata. Questo ovviamente si ripercuote sul raggiumento o meno dell'obiettivo dell'algoritmo. Ovviamente, quello a cui si punterà è il raggiungimento di un "costo" sempre minore, ovvero la minimizzazione di una qualche funzione della differenza tra valore stimato e valore vero. Formalmente questo si traduce minimizzando il valore di aspettazione del rischio, ovvero:

$$I[f] = \int_{X \times Y} V(f(\vec{x}), y) p(\vec{x}) \, \mathrm{d}\vec{x} \mathrm{d}y. \tag{2.1}$$

dove con $V(f(\vec{x}), y)$ intendiamo appunto la funzione costo, mentre $p(\vec{x})$ è la funzione densità di probabilità del processo che ha generato i dati.
In pratica, però, la distribuzione di probabilità è sconosciuta, quindi si utilizzano n dati indipendenti e identicamente distribuiti che chiameremo:
$S = (\vec{x}_1, y_1), ..., (\vec{x}_n, y_n)$ presi dal dataset. A questo punto la funzione che si va a minimizzare è:

$$I_s[f] = \frac{1}{n} \sum_{i=1}^{n} V(f(\vec{x}), y) \tag{2.2}$$

Per facilità, in ambito computazionale la funzione costo viene scritta come funzione di una sola variabile e diventa:

$$V(f(\vec{x}), y) = \phi(-yf(\vec{x}) \tag{2.3}$$

Nell'ambito della classificazione avremo solo due possibili valori della funzione costo, 0 se la previsione è corretta, 1 altrimenti. Ciò porta ad identificare la funzione

costo con la funzione a gradini di Heaviside. Il problema di questa funzione però è che non è convessa e non è derivabile. Ciò porta ad usare altri tipi di funzione convesse, più facilmente trattabili, che risolvono comunque il problema originalmente posto. Un esempio è la *funzione costo quadratica.* Definita come:

$$V(f(\vec{x}), y) = (1 - yf(\vec{x}))^2 \tag{2.4}$$

questa funzione è convessa e derivabile, anche se converge lentamente soprattutto con campioni complessi.
Vediamo ora più nel dettaglio alcuni tipi di classificatori.

2.2 LDA: linear discriminant analysis

Il *Linear Discriminant analysis* o *LDA* è un metodo di classificazione sviluppato nel 1936 da R. A. Fisher. Matematicamente robusto, spesso produce modelli la cui performance è la stessa di metodi più complessi.
L'LDA viene utilizzato, nella sua formulazione iniziale, su dati che seguono una distribuzione gaussiana e in cui ogni features delle classi ha la stessa varianza. Il suo obiettivo è quello di cercare la combinazione lineare di variabili $Z = \beta_1 x_1 + \beta_2 x_2 + ... + \beta_d x_d$ (predittori), che meglio separa le due o più classi (targets). Per fissare il concetto di separabilità, Fisher definisce la *score function*:

$$S(\beta) = \frac{\beta^T \mu_1 - \beta^T \mu_2}{\beta^T C \beta} \tag{2.5}$$

che equivale a:

$$S(\beta) = \frac{\bar{Z}_1 - \bar{Z}_2}{\sigma_Z^2} \tag{2.6}$$

dove con β indichiamo i coefficienti dell'LDA con C le matrici di covarianza (Appendice A) e con μ_1, μ_2 i vettori medi. Data la score function il problema diventa quello di stimare i β che massimizzano la funzione, risolvendo le seguenti equazioni:

$$\beta = C^{-1}(\mu_1 - \mu_2) \tag{2.7}$$

$$C = \frac{1}{n_1 + n_2}(n_1 C_1 + n_2 C_2) \tag{2.8}$$

Graficamente in figura 2.1 possiamo vedere come i dati sono ben classificabili linearmente per la feature sull'asse x, mentre risultano sovrapposti per la feature sull'asse y.

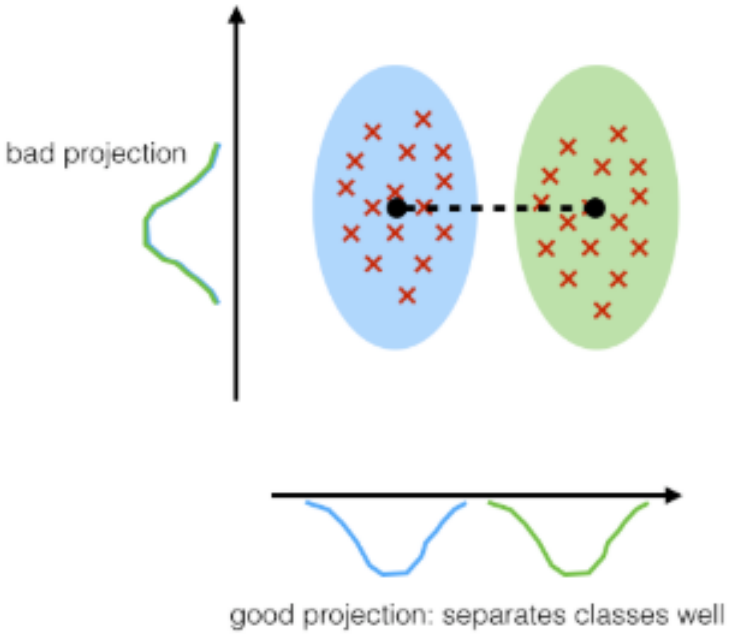

Figura 2.1: Esempio di dati classificabili con LDA (tratta da: [1])

Un modo di calcolare l'efficacia della discriminazione è quello di calcolare la distanza di Mahalanobis tra i due gruppi. Una distanza maggiore di tre significa che i due loro valori medi distano tra loro più di 3 deviazioni standard. Questo indica una piccolissima sovrapposizione di due gruppi quindi anche una bassissima probabilità di errore (*missclassification*). Vediamo graficamente quali dataset sono classificati in modo corretto da un algoritmo di questo tipo.
Se non si prendono in considerazione le assunzioni fatte all'inizio, non basta calcolare la distanza di Mahalanobis tra i due gruppi. Questi infatti potrebbero avere uno scatter molto grande che fa in modo che i gruppi si sovrappongano in modo consistente, rendendo molto alta la probabilità di *missclassification*.
In questo caso quello che si fa è andare a massimizzare la seguente funzione:

$$J(\vec{v}) = \frac{(\mu_1 - \mu_2)^2}{(s_1 + s_2)^2} \tag{2.9}$$

dove s_1, s_2 sono sono gli scatter dei due gruppi definiti come:

$$s_1^2 = \sum_{y_i \in Class1} (y_i - \mu_i)^2 \tag{2.10}$$

Dalla (2.9) possiamo vedere che si ha una buona classificazione, al crescere della distanza tra i valori medi e al diminuire del valore degli scatter come ci si aspetta. La retta che divide le due classi sarà quella dove giace il vettore $\vec{v}$.

2.3 QDA: quadratic discriminant analysis

La *quadratic discriminant analysis* o *QDA* è un algoritmo molto simile all'LDA, con capacità predittive maggiori ma un costo computazionale più alto. La differenza principale con l'LDA è che la superficie che separa le due classi sarà una sezione conica (per esempio in due dimensioni una parabola, un iperole ecc..).
Come possiamo vedere in figura 2.2 la classificazione di quel tipo di dati sarebbe stata molto meno performante se avessimo tentato di utilizzare un LDA, infatti l'iperpiano che divide bene i dati è una parabola.

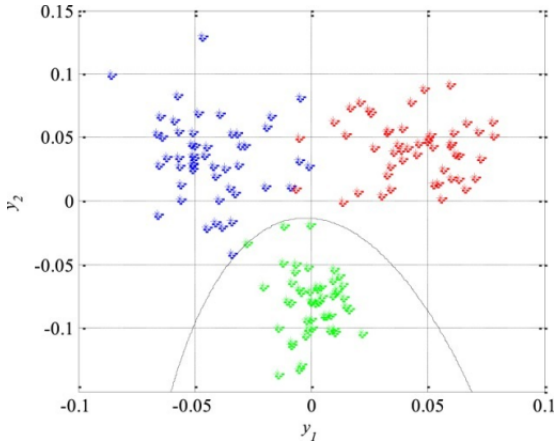

Figura 2.2: Esempio di dati classificabili tramite QDA: si noti che sono separabili tramite coniche (tratta da: [4])

2.4 KNN

Il KNN o K-Nearest Neighbour è un robusto e versatile classificatore che è usato di solito come confronto per algoritmi più complessi. A dispetto della sua semplicità, il KNN può avere performance migliori di molti altri classificatori ed è stato usato per molte applicazioni in economia, genetica, compressione dati ecc..
Il KNN fa parte della famiglia dell'apprendimento supervisionato. Ha due caratteristiche fondamentali:

- non fa assunzioni a priori sulla distribuzione dei dati, evitando errori di *mismodeling*.
- l'algoritmo non impara un modello esplicito. Quello che fa è memorizzare il dataset come "conoscenza" per fare previsioni. Questo si traduce in un elevato tempo di test perchè il KNN deve ricaricare tutto il database per fare

la previsione, il che lo porta ad essere non adatto se si vogliono previsioni in tempi brevi.

Il concetto di funzionamento è molto semplice e si riduce ad un confronto tra il campione da classificare e le K più simili istanze del dataset. Fatto questo il dato da classificare apparterrà alla classe a cui appartengono la maggior parte dei K elementi etichettati più vicini. I più vicini vengono appunto determinati proprio con la distanza tra due dati. La scelta di solito ricade sulla distanza Euclidea:

$$d(x, x') = \sqrt{(x_1 - x'_1)^2 + ... + (x_n - x'_n)^2} \tag{2.11}$$

Quella Euclidea è comunque un esempio di distanza ma in alcuni casi altre distanze come quella di Manhattan, di Chebyshev o di Hamming possono essere più performanti.
Il parametro da fornire all'algoritmo in conclusione è solo il K ovvero il numero dei vicini che "effettuano la votazione" per classificare il dato incognito. Si può intuire presto, come vedremo graficamente poi che un K piccolo, limita la regione di una data previsione e costringe il classificatore a considerare meno la distribuzione generale. Graficamente come possiamo vedere in figura 2.3, il nostro limite decisionale sarà più frastagliato. Ne consegue che la previsione sarà molto condizionata dai singoli dati del dataset e pochissimo dalla loro struttura statistica generale.

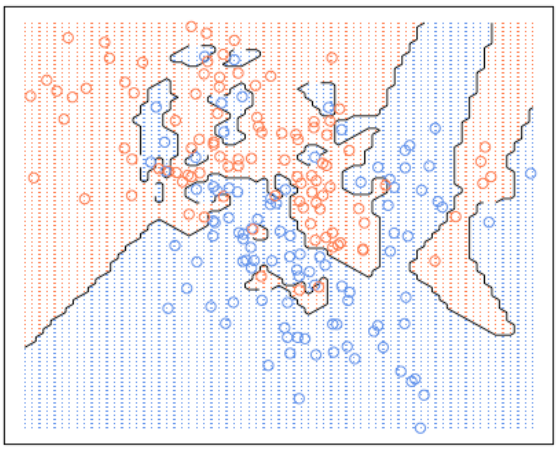

Figura 2.3: Regioni di previsione con K=1 (tratta da: [2])

D'altra parte, un K maggiore implicherà confini più uniformi come visibile in figura 2.4, che praticamente si tradurrà in una previsione più resistente ai valori anomali e più legata alla distribuzione generale.

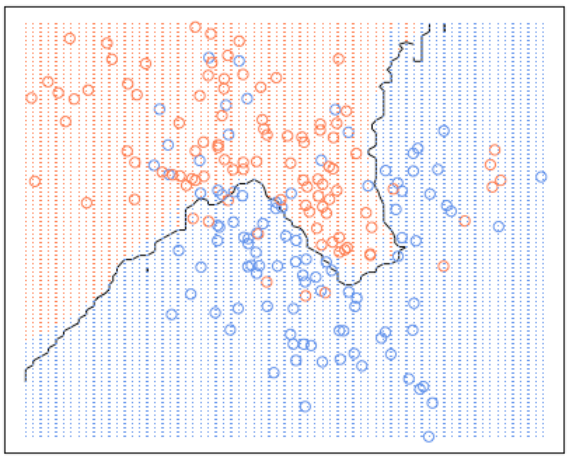

Figura 2.4: Regioni di previsione con K=20 (tratta da: [2])

2.5 SVM: Support Vector Machine

La Support Vector Machine è una famiglia di classificatori molto potenti che permettono di ricondursi alla classificazione lineare anche se si devono risolvere problemi di natura non lineare.

2.5.1 Spazi immagine e kernel

Per cercare di utilizzare una classificazione lineare anche per dati che avrebbero bisogno di funzioni non lineari per essere classificati si fa ricorso all'utilizzo degli *spazi immagine* o *feature space*. Questo metodo consiste nel mappare i dati in uno spazio di dimensione superiore rispetto a quello di input. Dovremo trovare quindi la funzione $\phi : \mathbb{R}^n \to \mathbb{R}^m$ con $m > n$ tale che partendo da dati non separati in $\mathbb{R}^n$ riesca a separare linermente i dati nello spazio $\mathbb{R}^m$ come in figura 2.5.

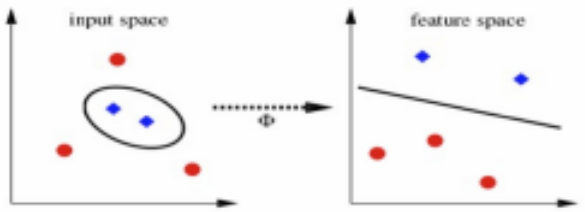

Figura 2.5: Esempio di feature space, si noti che nello spazio di partenza i dati non sono linearmente separati, mentre lo sono nello spazio di arrivo.

La tecnica degli spazi immagine risulta molto interessante per dati di training x_i esprimibili solo attraverso prodotti scalari $x_i \cdot x_j$. In questo caso non è necessario trovare sia $\phi(x_i)$ che $\phi(x_j)$, ma basta calcolare il loro prodotto scalare $\phi(x_i) \cdot \phi(x_j)$, che chiameremo *funzione kernel* e indicheremo con $K(x, y)$.
I kernel più utilizzati sono:

- Lineare: $K(x, y) = x \cdot y$
- Polinomiale: $K(x, y) = (x \cdot y)^d$
- Gaussian Radial Basis function: $K(x, y) = \exp -|x - y|^2/(2\sigma^2)$
- Sigmoide: $K(x, y) = \tanh(kx \cdot y - \delta)$

2.5.2 Formulazione Lagrangiana dell'SVM

In questa sezione daremo un'interpretazione della funzione costo in termini di energia. Per un algoritmo, trovare una soluzione che minimizzi l'energia, equivale ad ottenere la massima performance di classificazione. L'SVM cerca il minimo assoluto della funzione costo muovendosi nell'intorno di un certo punto scelto in modo random. L'ampiezza dell'intorno e il numero di punti che l'SVM controlla dipendono da parametri forniti in input. Concettualmente questo mostra come la Support Vector Machine sia un algoritmo ideale per l'ottimizzazione della classificazione. Per rendere possibile la generalizzazione di quanto detto finora anche nel caso non lineare in cui verranno utilizzate le funzioni kernel, è necessaria una formulazione Lagrangiana del problema, grazie alla quale i dati appariranno solo sotto forma di prodotto scalare. La Lagrangiana ottenuta nel caso di un problema lineare è la seguente:

$$L_p(w, b, \alpha) = \frac{1}{2}||w||^2 - \sum_{i=1}^{N} \alpha_i [y_i(w \cdot x_i + b)] + \sum_{i=1}^{N} \alpha_i \tag{2.12}$$

Per risolvere il problema di ottimizzazione, la prima cosa da fare è minimizzare L_p rispetto a w, b e contemporaneamente fare in modo che le derivate di L_p rispetto a tutti gli α_i si azzerino, il tutto soggetto al vincolo $\alpha_i > 0$ (insieme di vincoli Γ_1). E' possibile anche ricorrere alla sua espressione duale: si può cioè massimizzare L_p, imponendo che il suo gradiente rispetto a w, b sia 0 e che $\alpha_i > 0$ (insieme di vincoli Γ_2. Questa doppia formulazione del problema ha una proprietà fondamentale: il minimo di L_p soggetto al vincolo Γ_1 dà come risultato gli stessi α_i, w, b del massimo di L_p soggetto al vincolo Γ_2. Tutti i punti che soddisfano le varie condizioni vengono chiamati *vettori di supporto* (*support vectors*) e giacciono su un iperpiano separatore.

2.6 Decision Tree Classifiers

I *decision tree classifiers* sono classificatori con apprendimento supervisionato. Si rendono necessari quando per separare due o più classi non basta una semplice linea (o meglio iperpiano) ma ne serve più di una, come si può vedere nell'esempio in figura 2.6, dove l'obiettivo è dividere i cerchi dai quadrati.

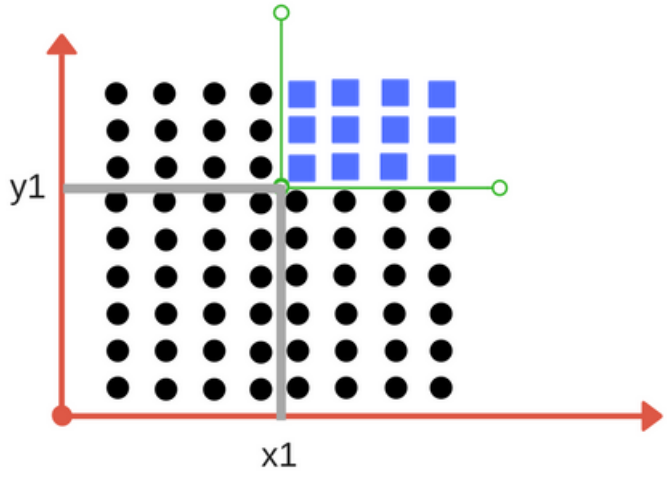

Figura 2.6: Esempio di dataset classificabile con Decision Tree (tratta da: [3])

Quello che fa un Decision Tree Classifier quindi è dividere ripetutamente l'area su cui si sta lavorando in sottospazi. il suo compito termina o quando la divisione si dice pura e quindi ogni area contiene elementi di una sola classe, o quando si raggiunge il livello di tolleranza di impurità impostato prima dell'esecuzione dell'algoritmo. In gergo dai dati di training per costruire il decisional tree si parte dalla radice dove si mette una prima condizione molto discriminante e via via si seguono i nodi che man mano rappresentano condizioni meno stringenti per la classificazione. Il punto di arrivo è chiamato foglia e rappresenta la classe di appartenenza del dato da classificare.

2.6.1 Information Gain

Una questione rilevante sul DTC riguarda la *quantità di informazione portata*. Questa è la capacità che ogni feature ha di separare un certo numero di classi. Se vogliamo classificare ad esempio rombi e triangoli, una feature che porta un'alta quantità di informazione sarà il numero di lati mentre una che ne porta poca può

essere il colore dei margini. Nel DTC le feature con la maggiore qantità di informazione devono essere controllate dai nodi più vicini alle radici e via via i nodi più lontani lavoreranno su feature che qualificano meno il campione. Il problema nel tempo è stato appunto stimare questa quantità. All'inizio si è provato a sistemare random le features sui vari nodi, con risultati pessimi in termini di performance.
I ricercatori hanno così costruito una grandezza chiamata *information gain* che quantifica proprio la quantità di informazione portata da ogni feature. Questa grandezza matematicamente è la *divergenza di Kullback-Leibler*.
Sia T il set di dati di training, ognuno dei quali nella solita forma $(x, y) = (x_1, x_2, \ldots, x_k, y)$. La information gain per un attributo a è definita in termini di entropia come:

$$IG(T, a) = H(T) - \sum_{v \in vals(a)} \frac{|\{x \in T | x_a = v\}|}{|T|} \cdot H(\{x \in T | x_a = v\}) \tag{2.13}$$

dove:

$$H(X) = -\sum_{x \in \mathbb{X}} p(x) \log p(x) \tag{2.14}$$

In pratica l'information gain quantifica la diminuzione di entropia dopo aver separato secondo la feature a. Si riesce così a determinare la posizione della feature sul livello opportuno.

2.7 Percettrone lineare

Il *percettrone* è un tipo di classificatore lineare molto semplice. Ha una struttura ripresa dal neurone biologico. Ha quindi una parte assimilata ai dentriti che riceve molti input i quali vengono in qualche modo sommati dando un valore che chiameremo x. A questo punto si serve di una una $f(x)$ per mappare i suoi ingressi in un valore di output calcolato come: $f(x) = \chi(< w, x > +b)$ dove w è il vettore dei pesi e b è il bias, cioè un termine costante che non dipende da alcun valore di input. $\chi(y)$ è la funzione di output e le scelte più comuni per questa sono:

- $\chi(y) = sign(y)$
- $\chi(y) = y\Theta(y)$ con $\Theta(y)$ funzione di Heaviside
- $\chi(y) = y$

Il bias può essere pensato come un settaggio della funzione di attivazione, una soglia da superare per accendere il percettrone che a questo punto come output darà un risultato binario acceso o spento (analogia caso biologico scarica o non scarica). Vediamo graficamente un esempio di percettrone.

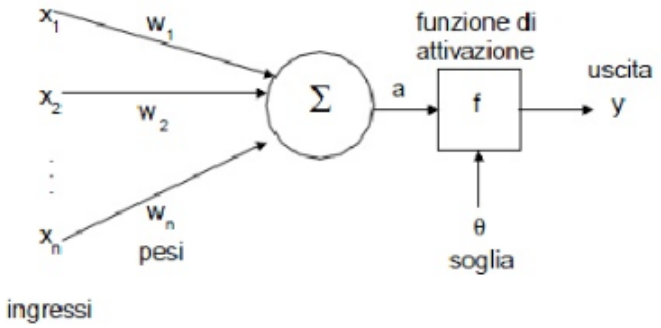

Figura 2.7: Schematizzazione di un percettrone (tratta da: [6])

2.8 MLP

Il *Multilayer Perceptron* o *MLP* è una classe di Reti Neurali Artificiali, con apprendimento supervisionato. Consiste in almeno tre strati di nodi. Ognuno di questi è un neurone che usa una funzione di attivazione non lineare. Questo fatto lo distingue dal Percetrone lineare che riconosce solo classi linearmente separate.

2.8.1 Funzione di attivazione

Se un MLP avesse una funzione di attivazione lineare in tutti i suoi neuroni, si dimostra che ogni rete potrebbe essere ridotta a un sistema a due strati input-output. Nell'MLP qualche neurone usa una funzione di attivazione non lineare, sviluppata sul modello del potenziale di azione biologico. Le due più comuni funzioni di attivazione sono entrambe delle sigmoidi e sono descritte come:

$$y(v_i) = \tanh v_i \tag{2.15}$$

$$y(v_i) = (1 + \exp -v_i)^{-1} \tag{2.16}$$

In questo caso $y(v_i)$ è l'output dell'i-esimo nodo o neurone e v_i è la somma pesata delle connessioni in input.

2.8.2 L'apprendimento

Abbiamo già detto che un MLP si compone di almeno tre strati, uno di input, uno di output e almeno un altro chiamato *hidden layer*. Ogni nodo di ogni strato è collegato al successivo strato. Ogni connessione ha un certo peso che chiameremo w_{ij}. L'apprendimento della rete consiste nel cambiare i pesi che le connessioni hanno, in base agli errori che vengono prodotti in output rispetto all'aspettativa.

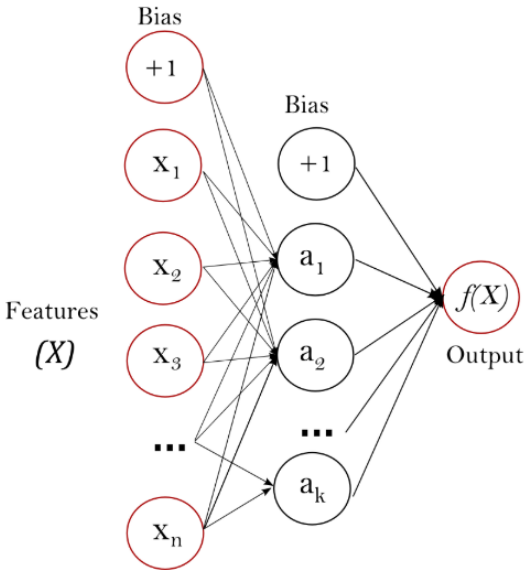

Figura 2.8: Esempio di un MLP (tratta da: [5])

Rappresentiamo l'errore nel nodo di output j dell'n-esimo dato con $e_j(n) = y_j(n) - d_j(n)$ dove y è il valore che ci aspettiamo e d il valore prodotto dal percetrone. A questo punto i valori dei pesi dei nodi vengono aggiustati in modo da minimizzare l'errore dell'intero output:

$$\epsilon(n) = \frac{1}{2}\sum_j e_j^2(n) \tag{2.17}$$

Usando la discesa del gradiente (Appendice B), il nuovo valore dei pesi risulta:

$$\Delta w_{ji}(n) = -\eta \frac{\partial \epsilon(n)}{\partial v_j(n)} d_i(n) \tag{2.18}$$

dove d_i è l'output del neurone precedente e η è il *learning rate*, che viene selezionato per garantire che i pesi convergano rapidamente ad un valore senza oscillazioni. L'espressione della derivata da calcolare risulta abbastanza semplice per un nodo di output e in particolare diventa:

$$-\frac{\partial \epsilon(n)}{\partial v_j(n)} = e_j(n)\phi'(v_j(n)) \tag{2.19}$$

dove ϕ' è la derivata della funzione di attivazione. Per un nodo di un hidden layer l'analisi diventa più complessa e può essere scritta come:

$$-\frac{\partial \epsilon(n)}{\partial v_j(n)} = \phi'(v_j(n)) \sum_k -\frac{\partial \epsilon(n)}{\partial v_k(n)} w_k j(n) \tag{2.20}$$

Gli MLP sono stati una classe di algoritmi molto usati negli anni '80 e hanno risolto problemi molto vari tra cui il riconoscimento immagini, riconoscimento del linguaggio ecc.. Ultimamente sono stati soppiantati dalle SVM. Gli MLP infatti hanno dei difetti:

- gli hidden node hanno una funzione costo non convessa, il che significa che presenta molti minimi locali. Dei pesi random potrebbero far classificare l'algoritmo in minimi diversi creando problemi nella scelta dei pesi stessi;
- l'MLP richiede un numero di iperparametri elevato come il numero di hidden layer, di neuroni e di iterazioni;
- l'MLP è sensibile al riscalamento delle variabili.

CAPITOLO 3

DATASET: PRESENTAZIONE E ANALISI

Il dataset a disposizione per questa tesi è stato fornito dall'Università di Padova. Quello che si va a studiare è il comportamento di un gruppo di Drosophile Melanogaster, i moscerini della frutta, all'interno di uno spazio chiuso, che chiameremo *arena*. Il moscerino della frutta è molto utilizzato in ambito di ricerca per via della completa e precisa mappatura del suo DNA, la veloce riproduzione e la presenza di soli 4 cromosomi.
L'esperimento mette a disposizione 4 tipi di genotipi da studiare:

- sani: drosophile con genoma non modificato;
- ELAV GAL4: modifica della proteina ELAV, centrale in moltissimi processi che riguardano il sistema nervoso centrale;
- SYX WT: modifica al DNA che fa in modo che questo tipo di drosophile sovraesprima la proteina sana syntaxina;
- SYX D253R: modifica che sostituisce un amminoacido carico in posizione 253 con uno di carica opposta. Il risultato è che la proteina non riesce più a legare gli stessi bersagli.

Si sono così preparate delle arene omogenee, ovvero contenenti tutte lo stesso tipo di drosophila. In ogni arena sono stati posti 10 moscerini e si è così proceduto alla presa dati.
I dati estrapolati sono stati i seguenti, formattati ognuno in una colonna:

- **id**: numero intero che identifica il moscerino all'interno dell'arena;
- **x**: numero che identifica la posizione del moscerino ad un certo istante, lungo l'asse delle ascisse (espresso in pixel);

- **y**: numero che identifica la posizione del moscerino ad un certo istante, lungo l'asse delle ordinate (espresso in pixel);
- **a**: immaginando il moscerino come un ellisse, a sarà il suo asse maggiore, ovvero la lunghezza (espressa in pixel);
- **b**: immaginando il moscerino come un ellisse, b sarà il suo asse minore, ovvero la larghezza (espressa in pixel);
- Θ: angolo che il moscerino istante per istante forma con un asse preso come riferimento (espresso in gradi).

Per ogni moscerino sono stati rilevati 9328 samples per ogni feature, uno ogni "time stamp" ovvero l'intervallo di campionamento del sistema utilizzato ottenendo un dataset come quello in figura 3.7. Grazie a questi dati si possono mappare le traiettorie di questi moscerini per avere idea del loro movimento.

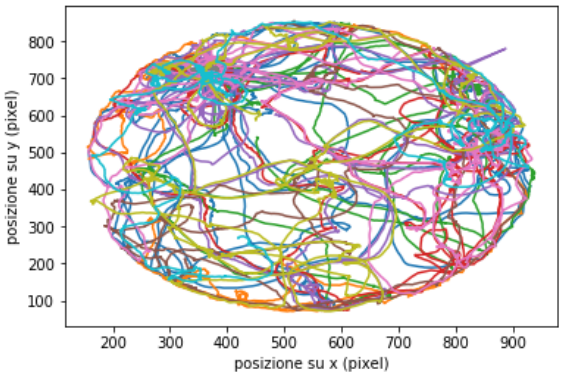

Figura 3.1: Insieme delle traiettorie nell'arena 0 sani

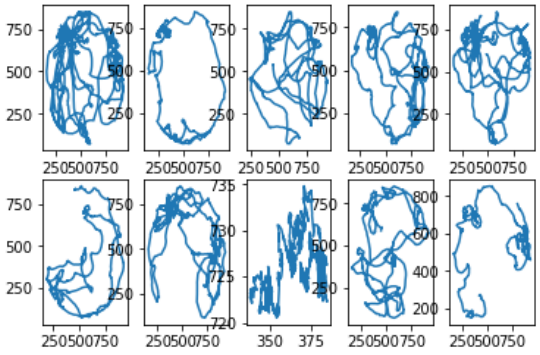

Figura 3.2: Tutte le traiettorie separate dei 10 moscerini dell'arena 0 sani

Le drosophile in un lavoro precedentemente svolto sono state classificate in base alla velocità e in base al numero di interazioni che i moscerini hanno tra di loro. Quello che si cercherà di fare in questo lavoro, invece, sarà classificarle in base all'angolo Θ formato dai moscerini con un asse preso come riferimento.
Prima di processarli con gli algoritmi di machine learning, i dati vanno studiati in maniera statistica per capire se la feature secondo cui stiamo classificando è buona, rende quindi in qualche modo differenti le due classi.
Θ essendo l'angolo tra la direzione del moscerino e un asse preso come riferimento, non risulta di per sè la feature secondo cui classificare. Questo, infatti, indica solo la direzione in cui la drosophila sta volando e risulta ininfluente per la classificazione. Quello che si fa allora è controllare il *cambio di direzione* della drosophila che da adesso chiameremo $\Delta\Theta$. Si definisce, dapprima $\Delta\Theta = \Theta_t - \Theta_{t-1}$. La grandezza, quindi, ci da informazione sulla variazione di direzione tra due time stamps successivi. Avendo però la presa dati un rate di campionamento molto alto si preferisce definire:

$$\Delta\Theta = \Theta_t - \Theta_{t-3} \tag{3.1}$$

Si plotta in un istogramma come nelle figure 3.3 e 3.4 la variazione di direzione per trarre alcune prime considerazioni. I moscerini tendono, durante il tempo di campionamento, a procedere in direzione "rettilinea", come si vede dal grande numero di occorrenze nei bins vicini allo 0, indipendentemente dalla classe di appartenenza. Si nota però una differenza abbastanza importante controllando ad

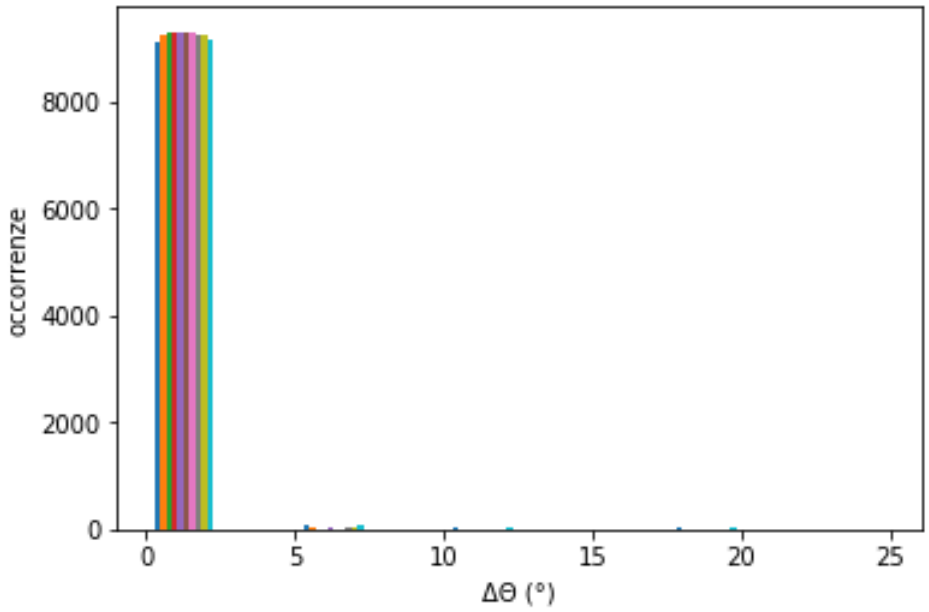

Figura 3.3: Istogramma che rappresenta la distribuzione dei $\Delta\Theta$ per drosophile sane

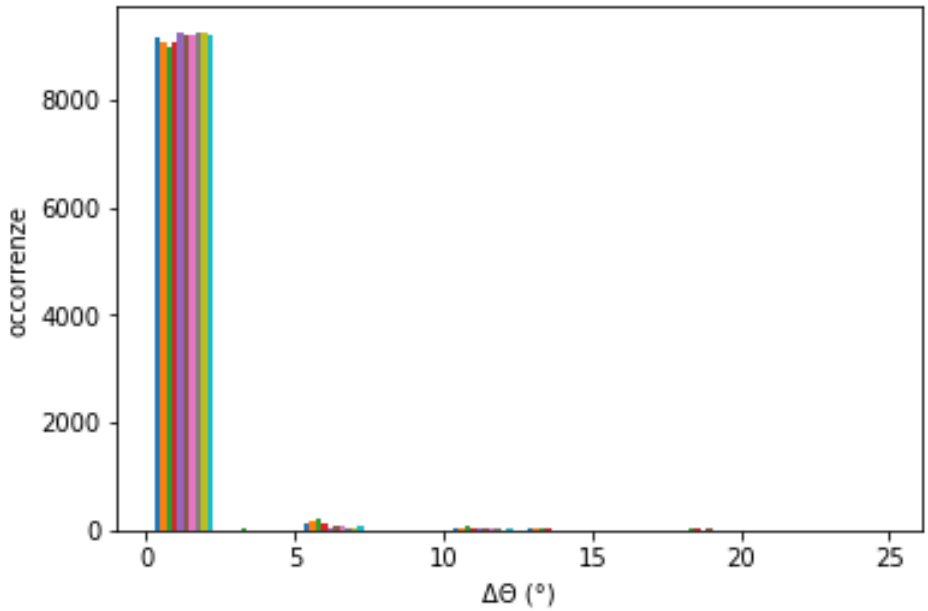

Figura 3.4: Istogramma che rappresenta la distribuzione dei $\Delta\Theta$ per drosophile geneticamente modificate

a $\Delta\Theta > 5$. I moscerini geneticamente modificati presentano molte più occorrenze in quella zona, come si può notare meglio dalle figure 3.5 e 3.6 che sono zoom nella zona $\Delta\Theta > 5$.

Ne concludiamo che le drosophile geneticamente modificate cambiano direzione più spesso rispetto a quelle sane e la feature $\Delta\Theta$ potrebbe essere un buon discriminante. Anche se non divide il dataset, infatti, la feature scelta determina due distribuzioni di proprietà diverse che permetteranno a qualche classificatore di riconoscere le due classi.

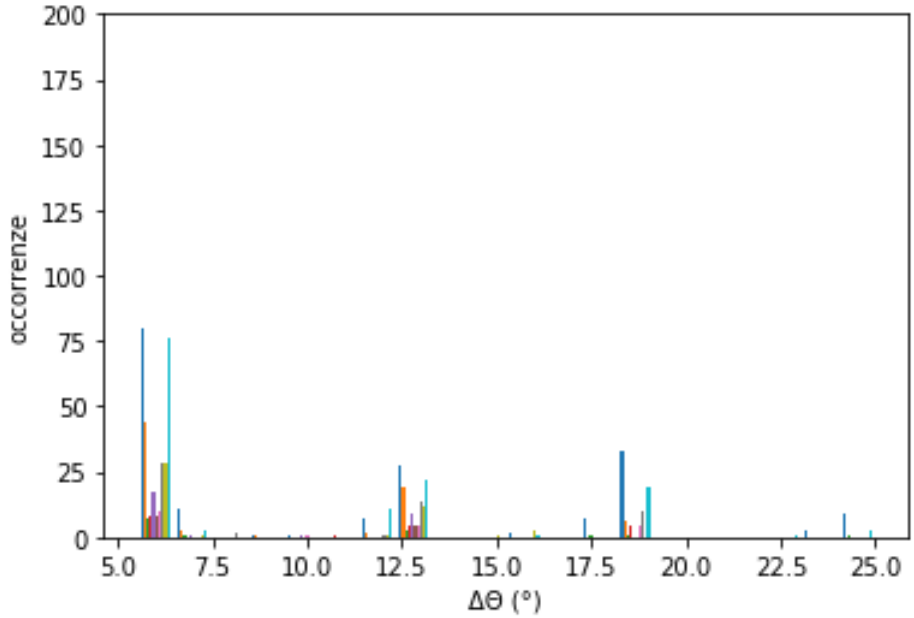

Figura 3.5: Istogramma che rappresenta la distribuzione dei $\Delta\Theta$ per drosophile sane, con $\Delta\Theta>5$

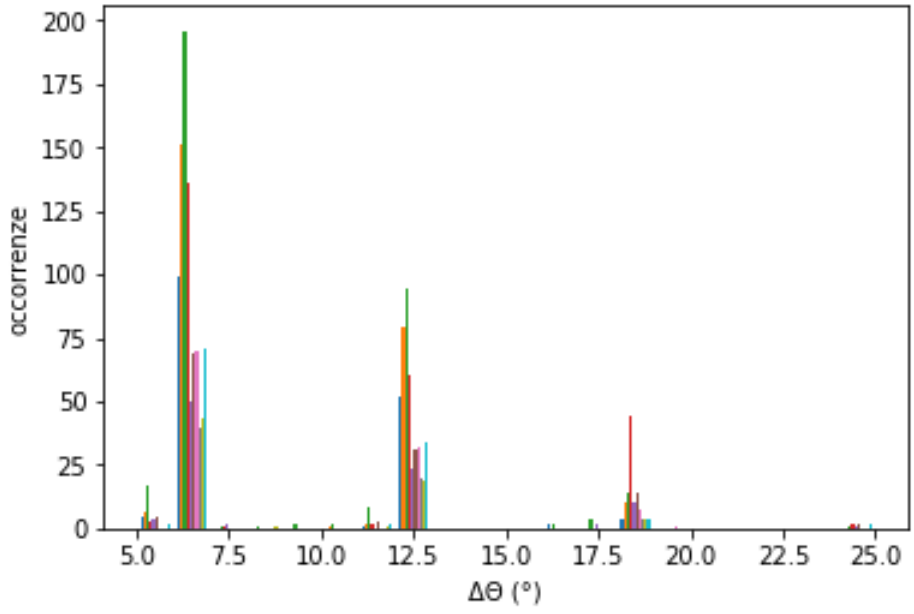

Figura 3.6: Istogramma che rappresenta la distribuzione dei $\Delta\Theta$ per drosophile geneticamente modificate, con $\Delta\Theta>5$

id	x	y	theta	a	b
0	511.561969	225.983299	0.210359	4.163292	1.593798
0	512.382658	227.631992	0.381381	4.180145	1.566307
0	513.053144	229.326412	0.528945	4.199513	1.553194
0	513.894121	231.142253	0.651953	4.264896	1.556491
0	514.270592	232.084324	0.772642	4.228124	1.542069
0	514.835963	232.975977	0.936313	4.189878	1.501749
0	515.113477	235.167655	1.006187	4.223483	1.565521
0	515.100616	237.433774	1.118042	4.273828	1.571400
0	515.474187	238.636596	1.196827	4.313267	1.535135
0	515.037866	241.010540	1.220337	4.314376	1.557852
0	514.565651	242.902916	1.300014	4.309741	1.569408

Figura 3.7: Parte del dataset utilizzato, si notino le precisioni dei dati

CAPITOLO 4

RISULTATI E CONCLUSIONI

Dopo lo studio statistico dei dati si è proceduto alla vera e propria fase di classificazione. I vari algoritmi sono già implementati in python, nella libreria scikit-learn. Si è quindi preparato il dataset nel seguente modo: un vettore di label e una matrice contenete tutti i dati, sia di drosophile sane che malate. A questo punto una funzione divide random la parte di dati da usare nel training (circa l'80%) e la parte di dati da usare nel test (il restante 20%). E' importante che nella fase di train si facciano allenare gli algoritmi su entrambe le classi di drosophile, sia quelle sane che quelle geneticamente modificate, altrimenti lo score dei classificatori rimarrà intorno al 50% ovvero una classificazione casuale.
Quello che ci viene fornito per ogni classificatore alla fine di ogni run sono due parametri:

- **accuratezza del classificatore sul train set**: dopo averlo allenato sul train set si prova il modello creato sul train set stesso e questo parametro ci fornisce la percentuale di elementi classificati in maniera corretta;

- **accuratezza del classificatore sul test set**: dopo averlo allenato sul train set si prova il modello sul test set e questo parametro ci fornisce la percentuale di elementi classificati con successo.

Ci aspettiamo che gli algoritmi lineari dopo lo studio statistico precedente non classifichino in modo troppo buono, visto che la feature scelta non separe nettamente le due classi. Vediamo ora nella tabella le performance di tutti gli algoritmi utilizzati usando come feature il $\Delta\Theta$.

ALGORITMO	ACCURACY TRAIN SET	ACCURACY TEST SET
LDA	0.59	0.56
QDA	0.57	0.55
KNN(3)	0.83	0.75
GaussianNB	0.56	0.54
SVC	0.54	0.52
DECISION TREE	1.00	0.87
RANDOM FOREST	0.99	0.90
AdaBoost	0.91	0.91
MLP	0.84	0.84

I parametri che qualificano un algoritmo sono due e sono chiamati AUC e MCC. Il primo non è altro che l'accuratezza del classificatore sul train set, mentre per l'MCC si faccia riferimento all'Appendice C. Nelle figure 4.1 e 4.2 si mostrano i coefficienti graficati per confrontare i vari algoritmi.

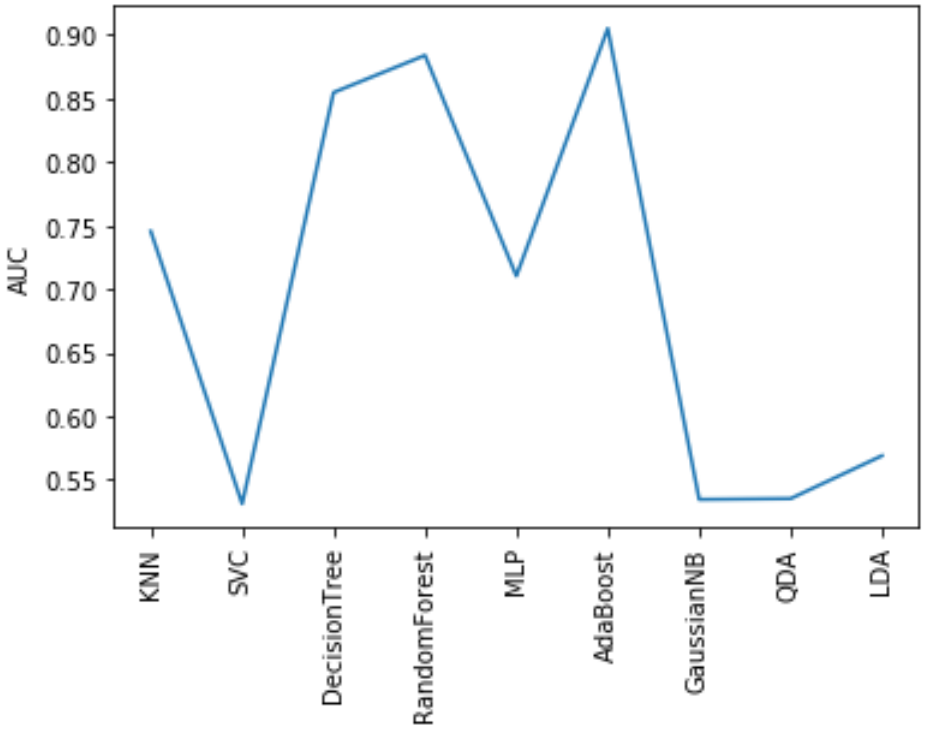

Figura 4.1: Grafico che rappresenta gli AUC dei classificatori utilizzati

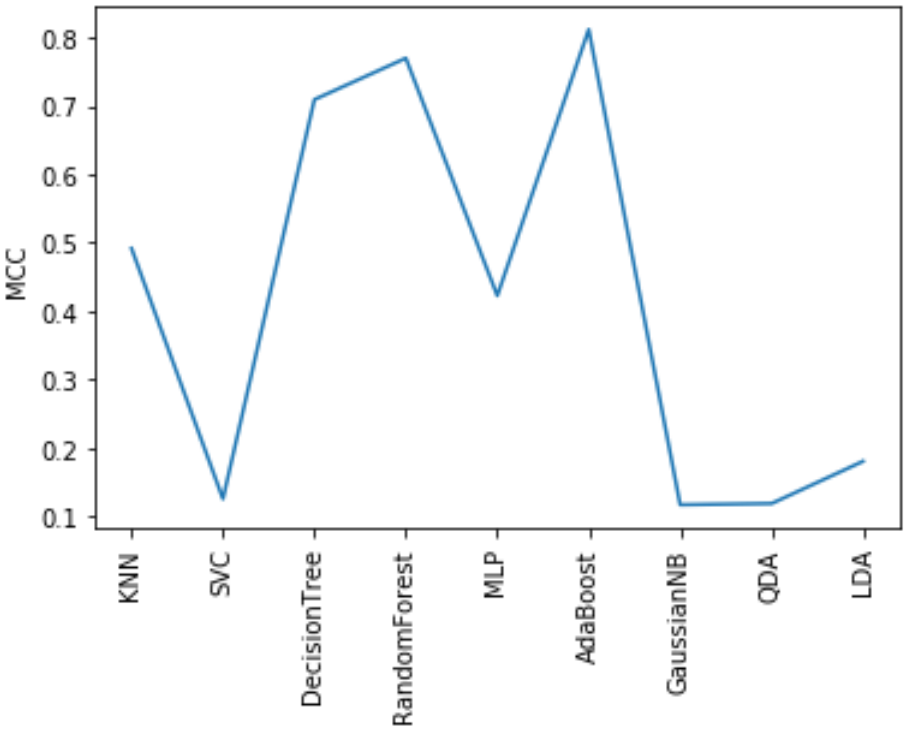

Figura 4.2: Grafico che rappresenta gli MCC dei classificatori utilizzati

Dal grafico dell'MCC, si può subito notare come gli algoritmi lineari o che si riconducono a questi falliscano con performance di classificazione molto basse (es. LDA e SVC). Anche classificatori che presuppongono dati distribuiti in modo gaussiano non ci danno classificazioni molto buone (es. GaussianNB).

Per il KNN in tabella sono riportati i valori di accuratezza per un K=3, ma vediamo in figura 4.3 come variano le performances dell'algoritmo sul nostro dataset in base al K scelto. Si nota bene che i KNN con performances migliori sono quelli con K più alto in accordo con la trattazione teorica. La feature scelta, infatti, non separa in modo netto il dataset, ma crea due distribuzioni di probabilità diverse. Il K più grande ci permette così, di avere un algoritmo più sensibile alla distribuzione di probabilità che alla separazione degli elementi.

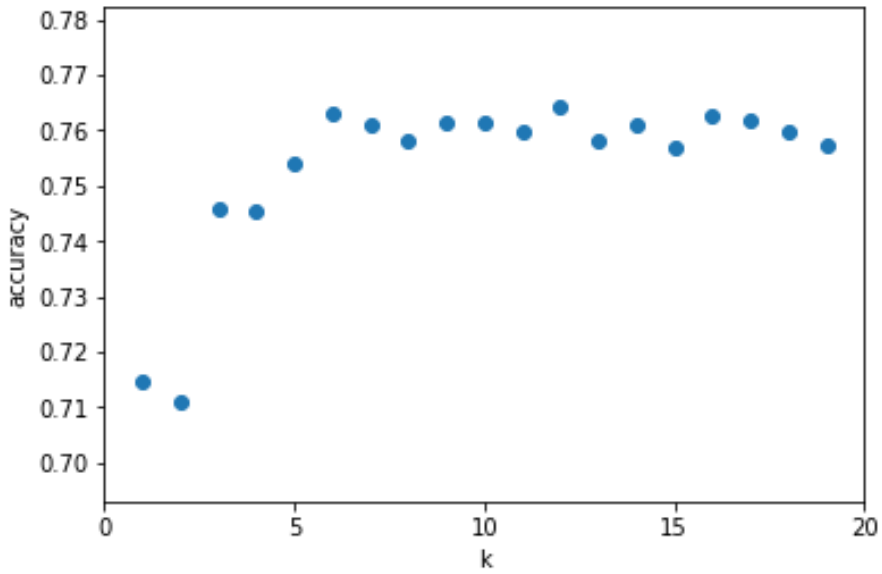

Figura 4.3: Grafico che rappresenta le performance del KNN infunzione del K scelto

APPENDICE A

MATRICI DI COVARIANZA

In statistica multivariata e in probabilità, la matrice delle covarianze (o matrice di varianza e covarianza) si indica di solito con Σ ed è una generalizzazione della covarianza al caso di dimensione maggiore di due. Essa è una matrice che rappresenta la variazione di ogni variabile rispetto alle altre (inclusa se stessa). È una matrice simmetrica.
Sia data una popolazione di n elementi su cui sono rilevati k caratteri quantitativi X_i. Cioè ogni X_i con $i = 1, \ldots, k$ è un vettore di n elementi, indicati con x_{hi} con $h = 1, \ldots, n$. L'elemento x_{hi} rappresenta quindi la modalità dell' h-esima unità statistica rispetto al carattere X_i. La matrice delle covarianze ha dimensione $k \times k$ e ogni elemento è definito come

$$\sigma_{ij} = \frac{1}{n} \sum_{h=1}^{n} (x_{hi} - \mu_i)(x_{hj} - \mu_j), \tag{A.1}$$

dove μ_i indica la media del carattere X_i.

APPENDICE B

DISCESA STOCASTICA DEL GRADIENTE

La discesa stocastica del gradiente (in lingua inglese stochastic gradient descent, SGD) è un metodo iterativo per l'ottimizzazione di funzioni differenziabili, approssimazione stocastica del metodo di discesa del gradiente (GD) quando la funzione costo ha la forma di una somma.
In molti problemi di statistica e di apprendimento automatico si ha la necessità di ottimizzare una funzione $Q(w)$ nei parametri w che ha la forma di una somma $Q(w) = \frac{1}{n}\sum_{i=1}^{n} Q_i(w)$, dove ogni addendo Q_i rappresenta il costo calcolato sull' i-esima osservazione in un dataset. Nel contesto della statistica classica, tale problema appare tipicamente in applicazioni del metodo dei minimi quadrati o del metodo della massima verosimiglianza, e gli stimatori ottenuti come soluzione di tali problemi sono chiamati stimatori M. Il metodo di discesa del gradiente esegue iterazioni nella forma

$$w := w - \eta \nabla Q(w) = w - \eta \frac{1}{n} \sum_{i=1}^{n} \nabla Q_i(w) \tag{B.1}$$

dove η è un iperparametro che controlla l'ampiezza di ogni passo, e nel contesto dell'apprendimento automatico è chiamato tasso di apprendimento (learning rate). In molti casi, gli addendi hanno un'espressione sufficientemente semplice, che consente una valutazione rapida del gradiente ad ogni iterazione (ad esempio, in statistica, le funzioni della famiglia esponenziale). Tuttavia, in altri contesti, la valutazione dell'intera somma può essere particolarmente costosa, ad esempio quando il dataset è particolarmente ampio e non esiste un'espressione elementare per il costo, richiedendo la valutazione di tutti i singoli addendi, fatto che si verifica comunemente, ad esempio, nei problemi di apprendimento profondo. La discesa stocastica del gradiente affronta tale problema approssimando il gradiente

dell'intera somma, valutandolo solo in un sottoinsieme casuale degli addendi ad ogni iterazione. La discesa del gradiente classica viene anche chiamata batch GD, in quanto l'intero dataset viene valutato ad ogni iterazione. Quando il gradiente viene invece approssimato stocasticamente, è possibile costruire diversi metodi a seconda del numero di addendi usati ad ogni iterazione. Un possibile approccio è la discesa stocastica del gradiente in linea (on-line), che usa un solo elemento del dataset alla volta, ed è così chiamato perché può essere usato quando il dataset non è prefissato e nuovi dati vengono generati sul momento (on-line, o on-the-fly). Il gradiente di $Q(w)$ è approssimato ad ogni iterazione con il gradiente calcolato su un singolo addendo della funzione costo (corrispondente ad un elemento del dataset):

$$w := w - \eta \nabla Q_i(w) \tag{B.2}$$

e i parametri vengono aggiornati dopo ogni valutazione. Il dataset può essere attraversato diverse volte, fino a quando il metodo converge. Tipicamente, l'ordine degli elementi è randomizzato ad ogni attraversamento del dataset, e il tasso di apprendimento η può essere variato dinamicamente, tipicamente riducendolo con il procedere delle iterazioni (annealing).

APPENDICE C

COEFFICIENTI DI CORRELAZIONE DI MATTHEWS

I coefficienti di correlazione di Matthews sono uno strumento usato in machine learning, per misurare la qualità di una classificazione binaria. Ideati dal biochimico Brian W. Matthews nel 1975, tengono in considerazione i veri e falsi positivi e negativi e generalmente sono importanti nel caso in cui la dimensione delle due classi sia notevolmente differente. L'MCC è essenzialmente un coefficiente che mette in relazione la previsione e i dati osservati. Assume valore compresi tra -1 e 1: un MCC uguale a 1 denota una classificazione perfetta, al contrario un valore di -1 sottolinea una classificazione esattamente contraria ai dati osservati, mentre se assume il valore di 0 indica una classificazione random. E' definito come:

$$|MCC| = \sqrt{\frac{\chi^2}{n}} \tag{C.1}$$

dove n rappresenta il numero totale di osservazioni.
Ha quindi lo stesso obiettivo dell'AUC ma una potenzialità maggiore. L'accuratezza, infatti, non ci da un valore veritiero delle performance di un classificatore nel caso le classi abbiano dimensioni molto diverse. Infatti, se questo classifica tutti i samples del test come appartenenti alla classe più grande avrà un AUC vicino a 1, ma la sua vera performance non è buona. L'MCC supera questo problema, infatti per calcolarlo si usano come precedentemente detto, i veri e falsi positivi e negativi come si vede dalla formula seguente:

$$MCC = \frac{TP \times TN - TF \times FN}{\sqrt{(TP + FP)(TP + FN)(TN + FP)(TN + FN)}} \tag{C.2}$$

dove con F e T si intendono falsi e veri e con N e P negativi e positivi. I coefficienti di Matthews sono stati generalizzati anche per classificazioni non binarie. La generalizzazione è nota come *statistica* R_k.

BIBLIOGRAFIA

[1] Christopher M Bishop. Pattern recognition and machine learning, 2006. , 60(1):78–78, 2012.

[2] Janez Brank, Marko Grobelnik, Natasa Milic-Frayling, and Dunja Mladenic. Interaction of feature selection methods and linear classification models. In Workshop on Text Learning held at ICML*, 2002.*

[3] Alan Julian Izenman. Linear discriminant analysis. In Modern multivariate statistical techniques*, pages 237–280. Springer, 2013.*

[4] Sotiris B Kotsiantis, I Zaharakis, and P Pintelas. Supervised machine learning: A review of classification techniques. Emerging artificial intelligence applications in computer engineering*, 160:3–24, 2007.*

[5] Dunja Mladenić, Janez Brank, Marko Grobelnik, and Natasa Milic-Frayling. Feature selection using linear classifier weights: interaction with classification models. In Proceedings of the 27th annual international ACM SIGIR conference on Research and development in information retrieval*, pages 234–241. ACM, 2004.*

[6] Nasser M Nasrabadi. Pattern recognition and machine learning. Journal of electronic imaging*, 16(4):049901, 2007.*

[7] Carl Edward Rasmussen. Gaussian processes in machine learning. In Advanced lectures on machine learning*, pages 63–71. Springer, 2004.*

[8] Stuart J Russell and Peter Norvig. Artificial intelligence: a modern approach*. Malaysia; Pearson Education Limited,, 2016.*

[9] H Taud and JF Mas. Multilayer perceptron (mlp). In Geomatic Approaches for Modeling Land Change Scenarios*, pages 451–455. Springer, 2018.*

[10] Zhou Yong, Li Youwen, and Xia Shixiong. An improved knn text classification algorithm based on clustering. Journal of computers*, 4(3):230–237, 2009.*

SITOGRAFIA

[1] *Dataset classificabile tramite LDA.* https://medium.com/journey-2-artificial-intelligence/lda-linear-discriminant-analysis-using-python-2155cf5b6398.

[2] *Dataset classificato tramite KNN, con K diversi.* https://ii.uni.wroc.pl/~lipinski/DM2017/uczenie_nadzorowane.pdf.

[3] *Dati classificabili tramite DTC.* https://medium.com/machine-learning-101/chapter-3-decision-trees-theory-e7398adac567.

[4] *Dati classificabili tramite QDA.* http://scikit-learn.org/stable/modules/lda_qda.html.

[5] *MLP.* https://medium.com/@annishared/build-your-first-neural-network-in-python-c80c1afa464.

[6] *Percettrone.* https://slideplayer.it/slide/570385/.