



UNIVERSITÀ DEGLI STUDI DI MILANO
FACOLTÀ DI SCIENZE E TECNOLOGIE

CORSO DI LAUREA IN INFORMATICA

*Confronto tra approcci basati su clustering e SVM
nell'induzione di insiemi fuzzy*

Relatore:
Prof. Anna Maria Zanaboni

Candidato:
Alessia Lombarda

Correlatore:
Prof. Dario Malchiodi

Matr. 908879

ANNO ACCADEMICO 2019/2020

Ringraziamenti

Prima di procedere con la descrizione del lavoro di tirocinio vorrei dedicare qualche riga a tutti coloro che mi hanno sostenuta lungo questo percorso. Ringrazio i miei genitori, che mi hanno permesso di portare avanti il percorso universitario e che mi hanno sostenuto in ogni momento. Ringrazio mia sorella Ilaria, che ha compreso le difficoltà di questi tre anni e ha saputo alleggerirle, e ne ha festeggiato con me le soddisfazioni. Ringrazio la mia relatrice e il mio correlatore, la professoressa Zanaboni e il professor Malchiodi, per il sostegno e la continua presenza durante questi mesi di tirocinio. Un sentito grazie a Rita, che mi ha accompagnato in questo percorso di tirocinio e ne ha condiviso con me ogni momento: lavorare con te è stato davvero un piacere! Un ringraziamento particolare va ad Andrea e Elisa, compagni di viaggio in questi anni di università: mi avete spronato a fare meglio e a non lasciarmi abbattere dalle difficoltà incontrate, avete condiviso con me ogni giorno di questo percorso universitario: senza di voi non sarei arrivata fino a qui!

Indice

Ringraziamenti	i
Introduzione	1
1 Apprendimento supervisionato	3
1.1 Apprendimento supervisionato	3
1.2 Insiemi e logica fuzzy	8
2 Induzione della funzione di membership	12
2.1 Clustering	12
2.1.1 Breve classificazione	13
2.1.2 Fuzzy clustering	15
2.2 Support Vector Machine	17
2.3 Fuzzy Support Vector Machine	19
3 Implementazione ed esperimenti	23
3.1 Dataset	23
3.2 Algoritmo basato su Fuzzy Clustering	25
3.3 Algoritmo basato su Fuzzy Support Vector Machine	28
3.4 Risultati	28
Conclusioni	30

Introduzione

Questo elaborato descrive il lavoro di tirocinio in cui mi sono occupata della ricerca, modifica e confronto di implementazioni di algoritmi per l'induzione di funzioni di membership a insiemi fuzzy. La logica fuzzy e gli insiemi fuzzy sono vastamente diffusi nel campo dell'apprendimento automatico (e in particolare di quello supervisionato) in quanto modellano l'incertezza propria di gran parte realtà che ci circonda. Se nella logica classica un elemento può appartenere o no a un dato insieme, gli insiemi fuzzy sono descritti da una funzione di membership che determina un grado di appartenenza di un oggetto all'insieme in questione.

Il problema dell'induzione della funzione di membership a un insieme fuzzy è complesso, anche a causa della mancanza di accordo sull'interpretazione della funzione stessa; il problema può essere approcciato in diversi modi: si sono approfonditi in questo elaborato i metodi riguardanti Fuzzy Clustering e Fuzzy Support Vector Machine.

Tra i diversi algoritmi individuati sono stati scelti il Fuzzy C-Means (algoritmo di clustering) e la Fuzzy Support Vector Machine for Class Imbalance Learning. A partire dal codice reperito in rete si sono quindi effettuate delle modifiche al fine di garantirne un funzionamento corretto e di migliorarne la leggibilità e le prestazioni. Si è infine svolta una valutazione sulle prestazioni degli algoritmi considerati, sui dataset Iris e Breast Cancer. Si è notato come l'algoritmo FCM ottiene prestazioni molto buone quando si lavora sulla membership alla classe setosa, che è linearmente separabile dalle altre, mentre ottiene prestazioni discrete sulla definizione della membership alle altre due classi così come sul dataset Breast Cancer. L'algoritmo FSVM-CIL ottiene invece prestazioni discrete su tutti i dataset, e il suo funzionamento è oneroso in termini di tempo.

L'elaborato è strutturato come segue: nel Capitolo 1 si descrivono le basi dell'apprendimento automatico (nello specifico di quello supervisionato); successivamente viene spiegato il funzionamento generale di un algoritmo di apprendimento, vengono definite le metriche che saranno poi utilizzate negli esperimenti e le tecniche di cross-validation e model selection. Viene inoltre introdotta la logica fuzzy e la sua estensione agli insiemi, e si inizia a delineare il problema dell'induzione della funzione di membership. Nel Capitolo 2 si introducono le macroclassi di algoritmi per l'induzione della funzione di membership: do-

po una panoramica sul funzionamento generale degli algoritmi di clustering e support vector machine e dei corrispettivi fuzzy, si analizzano nel dettaglio il metodo FCM e l'algoritmo FSVM-CIL. Nel Capitolo 3, infine, si mostrano i dataset utilizzati negli esperimenti effettuati, le problematiche riscontrate nel codice reperito e le soluzioni individuate. Successivamente si illustrano i risultati ottenuti, che vengono poi analizzati anche grazie all'uso di rappresentazioni grafiche.

Capitolo 1

Apprendimento supervisionato e induzione di insiemi fuzzy

1.1 Apprendimento supervisionato

L'apprendimento automatico è una branca dell'intelligenza artificiale che studia algoritmi che utilizzano l'esperienza per migliorare la propria performance o per fare predizioni più accurate. In questo caso l'esperienza viene rappresentata attraverso una collezione di dati resi disponibili all'analisi (si può definire ogni elemento dell'insieme di dati come *esempio*). Questi dati possono essere aumentati con delle *etichette* (dette anche *label*) impostate manualmente oppure venire estratti direttamente dall'ambiente: gli algoritmi avranno l'obiettivo di effettuare predizioni su dati non noti, sulla base della conoscenza acquisita. L'apprendimento automatico si applica a svariati campi quali il riconoscimento vocale, la guida autonoma di veicoli, i motori di ricerca o la ricerca scientifica in campo medico. Alcune delle classi principali di problemi che si possono affrontare con delle tecniche di apprendimento automatico sono:

- **classificazione:** l'assegnamento di oggetti a categorie; in questo caso di solito si hanno poche classi;
- **regressione:** la predizione di valori reali a partire da dati non noti; in questo caso l'errore associato ad una predizione errata dipende della differenza tra il valore predetto e quello reale;
- **ranking:** l'ordinamento di elementi in base a un criterio definito a priori;
- **clustering:** il partizionamento di elementi in gruppi omogenei.

L'apprendimento automatico può essere *supervisionato*, *non supervisionato* o *semi-supervisionato*. Nell'apprendimento non supervisionato gli esempi a disposizione nell'insieme di dati forniti sono degli oggetti non associati ad etichette; l'obiettivo dell'apprendimento è quindi quello di trovare dei raggruppamenti omogenei di tali oggetti (come nel caso del clustering) sulla base di un criterio di similarità o dissimilarità definito a priori. Nell'apprendimento semi-supervisionato, invece, si hanno solitamente a disposizione una gran quantità di dati senza etichetta e un numero ridotto di dati associati ad un'informazione (che rappresenta una quantità che si vuole predire); ciò si verifica spesso in situazioni reali in cui l'etichettatura dei dati è molto costosa e/o si dispone di un flusso costante di dati. Se ad esempio consideriamo i messaggi inappropriati in un social network come insieme di dati, possiamo notare come non ci sia modo di ottenere informazioni etichettate a mano su ogni messaggio, perché a causa della grande quantità di dati sarebbe troppo costoso. Possiamo quindi etichettare a mano un sottoinsieme di essi e, sfruttando le tecniche semi-supervisionate, utilizzare questo piccolo set di dati etichettati per comprendere il resto del contenuto dei messaggi.

Nell'apprendimento supervisionato l'informazione aggiuntiva (detta *etichetta* o *label*) è invece specificata per ogni dato a disposizione; lo scopo dell'algoritmo è quello di trovare un modello che permetta di effettuare predizioni anche su dati non noti. Si può modellare quindi una funzione f che associa ad ogni dato la corrispondente etichetta; lo scopo dell'apprendimento supervisionato è quello di determinare (a partire dai dati) una funzione h che approssimi in modo accettabile¹ la funzione f . Sia f che h sono funzioni di un vettore di input $X = (x_1, x_2, \dots, x_i, \dots, x_n)$ di n componenti (detti anche *feature*) [1]. Nel caso di apprendimento supervisionato si conoscono quindi (a volte solo in modo approssimato) i valori di f per gli m elementi del training set, Ξ . Assumiamo che, se possiamo trovare una funzione h che restituisce valori che approssimano in modo accettabile quelli di f per gli elementi di Ξ , allora questa funzione costituisce una buona approssimazione per f , specialmente se Ξ è grande.

Per quanto riguarda il vettore di input X , questo può essere costituito da valori reali, discreti o categorici, che a loro volta possono essere ordinati (ad esempio $\{\text{piccolo}, \text{medio}, \text{grande}\}$) o non ordinati (ad esempio $\{\text{nord}, \text{sud}, \text{est}, \text{ovest}\}$); un caso a sé è costituito dall'uso di valori booleani, che può essere considerato un caso particolare dell'utilizzo di numeri discreti (1,0) o di variabili categoriche (*True*, *False*).

L'output della funzione può essere invece un numero reale, ovvero una stima, oppure un valore categorico, tipico dei classificatori. Nel caso di output booleani, invece, possiamo distinguere *istanze positive*, ovvero quelle con label 1, e *istanze negative*, quelle con label 0; se anche l'input è booleano il classificatore implementa una *funzione booleana*.

Definiamo ora il *bias*, ovvero un insieme di informazioni che definisce la po-

¹definire una metrica che indichi la bontà dell'approssimazione è un problema significativo

tenza rappresentativa di un modello di apprendimento; il termine bias venne introdotto da Mitchell(1908) a significare "ogni informazione su cui ci si basa per scegliere un'ipotesi di generalizzazione rispetto ad un'altra". Si possono distinguere due tipi di bias: quello assoluto e quello relativo. Il *bias assoluto* è un'assunzione fatta dall'algoritmo di apprendimento che restringe l'insieme delle possibili funzioni target(ovvero quelle tra cui scegliere la funzione h che approssima f) a un sottoinsieme di funzioni (ad esempio le funzioni lineari, o quelle booleane). Il *bias relativo*, invece, assume che la funzione da apprendere sarà probabilmente in un insieme di funzioni invece che in un altro [2]. Se ad esempio si considera un algoritmo che deve approssimare una funzione booleana in n variabili, si potranno avere 2^n possibili input. Si supponga di non avere bias,

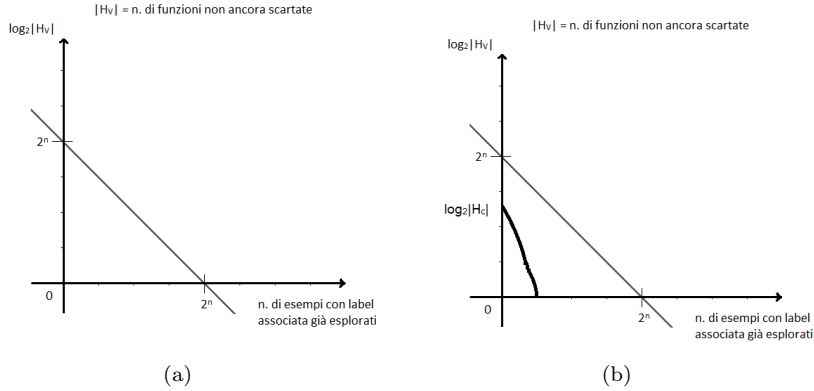


Figura 1.1: Relazione tra il numero di funzioni possibili e gli elementi del training set già esplorati in assenza e presenza di bias

che si stia considerando l'insieme \mathcal{H} delle 2^{2^n} possibili funzioni booleane e che non si abbiano preferenze tra quelle che danno corrispondenze sull'insieme dei dati usati per l'apprendimento (detto *training set*). In questo caso, dopo aver considerato un elemento del training set e la sua etichetta si potrà suddividere a metà l'insieme delle funzioni, ovvero distinguere quelle che classificherebbero correttamente quell'elemento dalle restanti. Presentando via via più elementi del training set, ad ogni passo del processo si dimezza il numero di funzioni considerabili come ipotesi di approssimazione per f , come mostrato in Figura 1.1a. Non è possibile in questo caso generalizzare, perché le corrispondenze già trovate non danno alcun indizio su quelle ancora da scoprire: è solo possibile memorizzare le prime, e ciò causa un apprendimento molto oneroso.

Se invece si limitasse l'insieme \mathcal{H} a un sottoinsieme \mathcal{H}_c , dipendentemente dal sottoinsieme e dall'ordine di presentazione dei dati del training set, la curva che rappresenta il numero di possibili ipotesi di funzioni apparirebbe come in Figura 1.1b: potrebbe infatti accadere che dopo aver visto meno di 2^n campioni si arrivi già a selezionare una funzione h che ben approssimi f . L'introduzione del bias, quindi, ci permette di considerare solo alcune classi di funzioni, rendendo

l'apprendimento più efficiente (come mostrato in Figura 1.1b).

Spieghiamo ora il funzionamento dell'algoritmo di apprendimento: inizialmente si ha una fase di addestramento sul *training set*, seguita da una fase di valutazione su un insieme distinto di dati, il *test set*. Si dice che una funzione *generalizza* se è in grado di predire “bene” anche i dati presenti nel test set.

In questo contesto è anche necessario selezionare le feature da considerare per ogni campione, in quanto feature rilevanti nella soluzione del problema che si sta considerando possono guidare l'algoritmo di apprendimento correttamente, mentre altre meno rilevanti possono portare a un errore maggiore. Risulta quindi necessario, prima di procedere con l'apprendimento, utilizzare un algoritmo di *feature selection*.

Altra questione da considerare è la possibile presenza di rumore nei vettori del training set: possiamo distinguere il *rumore di classe*, che altera il valore della funzione f , e quello di *attributo*, che altera in modo causale i valori del vettore di input X ; in entrambi i casi l'errore rende imprecisa la corrispondenza tra i valori di input e quelli della funzione applicata su di essi.

Fondamentale è poi la definizione di un metodo per la valutazione della performance dell'algoritmo: consideriamo a questo scopo l'*accuratezza* e le *funzioni di errore*. L'accuratezza consiste nel contare il numero di predizioni corrette, e dividere questo valore per la cardinalità del set di dati: si otterrà un valore compreso tra 0 e 1, che sarà maggiore se l'algoritmo predice in modo corretto. Un altro stimatore è l'*errore quadratico medio*, che si vuole minimizzare:

$$MSE = \frac{1}{n} \sum_{i=1}^n (y_i - h(x_i))^2,$$

dove y_i è la label attesa e $h(x_i)$ quella predetta dall'algoritmo, e la sua radice quadrata, l'RMSE:

$$RMSE = \sqrt{\frac{1}{n} \sum_{i=1}^n (y_i - h(x_i))^2}.$$

La scelta dello stimatore dipende dalla classe a cui appartiene il problema considerato: si utilizza infatti l'accuratezza nei problemi di classificazione, in cui data una label predetta possiamo dire con certezza se questa sia corretta oppure se è diversa da quella attesa; l'MSE è invece tipico dei problemi di regressione (anche se può comunque essere usato per la classificazione), in quanto considera quanto il valore reale predetto e quello atteso si discostano tra loro.

Per migliorare la performance dell'algoritmo è necessario evitare che questo si specializzi eccessivamente sui dati (*overfitting*). A causa della ridotta dimensione del training set disponibile, si è soliti usare una tecnica nota come *n-fold cross-validation*, usata sia per la model selection (la selezione dei parametri liberi

dell'algoritmo), sia per la fase di valutazione della generalizzazione. Se infatti il dataset disponibile ha una dimensione ridotta, la divisione in training e test set ridurrebbe ulteriormente i dati disponibili per l'apprendimento: questo si può evitare utilizzando la cross-validation, che considera nell'intero processo tutti i dati per la fase di training ma permette al contempo una fase di valutazione rigorosa, distinguendo ogni volta dati di training e di test.

Approfondiamo inizialmente l'uso della k-fold cross-validation per la fase di model selection [3]: la model selection serve per la definizione dei cosiddetti *iperparametri* o *parametri liberi*, ovvero parametri iniziali necessari su cui il processo di apprendimento si basa. Questi parametri sono distinti da quelli che l'algoritmo determina nel processo di apprendimento; tipici esempi sono il kernel e il parametro gamma per le Support Vector Machine.

Detto θ il vettore dei parametri liberi dell'algoritmo, ipotizziamo per semplicità che ci sia un solo parametro libero, da scegliere all'interno di un insieme di possibili valori. Per un fissato valore di θ si partiziona in modo causale un training S di m elementi con etichette associate in n sottogruppi (*fold*). L' i -esimo fold S_i sarà quindi un campione $((x_1, y_1), \dots, (x_{m_i}, y_{m_i}))$ di cardinalità m_i . Poi, per ogni $i \in \{1, \dots, k\}$ l'algoritmo viene addestrato su un *validation set* costituito da tutti i fold tranne l' i -esimo, per generare l'ipotesi h_i , e la performance di h_i viene valutata sull' i -esimo fold. Il valore del parametro θ è scelto sulla base dell'errore medio di h_i , chiamato *cross-validation error*, denotato da $\hat{R}_{CV}(\theta)$ e definito da

$$\hat{R}_{CV}(\theta) = \frac{1}{k} \sum_{i=1}^n \frac{1}{m_i} \sum_{(x,y) \in S_i} L(h_i(x), y)$$

I fold sono generalmente definiti della stessa taglia, ovvero $m_i = m/n$ per ogni $i \in [1, n]$. La scelta di n è molto importante ai fini dell'utilizzo di questo metodo: con un n grande ogni fold avrà taglia prossima ad m , la cardinalità dell'intero dataset, ma i fold saranno piuttosto simili, e quindi il metodo avrà un bias ridotto ed una grande varianza. Viceversa, valori bassi di n portano a training set differenziati, ma la loro taglia è significativamente più bassa di m e quindi il metodo tenderà ad avere una varianza minore ma un bias maggiore. I valori che generalmente si scelgono come n sono 5 o 10.

Per la model selection, quindi, la cross-validation è utilizzata in questo modo: l'intero dataset viene suddiviso inizialmente in training e test set, dove il training set ha taglia m . Questo viene poi utilizzato per computare il cross-validation error $\hat{R}_{CV}(\theta)$ per un certo numero di possibili valori di θ ; viene considerato come θ_0 il valore per cui $\hat{R}_{CV}(\theta)$ è minimo, e l'algoritmo viene addestrato con i parametri di θ_0 sull'intero training set di taglia m . La sua performance è poi valutata sul test set. Il metodo si può applicare allo stesso modo per quanto riguarda la valutazione della performance: si otterranno delle valutazioni parziali (accuratezze o errori), di cui possiamo calcolare la media per avere un'indicazione della bontà dell'apprendimento. Se vogliamo svolgere al contempo una cross-validation che valuti la performance e una prima fase di model selection, sarà necessario utilizzare due cross-validation annidate: la cross-validation esterna suddivide ogni volta i dati in training set e test set, e valuta la bontà dell'apprendimento; la

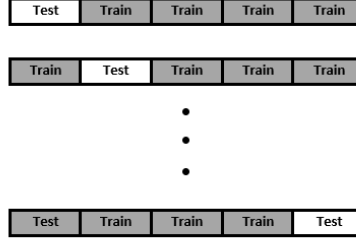


Figura 1.2: n-fold cross-validation

cross-validation interna, invece, svolge la model selection, suddividendo a sua volta il training set iniziale in fold e valutando gli iperparametri scelti di volta in volta.

1.2 Insiemi e logica fuzzy

La teoria classica degli insiemi, inizialmente formulata da Zadeh [4], è basata sul concetto fondamentale di *insieme* di cui un elemento fa parte oppure no, una distinzione netta che permette di identificare un chiaro confine tra ciò che appartiene e non appartiene all'insieme. Molti problemi reali non possono però essere descritti e gestiti dalla teoria classica degli insiemi, in quanto trattano elementi che hanno appartenenza solo parziale ad un dato insieme. La teoria dei *fuzzy set* (o insiemi sfocati), generalizza ed estende in questo senso la teoria classica degli insiemi [5, 6].

Nella teoria classica possiamo definire la funzione caratteristica di un insieme, ovvero la funzione di appartenenza a un insieme A come

$$X_A(x) = \begin{cases} 1, & \text{se } x \in A \\ 0, & \text{se } x \notin A \end{cases}$$

Ma se un elemento ha membership soltanto parziale ad un insieme dobbiamo generalizzare questa funzione per determinare il grado di membership dell'elemento stesso: un valore maggiore indicherà un grado di membership più alto all'insieme. Se consideriamo ad esempio l'insieme universo S di tutti gli esseri umani, e S_f come

$$S_f = \{ s \in S \mid s \text{ è vecchio} \},$$

S_f è un *sottoinsieme fuzzy* di S , perché la proprietà “vecchio” non è ben definita e non può essere misurata in modo preciso: dobbiamo quindi stabilire una funzione che assegni il “valore di vecchiaia” di un essere umano, ma non esiste un criterio unico o universale per scegliere questa funzione.

Un sottoinsieme fuzzy è quindi definito da un sottoinsieme e da una funzione di membership μ ad esso associata, che associa ad ogni elemento dell'insieme un valore reale nell'intervallo $[0, 1]$ (si vedano alcuni esempi in Figura 1.3).

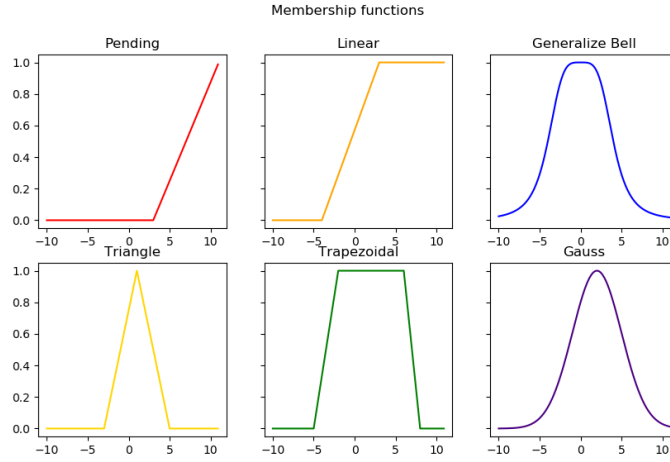


Figura 1.3: Forme di funzioni di membership comunemente usate

La definizione di insiemi fuzzy ha come base la logica fuzzy, che estende e potenzia la logica classica. La logica classica, infatti, riguarda proposizioni che possono essere vere o false ma che non possono mai assumere contemporaneamente entrambi i valori di verità, che hanno opposto e possono essere combinate tra loro. La logica fuzzy, che pone la base del ragionamento approssimato, si differenzia da quella classica in quanto ammette l'utilizzo termini linguistici imprecisi come

- *predicati fuzzy*: vecchio, giovane, alto, veloce,
- *quantificatori fuzzy*: molto, poco, quasi, di solito,
- *valori di verità fuzzy*: molto vero, probabilmente falso, sicuramente falso,

che permettono di trattare fenomeni della realtà difficilmente descrivibili in modo quantitativo utilizzando la logica classica, ma su cui la maggior parte del ragionamento umano si basa. In particolare si dicono *variabili linguistiche* gli attributi dei sistemi fuzzy che assumono *valori linguistici* espressi in linguaggio naturale. Questi valori hanno un significato e non un preciso valore numerico, e partizionano i possibili valori delle variabili linguistiche in modo soggettivo (ovvero solitamente basato sull'intuizione umana). Se ad esempio consideriamo come variabile linguistica la superficie abitabile A di un appartamento, e definiamo i valori linguistici $\{\text{molto piccolo}, \text{piccolo}, \text{medio}, \text{grande}, \text{molto grande}\}$ e le funzione di membership μ_k , con $k = mp, p, m, g, mg$ (in corrispondenza

dei termini linguistici), descritte come in Figura 1.4, possiamo notare che ogni $x \in A$ ha $\mu(x) \in [0, 1]$ per ogni valore. Se ad esempio consideriamo $x = 41m^2$, avremo che $\mu_{mp}(x) = \mu_p(x) = \mu_{mg}(x) = 0$, mentre $\mu_m(x) = 0.1$ e $\mu_g(x) = 0.9$.

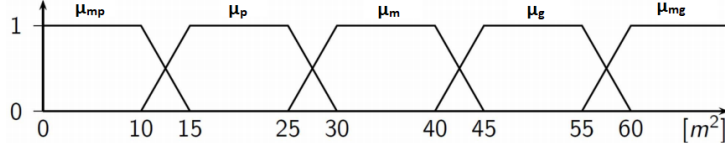


Figura 1.4: Esempio: funzioni di membership in riferimento alla superficie abitabile di un appartamento

L'applicazione della logica fuzzy a problemi di apprendimento passa attraverso l'utilizzo dei *FIS* (Fuzzy Inference Systems), che sono composti da cinque blocchi funzionali (Figura 1.5):

- *rule base*: contiene le regole if-then fuzzy definite da esperti,
- *database*: definisce le membership function dei fuzzy set usati nelle regole fuzzy,
- *inference engine*: esegue operazioni sulle regole,
- *fuzzifier*: converte le quantità crisp (ovvero valori numerici trattabili dalla logica classica) in quantità fuzzy,
- *defuzzifier*: converte le quantità fuzzy in quantità crisp.

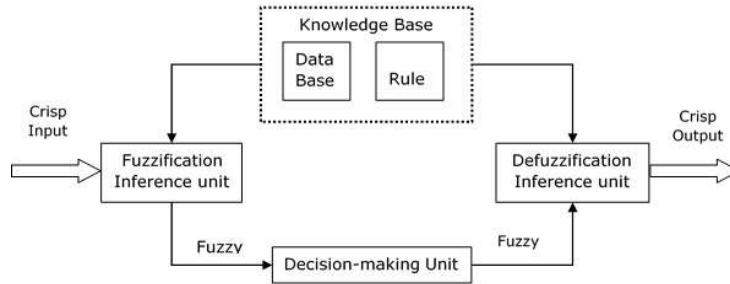


Figura 1.5: Struttura di un FIS

Il FIS quindi, e in particolare il fuzzifier converte l'input da crisp a fuzzy, aggiornando la Knowledge Base formata da Database e Rule base; l'input viene poi passato all'Inference Engine, che determina il match con le regole if-then (e aggiorna le regole stesse sulla base dell'input), ovvero associa input fuzzy ad output fuzzy, che viene poi ritrasformato in output crisp dal defuzzifier.

I concetti della teoria degli insiemi fuzzy applicati all'apprendimento automatico hanno portato alla nascita del *fuzzy machine learning*, che ha permesso l'estensione di tecniche di apprendimento non fuzzy a corrispondenti tecniche fuzzy: si può parlare di fuzzy clustering, fuzzy support vector machine, fuzzy k-nearest neighbour e così via. Queste tecniche sono utilizzate principalmente in tre ambiti: quelli di classificazione e data analysis, i problemi di decision-making e il ragionamento approssimato; questi problemi mostrano le tre semantiche attribuibili al grado di membership, ovvero la similarità, la preferenza e la possibilità. Per quanto riguarda la prima categoria, che approfondiremo in seguito, il grado di membership $\mu(u)$ di un elemento u indica la relazione tra oggetti di un universo, in termini di similarità, dissimilarità o in funzione della distanza; questa semantica è specialmente utilizzata per problemi di classificazione, clustering e regressione. Per quanto riguarda invece la preferenza, dato un insieme di oggetti e una funzione di membership quest'ultima rappresenta quanto si favorisce un determinato oggetto rispetto agli altri; interpretando la funzione di membership come grado di incertezza, infine, si intende il valore della funzione di membership μ come il grado di possibilità che un oggetto x abbia valore u [7].

Nel campo dell'apprendimento automatico, quindi, le tecniche di induzione di funzioni di appartenenza producono modelli che trasformano variabili di input in gradi di appartenenza ad un insieme fuzzy. Si possono distinguere a questo proposito tecniche induttive o deduttive: l'uso di metodi deduttivi implica la definizione di funzioni di appartenenza da parte di esperti, ovvero sulla percezione umana (tecniche *data-driven*). Le tecniche induttive, invece, prevedono la generazione delle funzioni di appartenenza sulla base dei dati (tecniche *data-driven*). Si approfondiranno in seguito queste ultime, più affidabili in quanto slegate dalle diverse interpretazioni che esperti possono dare agli stessi attributi o variabili.

Capitolo 2

Tecniche per l'induzione della funzione di membership

Determinare la funzione di membership a partire dai dati di training è uno dei problemi fondamentali legati all'applicazione della teoria dei fuzzy set. Non esistono però regole o linee guida che possano essere usate per scegliere l'appropriata tecnica di generazione, e il problema è reso più complesso dalla mancanza di accordo sulla definizione e interpretazione delle funzioni di membership: uno stesso concetto può infatti essere definito e quindi interpretato in modi diversi, portando alla determinazione di funzioni di appartenenza diverse tra loro. I diversi metodi presenti in letteratura spaziano da tecniche basate sulla percezione, cioè sul giudizio umano o di esperti, a tecniche euristiche, che usano forme di funzioni di membership predefinite, tecniche basate su istogrammi (che utilizzano appunto istogrammi per considerare la distribuzione delle features di input), trasformazione di distribuzioni di probabilità in distribuzioni di possibilità, tecniche basate su fuzzy nearest neighbour o su reti neurali (come approfondito in [8]), su clustering o support vector machine (che approfondiremo meglio in seguito); l'intera rassegna è presentata in [9].

2.1 Clustering

Gli algoritmi classici di clustering hanno come fine il raggruppamento di elementi omogenei in un insieme di dati. Dati in input N oggetti, un algoritmo di clustering restituisce in output un raggruppamento (detto *clustering*) $\mathcal{P} = \{P_1, \dots, P_c\}$ degli oggetti appartenenti a un insieme universo X , ovvero una partizione di X . L'insieme X è quindi lo spazio in cui giacciono i vettori di input, e dato un suo

generico elemento x_k sappiamo che questo appartiene a \mathbb{R}^n . Due elementi sono detti omogenei o disomogenei in relazione a una funzione di similarità o dissimilarità $\sigma : X \times X \mapsto \mathbb{R}$, detta *metrica*, che assegna un valore a ogni possibile coppia di punti: un esempio banale può essere costituito da una funzione che assegna il valore 1 a elementi omogenei e il valore -1 a elementi disomogenei. Si possono utilizzare svariati tipi di funzioni: un esempio comune è la norma euclidea, che è una funzione di dissimilarità e si può utilizzare quando le descrizioni degli oggetti u e v sono punti in \mathbb{R}^n per qualche n . Dati quindi due punti u e v , la norma euclidea è definita come:

$$\sigma(u, v) = \|u - v\|;$$

la funzione σ assumerà valori elevati in caso di elementi molto dissimili tra loro e viceversa valori più bassi se u e v sono simili.

2.1.1 Breve classificazione

Dati un elemento v e una metrica σ , si possono operare due diversi tipi di clustering: il clustering *partition based*, che utilizza una distanza da un punto rappresentativo del cluster (ad esempio il *centroide*) per determinare l'appartenenza di v a quel cluster, avendo prefissato il numero di cluster della partizione risultato, oppure clustering *gerarchico*, in cui viene costruita una gerarchia di partizioni visualizzabile mediante una rappresentazione ad albero (dendrogramma, come in Figura 2.1), in cui sono rappresentati i passi di accorpamento/divisione dei gruppi stessi.

Le tecniche di clustering si possono inoltre suddividere sulla base del modo in

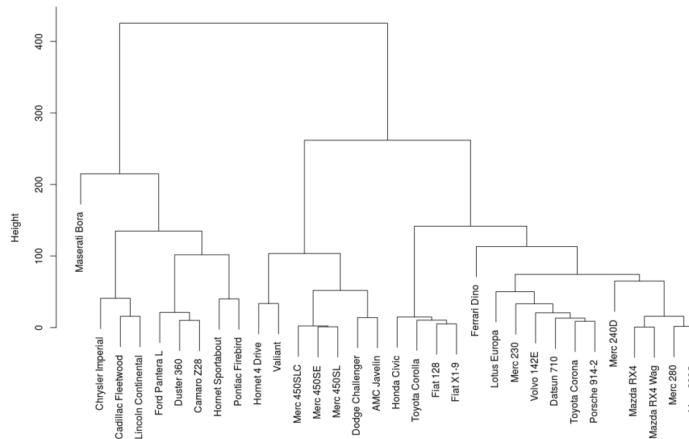


Figura 2.1: Dendrogramma che mostra il diverso consumo di carburante nei modelli di automobili considerati

cui operano: si distinguono *metodi aggregativi* (o *bottom-up*) e *metodi divisivi*

(o *top-down*). La prima classe comprende i metodi che considerano inizialmente tutti gli elementi come cluster a sé, e in cui poi l'algoritmo provvede a unire i cluster più vicini. L'algoritmo continua questo processo fino a ottenere un numero prefissato di cluster, oppure fino a che la distanza minima tra i cluster non supera un certo valore, o ancora in relazione a un determinato criterio di arresto prefissato. I metodi divisivi, invece, prevedono che all'inizio tutti gli elementi siano in un unico cluster: l'algoritmo, poi, inizia a dividere il cluster in tanti sottoinsiemi di dimensioni inferiori, cercando di ottenere gruppi sempre più omogenei. L'algoritmo procede fino a che non viene soddisfatto un generico criterio di arresto, come per i metodi aggregativi, o ad esempio fino al raggiungimento di un numero prefissato di cluster.

È possibile inoltre operare un'ulteriore classificazione delle tecniche di clustering: si distingue il clustering di tipo *esclusivo* o *hard clustering*, in cui ogni elemento può essere assegnato a uno e un solo gruppo (di conseguenza i cluster saranno a due a due disgiunti) e il clustering *non-esclusivo*, noto anche con il nome di *soft clustering* o *fuzzy clustering*.

La maggior parte dei metodi di fuzzy clustering esistenti fa riferimento al *Fuzzy C-Means*, la variante fuzzy del classico *k-means*. L'algoritmo k-means, proposto nel 1967 da MacQueen, è uno degli algoritmi di apprendimento base utilizzato per risolvere problemi di clustering; lo scopo è quello di classificare un dataset di cardinalità N in un certo numero c di cluster P_1, P_2, \dots, P_c , definito a priori. Si definiscono inizialmente c *centroidi*, uno per ogni cluster, a rappresentare i centri iniziali dei c cluster. Questi centroidi devono essere scelti in modo accurato, e preferibilmente inizialmente distanti gli uni dagli altri: la posizione dei centroidi iniziali influisce infatti sul risultato del clustering. Successivamente, si assegna ogni punto nel dataset al centroide a esso più vicino (utilizzando come funzione di similarità la norma); a questo punto si ricalcolano i c nuovi centroidi, ottenuti come baricentro dei punti in un dato cluster. Si ripetono queste due operazioni di riassegnazione dei punti e correzione dei centroidi finché non si hanno più cambiamenti significativi, ovvero finché il clustering è stabile. Lo scopo di questo algoritmo è infatti quello di minimizzare una funzione obiettivo J , espressa come

$$J = \sum_{i=1}^c \sum_{x_k \in P_i} \|x_k^{(i)} - v_i\|^2,$$

ovvero la somma dei quadrati delle distanze di ogni punto dal centro dei diversi cluster, dove $x_k^{(i)}$ è il k -esimo elemento appartenente all' i -esimo cluster e v_i è il centroide dell' i -esimo cluster. L'algoritmo k-means termina sempre, ma non trova necessariamente il clustering ottimo (quello che corrisponde al minimo globale della funzione obiettivo): il problema infatti è NP-hard, e l'algoritmo è molto sensibile alla posizione dei cluster iniziali. Per ridurre questo effetto è necessario ripetere più volte l'esecuzione dell'algoritmo.

2.1.2 Fuzzy clustering

I problemi di clustering possono avere applicazioni in svariati campi, come l'economia [10], psicologia [11], medicina [12, 13], biologia [14, 15]; ne è fatto in particolare un vasto utilizzo nel campo della bioinformatica, in cui il fuzzy clustering è preferito a quello classico, nel campo del marketing, in cui a esempio gli utenti possono essere suddivisi in cluster fuzzy sulla base di bisogni, preferenze e profili o ancora nell'analisi di immagini, in cui è usato in particolare per la segmentazione al fine di svolgere pattern recognition e object detection. Nel fuzzy clustering infatti, in opposizione alle tecniche classiche di clustering come il *k-means*, gli elementi di input possono appartenere a più di una partizione, con un certo valore di membership. La membership non rappresenta però una probabilità: la somma dei valori di membership assegnati a un elemento in relazione a classi opposte non deve perciò dare 1; la membership sarà più bassa se il punto si colloca sul limite esterno del cluster, e viceversa maggiore se l'elemento è vicino al centro del cluster.

Il *Fuzzy C-Means*, metodo alla base del fuzzy clustering, si basa come spiegato in precedenza sul *k-means* e generalizza l'algoritmo in senso fuzzy: gli oggetti che si collocano sul confine di un dato cluster non sono forzati ad appartenere a un solo cluster, ma possono invece appartenere a diversi insiemi con una membership compresa tra 0 e 1, garantendo una prestazione migliore specialmente in presenza di rumore nei dati o di outlier [16]. Gli elementi saranno quindi assegnati a ogni cluster con diversi valori di appartenenza: punti più vicini al centro del cluster avranno membership maggiore, e viceversa punti lontani avranno membership minore.

Fissato un dato numero di cluster c , dove $2 \leq c \leq N$, e selezionato un valore per il parametro m (che controlla la *fuzziness*, concetto che verrà approfondito in seguito), è necessario quindi inizializzare la *partition matrix* U . La matrice $U = [u_{ik}]$ è una matrice $c \times N$ di numeri reali (N è il numero di osservazioni in \mathbb{R}^n), e vale $0 \leq u_{ik} \leq 1$; x_k è il k -esimo feature vector, e

$$\begin{aligned} \sum_{i=1}^c u_{ik} &> 0 \quad \forall i, \\ \sum_{i=1}^c u_{ik} &= 1 \quad \forall k. \end{aligned}$$

Si possono scegliere varie funzioni come metriche per misurare la similarità; la più nota è l'uso dei *minimi quadrati generalizzati* (o Generalized Least Squares, GLS), definita come

$$J_m(U, v) = \sum_{k=1}^N \sum_{i=1}^c u_{ik}^m \|x_k - v_i\|_A^2,$$

dove:

- $X = x_1, x_2, \dots, x_N \subset \mathbb{R}^n$ è l'insieme dei dati,
- $2 \leq c < n$ è il numero di cluster desiderati,
- $1 \leq m < \infty$ è il parametro che esprime la fuzziness,
- U è la partizione fuzzy di X ,
- $v = (v_1, v_2, \dots, v_c)$ è il vettore dei centroidi,
- $v_i = (v_{i1}, v_{i2}, \dots, v_{in})$ è il centro del cluster i ,
- $\|\cdot\|_A^2$ è la norma indotta su \mathbb{R}^n da una matrice quadrata A di pesi positivi, corrisponde alla distanza quadratica tra x_k e v_i ed è calcolata come

$$d_{ik}^2 = \|x_k - v_i\|_A^2 = (x_k - v_i)^T A (x_k - v_i).$$

Ogni errore quadratico è poi pesato con $(u_{ik})^m$, l' m -esima potenza della membership degli elementi x_k nel cluster i .

Sono necessarie a questo punto delle considerazioni sui parametri m ed A . Si può infatti notare che per $m \rightarrow 1$ il clustering diventa più netto e con $m = 1$ si ottiene un *hard clustering*, ovvero una partizione dell'insieme universo in cui la funzione di membership di ogni sottoinsieme può assumere soltanto valore 0 o 1. Solitamente si utilizza $m \in \{2, \dots, 30\}$, anche se per la maggior parte dei dati un valore di m compreso tra 1.5 e 3.0 dà buoni risultati.

Per quanto riguarda invece la matrice A , questa controlla la forma che i cluster di Y assumeranno; si possono usare a questo proposito diversi tipi di norme.

Un clustering fuzzy ottimale è dunque definito da una coppia (U, v) che minimizza J_m ; possiamo descrivere quindi l'algoritmo FCM come segue. Definita con $r = 0, 1, \dots$ ogni iterazione dell'algoritmo, e fissati c, m e A , si definisce la matrice iniziale $U^{(0)}$. Successivamente è necessario calcolare a ogni passo il vettore dei c centri, e poi una partition matrix aggiornata $U^{(t+1)}$. Si itera l'esecuzione dell'algoritmo finché l'errore commesso nell'assegnamento delle membership (ovvero la norma calcolata sulla differenza tra la matrice delle membership calcolata al tempo t e $t+1$) non scende sotto una soglia ϵ [17].

Il Fuzzy C-Means si può quindi considerare l'algoritmo alla base del fuzzy clustering; a partire da questo si sono poi sviluppate una serie di varianti che trattano diverse forme di funzioni di appartenenza, che pongono un'attenzione particolare al rumore e agli outlier come RSFKM [18] e GEPFCM [19], algoritmi che risolvono il problema di stabilire a priori il numero di cluster e la sensibilità di FCM alla posizione dei centroidi iniziali, come in [20] o ancora che utilizzano come funzioni di similarità tecniche diverse dalla distanza euclidea, come in KFCM [21].

Oltre a questa categoria di algoritmi si distinguono algoritmi di fuzzy support vector clustering, una tecnica ibrida tra clustering e uso di support vector machine (come in [22, 23]).

Una rassegna completa dei vari tipi di algoritmi è disponibile in Tabella 2.1.

Tabella 2.1: Rassegna di metodi per fuzzy clustering

Categoria	Nome metodo	Anno di pubblicazione	Articolo
Estensioni a FCM	RSFKM	2016	[18]
	Agglomerative FCM	2008	[20]
	FKM con FIS	2013	[24]
	GIFP-FCM	2009	[19]
	BCPFCM, BCPNFCM	2016	[25]
	BCSP NFCM	2016	[25]
	REFCMFS	2019	[26]
	CSFCM	2015	[27]
	ICFFCM	2019	[28]
	KFCM	2003	[21]
	DOFCM	2011	[29]
Support Vector Clustering	SVC	2013	[22]
	MSVC	2016	[23]
Shadowed Set Induction	SSI	2018	[30]
Spectral Clustering	FKSC	2018	[31]
	SSC-PC	2016	[32]
Possibilistic FCM	PFCM	1993	[33]

2.2 Support Vector Machine

Le *Support Vector Machine* (SVM), sviluppate da Vapnik nel 1995, sono una tecnica di apprendimento supervisionato utilizzato soprattutto per problemi di classificazione, regressione e individuazione di outlier; esse sono utilizzate per un ampio spettro di applicazioni, dal riconoscimento facciale [34,35] all'elaborazione di dati biologici per diagnosi mediche (come in [36,37]).

Le SVM hanno come obiettivo quello di identificare un iperpiano “ottimo” come soluzione del problema di apprendimento, cioè un iperpiano che separi due classi di elementi dell'insieme dei dati di input. L'iperpiano migliore è quello che massimizza il *margin*, definito come la distanza tra i punti più vicini all'iperpiano e l'iperpiano stesso; anche punti lontani dal margine possono essere coinvolti nella determinazione dell'iperpiano. Se si considera uno spazio bidimensionale, ad esempio, un iperpiano è una retta che divide il piano in due semispazi. Se i punti iniziali definiti in uno *spazio delle features di input* sono linearmente separabili tra loro, la soluzione al problema è semplice e consiste nella determinazione di un iperpiano della forma

$$f(x) = w \cdot x + b,$$

individuato direttamente nello spazio delle feature di input. Se invece i dati non sono linearmente separabili, è necessaria una trasformazione dei dati stessi ad

uno spazio a più dimensioni: da uno spazio delle features di input si passa ad uno *spazio delle features*, a più dimensioni. I dati vengono quindi proiettati in uno spazio di dimensioni maggiori rispetto a quello di partenza, nel quale i dati siano linearmente separabili. In questo nuovo spazio (indotto da un *kernel*), si cercherà l'iperpiano ottimale, come mostrato in Figura 2.2. Nella ricerca dell'iperpiano ottimo è accettabile la classificazione errata di alcuni punti; la misura della tolleranza alla misclassificazione verrà approfondita in seguito. L'uso delle

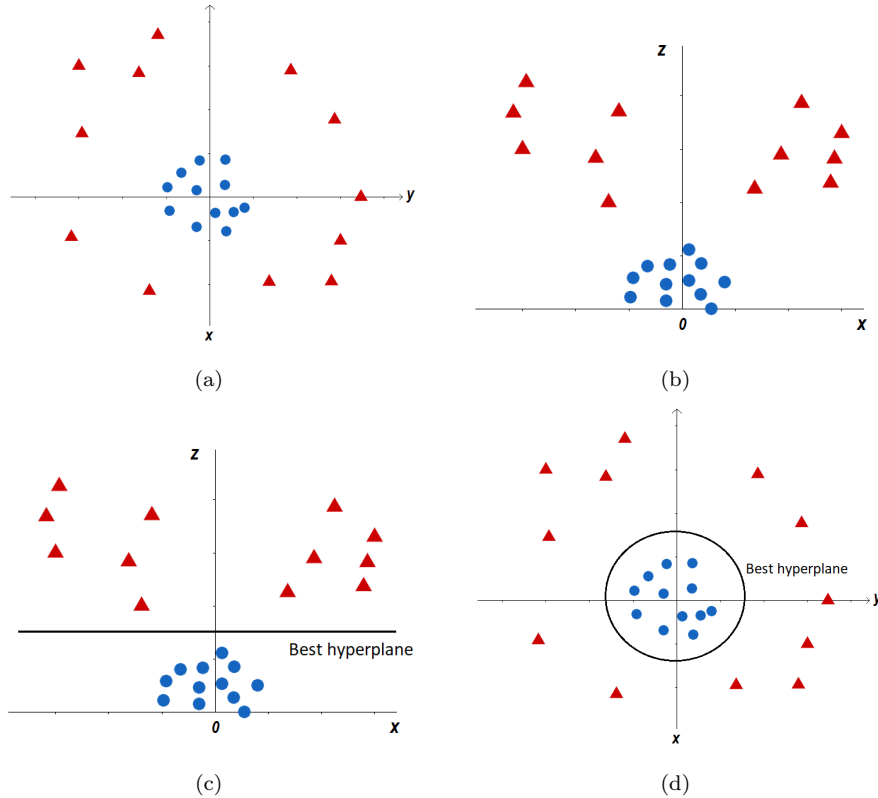


Figura 2.2: Passaggio dallo spazio delle feature di input allo spazio delle feature: spazio delle feature di input (a), utilizzo di un kernel per la trasformazione in uno spazio a più dimensioni (b), determinazione dell'iperpiano ottimo (c), trasformazione inversa per ritornare ad uno spazio con un numero di dimensioni pari a quelle delle feature di input (d)

SVM prevede inoltre la determinazione a priori di alcuni parametri, che in questo caso sono il *kernel*, il *parametro di regolarizzazione* e *gamma*. Per quanto riguarda il *kernel*, questo stabilisce il tipo di trasformazione che lo spazio iniziale subisce: si distinguono ad esempio kernel lineari, polinomiali, gaussiani ed esponenziali. Il *parametro di regolarizzazione* C , invece, esprime quanto è importante che ogni punto venga classificato in modo corretto: per valori grandi,

l'algoritmo sceglierà un iperpiano con un margine minore, affinché tutti i punti vengano assegnati alla classe corretta; viceversa per valori bassi del parametro l'iperpiano avrà margine maggiore, ma alcuni punti saranno classificati in modo non corretto, come si può vedere in Figura 2.3. Il parametro *gamma*, infine,

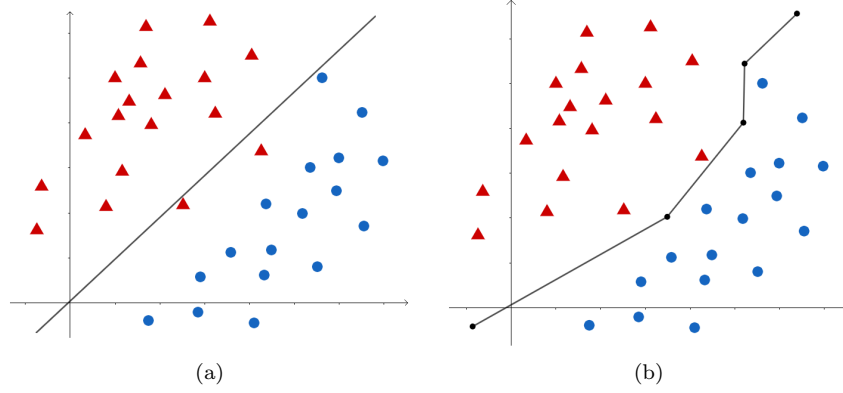


Figura 2.3: Sulla destra: valori bassi del parametro di regolarizzazione; sulla sinistra valori più elevati del parametro

indica quanto lontano dall'iperpiano arriva l'influenza del singolo punto, dove valori bassi del parametro indicano che punti lontani dal possibile iperpiano influenzano la scelta dell'iperpiano stesso, e viceversa.

2.3 Fuzzy Support Vector Machine

Le *Fuzzy Support Vector Machine* estendono il concetto di SVM, generalizzandolo e applicandolo alla logica fuzzy. Nelle FSVM, infatti, l'importanza di ogni elemento del training set alla determinazione della superficie separatrice è differente: a ogni elemento è infatti assegnato un grado di membership fuzzy, che denota il contributo fornito da quell'elemento all'apprendimento dell'iperpiano separatore. Se ad esempio un punto viene identificato come outlier, a esso verrà assegnato un valore di membership basso, in modo che il suo contributo all'errore complessivo sia poco rilevante. Supponiamo che i dati di training siano definiti da $S = \{(x_i, y_i, s_i) | i = 1, 2, \dots, l\}$, dove ogni $x_i \in \mathbb{R}^d$ è un elemento del training set, $y_i \in \{+1, -1\}$ è la label a esso associata e $s_i (i = 1, 2, \dots, l)$ è una membership fuzzy $r \leq s_i \leq 1$ (r è una costante dal valore molto basso ma maggiore di zero). Definiamo $Q = \{x_i | (x_i, y_i, s_i) \in S\}$; possiamo partizionare in Q due sottoinsiemi: la classe C^+ , ovvero la classe dei punti a cui è associata label $y_i = +1$, e quella Q^- , che conterrà gli x_i tali che $y_i = -1$. Il problema di

classificazione può quindi essere descritto come segue:

$$\begin{aligned} \min \frac{1}{2} \|w\|^2 + C \sum_{i=1}^l s_i \xi_i \\ \text{t.c. } y_i(w^T \cdot \Phi(x_i) + b) \geq 1 - \xi_i, i = 1, 2, \dots, l \\ \text{e } \xi_i \geq 0, \end{aligned}$$

dove C è il parametro di regolarizzazione (che definisce il costo della violazione del vincolo), ξ_i è una misura dell'errore nella FSVM e quindi il termine ξ_i è una misura pesata dell'errore [38, 39].

A partire dal metodo base delle FSVM si sono sviluppate varianti che mirano a ottimizzarlo e adattarsi a diverse situazioni: sono state introdotte ad esempio nuove funzioni di membership oppure algoritmi di ottimizzazione come in [40], che sfrutta l'algoritmo Firefly. Altri algoritmi si sono invece specializzati in problemi multiclasse [41], o ancora sono stati ottimizzati per dataset con rumore nei dati o outliers, come in [42, 43]. Una rassegna più esaustiva è mostrata in Tabella 2.2.

Tabella 2.2: Rassegna di metodi per fuzzy support vector machine

Nome metodo	Anno di pubblicazione	Articolo
FSVM	2011	[38]
FSVM con nuova funzione di membership	2013	[39]
FSVM using Firefly	2016	[40]
FSVM using APSO method	2014	[42]
FLST SVM	2016	[44]
CDFTSVM	2015	[45]
WCS FSVM	2013	[43]
FSVM for Multiclass Problem	2002	[41]

Consideriamo in particolare le Fuzzy Support Vector Machine per il *Class Imbalance Learning* (CIL [46]): questo algoritmo si specializza su apprendimento in caso di dataset nei quali una delle due classi è rappresentata con maggiore frequenza rispetto all'altra. In questo caso, con algoritmi tradizionali può accadere che l'apprendimento dia risultati molto accurati sulla predizione della classe maggiormente rappresentata, ma che commetta molti errori sull'altra classe (si consideri successivamente la classe positiva come classe rappresentata con minore frequenza, come nella maggior parte dei dataset reali).

Tra i metodi CIL, ovvero quelli che si occupano di questo tipo di problema, si possono distinguere metodi interni ed esterni: i *metodi esterni* effettuano un preprocessing dei dati in modo da bilanciarli, ad esempio eliminando in modo casuale alcuni elementi della classe più rappresentata finché un determinato

rapporto tra il numero degli elementi in ogni classe viene raggiunto; un altro metodo esterno consiste invece nella duplicazione degli esempi appartenenti alla classe meno rappresentata, sempre al fine di bilanciare meglio le classi tra loro. I *metodi interni*, invece, modificano l'algoritmo di apprendimento in modo da renderlo meno sensibile allo sbilanciamento tra classi.

L'algoritmo FSVM-CIL si pone l'obiettivo di risolvere sia il problema dello sbilanciamento tra classi sia quello della presenza di outlier e rumore nei dati. Come in ogni metodo di fuzzy SVM, i valori di membership (o *pesi*) assegnati agli esempi di training sono differenziati in modo da riflettere la loro diversa importanza. Possiamo quindi assegnare valori di membership più elevati in corrispondenza di elementi della classe meno rappresentata, e viceversa valori di membership più bassi a esempi della classe più rappresentata. Il valore di membership assegnato dipende inoltre dall'importanza intra-classe dei diversi elementi del training set, in modo che l'effetto del rumore e degli outlier sia minore.

Sia m_i^+ il valore della membership di un esempio x_i^+ appartenente alla classe positiva (la meno rappresentata), e m_i^- il valore di membership di un dato x_i^- nella classe negativa. Definiamo queste funzioni di membership come:

$$\begin{aligned} m_i^+ &= f(x_i^+)r^+, \\ m_i^- &= f(x_i^-)r^-, \end{aligned}$$

dove $f(x_i)$ genera un valore compreso tra 0 e 1, che riflette l'importanza di x_i nella sua classe di appartenenza, e r^+ e r^- riflettono lo sbilanciamento tra classi, tale che $r^+ > r^-$. Di conseguenza, un esempio nella classe positiva potrà avere un valore di membership $m_i^+ \in [0, r^+]$, mentre un esempio nella classe negativa avrà una membership $m_i^- \in [0, r^-]$. Per riflettere lo sbilanciamento tra classi, definiamo $r^+ = 1$ e $r^- = r$, dove r è il rapporto tra il numero di esempi nella classe positiva e il numero di quelli nella classe negativa. In questo modo, un elemento nella classe positiva potrà assumere un valore di membership compreso tra 0 e 1, mentre un esempio appartenente alla classe negativa potrà assumere valori di $m_i^- \in [0, r]$, dove $r < 1$. Fra i tre metodi espressi in [46] (funzione di membership basata sulla distanza dal centro della rispettiva classe, basata sulla distanza dall'iperpiano predetto, basato sulla distanza dall'iperpiano effettivo), nell'algoritmo che verrà analizzato nel Capitolo 3 si analizzerà è stato utilizzato quello che calcola la funzione di membership sulla base della distanza d_i^{sph} dall'iperpiano stimato. In questo caso, con d_i^{sph} si intende la distanza di x_i dal centro della "regione sferica", che può essere definita come un'ipersfera che copre la regione in cui si sovrappongono le due classi, per cui quindi l'iperpiano probabilmente passerà. Si possono utilizzare due differenti *funzioni di decadimento* per definire la funzione $f(x_i)$, una funzione lineare e una esponenziale:

$$\begin{aligned} f_{lin}^{sph}(x_i) &= 1 - \frac{d_i^{sph}}{\max(d_i^{sph}) + \Delta} \\ f_{exp}^{sph}(x_i) &= 1 - \frac{2}{1 + \exp(\beta d_i^{sph})}, \end{aligned}$$

dove $d_i^{sph} = \|x_i - \bar{x}\|^{\frac{1}{2}}$, e \bar{x} è il centro della regione sferica, stimato dal centro dell'intero dataset negativo e positivo; β e Δ sono due costanti definite a priori. Nel capitolo successivo si spiegherà in dettaglio il funzionamento di questo metodo e si mostreranno degli esperimenti eseguiti su vari dataset.

Capitolo 3

Implementazione ed esperimenti

3.1 Dataset

I dataset utilizzati durante il lavoro di tirocinio sono l'*Iris Dataset*¹ e il *Breast Cancer Dataset*².

Il dataset Iris è stato introdotto da Ronald Fisher nel 1936, e descrive 150 esemplari di Iris misurati da Edgar Anderson e classificati secondo tre specie: *Iris setosa*, *Iris virginica* e *Iris versicolor*. Ogni elemento del dataset è definito da quattro feature, la lunghezza e la larghezza del sepal e la lunghezza e la larghezza del petalo (misurate in centimetri) e da un'etichetta che indica la classe di appartenenza di quell'elemento; ogni classe è rappresentata da 50 istanze nel dataset. Nel dataset Iris la classe *iris setosa* è linearmente separabile dalle altre, che non lo sono invece tra loro. Si può osservare un estratto del dataset in Tabella 3.1.

Tabella 3.1: Estratto del dataset Iris

species	sepal_length	sepal_width	petal_length	petal_width
iris-setosa	5.1	3.5	1.4	0.2
iris-setosa	4.9	3.0	1.4	0.2
iris-versicolor	6.4	3.2	4.5	1.5
iris-versicolor	6.9	3.1	4.9	1.5
iris-virginica	6.5	3.0	5.2	2.0
iris-virginica	6.2	3.4	5.4	2.3

¹<https://archive.ics.uci.edu/ml/datasets/iris>

²<https://github.com/datasets/breast-cancer>

Il dataset Breast Cancer, invece, è stato creato da M.Zwitter e M.Soklic dell'Institute of Oncology University Medical Center Ljubljana, Yugoslavia, sulla base di dati raccolti da pazienti reali. In questo dataset si contano 286 istanze e 10 feature per ogni elemento; sono definite due classi che definiscono se la paziente ha avuto una recidiva (*recurrence-events*) oppure no (*no-recurrence-events*). I 10 attributi definiscono invece:

- la fascia d'età della paziente,
- *menopause*, che indica se la paziente sia o no in menopausa al momento della diagnosi,
- il diametro maggiore (in mm) del tumore,
- il numero di linfonodi contenenti metastasi del tumore,
- la presenza o meno di metastasi a linfonodi,
- il grado di gravità del tumore,
- la collocazione del tumore e il quadrante interessato,
- l'utilizzo di radioterapia nella cura del tumore.

Nel dataset sono presenti dei valori mancanti; la distribuzione delle istanze sulle classi è molto differente da quella del dataset Iris: si contano 201 istanze di *no-recurrence events* e 85 istanze di *recurrence-events*. Un estratto del dataset è mostrato in Tabella 3.2. Si è scelto di utilizzare quest'altro dataset per osservare il comportamento degli algoritmi considerati su un dataset di dimensioni maggiori e caratterizzato da un maggior numero di feature.

Tabella 3.2: Estratto del dataset Breast Cancer (per brevità sono stati tralasciati alcuni attributi)

age	menopause	tumor-size	deg-malig	breast	class
40-49	premeno	15-19	3	right	recurrence-events
50-59	ge40	15-19	1	right	no-recurrence-events
50-59	ge40	35-39	2	left	recurrence-events
40-49	premeno	35-39	3	right	no-recurrence-events
40-49	premeno	30-34	2	left	recurrence-events

Prima di essere utilizzati, entrambi i dataset hanno subito una fase di preprocessing: le feature descritte da valori categorici sono infatti state sostituite da feature a valori numerici (con l'utilizzo di appositi metodi di *sklearn*), così come le classi espresse dalle etichette associate ai dati; nel caso del dataset Breast Cancer le feature espresse da un range di valori sono state sostituite con il punto medio dell'intervallo considerato. Sono state inoltre eliminate le istanze del dataset contenenti valori mancanti.

Il dataset Iris, invece, è stato inizialmente considerato come composto da due classi: una classe caratterizzata da label 1 (rappresentante di volta in volta

ognuna delle tre classi possibili) e una associata alla label 0, contenente tutti gli elementi delle altre due classi non considerate. In questo modo si sono ottenuti tre diversi dataset a partire da *iris*, ognuno contenente due sole classi: considerando ad esempio la classe *iris-setosa*, uno dei tre dataset avrà come istanze positive quelle appartenenti a questa classe e come istanze negative tutte le altre (ovvero quelle appartenenti a *versicolor* e *virginica*). L'etichetta che rappresenta la classe di ogni elemento è stata inoltre posizionata come prima colonna del dataset, in modo da uniformare il modo in cui i vari dataset vengono presentati ai vari algoritmi considerati.

Nel repository pubblico annesso al lavoro di tirocinio³ si possono trovare:

- una cartella *Dataset*, contenente i dataset utilizzati nelle varie versioni dovute alle modifiche effettuate su questi al fine del trattamento,
- una cartella per ogni algoritmo analizzato, contenente il codice dell'algoritmo stesso ed eventuali funzioni utili,
- gli script *Preprocessing.py*, *iris_generator.py* e *iris_generator.py*, utilizzati per il preprocessing dei dati nei dataset,
- vari file per la visualizzazione dei risultati, uno per ogni metodo considerato,
- un file di logging, che contiene tutti gli esperimenti svolti,
- il file *final_script.py*, utilizzato per far eseguire automaticamente più algoritmi passandogli in input uno stesso dataset.

3.2 Algoritmo basato su Fuzzy Clustering

Nel lavoro di tirocinio è stato analizzato l'algoritmo *Fuzzy-C-Means*, da codice reperito in rete⁴. Il codice considerato presentava numerose criticità, legate sia a un'eccessiva specificità del codice sui dati considerati come input sia all'assenza di una classe che racchiudesse le varie funzioni. La fase preliminare di modifica del codice, quindi, è consistita nella realizzazione di un'adattatore, ovvero l'aggiunta di una parte di codice che permettesse di passare in input a tutti i metodi considerati dataset nello stesso formato (si sono considerati file *.csv* in cui l'etichetta fosse posta nella prima colonna del dataset). È stato poi necessario creare una classe (disponibile in *FCM.py*) che contenesse tutte le funzioni per il trattamento dell'input, per la fase di training e test, e tutte le funzioni utili in queste fasi; è stato anche necessario creare un costruttore che inizializzasse i parametri definiti a priori, cioè il parametro di arresto ϵ e un massimo globale, e introdurre degli attributi di classe che permettessero di salvare un modello addestrato per poi usarlo per effettuare predizioni. Si è reso inoltre necessario

³<https://github.com/ritafolisi/Tirocinio>, sotto licenza Creative Commons

⁴<https://github.com/AnanyaKirti/Fuzzy-C.git>

suddividere la funzione responsabile del training e test sul dataset in due diverse funzioni che svolgessero in modo isolato i due compiti, per poter utilizzare le due funzioni in modo distinto nella cross-validation.

Come metriche per valutare la performance si sono considerati l'accuratezza e l'RMSE sulle membership; per ottenere quest'ultimo è stata necessaria un'ulteriore modifica del codice iniziale, che invece prevedeva il calcolo dell'accuratezza sulle predizioni e inglobava in una funzione il passaggio dalle membership predette a una classificazione netta.

Per aumentare l'accuratezza del modello è stata poi introdotta nello script responsabile del funzionamento del FCM (*fuzzy_c.py*) la cross-validation (su 5 fold, con seme del generatore pseudocasuale impostato a 1): a ogni iterazione sono stati memorizzati l'accuratezza e l'RMSE sulle membership; come risultato finale si è ottenuta una media dei dati raccolti per ognuna delle due metriche. L'algoritmo analizzato utilizza nella fase di training dati di input senza etichetta: la label viene utilizzata solamente per controllare la bontà della predizione effettuata; ciò può talvolta influire sulle prestazioni dell'algoritmo, come approfondito nel Paragrafo 3.4.

Per controllare il buon funzionamento del codice scritto sono stati svolti unit test, che controllano l'ammissibilità dei valori restituiti per le metriche scelte, oltre al fatto che l'errore non superi una soglia prefissata. Il logging di tutte le operazioni ha permesso un controllo sui vari output degli esperimenti; gli esperimenti sono stati svolti più volte, utilizzando due o quattro feature (per quanto riguarda iris), i vari dataset a due classi generati a partire da Iris e l'intero dataset Breast Cancer. Per una migliore visualizzazione dei risultati, è stata realizzata una rappresentazione grafica con heatmap (Figura 3.1) e curve di livello, come in Figura 3.2.

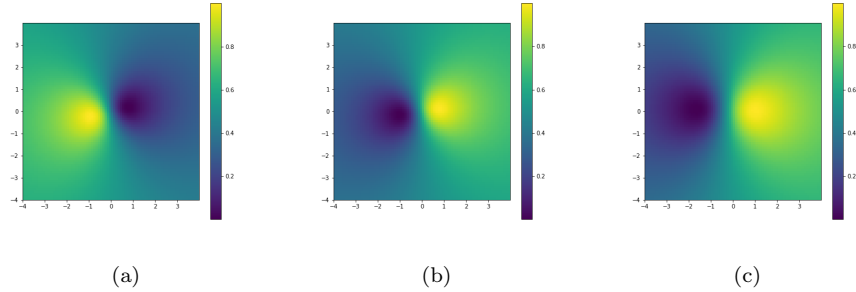


Figura 3.1: Heatmap corrispondenti ai tre dataset che considerano di volta in volta soltanto l'appartenenza o meno a una data classe (setosa, versicolor, virginica)

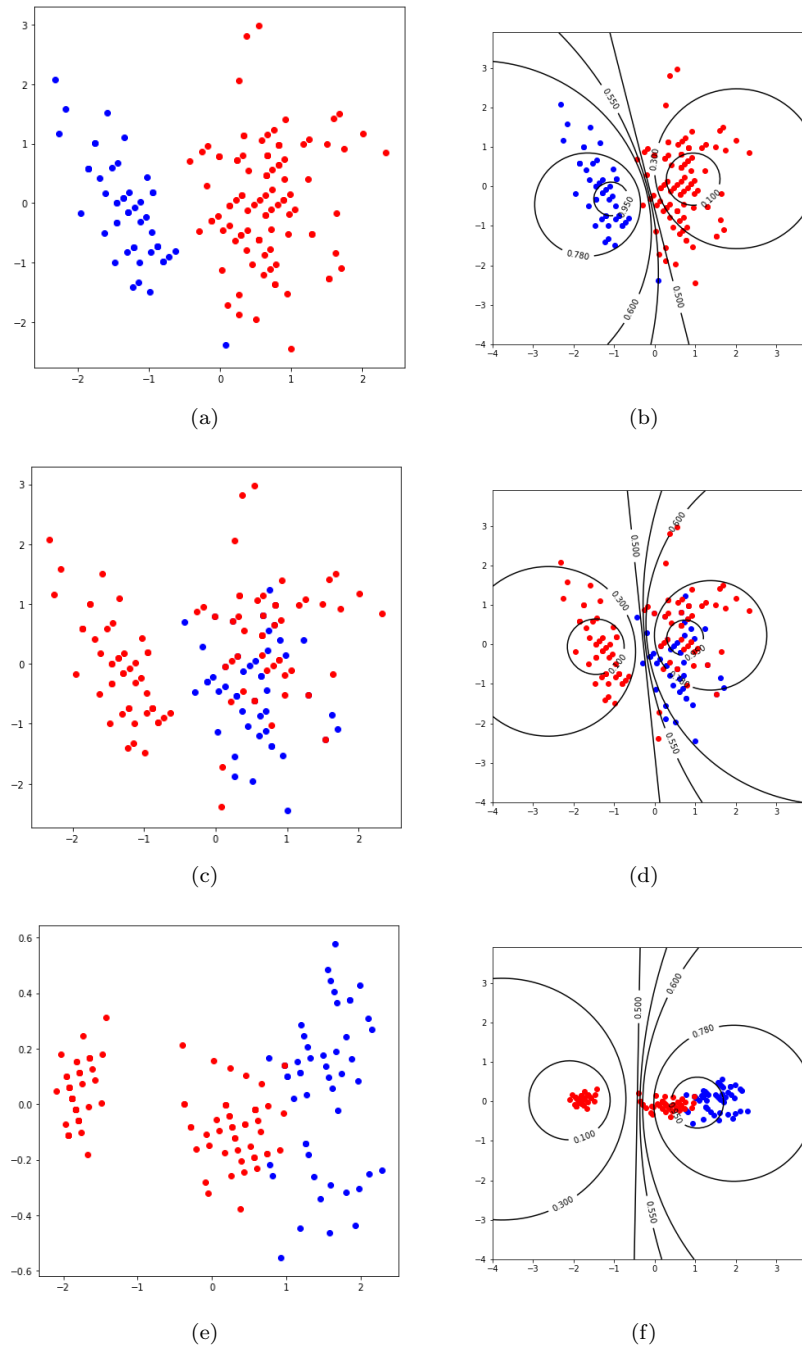


Figura 3.2: Il dataset iris (si considerano le prime due features), modificato in modo da avere due classi che indichino di volta in volta se un'istanza appartiene a setosa oppure no (a), versicolor oppure no (c), virginica oppure no (e), e le curve di livello risultanti dalle predizioni effettuate dall'algoritmo, a cui vengono passati i tre dataset (b, d, f)

3.3 Algoritmo basato su Fuzzy Support Vector Machine

Tra gli algoritmi basati su Fuzzy Support Vector Machine è stato analizzato il Fuzzy SVM per Class Imbalance Learning; il codice è stato reperito in rete⁵. L'algoritmo, implementato con l'uso di *Jupyter Notebook*, non permetteva un'esecuzione automatizzata: questo primo problema è stato gestito con l'introduzione del file *HYP_SVM*, contenente la classe in cui le funzioni necessarie all'algoritmo sono definite e da uno script, necessario per eseguire l'algoritmo stesso. Anche nel caso del codice trovato per SVM-CIL, inoltre, si è notata un'eccessiva specializzazione sui dati che l'algoritmo era progettato per utilizzare: nel codice si sono identificati valori costanti il cui significato non è reso esplicito, così come nomi di variabili non autoesplicativi; è stata quindi necessaria una modifica al fine di rendere il codice utilizzabile con qualsiasi dataset e di migliorarne la leggibilità. Come per il metodo descritto nel Paragrafo 3.2, si è reso inoltre necessario utilizzare una cross-validation, al fine di valutare in modo appropriato i risultati dell'algoritmo. Un'altra fase cruciale è stata quella della selezione degli iperparametri C e σ , svolta utilizzando il metodo *GridSearchCV* di *sklearn*. Per C la selezione avviene tra i valori $[1^{-2}, 1^{-1}, 1, 1^{+1}, 1^{+2}, 1^{+3}, 1^{+4}]$, mentre per σ (si è utilizzato un kernel gaussiano) in $[9^{-2}, 9^{-1}, 9, 9^{+1}, 9^{+2}, 9^{+3}, 9^{+4}]$. Per poter utilizzare *GridSearchCV*, è stato necessario introdurre nuovi metodi richiesti esplicitamente da *sklearn* (cioè *score*, *get_params* e *set_params*). La fase di model selection costituisce quindi una cross-validation annidata in un'altra più esterna, che invece suddivide training e test del modello.

L'algoritmo presenta un tempo di esecuzione notevole, dovuto solo in parte alla fase di model selection; sono stati fatti esperimenti sui tre dataset ottenuti da Iris e sul dataset Breast Cancer. In Tabella 3.3 si possono osservare i tempi di esecuzione dei vari algoritmi sui vari dataset.

Tabella 3.3: Tempi di esecuzione degli algoritmi FCM e FSVM-CIL sui dataset considerati

Dataset	FCM	FSVM-CIL
Iris-setosa	meno di 1s	circa 5 min
Iris-versicolor	meno di 1s	circa 5 min
Iris-virginica	meno di 1s	circa 5 min
Breast Cancer Dataset	12s	circa 50 min

3.4 Risultati

Di seguito si indicano i risultati degli esperimenti eseguiti su FCM (Tabella 3.4) e FSVM-CIL (Tabella 3.5). In seguito in Tabella 3.6 e Tabella 3.7 sono illustra-

⁵<https://github.com/adizz2407/Fuzzy-SVM>

ti gli stessi esperimenti eseguiti su Reti Neurali e Fuzzy k-Nearest Neighbour (approfonditi in [8]).

Dai risultati ottenuti dagli esperimenti si nota come FCM ottiene buoni risultati sul dataset *iris-setosa*, in quanto la classe è linearmente separabile dalle altre, mentre la performance peggiora sugli altri due dataset estratti da Iris.

L'algoritmo FSVM-CIL ottiene una performance discreta su tutti i dataset, e l'errore rilevato aumenta molto sui dataset *Iris-versicolor* e *Breast Cancer*. Il valore dell'RMSE non è indicato in quanto l'implementazione trovata si limita a indurre le membership e le usa per la classificazione, ma non permette di calcolarle in modo esplicito punto per punto.

Tabella 3.4: Risultati degli esperimenti eseguiti su FCM

Dataset	Accuratezza	RMSE
Iris-setosa	98.0%	0.12
Iris-versicolor	64.7%	0.55
Iris-virginica	68.7%	0.69
Breast Cancer Dataset	56.2%	0.54

Tabella 3.5: Risultati degli esperimenti eseguiti su FSVM-CIL

Dataset	Accuratezza	RMSE
Iris-setosa	80%	-
Iris-versicolor	66%	-
Iris-virginica	84%	-
Breast Cancer Dataset	53.9%	-

Tabella 3.6: Risultati degli esperimenti eseguiti su FKNN

Dataset	Accuratezza	RMSE
Iris-setosa	99%	0.08
Iris-versicolor	76%	0.44
Iris-virginica	75%	0.44
Breast Cancer Dataset	62%	4.6

Tabella 3.7: Risultati degli esperimenti eseguiti su Fuzzy min max

Dataset	Accuratezza	RMSE
Iris-setosa	99%	0.64
Iris-versicolor	76%	0.63
Iris-virginica	78%	0.57
Breast Cancer Dataset	71%	0.84

Conclusioni

Il lavoro di tirocinio ha riguardato l'approfondimento e il confronto di algoritmi per l'induzione della funzione di membership a insiemi fuzzy. Si sono analizzate nello specifico un'implementazione di Fuzzy C-Means e di Fuzzy Support Vector Machines for Class Imbalance Learning. Le implementazioni in questione sono state modificate al fine di migliorare le prestazioni degli algoritmi e la leggibilità del codice. Gli esperimenti hanno previsto una fase di cross-validation e di model selection, e sono stati eseguiti sui dataset Iris e Breast Cancer; a partire da Iris sono stati generati tre dataset contenenti ciascuno due classi, da usare per indurre la funzione di membership della classe di volta in volta considerata. I risultati ottenuti sono stati molto buoni sulla classe *iris-setosa* (linearmente separabile dalle altre), mentre la performance è sensibilmente diminuita sugli altri dataset. L'algoritmo FSVM-CIL ha mostrato performance generalmente più basse, e un tempo di esecuzione notevole.

Possibili estensioni di questo lavoro potrebbero essere la ricerca e implementazione di nuovi metodi (alcuni sono già citati in Tabella 2.1 e in Tabella 2.2) o l'uso di dataset di dimensioni maggiori, che permetterebbero di osservare comportamenti differenti degli algoritmi.

Bibliografia

- [1] Nils J. Nilsson. *Introduction to Machine Learning: an early draft of a proposed textbook*. Robotics Laboratory, Department of Computer Science, Stanford, CA 94305, November 3 1998.
- [2] T. G. Dietterich and E. B. Kong. Machine learning bias, statistical bias, and statistical variance of decision tree algorithms. Technical report, Department of Computer Science, Oregon State University, 1995.
- [3] M. Mohri, A. Rostamizadeh, and A. Talwalkar. *Foundation of Machine Learning*. The MIT Press, 2012.
- [4] J. A. Goguen. L. a. zadeh. fuzzy sets. information and control, vol. 8 (1965), pp. 338–353. - l. a. zadeh. similarity relations and fuzzy orderings. information sciences, vol. 3 (1971), pp. 177–200. *Journal of Symbolic Logic*, 38(4):656–657, 1973.
- [5] Guanrong Chen and Trung Tat Pham. *Introduction to Fuzzy Sets, Fuzzy Logic and Fuzzy Control Systems*. CRC Press, 2001.
- [6] Tzung-Pei Hong and Shyue-Liang Wang. *Determining Appropriate Membership Functions to Simplify Fuzzy Induction*. 2020.
- [7] D.Dubois and H.Prade. The three semantics of fuzzy sets. *Fuzzy Sets and Systems 90*, pages 141–150, 1997.
- [8] Rita Folisi. Confronto tra approcci basati su nearest neighbour e reti neurali nell’induzione di insiemi fuzzy, 2020.
- [9] S. Medasani, J. Kim, and R. Krishnapuram. An overview of membership function generation techniques for pattern recognition. *Int. J. Approx. Reason.*, 19:391–417, 1998.
- [10] S. Safari and A. R. Erfani. A new method for fuzzification of nested dummy variables by fuzzy clustering membership functions and its application in financial economy. *Iranian Journal of Fuzzy Systems*, 17(1):13–27, 2020.

- [11] Einollah Pasha and Adel Fatemi. Intuitionistic fuzzy sets clustering (ifsc) with an application in psychology. *Journal of Mathematics and Applications*, 28, 01 2006.
- [12] Yamid Hernández Julio, Helmer Muñoz Hernández, Wilson Bernal, Javier Guzmán, Romel González Díaz, and Patrícia Ferraz. Fuzzy knowledge discovery and decision-making through clustering and dynamic tables. application to medicine. 02 2019.
- [13] Thomas Villmann and Conny Albani. Clustering of categoric data in medicine — application of evolutionary algorithms. volume 2206, pages 619–627, 09 2001.
- [14] Doris Damian, Matej Oresic, Elwin Verheij, Jacqueline Meulman, Jerome Friedman, Aram Adourian, Nicole Morel, Age Smilde, and Jan van der Greef. Applications of a new subspace clustering algorithm (cosa) in medical systems biology. *Metabolomics*, 3:69–77, 03 2007.
- [15] Mehmet Gönen and Adam A Margolin. Localized data fusion for kernel k-means clustering with application to cancer biology. In Z. Ghahramani, M. Welling, C. Cortes, N. D. Lawrence, and K. Q. Weinberger, editors, *Advances in Neural Information Processing Systems 27*, pages 1305–1313. Curran Associates, Inc., 2014.
- [16] Rimbun Siringoringo and Jamaluddin Jamaluddin. Initializing the fuzzy c-means cluster center with particle swarm optimization for sentiment clustering. *Journal of Physics: Conference Series*, 1361, 11 2019.
- [17] J. Bezdek, R. Ehrlich, and W. E. Full. Fcm: The fuzzy c-means clustering algorithm. *Computers e Geosciences*, 10:191–203, 1984.
- [18] Jinglin Xu, J. Han, Kai Xiong, and F. Nie. Robust and sparse fuzzy k-means clustering. In *IJCAI*, 2016.
- [19] Lin Zhu, Fu-Lai Chung, and Shitong Wang. Generalized fuzzy c-means clustering algorithm with improved fuzzy partitions. *Trans. Sys. Man Cyber. Part B*, 39(3):578–591, June 2009.
- [20] Mark Junjie Li, Michael Ng, Yiu-ming Cheung, and Joshua Huang. Agglomerative fuzzy k-means clustering algorithm with selection of number of clusters. *Knowledge and Data Engineering, IEEE Transactions on*, 20:1519 – 1534, 12 2008.
- [21] Dao-Qiang Zhang and Song-Can Chen. Kernel-based fuzzy and possibilistic c-means clustering. 2003.
- [22] Asa Ben-Hur, David Horn, Hava T. Siegelmann, and Vladimir Vapnik. Support vector clustering. *J. Mach. Learn. Res.*, 2:125–137, March 2002.
- [23] Dario Malchiodi and Witold Pedrycz. Learning membership functions for fuzzy sets through modified support vector clustering”, editor=”masulli,

- francesco and pasi, gabriella and yager, ronald. In *Fuzzy Logic and Applications. 10th International Workshop, WILF 2013, Genoa, Italy, November 19–22, 2013. Proceedings.*, page 52–59. Springer International Publishing, Switzerland, series="Lecture Notes on Artificial Intelligence, 2013.
- [24] S.S. Jamsandekar and R.R. Mudholkar. Fuzzy classification system by self generated membership function using clustering technique. *BIJIT - BVICAM's International Journal of Information Technology*, 2013.
 - [25] Jeetashree Aparajeeta, Pradipta Kumar Nanda, and Niva Das. Modified possibilistic fuzzy c-means algorithms for segmentation of magnetic resonance image. *Appl. Soft Comput.*, 41(C):104–119, April 2016.
 - [26] Jinglin Xu, Junwei Han, Feiping Nie, and Xuelong Li. Robust and efficient fuzzy c-means clustering constrained on flexible sparsity, 08 2019.
 - [27] S.K.Adhikari, J.K.Sing, D.K.Basub, and M.Nasipuri. Conditional spatial fuzzy c-means clustering algorithm for segmentation of mri images. *Applied Soft Computing*, 34:758–769, September 2015.
 - [28] X. Bai, Yuxuan Zhang, H. Liu, and Yingfan Wang. Intuitionistic center-free fcm clustering for mr brain image segmentation. *IEEE Journal of Biomedical and Health Informatics*, 23:2039–2051, 2019.
 - [29] Prabhjot Kaur, I. Lamba, and Anjana Gosain. Dofcm: A robust clustering technique based upon density. *International Journal of Engineering and Technology*, 3:297–303, 01 2011.
 - [30] Dario Malchiodi and Anna Maria Zanaboni. Data-driven induction of shadowed sets based on grade of fuzziness. In Robert Fullér, Silvio Giove, and Francesco Masulli, editors, *Fuzzy Logic and Applications — 12th International Workshop, WILF 2018 Genoa, Italy, September 6–7, 2018 — Revised Selected Papers*, Lecture Notes in Artificial Intelligence 11291, page 17–28, Cham, 2019. Springer Nature Switzerland AG.
 - [31] Muhan Guo, Rui Zhang, Feiping Nie, and Xuelong Li. Embedding fuzzy k-means with nonnegative spectral clustering via incorporating side information. *CIKM '18*, page 1567–1570, New York, NY, USA, 2018. Association for Computing Machinery.
 - [32] Rui Zhang, Feiping Nie, and Xuelong Li. Self-weighted spectral clustering with parameter-free constraint. *Neurocomputing*, 241, 02 2017.
 - [33] R. Krishnapuram and J. Keller. A possibilistic approach to clustering. *IEEE Trans. Fuzzy Syst.*, 1:98–110, 1993.
 - [34] İsmail Babaoğlu, Oğuz Fındık, and Mehmet Bayrak. Effects of principle component analysis on assessment of coronary artery diseases using support vector machine. *Expert Systems with Applications*, 37:2182–2185, 03 2010.
 - [35] Yongmin Li, Shaogang Gong, and Jamie Sherrah. Support vector machine based multi-view face detection and recognition yongmin li. 06 2004.

- [36] İsmail Babaoğlu, Oğuz Fındık, and Mehmet Bayrak. Effects of principle component analysis on assessment of coronary artery diseases using support vector machine. *Expert Systems with Applications*, 37:2182–2185, 03 2010.
- [37] Babak Asl, Kamal Setarehdan, and Maryam Mohebbi. Support vector machine-based arrhythmia classification using reduced features of heart rate variability signal. *Artificial intelligence in medicine*, 44:51–64, 07 2008.
- [38] Wan Tang. Fuzzy svm with a new fuzzy membership function to solve the two-class problems. *Neural Processing Letters*, 34:209–219, 12 2011.
- [39] Xiufeng Jiang, Zhang Yi, and Jian Lv. Fuzzy svm with a new fuzzy membership function. *Neural Computing and Applications*, 15:268–276, 06 2006.
- [40] O. Almasi and M. Rouhani. A new fuzzy membership assignment and model selection approach based on dynamic class centers for fuzzy svm family using the firefly algorithm. *Turkish Journal of Electrical Engineering and Computer Sciences*, 24:1797–1814, 2016.
- [41] Shigeo Abe and Takuya Inoue. Fuzzy support vector machines for multiclass problems. pages 113–118, 01 2002.
- [42] Omid Almasi, Hamed Gooqeri, Behnam Asl, and Wan Tang. A new fuzzy membership assignment approach for fuzzy svm based on adaptive pso in classification problems. *Journal of Mathematics and Computer Science*, 14:171–182, 02 2015.
- [43] Wenjuan An and Mangui Liang. Fuzzy support vector machine based on within-class scatter for classification problems with outliers or noises. *Neurocomputing*, 110:101–110, 06 2013.
- [44] Javad Salimi Sartakhti, Homayun Afrabandpey, and Nasser Ghadiri. Fuzzy least squares twin support vector machines, 2015.
- [45] Bin-Bin Gao, Jianjun Wang, Yao Wang, and Chan-Yun Yang. Coordinate descent fuzzy twin support vector machine for classification. pages 7–12, 12 2015.
- [46] Rukshan Batuwita and Vasile Palade. Fsvm-cil: Fuzzy support vector machines for class imbalance learning. *IEEE Transactions on Fuzzy Systems*, 18(3):558–571, 6 2010.