UNIVERSITÀ DEGLI STUDI DI MILANO

FACOLTÀ DI SCIENZE E TECNOLOGIE

Corso di Laurea in Informatica

Apprendimento di insiemi fuzzy nell'ambito del web semantico

Relatore: Prof. Dario Malchiodi

Correlatore: Prof.ssa Anna Maria Zanaboni

Tesi di Laurea di: Alessia Cecere
Matr. 923563

Anno Accademico 2020/2021

Indice

Introduzione

Capitolo 1

Apprendimento di insiemi fuzzy

1.1 Gli insiemi fuzzy

Una collezione di elementi rappresenta un insieme se esiste un criterio oggettivo che permette di definire univocamente se un qualunque elemento appartiene o meno al raggruppamento.[1] Immaginiamo "l'insieme dei laureati": è possibile definire, preso un elemento dell'universo del discorso, se è vero o falso che questo ne fa parte, tramite il criterio oggettivo di aver conseguito la laurea. O ancora, consideriamo "l'insieme di persone con la febbre": come stabiliamo se un elemento vi appartiene? In questi casi occorre identificare una soglia: se la temperatura corporea di una persona supera i 37 gradi, allora diciamo che questa ha la febbre, ovvero appartiene all'insieme.

L'esperienza quotidiana del mondo racconta, tuttavia, una realtà più composita: non a tutti i concetti è possibile applicare in maniera netta una classificazione. Pensiamo a una "classe degli uomini alti"[18]: quale soglia di appartenenza si potrebbe fissare? 1.80, 1.85 metri? L'operazione appare molto più complessa e, già a primo impatto, di scarso significato. Questo perché "uomo alto" è un concetto molto più qualitativo dei precedenti, per il quale non ha particolare attinenza dare una risposta binaria: ha molto più senso interrogarsi su *in che misura* l'elemento appartenga all'insieme. Potremmo così intuitivamente dire che un uomo di 1.70 metri appartiene meno all'insieme di uno alto 1.80.

Gli insiemi fuzzy (o sfocati, nella terminologia italiana) sono una variante degli insiemi classici introdotta da Lotfi Zadeh del 1965, che li definisce "una classe di oggetti con un continuo di gradi di appartenenza"[18]. Dato

[1] Si tratta, in realtà, di una definizione informale, introdotta per brevità di esposizione. Una definizione rigorosa richiederebbe una trattazione molto più lunga, che va al di fuori degli scopi di questo elaborato.

un insieme classico e un oggetto appartenente a un universo del discorso X, è possibile stabilire una funzione $f : X \mapsto \{0, 1\}$, o funzione di appartenenza: questa restituisce 1 se l'elemento appartiene all'insieme, 0 altrimenti. Nel caso di un insieme fuzzy, la funzione di appartenenza si presenta nella generalizzazione $f : X \mapsto [0, 1]$: invece di un valore booleano, restituisce un valore nell'intervallo continuo tra 0 e 1, che sta a indicare il grado di appartenenza all'insieme; più il valore risulterà vicino a 1 più il grado di appartenenza sarà elevato, e viceversa. Questa generalizzazione rende molto più facile esprimere concetti che, come nel caso degli uomini alti descritto precedentemente, evadono la classificazione binaria a causa del loro carattere vago e qualitativo: proprio per questo parliamo di insiemi sfocati. Grazie alla formalizzazione matematica di questi concetti, è possibile aggiungere un potere descrittivo enorme. Non solo, gli insiemi fuzzy rappresentano uno strumento utile in contesti di informazione incerta: un elemento può appartenere a più insiemi che esprimono caratteristiche opposte (ad esempio "l'insieme delle persone vecchie" e "l'insieme delle persone giovani"[6]) con gradi differenti mantenendo il sistema coerente, proprio come avverrebbe in un contesto reale.

1.2 Il Web semantico

Nel 2001, Tim Berners Lee definiva il Web semantico come "un'estensione del web attuale, nella quale all'informazione viene dato un significato ben definito, permettendo a computer e utenti di cooperare meglio"[17]. Da quel momento, il termine è stato utilizzato per riferirsi all'idea di un web nel quale applicazioni intelligenti fossero in grado di comprendere il senso di un testo, guidando l'utente verso l'informazione cercata o addirittura sostituendosi ad esso nel processo. Condizione necessaria perché ciò avvenga è essere in grado di rappresentare l'informazione in un modo strutturato, che permetta ai computer di fare inferenze in modo automatico. Per questo motivo, ai dati in senso stretto devono essere associati dei metadati, ovvero descrizioni del contenuto che riescano a riportare l'enorme varietà dei concetti ad uno schema ben definito; ciò avviene a diversi livelli, che sono oggetto di studio dell'ingegneria della conoscenza. In Figura 1.1 vi è una schematizzazione di questi livelli. Alla base troviamo il concetto di URI (Uniform resource identifier), ovvero un insieme di identificatori unificato: nel momento in cui si scrive un testo su un argomento, infatti, è necessario identificarlo in maniera univoca. A salire, troviamo la tecnologia XML (eXtensible Markup Language): si tratta di un linguaggio di markup che permette di creare tag personalizzati per descrivere una risorsa; si è così in grado di conferire una struttura ar-

bitraria ai propri dati, pur senza dare informazioni su cosa questa struttura stia effettivamente a significare. Il significato è demandato a RDF, standard proposto dal W3C; si tratta di un set di linguaggi basato su sintassi XML, principalmente utilizzato per fare delle asserzioni descrittive di un contenuto. RDF utilizza delle triple soggetto-verbo-oggetto, ad esempio "Apprendimento di insiemi fuzzy nell'ambito del web semantico" (soggetto) ha come autore (verbo) Alessia Cecere (oggetto): questa particolare struttura risulta essere adatta a descrivere la maggior parte dei dati che devono essere elaborati da macchine. I tre componenti di una tripla sono spesso rappresentati da URI, in modo da rappresentare elementi specifici ed evitare l'ambiguità tra termini che è tipica del linguaggio naturale. Il passo successivo è dato dalla ontologie o OWL (Web ontology language), che permettono di superare la specificità dei riferimenti (ad esempio diversi modi di esprimere lo stesso concetto), tramite una tassonomia e un insieme di regole di inferenza. La tassonomia descrive classi di oggetti e le relazioni che queste classi hanno tra loro (ad esempio, "sedia è una sottoclasse di mobile"), mentre le regole di inferenza permettono di stabilire vincoli ulteriori relativi alla realtà di riferimento: ad esempio, se un prefisso è associato a una città, e un indirizzo utilizza quel prefisso, allora anche l'indirizzo ha associato il prefisso di quella città [17]. L'utilizzo delle ontologie permette di salire al livello successivo, ovvero recuperare documenti esprimendo query complesse, che tengano conto non solo degli oggetti del discorso coinvolti ma anche del legame tra essi.

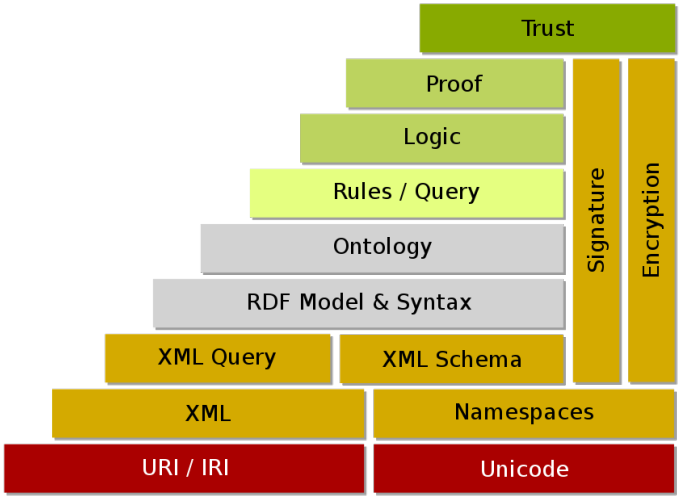

Figura 1.1: Livelli del web semantico

Si tratta del primo passo verso lo strato verde in Figura 1.1, ovvero il livel-

lo logico, non ancora sviluppato e oggetto di speculazione teorica. A questo livello l'obiettivo è realizzare sistemi composti da formulazioni di concetti logici e rendere i computer in grado di ragionare per inferenza su tali principi, attraverso un insieme di processori di informazioni. Naturalmente, in questo contesto dovrà essere garantito un grado di fiducia nelle informazioni, attraverso l'uso trasversale di crittografia e firma digitale: questa attesterà che un determinato utente ritiene veritiero un documento o un'istruzione, dando alle macchine dei parametri per decidere in cosa e quanto credere. Lo sviluppo di questi concetti porterà nel suo complesso alla realizzazione *Web of trust*, apice della gerarchia.

1.3 Applicazione alla ricerca di assiomi in un insieme di formule

L'ontology learning è un ambito di ricerca che punta alla generazione automatica di ontologie, a partire da un testo o un'ontologia preesistente (tipicamente espressa in OWL), insieme alla sua istanza (espressa in RDF). Sono stati studiati diversi metodi per ricavare dei pattern da questi input, e tutti si basano fortemente sul calcolo dello score di candidati assiomi, ovvero sull'identificazione del valore di verità delle formule di un insieme considerato, allo scopo di ricavarne degli assiomi. Negli ultimi anni sono stati proposti diversi approcci alla risoluzione del problema, che utilizzano principalmente euristiche fondate su inferenza statistica. Un'euristica basata sulla teoria della possibilità e sulla falsificazione nel contesto della semantica del mondo aperto viene proposta in [9] e riassunta in [15]. Sintentizzando ulteriormente, immaginiamo di avere un candidato assioma ϕ che esprime una possibile relazione tra due elementi del dominio, e di dover valutare la sua credibilità sulla base di un dataset a disposizione Γ. L'assioma ϕ verrà definito come l'insieme finito di affermazioni ψ che ne sono conseguenza: perciò, se $\Gamma \models$ [2]ψ allora ψ è una conferma di ϕ, se $\Gamma \models \neg\psi$ allora ψ è un controesempio di ϕ, mentre se $\Gamma \not\models \psi$ e $\Gamma \not\models \neg\psi$ non si tratta né di un controesempio né di una conferma. Inseriamo nel quadro la teoria della possibilità, ovvero quella teoria matematica per cui a ogni evento si associa un grado di possibilità tra 0 (impossibile) e 1 (completamente possibile, normale)[8]: avendo a disposizione una misura distribuzione di possibilità π, induciamo una misura di possibilità Π per ogni evento. Indichiamo con u_ϕ la cardinalità degli ψ, con

[2]Il simbolo $\models$ esprime la relazione di conseguenza logica. In questo caso indica che, se tutte le proposizioni contenute nel dataset Γ sono vere, allora ne deve conseguire che anche ψ è vera.

u_ϕ^+ la cardinalità delle conferme di ϕ e con u_ϕ^- la cardinalità dei controesempi di ϕ; allora, se $u_\phi > 0$,

$$\Pi(\phi) = 1 - \sqrt{1 - \frac{u_\phi - u_\phi^-}{u_\phi}},$$

mentre se $u_\phi > 0$, allora $\Pi(\phi) = 1$. Il problema di questa tecnica è che, a meno di alcuni trucchi di implementazione, è impossibile da applicare nella pratica, a causa dell'eccessivo costo computazionale.

L'alternativa considerata per il lavoro che si andrà a documentare, esposta in [15], è quella di allenare un modello surrogato tramite un algoritmo di machine learning; l'allenamento viene fatto in modo induttivo a partire da un insieme di possibili assiomi e delle loro negazioni, etichettati con i rispettivi valori di possibilità precedentemente calcolati. L'obiettivo è rendere l'algoritmo in grado di ricavare valori di possibilità per nuovi candidati assiomi.

Il metodo è promettente perché il grande costo computazionale viene concentrato nella fase iniziale e molto mitigato in quelle successive: una volta calcolati i valori di possibilità per le formule con cui allenare l'algoritmo ed effettuato il training, l'inferenza per i nuovi candidati assiomi sarà pressoché immediata.

L'algoritmo utilizzato, descritto nel Paragrafo 2.1, si basa sull'inferenza di una funzione di appartenenza a un insieme fuzzy: l'insieme fuzzy è la classe degli assiomi, mentre il valore di possibilità con cui le formule sono inizialmente etichettate rappresenta il grado di appartenenza al suddetto insieme. Il problema risulta adatto a essere modellato in senso fuzzy perché l'appartenenza alla categoria degli assiomi non è booleana: l'obiettivo non è suddividere le formule in categorie binarie di vero o falso, ma assegnare a ognuna di esse un certo valore di possibilità, grado di confidenza nel contesto incerto della modellazione della conoscenza. In questo senso, la confidenza nel candidato assioma sarà tanto più grande quanto il grado di appartenenza all'insieme fuzzy degli assiomi.

Capitolo 2

Risoluzione del problema

2.1 L'algoritmo di apprendimento

Per affrontare il problema sopra descritto, si è utilizzata una variante dell'algoritmo di support vector clustering, descritta in [14].

Dati un campione $\{x_1, \ldots, x_n\}$ di elementi appartenenti a un dominio X e i rispettivi gradi di appartenenza $\{\mu_1, \ldots, \mu_n\}$ a un fuzzy set sconosciuto A, obiettivo dell'algoritmo è indurre un'approssimazione del fuzzy set A, inferendo la sua funzione di appartenenza μ_A. Per spiegare la tecnica, immaginiamo che esista una sfera di raggio R e centro a sconosciuti che contiene i punti con membership elevata, in modo che il loro grado di membership dipenda dalla distanza dal centro a. Un primo modo di formulare il problema sarebbe, ignorando i valori di membership, la seguente ottimizzazione vincolata:

$$\min R^2 + C \sum_i \xi_i$$

$$s.t.$$

$$||x_i - a||^2 \leq R^2 + \xi_i \; \forall i = 1, \ldots, n,$$

$$\xi_i \; \geq 0 \; \forall i = 1, \ldots, n.$$

Il problema diventa dunque trovare la sfera di raggio minimo, al variare di R^2 e a, che raccolga la maggior parte dei punti rappresentanti gli elementi del campione. Le ξ_i corrispondono a variabili di slack, che vengono aggiunte per rilassare i vincoli ove necessario, rendendoli ridondanti e permettendo che vengano rispettati, per alcune istanze, nonostante queste non si trovino all'interno della sfera. Per ridurre il numero di punti all'esterno della sfera, minimizziamo la somma delle ξ_i all'interno della funzione obiettivo, moltiplicandola per un valore C, che funge da bilanciamento tra il rilassamento

precedentemente descritto e il rispetto del vincolo, identificando se sia più importante mantenere vincoli stringenti o mappare tutti i punti nella sfera. In realtà, l'obiettivo è che la distanza dal centro della sfera rappresenti un grado di appartenenza all'insieme fuzzy, e dunque andiamo a inserire le μ_i nella formulazione, che diventa:

$$\min R^2 + C \sum_i (\xi_i + \tau_i)$$

$$s.t.$$

$$\mu_i ||x_i - a||^2 \leq \mu_i R^2 + \xi_i \; \forall i = 1, \ldots, n,$$

$$(1 - \mu_i) ||x_i - a||^2 \geq \mu_i R^2 - \tau_i \; \forall i = 1, \ldots, n,$$

$$\xi_i \; \geq 0, \tau_i \; \geq 0 \; \forall i = 1, \ldots, n.$$

La formulazione cattura in parte l'insieme fuzzy, infatti:

- Se $\mu_i = 1$, il secondo vincolo diventa ridondante, e si torna alla formulazione iniziale, nella quale si chiede di trovare il centro e il raggio della sfera più piccola che contiene tutti i punti con membership 1;
- Se $\mu_i = 0$, è il primo vincolo a diventare ridondante, e il vincolo modula la non appartenenza alla sfera;
- Se $\mu_i = \frac{1}{2}$, moltiplicando entrambi i vincoli per 2 si ottiene:

$$||x_i - a||^2 \leq R^2 + 2\xi_i \; \forall i = 1, \ldots, n,$$

$$||x_i - a||^2 \geq R^2 - 2\tau_i \; \forall i = 1, \ldots, n.$$

Dal momento che entrambe le variabili di slack devono essere il più possibile vicine e zero, questo sta a significare che per la membership $\frac{1}{2}$ i punti tendono a stare esattamente sulla superficie della sfera. Dunque, risolvendo il problema di ottimizzazione, troviamo una prima approssimazione dell'insieme fuzzy, in termini di una sfera che conterrà i punti con membership elevata ed escluderà quelli con membership bassa, la cui frontiera identificherà il luogo dei punti con incertezza massima (con membership $\frac{1}{2}$). Rimane il problema di modellazione delle membership intermedie, che si affronterà in seguito. Ottenuta quest'ultima formulazione del problema di ottimizzazione, si risolve il corrispondente problema duale, attraverso il metodo di Wolfe. Per farlo, costruiamo la funzione lagrangiana, sottraendovi obiettivo e vincoli, moltiplicati per altrettante variabili:

$$L = R^2 + C\sum_i(\xi_i + \tau_i) - \sum_i \alpha_i(\mu_i + R^2 + \xi_i - ||\xi_i - a||^2) -$$

$$\sum_i \beta_i[(1-\mu_i)||x_i - a||^2 - (1-\mu_i)R^2 + \tau_i] - \sum_i \gamma_i\xi_i - \sum_i \delta_i\tau_i.$$

Tale metodo richiede che tutte le derivate parziali della lagrangiana rispetto alle variabili del problema originale siano poste uguali a zero. Le suddette derivate assumono la seguente forma:

$$\frac{\partial L}{\partial R^2} = 1 - \sum_i \alpha_i\mu_i + \sum_i \beta_i(1-\mu_i),$$

$$\frac{\partial L}{\partial \xi_k} = C - \alpha_k - \gamma_k \ \forall k = 1, \ldots, n,$$

$$\frac{\partial L}{\partial \tau_k} = C - \beta_k - \delta_k \ \forall k = 1, \ldots, n.$$

Sostituendo il pattern $\sum_i \alpha_i\mu_i - \beta_i(1-\mu_i)$ con ϵ_i e imponendo che derivata parziale rispetto a R^2 si annulli, otteniamo $\sum_i \epsilon_i = 1$. Calcoliamo la derivata parziale della lagrangiana rispetto alla variabile originale a:

$$\frac{\partial L}{\partial a} = \sum_i ||x_i - a||^2\epsilon_i.$$

Imponendo che questa sia uguale a zero e tenendo conto del vincolo sulla sommatoria delle ϵ_i appena ricavato, arriviamo all'equazione $\sum_i x_i\epsilon_i = a$. Dunque, nel momento in cui si trovano le variabili ottimali ϵ_i, si sa di riuscire a identificare anche il centro della sfera cercata. Il problema duale, da massimizzare, è dunque:

$$\max \sum_i \epsilon_i x_i \cdot x_i - \sum_{i,j} \epsilon_i\epsilon_j x_i \cdot x_j$$

$$s.t.$$

$$\sum_i \epsilon_i = 1.$$

Una volta risolto il problema e ottenute le ϵ_i^* ottimali, a partire da un qualunque elemento x del dominio si può calcolare il valore:

$$R^2(x) = x \cdot x - \sum_i \epsilon_i^* x_i \cdot x + \sum_{i,j} \epsilon_i^*\epsilon_j^* x_i \cdot x_j.$$

Si può dimostrare che questa quantità è esattamente uguale al quadrato della distanza tra a^* (valore ottimale del centro) e x fornito. Inoltre, se si prende un i tale che $0 < \alpha_i < C$ o $0 < \beta_i < C$, allora $R^2(x_i) = R^*$, quadrato del raggio ottimale della sfera. Perciò, dato un punto, è possibile stimare il suo grado di appartenenza alla sfera: se $R^2(x_i) > R^*$ ha membership $> \frac{1}{2}$, se $R^2(x_i) = R^*$ la membership è $\frac{1}{2}$, se $R^2(x_i) < R^*$ la membership è $< \frac{1}{2}$. A questo punto, per ottenere un'approssimazione migliore, è necessario stabilire una famiglia per la funzione di appartenenza che vogliamo inferire, ed eseguire una regressione all'interno di questa famiglia. Nel Paragrafo 2.1.1 vi è una descrizione delle famiglie di funzioni che sono state utilizzate allo scopo.

In realtà, l'ipotesi di racchiudere la maggior parte dei punti all'interno di una sfera è eccessivamente restrittiva: si rende pertanto necessario introdurre una trasformazione Φ, che mappi in modo non lineare tali punti in uno spazio di dimensione più elevata rispetto al dominio considerato; in questo spazio cercheremo la sfera che includa la maggior parte delle immagini. La trasformazione Φ deve essere non lineare, in modo che la nuova sfera corrisponda, nello spazio originale, a un insieme che meglio si possa adattare ai dati. Definiamo kernel la funzione che accetta due argomenti e calcola il prodotto scalare delle loro immagini attraverso Φ: questa si può calcolare anche non conoscendo la trasformazione specifica, ma sapendo a quale famiglia appartiene. Ad esempio, nel caso sia una funzione polinomiale di grado al massimo p, $k(x_i, x_j) = \Phi(x_i) \cdot \Phi(x_j) = (1 + x_i \cdot x_j)^p$. Di conseguenza, la formulazione finale della funzione obiettivo diventa

$$\max \sum_i \epsilon_i \Phi(x_i) \cdot \Phi(x_j) - \sum_{i,j} \epsilon_i \epsilon_j \Phi(x_i) \cdot \Phi(x_j),$$

mentre il calcolo di $R^2(x)$ viene così modificato:

$$R^2(x) = \Phi(x) \cdot \Phi(x) - \sum_i \epsilon_i^* \Phi(x_i) \cdot x + \sum_{i,j} \epsilon_i^* \epsilon_j^* \Phi(x_i) \cdot \Phi(x_j).$$

Nel Paragrafo 2.2 sono elencati e descritti i diversi kernel utilizzati durante gli esperimenti.

2.1.1 Utilizzo della libreria mulearn

Per eseguire gli esperimenti, è stata utilizzata la libreria mulearn[7]. Come si può leggere dalla documentazione, al suo interno vi è una classe FuzzyInductor, che raccoglie la funzionalità fondamentale del processo di apprendimento: una volta allenato (attraverso l'operazione fit) su una serie di esempi, il FuzzyInductor è in grado, ricevendo un vettore di valori, di inferire una funzione

di appartenenza all'insieme fuzzy, e quindi di restituire, tramite la funzione predict, un grado di appartenenza per ogni vettore dato. Per la costruzione del FuzzyInductor devono essere specificati una serie di parametri; il primo dei quali è il parametro c (l'equivalente di C dei paragrafi precedenti), che fa da tradeoff tra la grandezza della sfera di appartenenza e il rispetto dei vincoli imposti.

Ulteriore parametro è il fuzzifier, ovvero la funzione che riceve come argomento la distanza tra l'immagine di un vettore x e il centro della sfera (indicata con $R^2(x)$ nel paragrafo precedente), restituendo un'approssimazione del grado di appartenenza di x al fuzzy set. Esistono diversi tipi di fuzzificatori, più o meno complessi a seconda delle esigenze del problema.

- *Crisp Fuzzifier* (grafico esemplificativo in Figura 2.1): corrisponde a un insieme nel senso classico del termine, che ha valore 1 se l'elemento appartiene all'insieme, 0 se non vi appartiene;

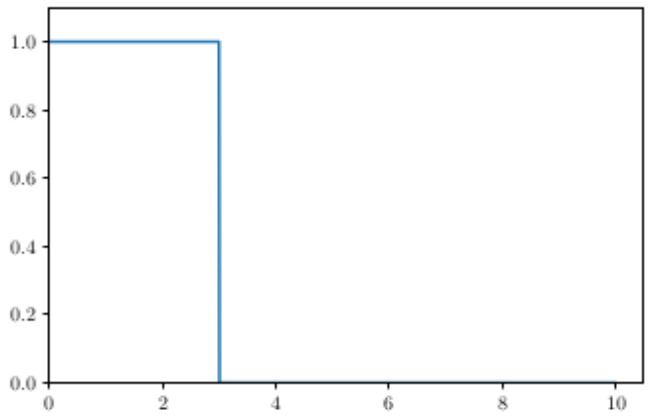

Figura 2.1: Grafico esemplificativo di un crisp fuzzifier

- *Linear Fuzzifier*(grafico esemplificativo in Figura 2.2): corrisponde a un fuzzy set la cui membership decresce linearmente da 1 a 0;

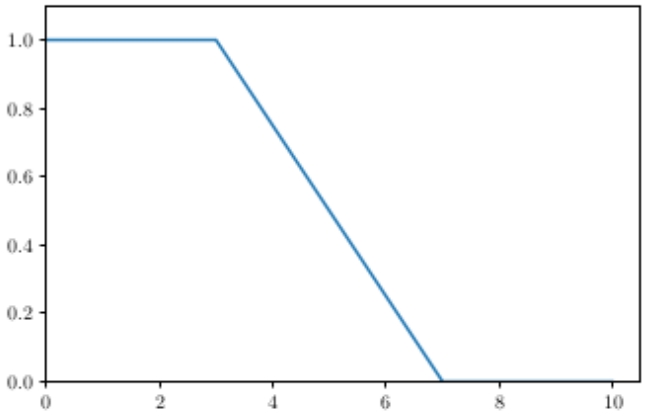

Figura 2.2: Grafico esemplificativo di un linear fuzzifier

- *Exponential Fuzzifier*(grafico esemplificativo in Figura 2.3): corrisponde a un fuzzy set la cui membership diminuisce da 1 a 0 in maniera esponenziale. Il parametro α indica il decadimento esponenziale fissato;

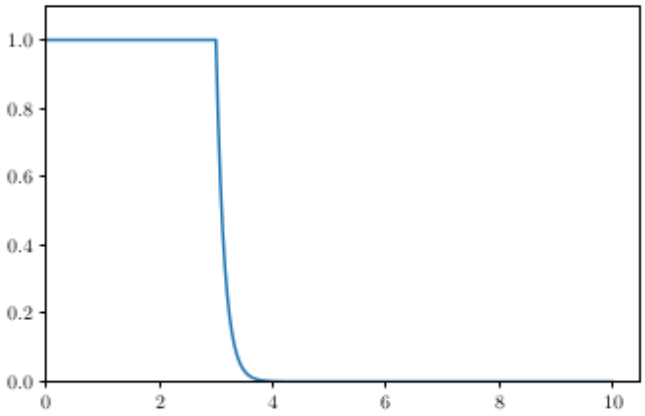

Figura 2.3: Grafico esemplificativo di un exponential fuzzifier con $\alpha = 0.001$

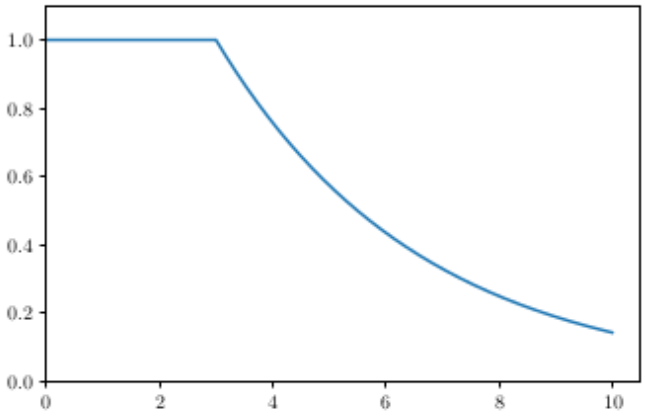

Figura 2.4: Grafico esemplificativo di un exponential fuzzifier con $\alpha = 0.5$

- *Quantile Constant Piecewise Fuzzifier*(grafico esemplificativo in Figura 2.5): corrisponde a un fuzzy set che ha una funzione di membership costante a tratti, per la quale gli step sono definiti in base ai quartili delle distanze al quadrato tra le immagini dei punti e il centro della sfera inferita;

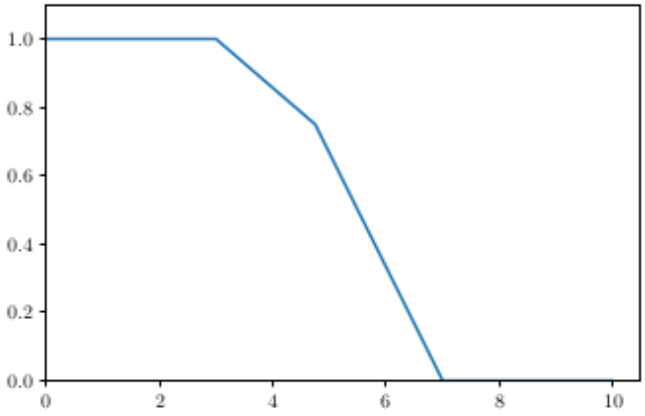

Figura 2.5: Grafico esemplificativo di un quantile constant piecewise fuzzifier

- *Quantile Linear Piecewise Fuzzifier*(grafico esemplificativo in Figura 2.6): corrisponde a un fuzzy set che ha una funzione di membership lineare a tratti, per la quale gli step sono definiti in base ai quartili

delle distanze al quadrato tra le immagini dei punti e il centro della sfera inferita.

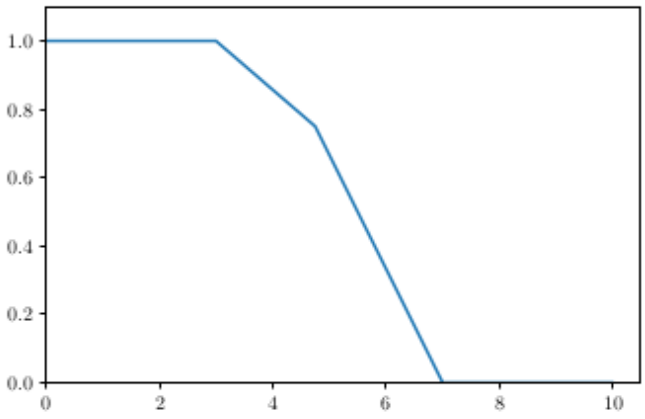

Figura 2.6: Grafico esemplificativo di un quantile linear piecewise fuzzifier

Nella costruzione del FuzzyInductor si può optare per due tipi diversi di solver che risolvano il problema di ottimizzazione: il solver basato su Gurobi e quello basato su TensorFlow.

Gurobi[1] è un solutore commerciale di ottimizzazione sviluppato nel 2008. Risolve problemi di programmazione lineare (LP), quadratica (QP), quadratica con vincoli (CQP), programmazione lineare intera mista (MILP), programmazione intera quadratica mista (MIQP) e programmazione intera quadratica con vincoli (MIQCP).

TensorFlow [3] è una piattaforma end-to-end open source per il machine learning. Viene utilizzata in ambiti di produzione e di ricerca scientifica, principalmente allo scopo di realizzare algoritmi per compiti percettivi e di comprensione del linguaggio. Viene tipicamente utilizzato per allenare modelli di reti neurali profonde, ma, dal momento che il loro allenamento richiede l'ottimizzazione di una funzione, è possibile utilizzarlo per risolvere problemi generici di ottimizzazione, come il quello descritto nel Paragrafo 2.1.

2.2 Il Kernel

Il parametro kernel (equivalente della funzione k dei paragrafi precedenti) della classe FuzzyInductor definisce quale sarà la funzione attraverso cui i valori verranno mappati nello spazio a più dimensioni. La libreria mulearn implementa i seguenti kernel:

- *kernel lineare*: il valore $k(x_1, x_2)$ equivale al prodotto $x_1 \cdot x_2$, ovvero $\sum_{i=1}^{N}(x_1)_i \cdot (x_2)_i$, dove N è la dimensione dei vettori x_1 e x_2;
- *Kernel polinomiale*: il valore $k(x_1, x_2)$ equivale a $(x_1 \cdot x_2 + 1)^d$, dove d è il grado polinomiale del kernel;
- *kernel polinomiale omogeneo*: il valore $k(x_1, x_2)$ equivale a $(x_1 \cdot x_2)^d$;
- *kernel Gaussiano*: il valore $k(x_1, x_2)$ equivale a $e^{-\frac{||x_1 - x_2||^2}{2\sigma^2}}$, dove σ è la deviazione standard del kernel;
- *kernel iperbolico*: il valore $k(x_1, x_2)$ equivale a $\tanh(\alpha x_1 \cdot x_2 + \beta)$, dove α è la scala e β l'offset.

La maggior parte dei kernel qui descritti richiedono di essere inizializzati con opportuni iper-parametri: nel Capitolo 3 è stato esposto il procedimento di model selection effettuato su di essi.

2.2.1 Kernel precomputato

Per ottimizzare i tempi di computazione (i punti da mappare, nel caso di un insieme di formule, non corrispondono a valori numerici i cui valori del kernel sono semplici da calcolare) si è utilizzato un kernel precomputato, per il quale $k(x_1, x_2)$ equivale a un valore precedentemente calcolato e salvato all'interno di una matrice, detta *matrice di Gram*. In generale, il valore restituito da un kernel è interpretabile come una misura della similarità tra i suoi argomenti: per calcolarlo tra due formule si è dunque fatto riferimento a un criterio ispirato alla similarità di Jaccard, descritto in [15]. Per tutte le coppie di assiomi ϕ e ψ, la definizione di similarità deve rispettare le seguenti proprietà:

- $0 \leq \text{sim}(\psi, \phi) \leq 1$, si sta infatti operando in un ambito possibilistico, nel quale a ogni assioma viene associato un grado di possibilità compreso tra 0 e 1;
- $\text{sim}(\psi, \phi) = 1$ sse $\psi \equiv \phi$;
- $\text{sim}(\psi, \phi) = \text{sim}(\phi, \psi)$.

Se si riuscisse a definire una funzione $\text{Impl}(\phi, \psi)$, allora si potrebbe dire $\text{sim}(\psi, \phi) = \min\{\text{Impl}(\phi, \psi), \text{Impl}(\psi, \phi)\}$, dove il minimo traduce la congiunzione delle due condizioni logiche. Per definire l'implicazione, ci si può avvalere della definizione classica di implicazione materiale, ovvero $\text{Impl}(\psi, \phi) = 1$ se $\models \neg\phi \vee \psi$, 0 altrimenti.

Si può restringere il campo agli assiomi di inclusione di cui ci si è occupati nello specifico, ovvero formule del tipo Subclass(B,C), o $B \sqsubseteq C$ in forma più compatta, e le loro negazioni ¬Subclass(B, C) ($B \not\sqsubseteq C$). Avendo a disposizione un insieme di assiomi, si può dire che un individuo della base di conoscenza a conferma $B \sqsubseteq C$ (e contraddice $B \not\sqsubseteq C$) se vale $B(a) \wedge C(a)$, a contraddice $B \sqsubseteq C$ (e conferma $B \not\sqsubseteq C$) se vale $B(a) \wedge \neg C(a)$. Dunque, sfruttando la definizione precedentemente data di implicazione materiale, si può affermare:

$$Impl(A \sqsubseteq B, C \sqsubseteq D) = \frac{\| \{a : (A(a) \wedge \neg B(a)) \vee (C(a) \wedge D(a))\} \|}{\| \{a : (A(a) \vee C(a))\} \|}.$$

Se si definisce

$$[C] = \{a : C(a)\},$$

allora si può scrivere la formula precedente come:

$$\frac{\| [A] \cup [\overline{B}] \cup [C] \cap [D] \|}{\| [A] \cup [C] \|}.$$

Avendo precedentemente definito l'operatore di similarità tra due assiomi in relazione al minimo tra le due implicazioni, risulta:

$$sim(A \sqsubseteq B, C \sqsubseteq D) = min\{Impl(A \sqsubseteq B, C \sqsubseteq D), Impl(C \sqsubseteq D, A \sqsubseteq B)\}$$

$$= min\left\{ \frac{\| [A] \cap [\overline{B}] \cup [C] \cap [D] \|}{\| [A] \cup [C] \|}, \frac{\| [C] \cup [\overline{D}] \cup [C] \cap [D] \|}{\| [A] \cup [C] \|} \right\}.$$

$$= \frac{min\{\| [A] \cap [\overline{B}] \cup [C] \cap [D] \|, \| [C] \cup [\overline{D}] \cup [C] \cap [D] \|\}}{\| [A] \cup [C] \|}.$$

Nella pratica, per calcolare i valori del kernel precomputato si è utilizzata una versione semplificata della formula precedente, ovvero:

$$sim(A \sqsubseteq B, C \sqsubseteq D) = \frac{\| [A] \cap [B] \cup [C] \cap [D] \|}{\| [A] \cup [C] \|}.$$

2.2.2 Kernel alternativi

Nel corso degli esperimenti sono state utilizzate altre definizioni di similarità, diverse da quella di Jaccard, descritte in [13].

Length-based similarity

Una prima forma ingenua di similarità utilizzata è quella basata sulla lunghezza degli assiomi. Si definisce dunque

$$s_{len}(\phi_1, \phi_2) = 1 - \frac{|\#\phi_1 - \#\phi_2|}{max\{\#\phi_1, \#\phi_2\}},$$

ovvero il valore assoluto della differenza tra le due lunghezze, normalizzato. Tale formulazione nasconde una problematica: quando un assioma è il negato dell'altro la funzione restituisce un valore di similarità alto. Occore, pertanto, introdurre un'ulteriore casistica: se i segni dei due assiomi sono diversi, la funzione di similarità vale $1 - s_{len}(\phi_1, \phi_2)$.

Hamming similarity

Altro modo di definire la similarità tra due assiomi è la distanza di Hamming, ovvero la distanza tra le rappresentazioni testuali delle due formule, intesa come il numero di caratteri diversi tra una e l'altra, trascurando i caratteri extra della stringa più lunga. Definendo H la distanza di Hamming, si fa la stessa distinzione descritta precedentemente per gli assiomi di segno opposto: $\mathrm{sim_H}(\phi_1, \phi_2) = H(\mathrm{abs}(\phi_1), \mathrm{abs}(\phi_2))$ se i segni delle due formule sono opposti, $1 - H(\phi_1, \phi_2)$ altrimenti.

Levenshtein similarity

L'ultima funzione di similarità utilizzata è la distanza di Levenshtein tra due stringhe, ovvero il numero più piccolo di operazioni atomiche che vanno fatte per trasformare una nell'altra. Chiamando questa funzione *Lev*, $sim_{edit}(\phi_1, \phi_2) = 1 - Lev(\phi_1, \phi_2)$ se i due segni sono diversi, $Lev(abs(\phi_1), abs(\phi_2))$ altrimenti.

Capitolo 3

Esperimenti

3.1 Model selection e model evaluation

La model selection è il processo di scelta di un modello definitivo tra un insieme di candidati modelli di machine learning per un training set; può essere applicata sia tra modelli sia tra parametri di uno stesso modello. La model evaluation è, invece, la valutazione delle performance di un modello esistente.

Effettuare model selection e model evaluation sugli stessi dati che vengono utilizzati per il training è un errore metodologico [16] che rende difficile identificare una condizione di overfitting, nella quale il modello è in grado di associare molto bene etichette a individui del campione che ha già visto, semplicemente ripetendole, ma fallisce nell'identificare i valori associati a nuovi elementi su cui non è stato allenato; spesso l'overfitting si verifica quando il numero di parametri è eccessivo rispetto a quello di osservazioni, dunque il modello risulta troppo complesso e finisce per adattarsi ai dati del campione, imparandone anche il "rumore" (o informazioni irrilevanti) [2]. Per questo motivo si suddividono in dati in training, validation e test set, utilizzati rispettivamente per allenare, selezionare e valutare i modelli. Quest'ultimo è un procedimento che può essere applicato in condizioni di disponibilità di dati: in molte situazioni, tuttavia, le quantità di dati disponibili per training e test saranno limitate. Per costruire dei buoni modelli vorremmo usare gran parte dei dati disponibili per il training, ma allo stesso tempo la scarsità di dati di test ci porterebbe a una valutazione poco attendibile dei modelli stessi [5]. Per far fronte a questa problematica, si deve fare riferimento ad altre strategie.

3.1.1 Metodi probabilistici

Un metodo di risolvere il problema è ricorrere a un approccio probabilistico, nel quale la valutazione del modello avviene basandosi contemporaneamente sulla sua complessità e sulle sue performance in fase di allenamento. Lo scoring delle performance viene penalizzato sulla base di quanto alto si suppone il bias di ottimismo per l'errore del training set, che tende notoriamente ad essere elevato, mentre la complessità dipende tipicamente dal numero di gradi di libertà nei parametri del modello. Il vantaggio di questo approccio sta nel fatto che non necessita di isolare una porzione di dati di test: tutti i dati a disposizione possono essere utilizzati per il training. I metodi probabilistici, tuttavia, non permettono di utilizzare lo stesso criterio per valutare più modelli (i criteri di valutazione tendono ad essere specifici per ogni modello, proprio a causa della loro variabilità) e non considerano l'incertezza dei parametri: nella pratica, questi metodi tendono a favorire modelli troppo semplificati [12].

3.1.2 Metodi di sottocampionamento

I metodi di ricampionamento stimano le performance di un modello tramite elementi fuori dal campione di training. Il traning set viene suddiviso in sottoinsiemi di training e di test, sui quali viene allenato e valutato, e questa suddivisione avviene più volte, riportando come errore finale la media delle performance degli esperimenti effettuati con ogni suddivisione. Tra i metodi di ricampionamento contiamo:

- *random train/test splits*: fissate le percentuali di dati che devono essere inserite nei sottoinsiemi di training e test, l'assegnamento di un'osservazione a un campione o all'altro avviene in maniera casuale;
- *bootstrap*: per ognuno dei sottoinsiemi di training e test, si sceglie una grandezza del campione n, dopodiché, scelta casualmente n volte un'osservazione dal dataset, questa viene aggiunta al campione (la stessa osservazione può così trovarsi in più sottoinsiemi contemporaneamente);
- *cross-validation*: in quanto metodo utilizzato per effettuare la maggior parte degli esperimenti di cui si tratterà in questo elaborato, la sua forma di base (k-Fold cross-validation) viene trattata nella sezione seguente.

k-Fold cross-validation

Nella k-Fold cross-validation, il training set viene suddiviso in k sottoinsiemi più piccoli: per ogni sottoinsieme il modello viene allenato su $k-1$ sottoinsiemi del training set, mentre la valutazione avviene sulla porzione rimanente dei dati; l'errore finale è dato dalla media degli errori calcolati per ogni ripetizione del processo. Tale metodo risulta essere computazionalmente oneroso, ma permette di utilizzare tutti i dati a disposizione per allenare l'algoritmo, valore aggiunto che risulta determinante in problemi per i quali il numero di elementi del campione risulta basso [16]. Una schematizzazione grafica di questa tecnica viene riportata in Figura 3.1.

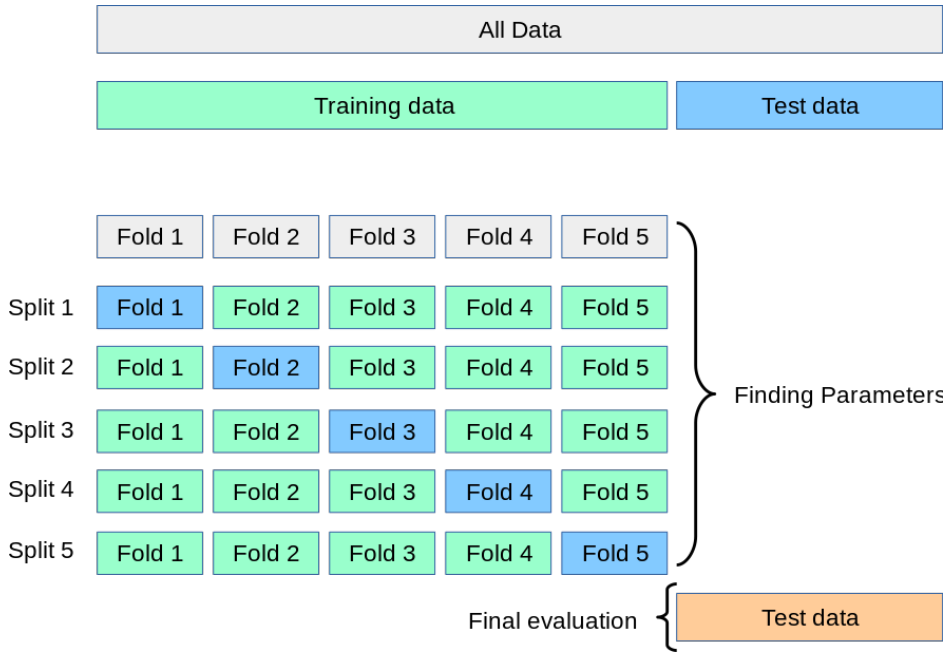

Figura 3.1: Schematizzazione della suddivisione in fold nella k-fold cross validation (con $k = 5$) [16].

3.2 Riproduzione degli esperimenti originali

Una delle prime attività svolte è stata quella di ripetere gli esperimenti originali descritti in [15], i cui valori medi di RMSE sono riportati nella Tabella 3.2, per confrontarne i risultati al netto dell'utilizzo di una nuova libreria: quegli esperimenti, infatti, si basavano una versione vecchia del codice, che non era ancora stato organizzato sistematicamente come nella libreria mulearn [7].

Sono stati considerati 722 candidati assiomi del tipo *SubClassOf* con le loro negazioni, per un totale di 1444 formule; tra ciascuna coppia è stata calcolata la similarità di Jaccard e inserita all'interno di una matrice di Gram K. Il valore precalcolato di possibilità è stato utilizzato come etichetta per il grado di appartenenza di ogni formula all'insieme fuzzy degli assiomi. Si è poi effettuata la model selection sugli iperparametri con cui inizializzare i componenti dell'algoritmo; le formule sono state permutate e divise in tre gruppi dedicati a training, model selection e model validation, contenenti rispettivamente l'80%, il 10% e il 10% del campione originale. La model selection è stata effettuata a questo stadio sull'iperparametro c, in termini di quale tra quelli testati (0.005, 0.007, 0.01, 0.03, 0.05, 0.07, 0.1, 0.3, 0.5, 0.7, 1, 10, 100) desse luogo all'RMSE minore nella previsione del grado di membership dell'assioma, dove RMSE sta a indicare lo scarto quadratico medio posto sotto radice, per riportarlo all'unità di misura dei dati. Questo procedimento è stato ripetuto dieci volte per ogni fuzzificatore, concludendo per ciascuno di esso i valori minimi dell'RMSE sui valori di test, che vengono riportati nella tabella 3.1

	RMSE test(+/- std)
Crisp Fuzzifier	0.41536 +/- 0.06
Quantile Constant Piecewise Fuzzifier	0.33295 +/- 0.048
Quantile Linear Piecewise Fuzzifier	0.30343 +/- 0.032
Linear Fuzzifier	0.34012 +/- 0.054
Exponential Fuzzifier	0.32827 +/- 0.04

Tabella 3.1: Valori minimi RMSE per fuzzificatore, esperimenti nuovi

	RMSE test(+/- std)
Crisp Fuzzifier	0.377 +/- 0.33
Quantile Constant Piecewise Fuzzifier	0.314 +/- 0.179
Quantile Linear Piecewise Fuzzifier	0.314 +/- 0.164
Linear Fuzzifier	0.313 +/- 0.49
Exponential Fuzzifier	0.362+/- 0.321

Tabella 3.2: Valori minimi RMSE per fuzzificatore, esperimenti 2018

Come si può notare confrontando le tabelle 3.1 e 3.2, i risultati ottenuti nei due esperimenti si discostano, anche se non eccessivamente. Le maggiori differenze si riscontrano per il Crisp Fuzzifier e per il Linear Fuzzifier; i risultati

ottenuti mediante Exponential Fuzzifier sono solo parzialmente confrontabili poiché nel primo caso è stato utilizzato un fuzzificatore con decadimento esponenziale non fissato manualmente, mentre i risultati degli esperimenti del 2018 si riferiscono a un decadimento esponenziale di 0.001.

3.3 Esperimenti sul kernel

Sono poi state effettuate model selection e model validation tramite k-Fold cross-validation, con $k = 5$, per ogni fuzzificatore.

La model selection è stata eseguita sul parametro c, scegliendo tra sette possibili valori in un range da -1 a 1, spaziati uniformemente su una scala logaritmica (dunque tra 0.1, 0.21544347, 0.46415888, 1, 2.15443469, 4.64158883 e 10). Per ogni fuzzificatore, la model evaluation ha portato ai valori medi di RMSE in fase di test e in fase di train per i valori migliori di c, con le rispettive deviazioni standard.

Il procedimento è stato ripetuto utilizzando diversi kernel precomputati e ha portato ai risultati che verranno esposti nelle sezioni successive.

3.3.1 Esperimenti con Jaccard similarity

Nella Tabella 3.3 sono riportati i risultati ottenuti utilizzando come kernel la similarità di Jaccard definita nel Paragrafo 2.2.1, precomputata e inserita all'interno di una matrice di Gram.

Fuzzifier	RMSE test(+/- std)	RMSE train (+/- std)
Crisp	0.26974 +/ 0.27	0.21694 +/- 0.024
Quantile Constant Piecewise	0.11759 +/- 0.108	0.10881 +/- 0.02
Quantile Linear Piecewise	0.1032 +/- 0.08	0.09935 +/- 0.014
Linear	0.15256 +/- 0.172	0.13883 +/- 0.024
Exponential	0.13603 +/- 0.17	0.13391 +/- 0.02

Tabella 3.3: Valori medi di RMSE in fase di test e train per ogni fuzzificatore, con la similarità di Jaccard come kernel.

3.3.2 Esperimenti con Length-based similarity

La Tabella 3.4 mostra i risultati ottenuti utilizzando come kernel la similarità basata su lunghezza definita al Paragrafo 2.2.2. L'ingenuità del metodo faceva supporre un generale peggioramento dei risultati rispetto alla similarità

di Jaccard in Tabella 3.3, che anzi si presupponeva più consistente di quello effettivamente riscontrato.

Fuzzifier	RMSE test(+/- std)	RMSE train (+/- std)
Crisp	0.38573 +/- 0.466	0.391 +/- 0.108
Quantile Constant Piecewise	0.31572 +/- 0.288	0.31 +/- 0.058
Quantile Linear Piecewise	0.30646 +/- 0.252	0.3 +/- 0.052
Linear	0.24792 +/- 0.092	0.225 +/- 0.014
Exponential	0.24521 +/- 0.11	0.224 +/- 0.014

Tabella 3.4: Valori medi di RMSE in fase di test e train per ogni fuzzificatore, con la Length-similarity come kernel.

Una nuova classe kernel è stata implementata per essere utilizzata in modo da computare direttamente il valore di similarità basato su lunghezza dei due assiomi, senza passare per la creazione o il caricamento della matrice di Gram con i valori precalcolati. Per velocità di computazione si è scelto di implementare solamente la prima versione della funzione, ignorando il problema del valore di similarità per assiomi opposti l'uno all'altro esposto nel Paragrafo 2.2.2. Anche agendo in questo modo, il tempo di computazione è risultato accettabile eseguendo gli esperimenti su un massimo di 50 assiomi su 1444; si è scelto, pertanto, di utilizzare matrice e kernel precomputato anche per questa definizione di similarità.

3.3.3 Esperimenti con Hamming similarity

Gli esperimenti con la Hamming similarity descritta al Paragrafo 2.2.2 hanno invece condotto ai risultati riportati nella Tabella 3.5. I valori di RMSE e varianza risultano complessivamente peggiori rispetto alla similarità di Jaccard, e stranamente non molto migliori di quelli basati sulla lunghezza: anzi, utilizzando il CrispFuzzifier risultano essere perfino peggiori di questi ultimi.

Fuzzifier	RMSE test(+/- std)	RMSE train (+/- std)
Crisp	0.54868 +/- 0.418	0.713 +/- 0.118
Quantile Constant Piecewise	0.35935 +/- 0.198	0.448 +/- 0.064
Quantile Linear Piecewise	0.35008 +/- 0.162	0.42 +/- 0.06
Linear	0.23933 +/- 0.11	0.191 +/- 0.066
Exponential	0.22399 +/- 0.088	0.198 +/- 0.016

Tabella 3.5: Valori medi di RMSE in fase di test e train per ogni fuzzificatore, con la Hamming similarity come kernel.

Anche in questo caso, l'implementazione della classe *HammingKernel* non ha portato ai risultati sperati, a causa dell'eccessivo costo computazionale di calcolare di volta in volta la distanza di Hamming per ognuno dei 1444 assiomi.

3.3.4 Esperimenti con Levenshtein similarity

I valori dell'RMSE ottenuti tramite similarità di Levenshtein (Paragrafo 2.2.2) sono mostrati nella Tabella 3.6 e risultano valere le stesse considerazioni fatte per la distanza di Hamming.

Fuzzifier	RMSE test(+/- std)	RMSE train (+/- std)
Crisp	0.50254 +/- 0.388	0.73 +/- 0.076
Quantile Constant Piecewise	0.36324 +/- 0.122	0.439 +/- 0.058
Quantile Linear Piecewise	0.35503 +/- 0.138	0.412 +/- 0.05
Linear	0.23958 +/- 0.094	0.201 +/- 0.068
Exponential	0.23807 +/- 0.11	0.198 +/- 0.07

Tabella 3.6: Valori medi di RMSE in fase di test e train per ogni fuzzificatore, con la Levenshtein similarity come kernel.

Come nei due casi precedenti, si è continuato a dover utilizzare una matrice di Gram precalcolata per definire i valori del kernel in un tempo accettabile.

3.3.5 Il problema del raggio negativo

Nel corso degli esperimenti, ci si è trovati di fronte a una problematica precedentemente inaspettata. Al momento del calcolo della distanza dei punti, nello spazio delle feature, dal centro dello spazio di riferimento, si è riscontrato che questi valori risultavano negativi. Questo non dovrebbe chiaramente avvenire per il valore di una distanza, e ha fatto sì che risultasse problematico considerare valido il valore di membership inferito dall'algoritmo per alcuni assiomi.

Il problema di definire perché ciò avvenga è ancora aperto, ma in prima istanza è stata fatta la seguente ipotesi: i valori del kernel all'interno della matrice di Gram sono stati precalcolati seguendo delle euristiche, che per definizione risultano intuitivamente corrette, ma non supportate da una dimostrazione formale. L'ipotesi è, dunque, che i valori precomputati della matrice di Gram non rappresentino effettivamente un kernel, non esistendo una dimostrazione del fatto che ne rispettino le caratteristiche formali; perciò

è possibile che, nel momento in cui si è andati a cercare di ritrovare delle proprietà matematiche che si riconoscevano immediate, queste non siano invece state riscontrate. Per cercare di comprendere meglio le ragioni che portavano al problema sopra descritto, sono stati condotti ulteriori esperimenti.

Eliminazione di formule

Contestualmente alla negatività del raggio, ci si è anche resi conto di un problema riguardante la matrice di Gram in sé: questa non era semidefinita positiva, requisito necessario per effettuare il processo di ottimizzazione. Teoricamente è possibile perturbare la diagonale della matrice al fine di renderla semidefinita positiva, ma l'aggiustamento necessario è risultato di un ordine di grandezza talmente elevato da far perdere di significato ai valori del kernel. Si è quindi supposto che il problema riguardasse la matrice, e, per convalidare l'ipotesi, si è pensato eliminare da quest'ultima alcune formule, allo scopo di capire se rimuovendole sarebbe cambiato anche l'aggiustamento. Inizialmente si è intrapresa una strada combinatoria, che consisteva nell'eliminare tutte le possibili formule una alla volta, successivamente tutte le coppie e a procedere fino a gruppi di dimensione più elevata; eliminando una formula alla volta non si sono riscontrati cambiamenti dell'aggiustamento nel suo ordine di grandezza, mentre il tempo di esecuzione per l'eliminazione combinatoria delle coppie di formule si è rivelato eccessivamente elevato. Si è perciò deciso di eliminare coppie di formule a campione, scegliendole diecimila volte casualmente dal gruppo di candidati assiomi: anche con questa tecnica, l'indice di aggiustamento non ha subito delle modifiche rilevanti; si è perciò deciso di abbandonare questa strada.

Valori di similarità come vettori in input all'algoritmo

Si è poi provato a cambiare completamente approccio: non considerare più i valori precomputati della matrice di Gram come un kernel, ma come descrizione delle formule stesse; le righe della matrice di similarità sono dunque diventate il vettore di valori numerici utilizzati per allenare l'algoritmo di apprendimento. Procedendo in questo modo, si sono ottenuti i risultati riportati nella tabella 3.7, che non soffrono del problema del raggio negativo ma sono globalmente meno buoni, in termini di RMSE, di quelli ottenuti utilizzando l'approccio originale.

Fuzzifier	RMSE test(+/- std)	RMSE train (+/- std)
Crisp	0.4714 +/- 0.402	0.243 +/- 0.06
Quantile Constant Piecewise	0.52855 +/- 0.224	0.067 +/- 0.016
Quantile Linear Piecewise	0.46104 +/- 0.228	0.072 +/- 0.016
Linear	0.45253 +/- 0.33	0.165 +/- 0.028
Exponential	0.42252 +/- 0.302	0.167 +/- 0.028

Tabella 3.7: Valori ottenuti utilizzando la matrice di Gram come descrizione delle formule

3.4 Ambiente utilizzato

Gli esperimenti sono stati eseguiti su una macchina con CPU Intel(R) Core(T) i7-8550U, 8 GB di RAM, sistema operativo Windows 10 Home versione 20H2. Il linguaggio utilizzato è stato Python 3.7.9, in environment Anaconda [4] versione 4.9.2. Le principali librerie utilizzate sono state scikit-learn [16] versione 0.23.2, numpy [10] versione 1.17.0, matplotlib [11] versione 3.3.2, mulearn [7] versione 0.2.9.4.

3.5 Risultati ottenuti

A seguito degli esperimenti svolti, si può concludere che il kernel precomputato all'interno della matrice di Gram è necessario per evitare l'eccesso di tempo di computazione impiegato nel calcolare di volta in volta i valori richiesti; è inoltre emerso che, tra le possibili definizioni di kernel, quella che porta ai valori medi migliori di RMSE è la similarità di Jaccard. Il problema della negatività del raggio risulta ancora aperto, dal momento che le modifiche della matrice tramite eliminazione di formule non lo hanno eliminato e la soluzione proposta al paragrafo 3.3.5 non si è rivelata ottimale in termini di RMSE: la risoluzione del problema necessiterebbe di ulteriore ricerca.

Conclusione

Bibliografia

[1] Gurobi website. https://www.gurobi.com/products/gurobi-optimizer/.

[2] Ibm cloud education website. https://www.ibm.com/cloud/learn/overfitting.

[3] Tensor flow website. https://www.tensorflow.org/.

[4] Anaconda software distribution, 2020.

[5] Christopher M. Bishop. *Pattern recognition and machine learning*. 2006.

[6] Andrea Bonarini. Sistemi fuzzy. *Mondo digitale*, (1), 2003.

[7] Malchiodi Dario. Mulearn documentation. https://mulearn.readthedocs.io/en/latest/.

[8] Didier Dubois. Possibility theory and statistical reasoning. 2006.

[9] Andrea G. B. Tettamanzi Catherine Faron-Zucker and Fabien Gandon. Possibilistic testing of owl axioms against rdf data. *International Journal of Approximate Reasoning*, page 114–130, 2017.

[10] Charles R. Harris, K. Jarrod Millman, Stéfan J. van der Walt, Ralf Gommers, Pauli Virtanen, David Cournapeau, Eric Wieser, Julian Taylor, Sebastian Berg, Nathaniel J. Smith, Robert Kern, Matti Picus, Stephan Hoyer, Marten H. van Kerkwijk, Matthew Brett, Allan Haldane, Jaime Fernández del Río, Mark Wiebe, Pearu Peterson, Pierre Gérard-Marchant, Kevin Sheppard, Tyler Reddy, Warren Weckesser, Hameer Abbasi, Christoph Gohlke, and Travis E. Oliphant. Array programming with NumPy. *Nature*, 585(7825):357–362, September 2020.

[11] J. D. Hunter. Matplotlib: A 2d graphics environment. *Computing in Science & Engineering*, 9(3):90–95, 2007.

[12] Max Kuhn and Kjell Johnson. *Applied predictive modeling*. 2013.

[13] Dario Malchiodi, Célia da Costa Pereira, , and Andrea G.B. Tettamanzi. Classifying candidate axioms, dimensionality reduction techniques. 2020.

[14] Dario Malchiodi and Witold Pedrycz. Learning membership functions for fuzzy sets through modified support vector clustering. 2013.

[15] Dario Malchiodi and Andrea G.B. Tettamanzi. Predicting the possibilistic score of owl axioms through modified support vector clustering. *SAC 2018*, pages 1984–1991, 2018.

[16] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.

[17] James Hendler Tim Berners Lee and Ora Lassila. The semantic web. *Scientific American*, 2001.

[18] Lotfi Zadeh. Fuzzy sets. *Information and control*, (8(3)):891–921, 1965.