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

1.2 Applicazione alla ricerca di assiomi in un insieme di formule

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 [5].

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 di mappare i punti x_i su una sfera di raggio R e centro a sconosciuti, e che vi sia un modo di far sì che il loro grado di membership dipenda dalla distanza dal centro a. A questo punto, la ricerca si trasformerebbe nella risoluzione di un problema di ottimizzazione vincolata, di cui si può dare una prima formulazione:

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

$$s.t.$$

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

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

Il problema diventa dunque trovare la sfera di raggio minimo, al variare di R^2 e a, che raccolga i 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. 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. Calcoliamo:

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

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

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:

$$max \sum_i \epsilon_i x_i x_i - \sum_{i,j} \epsilon_i \epsilon_j x_i x_j$$
$$s.t.$$
$$\sum_i \epsilon_i = 1.$$

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:

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

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 $R^2(x) = R^*$, quadrato del raggio ottimale della sfera. Perciò, dato un punto, è possibile stimare il suo grado di appartenenza alla sfera: se $R^2(x) > R^*$ ha membership $> \frac{1}{2}$, se $R^2(x) = R^*$ la membership è $\frac{1}{2}$, se $R^2(x) < R^*$ 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 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, $k(x_i, x_j) = \Phi(x_i) \cdot \Phi(x_j) = (1 + x_i + x_j)^p$. Di conseguenza, la formulazione finale della funzione obiettivo diventa

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

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

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

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[3]. 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, 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 approssima gli ouput del processo di regressione, al fine di prevedere l'appartenenza di nuovi

input. Esistono diversi tipi di fuzzificatori, più o meno complessi a seconda delle esigenze del problema.

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

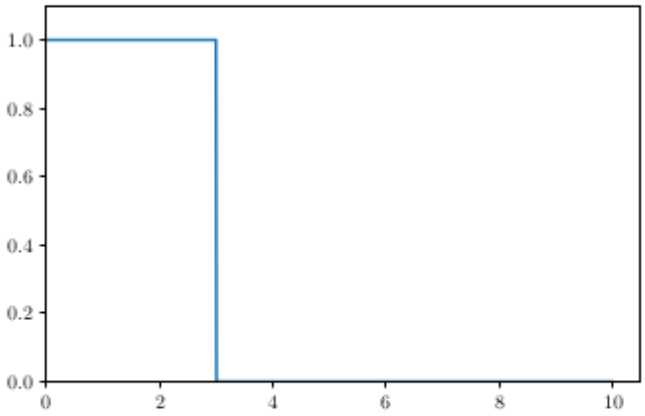

Figura 2.1: Grafico esemplificativo di un crisp fuzzifier

- *Linear Fuzzifier*: corrisponde a un fuzzy set la cui membership decresce linearmente da 1 a 0;

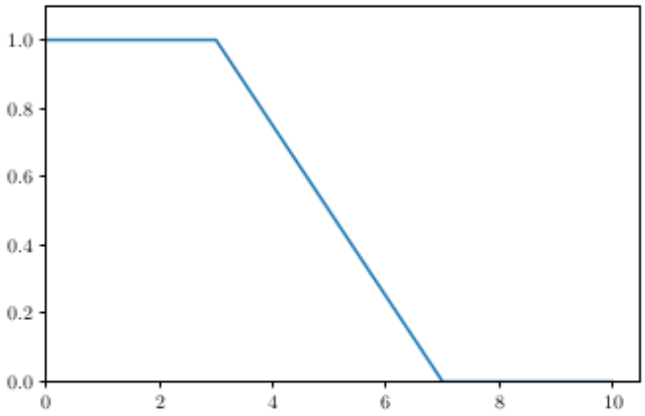

Figura 2.2: Grafico esemplificativo di un linear fuzzifier

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

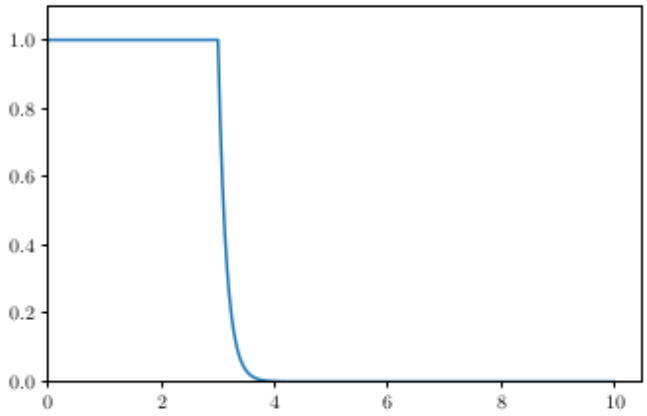

Figura 2.3: Grafico esemplificativo di un exponential fuzzifier con $\alpha = 0.001$

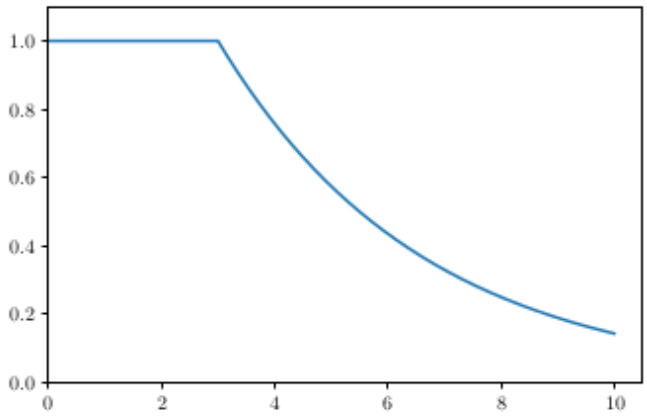

Figura 2.4: Grafico esemplificativo di un exponential fuzzifier con $\alpha = 0.5$

- *Quantile Constant Piecewise Fuzzifier*: 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;

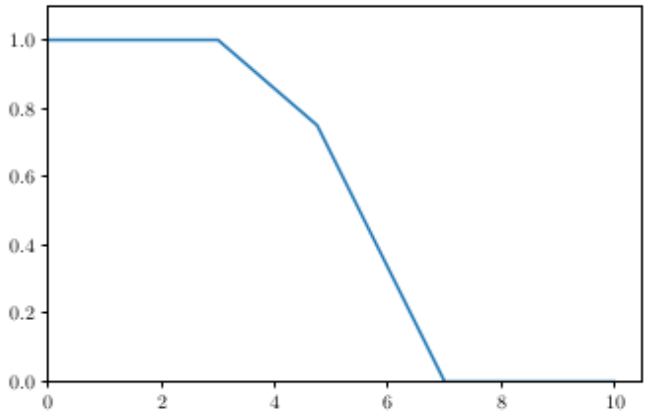

Figura 2.5: Grafico esemplificativo di un quantile constant piecewise fuzzifier

- *Quantile Linear Piecewise Fuzzifier*: 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.

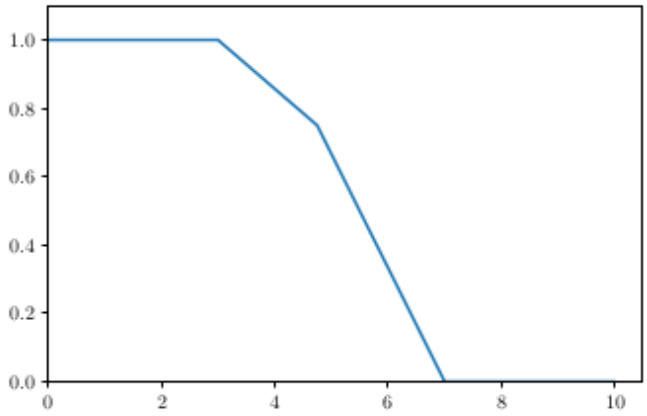

Figura 2.6: Grafico esemplificativo di un quantile linear piecewise fuzzifier

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 fondato nel 2008. Risolve problemi di programmazione lineare (LP), quadratica (QP), qua-

dratica con vincoli (CQP), programmazione lineare intera mista (MILP), programmazione intera quadratica mista (MIQP) e programmazione intera quadratica con vincoli (MIQCP).

TensorFlow[2] 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.

2.2 Il Kernel

Il parametro kernel 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*. Per calcolarlo, ovvero trovare un criterio per definire il grado di similarità tra due assiomi, si utilizza un indice di similarità ispirato alla similarità di Jaccard, descritta in [6]. Per tutte le

coppie di assiomi ϕ e ψ, la definizione di similarità deve rispettare le seguenti proprietà:

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

Se si riuscisse a definire una funzione $Impl(\phi, \psi)$, allora si potrebbe dire $sim(\psi, \phi) = min\{Impl(\phi, \psi), Impl(\psi, \phi)\}$, 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 $Impl(\psi, \phi) = 1$ 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 $\neg Subclass(B, C)$ e $B \not\sqsubseteq C$. Avendo a disposizione un insieme di assiomi, si può dire che 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:

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

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

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

Tabella 2.1: Valori ottenuti con la similarità di Jaccard

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

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

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

Utilizzando questa definizione di similarità, i risultati ottenuti in termini di RMSE per i valori di test e per i valori di train sono riportati nella tabella 2.1.

2.2.2 Kernel alternativi

Nel corso degli esperimenti sono state utilizzate altre definizioni di similarità, diverse da quella di Jaccard, descritte in [4].

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 $1 - s_{len}(\phi_1, \phi_2)$. 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. Anche agendo in questo modo, il tempo di computazione è risultato accettabile per un massimo di 50 assiomi su 1444; si è scelto, pertanto, di utilizzare matrice e kernel precomputato anche per questa definizione di similarità. La tabella 2.2 mostra i risultati ottenuti. L'ingenuità del metodo faceva supporre un generale peggioramento dei risultati rispetto alla similarità di Jaccard, che anzi si presupponeva più consistente di quello effettivamente riscontrato.

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

Tabella 2.2: Valori ottenuti con Length Similarity

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: $sim_H(\phi_1, \phi_2) = H(abs(\phi_1), abs(\phi_2))$ se i segni delle due formule sono opposti, $1 - H(\phi_1, \phi_2)$ altrimenti. Anche in questo caso, l'implementazione della classe *HammingKernel* non ha portato ai risultati sperati, a causa dell'eccessivo costo computazionale di calcolare la distanza di Hamming per ognuno dei 1444 assiomi. Gli esperimenti con la matrice di Gram hanno invece condotto ai risultati riportati nella tabella 2.3. 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.

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*,

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

Tabella 2.3: Valori ottenuti con la similarità di Hamming

$sim_{edit}(\phi_1, \phi_2) = 1 - Lev(\phi_1, \phi_2)$ se i due segni sono diversi, $Lev(abs(\phi_1), abs(\phi_2))$ altrimenti. 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. I valori dell'RMSE ottenuti sono mostrati nella tabella 2.4 e risultano valere le stesse considerazioni fatte per la distanza di Hamming.

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

Tabella 2.4: Valori ottenuti con la similarità di Levenshtein

Capitolo 3

Esperimenti

Qui andrei a mostrare gli esperimenti e i relativi risultati conseguiti tramite cross validation e model selection.

3.1 Riproduzione degli esperimenti originali

3.2 Esperimenti sul kernel

3.2.1 Kernel alternativi

3.2.2 Possibili soluzioni al problema del fitting

Eliminazione combinatoria di formule

Eliminazione a campione di formule

Valori di similarità come vettori in input all'algoritmo

Conclusione

Bibliografia

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

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

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

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

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

[6] 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.