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 [3]

Dati $\{x_1, \ldots, x_n\}$ un campione di elementi appartenenti a un dominio X e $\{\mu_1, \ldots, \mu_n\}$ i rispettivi gradi di appartenenza a un fuzzy set sconosciuto A, duplice obiettivo dell'algoritmo è trovare il fuzzy set A, e inferire i parametri della funzione di appartenenza μ_A. Per spiegare la tecnica, immaginiamo di mappare i punti x_i su una circonferenza 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 possiamo dare una prima formulazione:

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

$$s.t.$$

$$||x_i - a||^2 \leq R^2 + \xi_i$$

$$\xi_i \ \geq 0, C > 0$$

Ovvero, il problema diventa trovare la circonferenza di raggio minimo, al variare di R^2 e a, che raccolga i punti rappresentanti gli elementi del campione. Le ξ_i, la cui sommatoria minimizzo nella funzione obiettivo, corrispondono a variabili di slack che vengono aggiunte per far rispettare i vincoli di appartenenza; il valore C funge da bilanciamento tra questo adattamento e il rispetto del vincolo, identificando se sia più importante mantenere vincoli stringenti o mappare tutti i punti nella circonferenza. In realtà, l'obiettivo è che la distanza dal centro della circonferenza rappresenti un grado di appartenenza

all'insieme fuzzy, e dunque andiamo a inserire le μ_i nella formulazione, che diventa, moltiplicando per μ_i:

$$minR^2 + C\sum_i(\xi_i + \tau_i)$$

$$s.t.$$

$$\mu_i||x_i - a||^2 \leq \mu_i R^2 + \xi_i$$

$$(1-\mu_i)||x_i - a||^2 \geq \mu_i R^2 - \tau_i$$

$$\xi_i \ \geq 0, \tau_i \ \geq 0, C > 0$$

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 circonferenza più piccola che contiene tutti i punti con membership 1;
- Se $\mu_i = 0$, è il primo vincolo a diventare ridondante, e la formulazione è volta a cercare punti che stanno fuori dalla ciconferenza;
- Se $\mu_i = \frac{1}{2}$, moltiplicando entrambi i vincoli per 2 si ottiene:

$$||x_i - a||^2 \leq R^2 + 2\xi_i$$

$$||x_i - a||^2 \geq R^2 - 2\tau_i$$

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 circonferenza. Rimane il problema di modellazione delle membership intermedie, che si affronterà in seguito. Ottenuta quest'ultima formulazione del problema di ottimizzazione, si risolve il problema duale, attraverso il metodo di Wolfe. Per farlo, costruiamo la funzione lagrangiana, sottraendovi obiettivo e vincoli, moltiplicati per altrettante variabili:

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

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

I vincoli richiedono che vengano messe a zero tutte le derivate parziali della lagrangiana rispetto al problema originale. Sostituendo il pattern

$\sum_i \alpha_i \mu_i - \beta_i(1-\mu_i)$ con ϵ_i, sapendo che $\sum_i \epsilon_i = 1$, a partire dal vincolo sulla derivata parziale rispetto ad a 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 circonferenza cercata. Il problema duale, da massimizzare, è dunque:

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

$$s.t.$$

$$\sum_i \epsilon_i = 1$$

$$C \geq 0, 0 \leq \alpha i \ \leq C, 0 \leq \beta_i \ \leq C$$

Una volta risolto il problema e ottenute le ϵ_i^* ottimali, 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 raggio) e x fornito. Inoltre, se si prende un i tale che $0 \leq \alpha_i \ \leq C$ o $0 \leq \beta_i \ \leq C$, allora $R^2(x) = R^*$, raggio ottimale della sfera. Perciò, dato un punto, è possibile stimare il suo grado di appartenenza alla sfera: se $R(x)^2 > R^*$ ha membership $> \frac{1}{2}$, se $R(x)^2 = R^*$ la membership è $\frac{1}{2}$, se $R(x)^2 < R^*$. Se si trova una funzione che si comporta in questo modo, si può calcolare la membership in ogni punto.

In realtà, nell'algoritmo i punti non vanno visti come inscritti in una circonferenza ma in una sfera, e mappati nello spazio tridimensionale tramite una trasformazione Φ, che viene definita kernel. Anche non conoscendo la funzione specifica, ma essendo consapevoli della sua famiglia, si può ricavare il prodotto scalare. Ad esempio, nel caso sia una funzione polinomiale di grado al massimo p, $\Phi(x_i) \cdot \Phi(x_j) = (1 + x_i + x_j)^p$. Di conseguenza, la formulazione finale della funzione obiettivo diventa:

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

2.1.1 Utilizzo della libreria mulearn

Per eseguire gli esperimenti, è stata utilizzata la libreria mulearn.[1] 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 input, il FuzzyInductor è in grado, ricevendo un vettore di valori, di inferire una funzione di appartenenza al fuzzyset, e quindi di restituire, tramite la funzione predict, un grado di appartenenza per ogni input dato. Per la costruzione del FuzzyInductor, devono essere passati una serie di parametri; il primo di questi 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 set semplice, ce ha valore 1 se l'elemento 0 appartiene all'insieme, 0 se non vi appartiene;
- *Linear Fuzzifier*: corrisponde a un fuzzy set la cui membership decresce linearmente da 1 a 0;
- *Exponential Fuzzifier*: la membership diminuisce da 1 a 0 in maniera esponenziale;
- *Quantile Constant Piecewise Fuzzifier*: corrisponde a un fuzzy set che ha una funzione di membership costante definita 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;
- *Quantile Linear Piecewise Fuzzifier*: la funzione definita a tratti è in questo caso lineare.

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.

2.2 Il Kernel

Il parametro kernel definisce quale sarà la funzione attraverso cui i valori verranno mappati nello spazio tridimensionale. La libreria mulearn implementa:

- *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.

2.2.1 Kernel precomputato

Per ragioni di tempi di computazione (i punti da mappare, nel caso di un insieme di formule, non corrispondono a valori numerici di cui i 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 calcolare questo valore, ovvero trovare un criterio per definire il grado di similarità tra due assiomi, si utilizza la similarità di Jaccard, descritta in [4]. 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:

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

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

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

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 seguente.

	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

2.2.2 Kernel alternativi

Nel corso degli esperimenti sono state utilizzate altre definizioni di similarità, diverse da quella di Jaccard, descritte in [2].

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. Ciò però nasconde una problematica: quando un assioma è il negato dell'altro la funzione restituisce un valore di similarità alto, cosa che chiaramente non dovrebbe accadere. Ciò fa sì che sia opportuno 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. Seguono i risultati ottenuti.

	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

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, e trascurando i caratteri extra della stringa più lunga. Definendo H come la distanza di Hamming calcolata in questo modo, si fa la stessa distinzione di prima 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 è stata utile per computare

direttamente il kernel, 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 condotto, invece, ai seguenti risultati:

	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

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. 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.

	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

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

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

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

[4] 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.