Documentazione Caso di Studio
Ingegneria della Conoscenza
A.A. 2024/2025
SentiView

Autore:

Pasquale Caringella, 777038, p.caringella@studenti.uniba.it

Repository:

https://github.com/CariP03/Uniba_ICon_SentiView.git

Sommario

Introduzione

Sommario

SentiView è un prototipo di *sentiment analysis* per recensioni IMDb in lingua inglese. Tale dominio presenta sfide peculiari; come la presenza di termini generalmente fortemente negativi che in certi generi assumono connotazione positiva o una forte soggettività del linguaggio.

Il progetto punta ad esplorare il concetto di un classificatore *rule-based* (basato su regole logiche implementate in Prolog) che non si basa su rappresentazioni statiche come TF-IDF; ma sfrutta regole logiche per le predizioni. Un modello così costruito risulterebbe facilmente interpretabile e richiederebbe relativamente pochi dati etichettati e tempi di training ridotti rispetto a modelli più complessi.

Inoltre, si vuole valutare l'apporto che features estratte dalla base di conoscenza possano avere su modelli tradizionali che sfruttano vettorizzazione TF-IDF. Questo permette, inoltre, di confrontare il classificatore *rule-based* con modelli tradizionali addestrati sugli stessi dati.

Dataset

Per questo progetto è stato utilizzato il dataset "IMDb Movie Reviews" [1], una raccolta ampiamente usata nel campo della *sentiment analysis*. Il dataset contiene recensioni testuali in lingua inglese tratte dal sito IMDb e annotate manualmente con etichette binarie che indicano il sentimento positivo o negativo dell'autore della recensione. Le recensioni sono bilanciate tra le due classi, facilitando il training e la valutazione dei modelli.

Il dataset presenta una notevole varietà linguistica e stilistica, con recensioni che spaziano da poche righe a diversi paragrafi. Tale eterogeneità rende il task di classificazione più realistico e complesso.

Elenco Argomenti di Interesse

- **Ontologie**
- **Rappresentazione e Ragionamento Relazionale**
- **Apprendimento Supervisionato**

Ontologia

Sommario

Questa sezione descrive l'ontologia utilizzata per estrarre la conoscenza sulla quale si base la *knowledge base* (KB); in particolare, come tale conoscenza è estratta.

Strumenti Utilizzati

- WordNet [2]; ontologia di ordine superiore della lingua inglese
- SentiWordNet [3]; risorsa lessicale che associa polarità a *synsets* WordNet [2]
- Natural Language Toolkit (NLTK) [4]; libreria python per NLP

Decisioni di Progetto

SentiWordNet [3] assegna a ciascun *synset* un punteggio di polarità, compreso nell'intervallo [-1, 1]. È importante sottolineare che una stessa parola (*word form*) può essere associata a più *synset*. Questo è un aspetto importante da considerare in quanto *synset* diversi hanno polarità diverse e una attribuzione errata può impattare sulle performance del classificatore.

Si sono considerate diverse soluzioni e se ne sono valutati vantaggi e svantaggi. Si è valutato l'utilizzo di algoritmi di disambiguazione, di considerare unicamente il primo *synset* e di effettuare una media tra le polarità di tutti i *synsets.* Il primo metodo è stato scartato in quanto avrebbe richiesto una elevata mole di risorse in fase di predizione. Il secondo metodo non è stato adottato perché considerato troppo aleatorio e privo di fondamento. L'ultima tecnica è risultata un'adeguata via intermedia tra la precisione del primo metodo e l'efficienza del secondo.

Tali punteggi costituiscono i fatti della KB; come approfondito nella sezione successiva.

Knowledge Base

Sommario

Questa sezione descrive la struttura della *knowledge base*. Illustra come i fatti sono inseriti al suo interno e il *pre-processing* applicato ai *tokens* al fine di avere il maggior numero di corrispondenze durante la fase di predizione.

Strumenti Utilizzati

- Prolog [5]; linguaggio di programmazione in cui è scritta la KB
- SpaCy [6]; libreria per il NLP
- Contractions [7]; libreria per la conversione di forme contratte

Decisioni di Progetto

I fatti della KB sono descritti da tre predicati:

- word_score/2: Rappresenta l'associazione tra *tokens* e il loro score di polarità. Sono il fulcro del classificatore e la base su cui sono calcolate tutte le metriche.

```
word_score("tuneful", 0.5000).
word_score("mayo", -0.1250).
word_score("remorseful", -0.7500).
word_score("tremble", -0.1875).
word_score("procrastinate", -0.1250).
word_score("pedestal", 0.1667).
word_score("unattainable", -0.6250).
```

- Intensifier/2: Rappresenta il moltiplicatore associato a ciascun intensificatore. Gli intensificatori sono *tokens* speciali che non sommano la loro polarità; ma moltiplicano quella del *token* successivo secondo il moltiplicatore indicato nel predicato.

```
% Intensifiers
intensifier("very", 1.5).
intensifier("extremely", 2.0).
intensifier("really", 1.3).
intensifier("so", 1.2).
intensifier("quite", 1.2).
intensifier("too", 1.2).
```

- Negator/1: Indica che il *token* è un negatore. I negatori non sommano la loro polarità; ma invertono quella del *token* successivo.

```
% Negators
negator("no").
negator("little").
negator("scarcely").
```

Ad eccezione degli intensificatori e dei negatori, i word_score sono creati basandosi sui dati di training. Per ogni recensione sono estratti i *tokens* e calcolati i punteggi di polarità. Questi sono quindi inseriti nella base di conoscenza. La creazione di questo *lexicon* è, di fatto, la fase di addestramento del classificatore.

Di particolare rilevanza è come le parole presenti nelle recensioni sono trasformate in *token*. Un *tokenizer* adeguato risulta in un numero più elevato di corrispondenze tra i termini presenti nelle recensioni e quelli presenti nella KB; inoltre può facilitare la gestione delle regole logiche.
Il prototipo utilizza un *tokenizer* personalizzato che si sviluppa in più fasi:

1. Il testo è convertito in minuscolo;
2. Le contrazioni sono espanse; ovvero espressioni del tipo “don’t” sono convertite in due *token* “do” e “not”;
3. Utilizzando spaCy [6] sono identificati tutti i termini che sono negati;
4. Negatori e intensificatori non sono ulteriormente trasformati;
5. Sono ricavati i lemmi dei *tokens;*
6. Gli avverbi sono convertiti ad aggettivi;
7. I lemmi negati sono convertiti alla forma “not_{lemma}”; questo è fondamentale per la gestione delle negazioni.

Valutazione e Considerazioni

In alternativa alla generazione del *lexicon* sui dati di training, si potrebbe inserire l’intera risorsa di SentiWordNet [3] nella base di conoscenza; ma questo creerebbe una KB molto ampia e sarebbero presenti molti termini che nel dominio di applicazione non sono utilizzati o comuni. In compenso, non vi sarebbe una fase di addestramento del classificatore.

Al momento il processo di tokenizzazione è molto impattante sul tempo richiesto dal classificatore per effettuare le predizioni. Inizialmente si utilizzavano le funzioni offerte da NLTK [4] piuttosto che il più esoso parser di spaCy [6]. Purtroppo questo non permetteva di identificare forme di negazioni più complesse, come nel caso di “not really good”; infatti la negazione era attribuita a “really” piuttosto che “good”. Si è quindi deciso di preferire un processo più preciso a uno più rapido; come si vedrà in seguito questo sarà fattore decisivo nel confrontare il classificatore *rule-based* al modello di ML.

Regole e Query

Sommario

Questa sezione descrive le regole presenti nella KB e ne spiega il funzionamento. Si osserva anche come la KB è interrogata in Python e quali features sono estratte da essa.

Strumenti Utilizzati

- PySwip [8]; interfaccia Python-Prolog che permette di interrogare basi di conoscenza
- Matplotlib [9]; libreria per la visualizzazione di grafici e metriche

Decisioni di Progetto

Si procede ora a descrivere ogni regola presente nella base di conoscenza:

```
% Base: if the word has a score, otherwise 0
token_base_score(Word, Score) :-
    word_score(Word, Score), !.
token_base_score(Word, Score) :-
    string_concat("not_", Base, Word),
    word_score(Base, BaseScore),
    Score is -BaseScore, !.
token_base_score(_, 0).
```

- Questo predicato determina il punteggio di polarità assegnato a una parola. Sono presenti tre diverse clausole che coprono tutti i casi possibili. Ogni clausola termina con il carattere “!” che indica a Prolog [5] di non provare altre clausole nel momento in cui una viene derivata.

  La prima clausola restituisce il punteggio di polarità di una parola qualora questa sia presente nella KB. La seconda clausola gestisce il caso in cui una parola sia negata; per fare ciò recupera la parola base e ne inverte il punteggio. Infine, la terza clausola gestisce il caso in cui una parola non è presente nella KB.

```
:- dynamic token_at/2.
```

- Questo predicato è di tipo dinamico; cioè i fatti che lo coinvolgono sono specificati a *runtime*. Esso è utilizzato per inserire la recensione nella KB. Ogni *token* che compone la recensione è inserito indicando la sua posizione e il *token* stesso.

```
% Manage intensifiers
token_effective_score(Index, ScoreEff) :-
    token_at(Index, Word),
    token_base_score(Word, Base),
    % Intensifiers
    ( Index > 1,
      Prev is Index - 1,
      token_at(Prev, PrevW),
      intensifier(PrevW, M) ->
        ScoreEff is Base * M ;
        ScoreEff = Base ).
```

- Questa regola gestisce gli intensificatori.

  Dato un indice, recupera il punteggio di base del *token* presente in quell'indice e se l'indice è maggiore di 1, ovvero non è il primo *token*, recupera il token precedente. Se il *token* precedente è un intensificatore, moltiplica il punteggio di base per l'effetto del moltiplicatore.

```
% Total Sum
sentiment_sum(Sum) :-
    findall(S, token_effective_score(_, S), Scores),
    sum_list(Scores, Sum).
```

- Questo predicato restituisce la prima metrica utilizzata nel classificatore: la somma della polarità di tutti i *token*.

  Per fare ciò è utilizzato il predicato findall il quale cerca tutti i valori di S per cui token_effective_score(_, S) sia vero e li raccoglie nella lista Scores.
  Quindi tutti i punteggi raccolti nella lista sono sommati mediante sum_list e il risultato è Sum.

```
% Non-zero tokens count
sentiment_count_nonzero(Count) :-
    findall(1, (token_effective_score(_, S), S \= 0), L),
    length(L, Count).
```

- Questa regola restituisce il numero di *token* il cui valore di polarità è diverso da 0.

  Come nel predicato precedente è utilizzato il findall; ma in questo caso all'interno della lista sono inseriti degli 1 e non la polarità dei *token*. Quindi è restituita la lunghezza della lista.

```
% Total token count
sentiment_total_tokens(Tot) :-
    findall(1, token_at(_, _), L),
    length(L, Tot).
```

- Questo predicato restituisce il numero totale di *token* presenti in una recensione.

```
% Classifier
is_positive :-
    sentiment_sum(S),
    threshold(T),
    S >= T.
```

- Questo predicato è il fulcro del classificatore è restituisce vero se la somma delle polarità dei *token* è maggiore od uguale a una certa soglia. La soglia, come si vedrà in seguito, è determinata sperimentalmente.

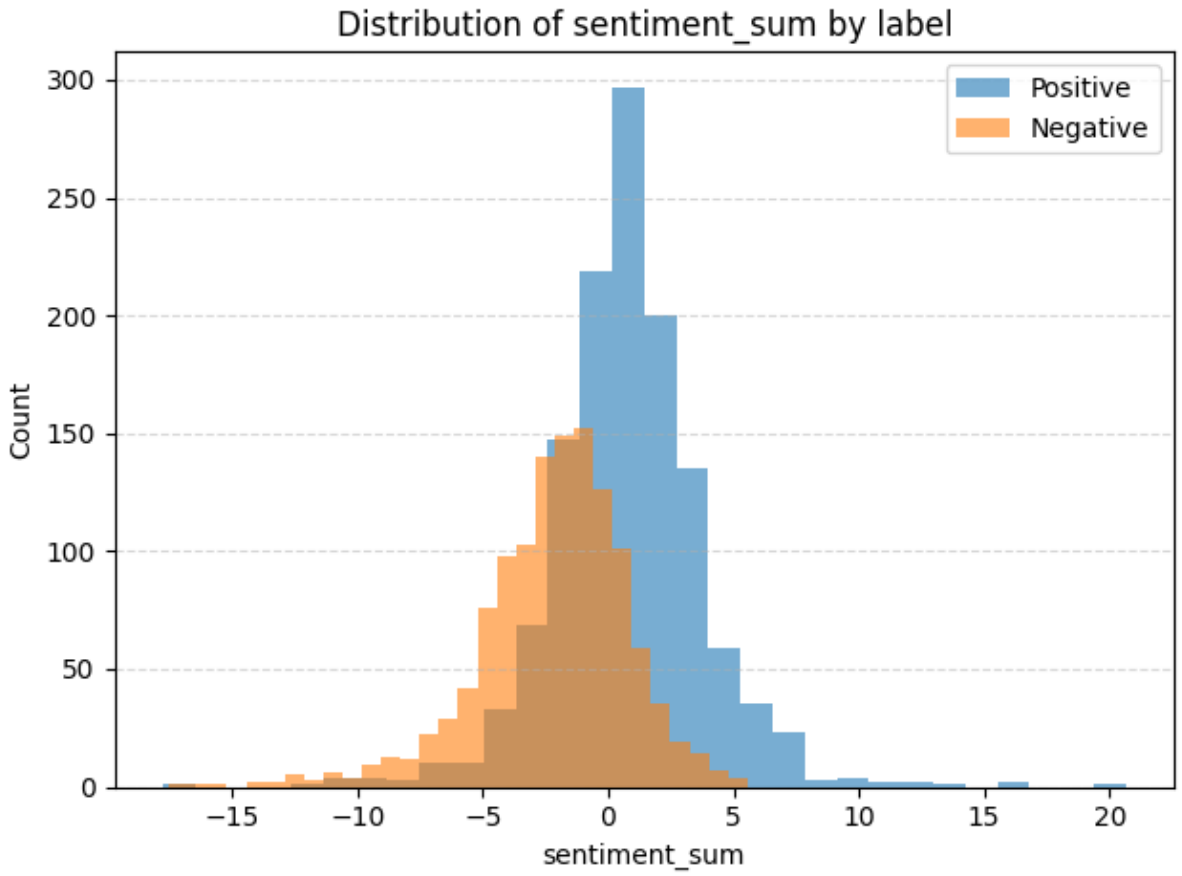
Oltre a queste tre metriche calcolate in Prolog [5], sono presenti altre due metriche che combinano le tre basilari:

- Avg_nonzero: il valore medio di polarità dei *token* calcolato escludendo quelli aventi polarità nulla;
- Ratio: il rapporto tra il numero di *token* aventi polarità non nulla e il numero totale di *token*.

Valutazione e Considerazioni

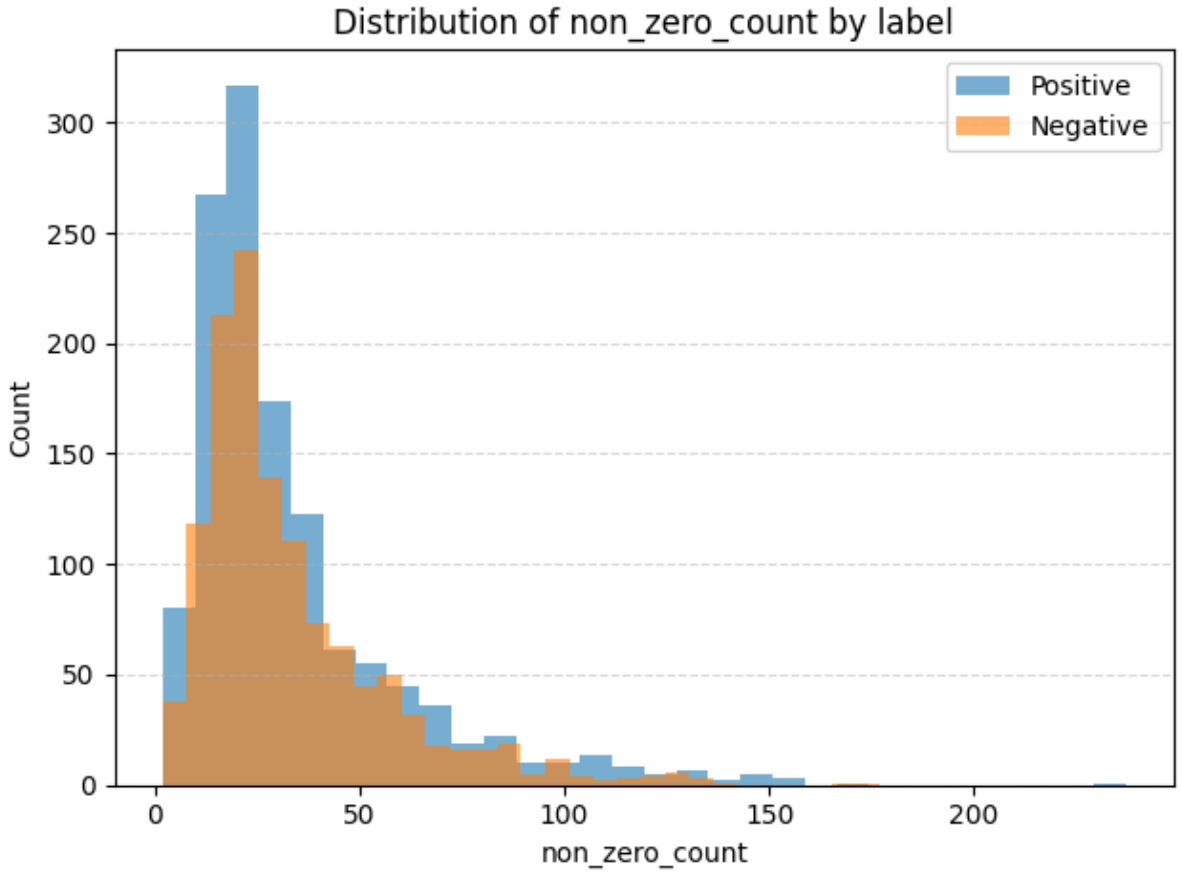
È di fondamentale importanza comprendere le metriche estratte dalla KB; in particolare osservare la loro distribuzione e le statistiche associate per intuire la loro utilità e quanto discriminati siano nell'identificare il sentimento associato alle recensioni.

Seguono le osservazioni sulle metriche del classificatore *baseline*. Queste metriche sono calcolate su un set di validazione estratto dal set di training mediante strategia *Hold-Out*. Sono stati, quindi creati, un set di addestramento di 22500 esempi e un set di validazione di 2500 esempi.
In questa fase del progetto è stata scelta questa strategia, piuttosto che una *Cross-Validation*, in quanto più rapida e sufficientemente precisa per un modello *baseline*; inoltre, considerato l'elevato numero di esempi presenti nel set di validazione, essa fornisce risultati adeguati.



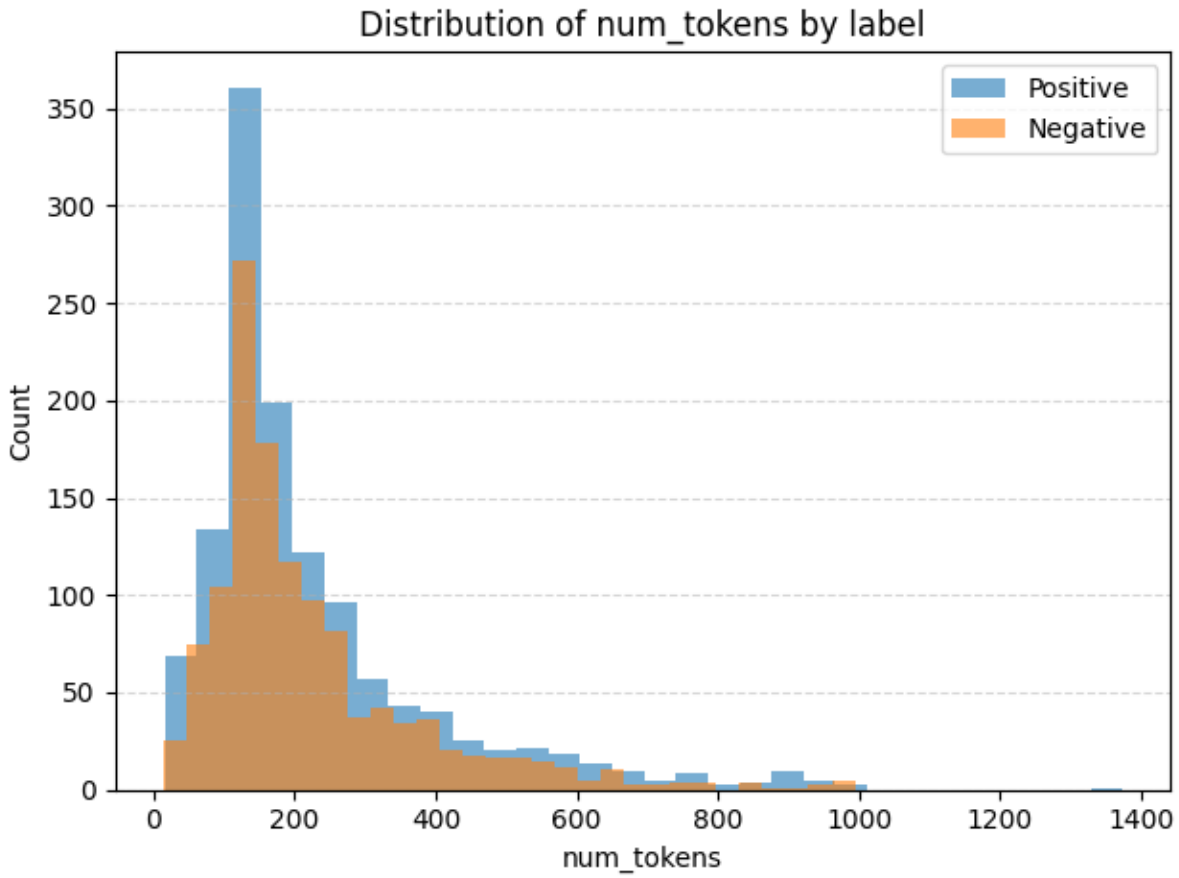
label	count	mean	std	min	25%	50%	75%	max
0	1235.0	-2.198099	2.973206	-17.5257	-3.78355	-1.8926	-0.27775	5.55300
1	1265.0	0.757176	3.044595	-17.7789	-0.91010	0.7643	2.31395	20.65038

- Si osserva che la metrica “sentiment_sum” è abbastanza efficace nel discriminare gli esempi; infatti nel grafico è evidente che le due etichette abbiano distribuzioni diverse, come mostrato anche dalle differenze nelle medie.

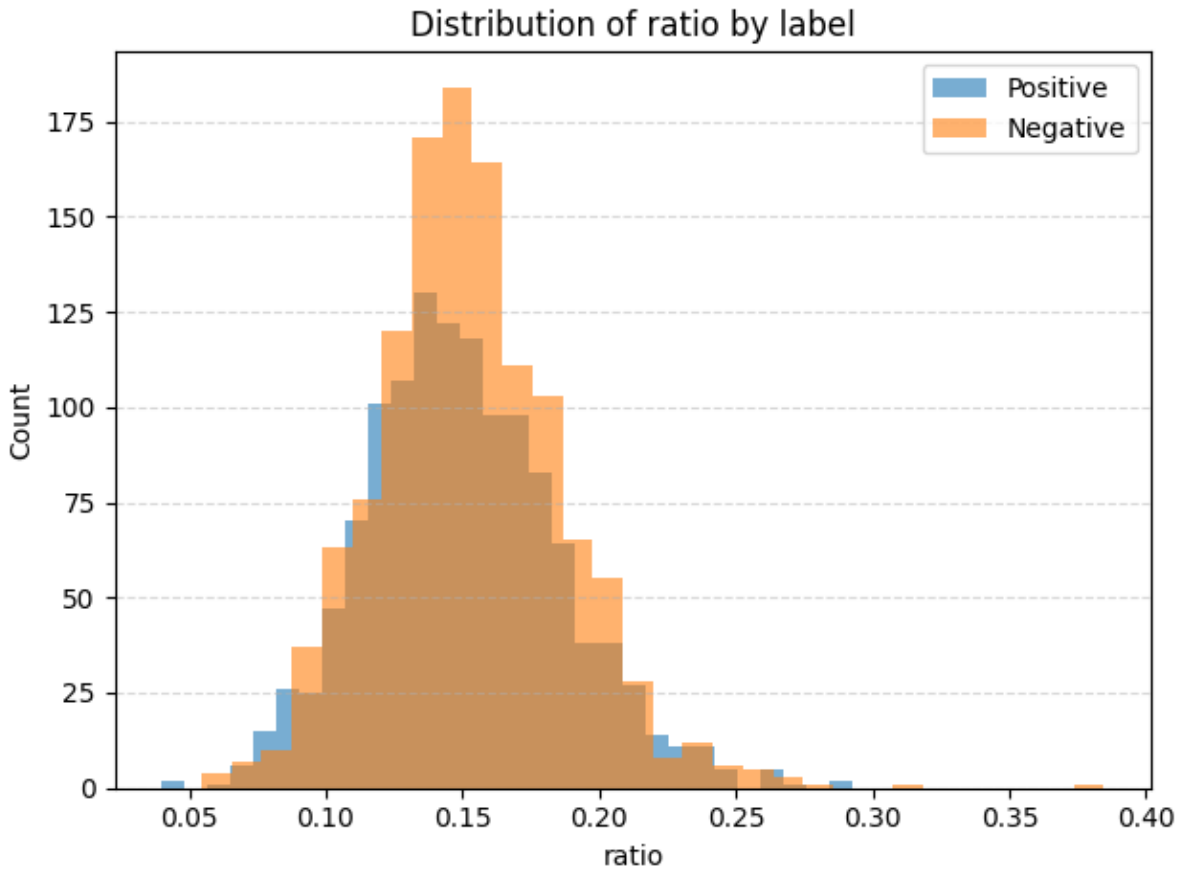


label	count	mean	std	min	25%	50%	75%	max
0	1235.0	33.307692	23.628285	2.0	18.0	26.0	42.0	177.0
1	1265.0	33.682213	26.863082	2.0	17.0	25.0	40.0	237.0

0

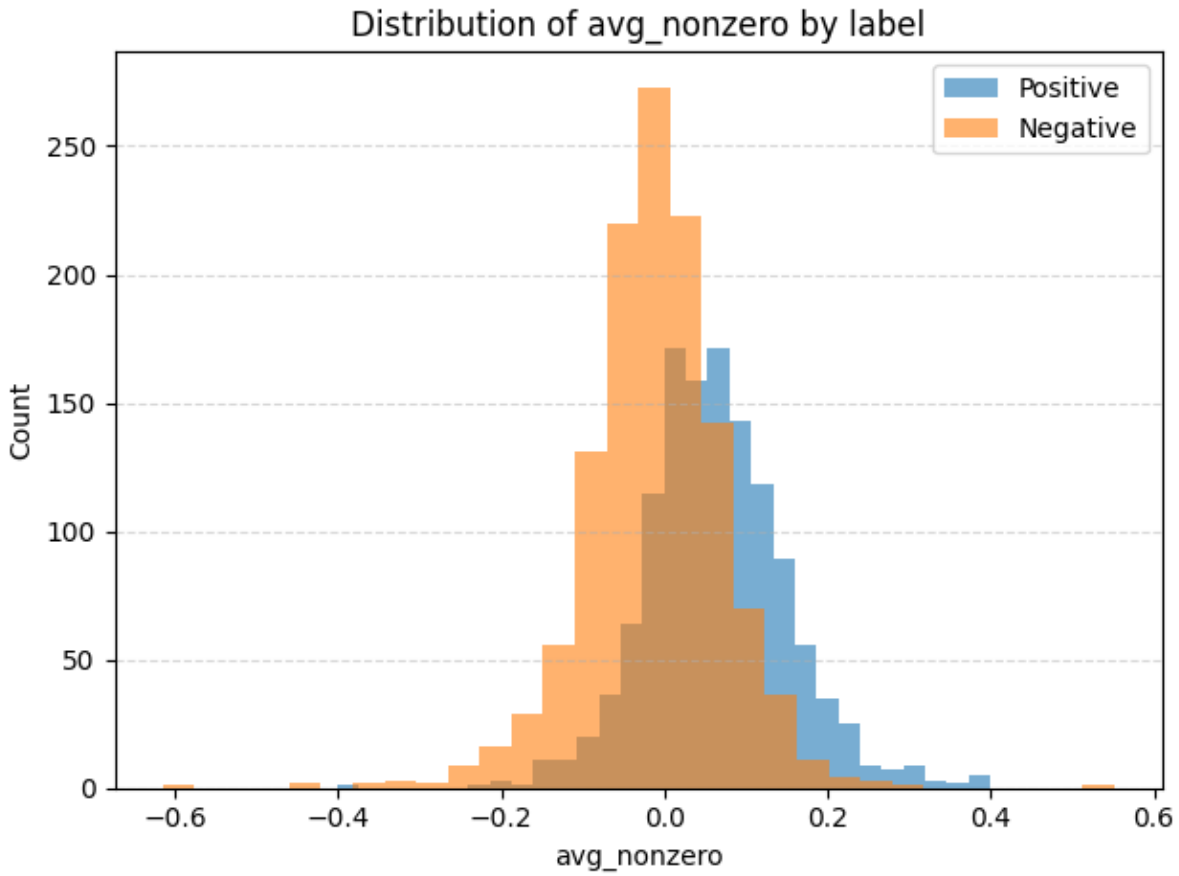


label	count	mean	std	min	25%	50%	75%	max
0	1235.0	221.119838	157.093759	13.0	125.0	168.0	265.5	994.0
1	1265.0	225.949407	173.414789	16.0	122.0	163.0	269.0	1374.0



label	count	mean	std	min	25%	50%	75%	max
0	1235.0	0.152689	0.034640	0.054545	0.130993	0.150510	0.173112	0.384615
1	1265.0	0.150415	0.035502	0.040000	0.126183	0.148148	0.172840	0.292308

- È evidente che “ratio”, “non_zero_count” e “num_tokens” non siano metriche utili al task; infatti le distribuzioni sono sovrapposte e non forniscono informazioni aggiuntive.



label	count	mean	std	min	25%	50%	75%	max
0	1235.0	-0.013132	0.086401	-0.615217	-0.059926	-0.010979	0.037041	0.551154
1	1265.0	0.063768	0.086949	-0.400480	0.009297	0.058679	0.114479	0.399689

- Questa metrica derivando da “sentiment_sum” e “non_zero_count”, ed essendo quest’ultima pressoché identica per entrambe le classi, non aggiunge informazioni aggiuntive alla prima e ne è equivalente.

Nonostante alcune metriche non siano significative in questo contesto, ciò non esclude che possano esserlo in altri contesti. Infatti, l’ipotesi iniziale supponeva che recensioni negative potessero avere un numero maggiore di termini polarizzanti rispetto alle recensioni positive; risultando quindi in distribuzioni diverse. Si è invece osservato che in tale dominio questo non sia vero.

Classificatore Rule-Based

Sommario

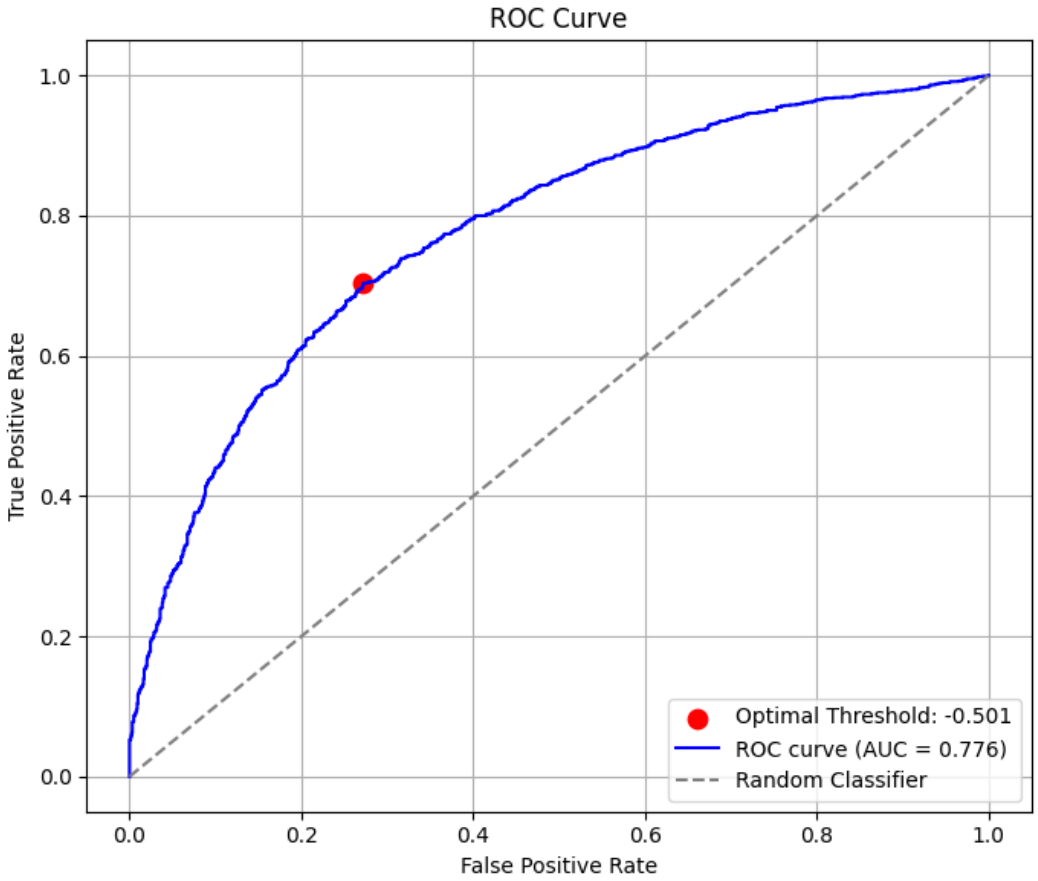
Questa sezione esplora la calibrazione del classificatore *rule-based*; in particolare come è stata scelta la soglia di classificazione. Sono anche analizzate le performance del classificatore.

Strumenti Utilizzati

- Scikit-learn [10]; libreria per il machine learning

Decisioni di Progetto

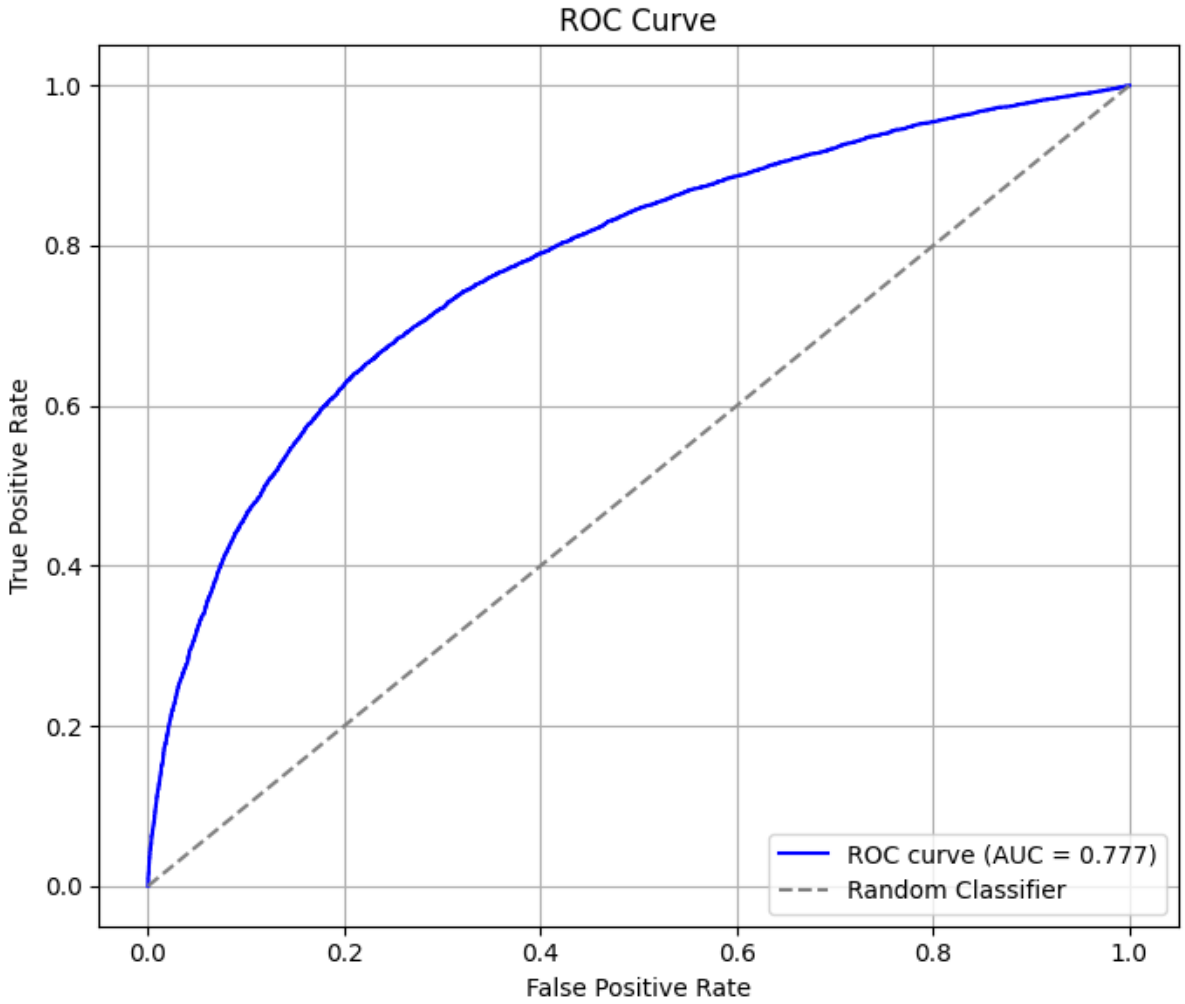
Lo strumento utilizzato per determinare la soglia di classificazione è la curva ROC. Essa è stata calcolata sui valori di "sentiment_sum" sull'intero set di validazione.



Quindi è stato calcolato l'indice di Youden [11] per ogni punto della curva e utilizzato il massimo indice come soglia.

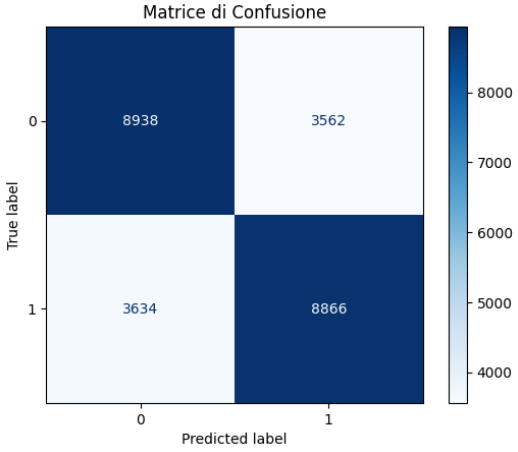
Valutazione e Considerazioni

Dopo l'individuazione della soglia, il classificatore è stato nuovamente addestrato, ovvero il lexicon rigenerato, sull'intero set di addestramento, incluso il set di validazione. Il modello è stato quindi testato sul test set che consta di 25000 esempi.



È stato ricalcolata la curva ROC e presa in considerazione la metrica AUC. Questa è pari a circa 0.8; un valore soddisfacente per un classificatore che non si basa su machine learning e che, in linea di massima, è privo di addestramento.

Considerato il task, altre metriche sono state prese in considerazione; quali accuratezza, precisione e recall. Segue anche una matrice di confusione dalla quale si evince che il classificatore performa equamente sui due tipi di recensioni.




```
Accuracy: 0.712
Precision: 0.713
Recall: 0.709
```

Apprendimento Supervisionato

Sommario

Questa sezione esplora la scelta e l'addestramento di un modello di machine learning per il task di classificazione. In aggiunta, si valuta l'effetto delle feature estratte dalla KB sulle performance del modello.

Strumenti Utilizzati

- LogisticRegression da Scikit-learn
- TfidfVectorizer da Scikit-learn
- StandardScaler da Scikit-learn
- GridSearchCV da Scikit-learn [10]

Decisioni di Progetto

Come modello è stato scelto un modello di regressione logistica. Questo per diversi motivi: esso è facilmente interpretabile, efficiente su feature sparse come quelle generate dalla vettorizzazione TF-IDF e costituisce la baseline nel NLP. Inoltre ha pochi iperparametri; ciò ha ridotto notevolmente i tempi di esplorazione, permettendo anche una *grid search.*

In aggiunta alle feature estratte dalla KB, è stato utilizzato un TfidfVectorizer per ottenere features aggiuntive; come di prassi in questa tipologia di task.

Al fine di addestrare ed utilizzare il modello, è stata costruita una pipeline costituita da due passi: un primo passo di pre-processing che contiene il TfidfVectorizer e uno StandardScaler per le restanti feature; ed un secondo passo di addestramento del modello.

Il dataset è stato suddiviso in 60% training, 20% validazione e 20% test; ovvero 30000 esempi per il training e 10000 esempi ciascuno per validazione e test. La suddivisione è stata stratificata così da mantenere la proporzione delle classi.

Gli iperparametri sono stati scelti mediante *Cross-Validation* a 3 *fold*. In particolare, si sono valutate 6 diverse possibili combinazioni di iperparametri: due correlate al TfidfVectorizer

con la scelta del range degli *n-grams* e tre correlate al fattore C del modello; ovvero l'intensità della regolarizzazione. Segue una griglia con le 6 possibili combinazioni esplorate.

N-grams Range	Min Document Freq	Max Document Freq	C
(1, 1)	5	0.9	0.1
(1, 1)	5	0.9	1
(1, 1)	5	0.9	10
(1, 2)	5	0.9	0.1
(1, 2)	5	0.9	1
(1, 2)	5	0.9	10

I migliori iperparametri sono risultati i seguenti:

(1,2)	5	0.9	10

Individuati i migliori iperparametri, questi sono stati validati sul set di validazione. In seguito, il modello è stato nuovamente addestrato su training e validation e testato sul test set.

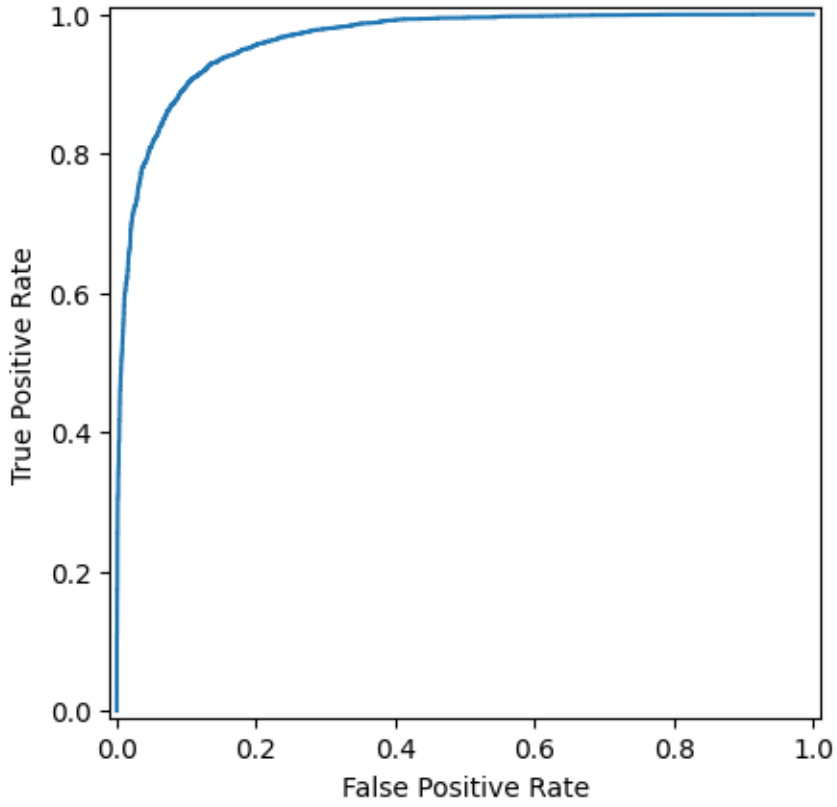
Al fine di valutare l'apporto delle feature estratte dalla KB, sono stati addestrati altri due modelli: uno con solo vettorizzazione TF-IDF ed uno con solo feature estratte dalla KB. Questi sono stati testati sul set di validazione.

Inoltre, si è costruito un classificatore *rule based* avente come *lexicon* i pesi estratti dalla vettorizzazione TF-IDF del modello di regressione logistica.

Valutazione e Considerazioni

Le metriche calcolate sono le stesse di quelle utilizzate per il classificatore *rule-based*; ovvero AUC, accuratezza, precisione e *recall*.

Per il modello TF-IDF + KB, seguono la curva ROC e le metriche.



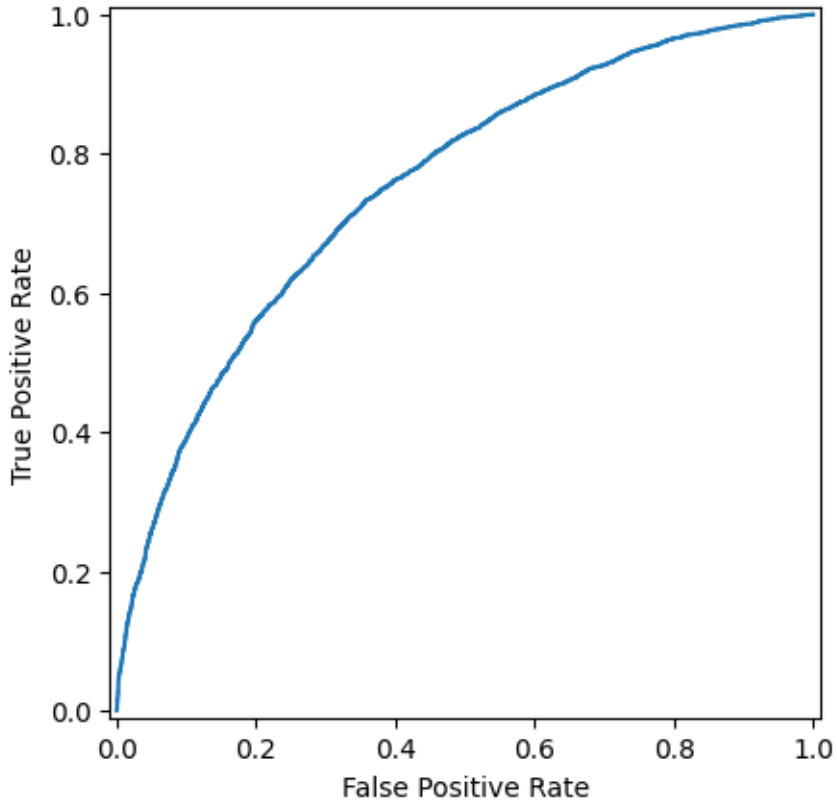

```
Acc val: 0.8995
AUC val: 0.9637908399999999
LogLoss: 0.245416453215184
              precision    recall  f1-score   support

           0       0.90      0.90      0.90      5000
           1       0.90      0.90      0.90      5000

    accuracy                           0.90     10000
   macro avg       0.90      0.90      0.90     10000
weighted avg       0.90      0.90      0.90     10000
```

- Osserviamo un valore di AUC pari a 0.96 e un'accuratezza del 90%. Questi sono valori che si allineano con quelli presenti in letteratura e solo modelli più complessi di deep learning ottengono performance migliori.

Per il modello solo KB, osserviamo:



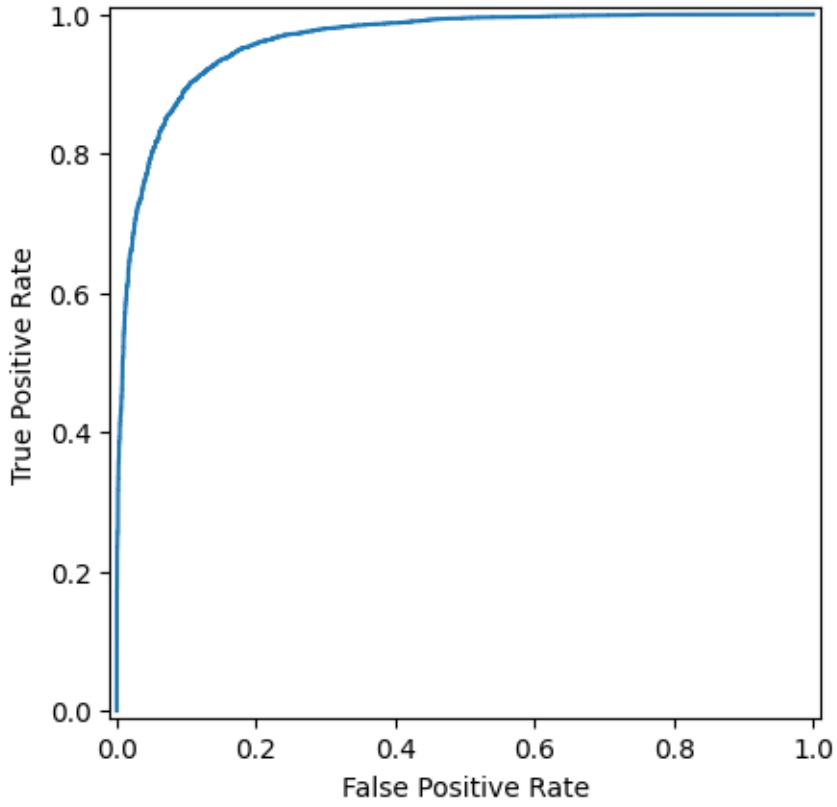

```
Acc val: 0.6855
AUC val: 0.7532050000000001
LogLoss: 0.5888985264621668
              precision    recall  f1-score   support

           0       0.68      0.69      0.69      5000
           1       0.69      0.68      0.68      5000

    accuracy                           0.69     10000
   macro avg       0.69      0.69      0.69     10000
weighted avg       0.69      0.69      0.69     10000
```

- Notiamo che tali performance non sono dissimili da quelli ottenuti dal classificatore *rule-based*. Possiamo concludere che le metriche utilizzate non riescono a catturare pattern troppo complessi o le sfumature presenti nei testi.

Per il modello solo TF-IDF, segue:



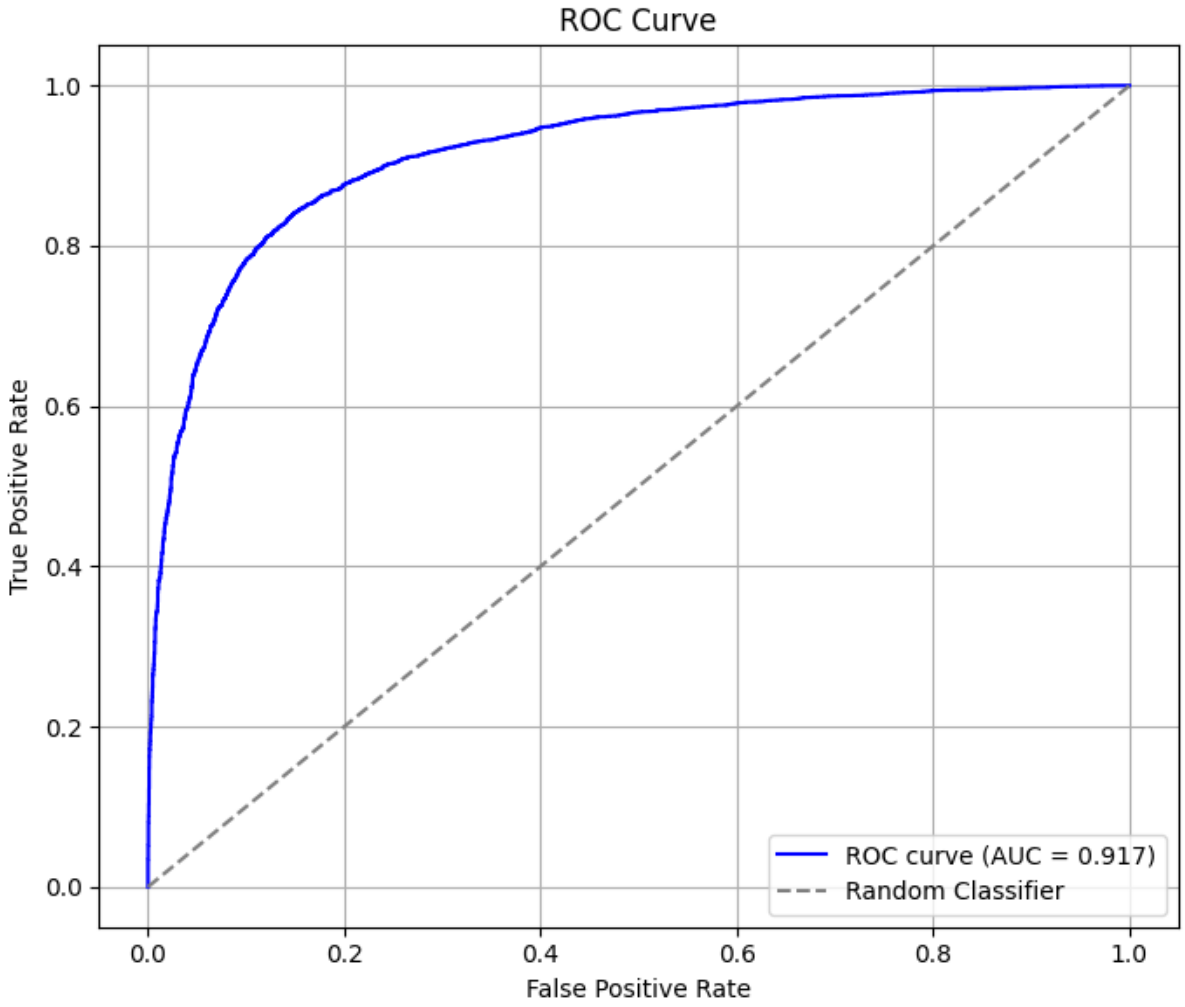

```
Acc val: 0.8963
AUC val: 0.9616990000000001
LogLoss: 0.283430585858576
              precision    recall  f1-score   support

           0       0.91      0.89      0.90      5000
           1       0.89      0.91      0.90      5000

    accuracy                           0.90     10000
   macro avg       0.90      0.90      0.90     10000
weighted avg       0.90      0.90      0.90     10000
```

- Si osserva come le performance di questo modello siano le stesse di quello che sfrutta anche la KB. Ne consegue che le feature estratte dalla KB non aggiungono informazioni rispetto alla vettorizzazione TF-IDF.

Per il classificatore *rule-based* con i pesi estratti dalla vettorizzazione TF-IDF, segue:




```
Optimal threshold: -1.366
Accuracy for optimal threshold: 0.843
Precision for optimal threshold: 0.836
Recall for optimal threshold: 0.855
```

- Questo modello ottiene performance non lontane da quelle ottenute mediante ML; ma rinuncia al basso costo di addestramento; infatti è necessario addestrare un modello di ML per ottenere i pesi della vettorizzazione. Inoltre, misurati i tempi di predizione sul test set, questo risulta notevolmente più lento rispetto alla sua controparte di ML come si può osservare dalla tabella sottostante:

Tempi Modello ML solo TF-IDF	Tempi Classificatore Rule Based
324 secondi	837 secondi

- Questa differenza va attribuita all'*overhead* del calcolo delle metriche; infatti questa richiede un passaggio mediante Prolog [5] e l'interfaccia PySwip [8].

Conclusioni e Possibili Sviluppi

In conclusione, si è osservato che il modello di machine learning ottiene performance migliori del classificatore *rule-based*. Nonostante questo, l'esperimento fatto utilizzando i pesi della vettorizzazione TF-IDF per costruire la KB dimostra l'esistenza della possibilità di un classificatore *rule-based* in grado di performare alla pari di un modello di ML.

Vi sono svariati sviluppi possibili; soprattutto che puntino a migliorare la KB e le sue regole:

- Si potrebbe utilizzare una ontologia o una risorsa lessicale diversa; ad esempio VADER [12].
- Si può puntare a introdurre una logica più complessa che rimpiazzi il più esoso parser spaCy [6]; ottenendo così risultati migliori con costi ridotti. Ad esempio, gestendo le congiunzioni coordinanti avversative; infatti il vero pensiero dell'autore in una frase è generalmente espresso dopo di queste.
- Introdurre nuove metriche o combinare quelle già esistenti.

In generale, ciò a cui si dovrebbe puntare è un modello privo di addestramento che sfrutti conoscenza pregressa per effettuare le predizioni. È fondamentale valutare il guadagno ottenuto sfruttando strumenti più complessi ed esosi per il NLP; e, eventualmente, sostituirli con strumenti meno precisi ma che richiedano meno risorse.

Riferimenti Bibliografici

[1] A. L. Maas, R. E. Daly, P. T. Pham, D. Huang, A. Y. Ng e C. Potts, “Large Movie Review Dataset,” Stanford University AI Lab, 2011. [Online]. Available: https://ai.stanford.edu/~amaas/data/sentiment/ Accesso: 13 Giugno 2025.

[2] Princeton University, “About WordNet,” WordNet, Princeton University, 2010. [Online]. Available: http://wordnet.princeton.edu. Accessed: 13 Giugno 2025.

[3] A. Esuli and F. Sebastiani, “SentiWordNet: A Publicly Available Lexical Resource for Opinion Mining,” in Proceedings of the 5th Conference on Language Resources and Evaluation (LREC’06), Genova, Italia, Maggio 2006, pp. 417–422. [Online]. Available: http://www.lrec-conf.org/proceedings/lrec2006/pdf/384_pdf.pdf. Accessed: 13 Giugno 2025.

[4] S. Bird, E. Klein, and E. Loper, *Natural Language Processing with Python*. Sebastopol, CA, USA: O’Reilly Media, 2009.

[5] J. Wielemaker, T. Schrijvers, M. Triska, and J. Lager, “SWI-Prolog,” *Theory and Practice of Logic Programming*, vol. 12, no. 1–2, pp. 67–96, 2012. [Online]. Available: https://www.swi-prolog.org. Accessed: 13 Giugno 2025.

[6] Explosion, “spaCy: Industrial-strength Natural Language Processing in Python,” [Online]. Available: https://spacy.io. Accessed: 17 Giugno 2025.

[7] P. van Kooten, “contractions: Fixes contractions in text,” 2016. [Online]. Available: https://github.com/kootenpv/contractions. Accessed: 13 Giugno 2025.

[8] Y. Tekol, “PySwip: A Python - SWI-Prolog bridge,” 2008. [Online]. Available: https://github.com/yuce/pyswip. Accessed: 13 Giugno 2025.

[9] J. D. Hunter, “Matplotlib: A 2D Graphics Environment,” *Computing in Science & Engineering*, vol. 9, no. 3, pp. 90–95, 2007.

[10] F. Pedregosa et al., “Scikit-learn: Machine Learning in Python,” *Journal of Machine Learning Research*, vol. 12, pp. 2825–2830, 2011.

[11] W. J. Youden, “Index for rating diagnostic tests,” *Cancer*, vol. 3, no. 1, pp. 32–35, 1950.

[12] C. J. Hutto and E. Gilbert, “VADER: A Parsimonious Rule-based Model for Sentiment Analysis of Social Media Text,” in *Proceedings of the Eighth International AAAI Conference on Weblogs and Social Media*, Ann Arbor, MI, USA, 2014, pp. 216–225.