Relazione del progetto

Linguistica Computazionale II (2023/24)

UNIVERSITÀ DI PISA

Docenti:

Prof. Felice dell'Orletta

Prof. Simonetta Montemagni

Prof. Giulia Venturi

Consegna di: Francesco Maggio 605001

7 agosto 2024

Indice

Introduzione **2**

1 Classificatore basato su SVM lineari con Profiling-UD **3**

2 Classificatore basato su SVM lineari con n-grammi **8**

3 Classificatore basato su SVM lineari con word embeddings **11**

4 Fine tuning su Neural Language Model **15**

Introduzione

Questa è una relazione relativa al progetto svolto per il corso in Linguistica Computazionale II (2023/24).

La relazione tratterà del progetto e delle sue implementazioni, esponendo risultati e analisi.

La relazione sarà divisa in 4 sezioni, una per ogni compito.

Il task scelto è HaSpeeDe - Hate Speech Detection da EVALITA 2020 (link); in particolare, il sotto-task a cui il progetto fa riferimento è il sotto-task A, Hate Speech Detection, un task di classificazione binaria per determinare se il messaggio scelto contiene hate speech o meno[3].

Francesco Maggio (605001), laurea magistrale in Informatica umanistica

Capitolo 1

Classificatore basato su SVM lineari con Profiling-UD

Il primo compito chiedeva di sviluppare un classificatore basato su SVM lineari che prende in input una rappresentazione del testo estratta da Profiling-UD[1].

Partendo dal training set originale, *haspeede2_dev_taskAB.tsv*, composto da 6837 righe e 4 colonne (id, text, hs, stereotype), ho sviluppato del codice che scrivesse, in una cartella *profiling_input*, un file di testo (*.txt*) per ogni riga del dataset, nel seguente formato:

- **Nome del file**: id#hs (*id* rappresenta l'id numerico del messaggio, mentre *hs* è 0 se il messaggio non contiene hate speech e 1 nel caso contrario).
- **Contenuto del file**: text (che rappresenta il contenuto effettivo del messaggio). Il nome della colonna *text* nel file originale si chiamava

in realtà "text ", con uno spazio alla fine, il quale è stato rimosso per evitare confusione.

La colonna *stereotype* viene ignorata in quanto non rilevante ai fini della nostra ricerca.

Generati i file e compressi in una cartella *.zip*, li ho passati a Profiling-UD, il quale ha prodotto in output il file *11925.csv*, contenente la rappresentazione del testo basata solo su informazioni linguistiche non lessicali che cercavo.

A partire da questa, inserita nella cartella *profiling_output*, ho estratto le features e le labels (ovvero le classificazioni 0 o 1) di ogni documento.

Ho poi effettuato una divisione tra training e test set con un rapporto di 80/20 attraverso *train_test_split*, normalizzato i valori delle features (portandole a valori compresi tra 0 e 1) tramite *MinMaxScaler*, e addestrato un modello *LinearSVC* sui dati di training.

Facendo una prima previsione sul test set (e stampando classification report e confusion matrix), ho notato che il modello ha raggiunto un'accuracy e un F1-score weighted average del 66%, dati comunque superiori a quelli di una baseline posta attraverso l'utilizzo di un modello *DummyClassifier* con la strategia di predirre sempre la classe più frequente, il quale in questo caso ha previsto sempre la classe 0, ottenendo un'accuracy del 58% e un F1-score weighted average del 42%..

Ho poi provato a valutare il sistema attraverso un processo di 5-fold cross validation sull'intero training set, ottenendo i risultati mostrati in tabella 1.1. Da notare che, anche in questo caso, è stato considerato un modello *DummyClassifier* con strategia "most frequent" come baseline.

Fold	**Accuracy**	**Baseline**
1	0.66	0.57
2	0.67	0.60
3	0.68	0.62
4	0.66	0.59
5	0.66	0.57

Tabella 1.1: Accuracy e baseline per ciascun fold.

L'accuracy del report di classificazione complessivo è risultata del 67%, al pari dell'F1-Score weighted average, quindi ho notato che l'applicazione di una 5-fold cross validation ha aumentato solo marginalmente le prestazioni del primo tentativo.

Ho inoltre estratto anche il test set ufficiale del task, per poter valutare su di esso il modello. Il task dispone in realtà di due test sets differenti: uno, più piccolo, nell'ambito delle news (con 500 righe), e uno, più grande, nell'ambito dei tweet (con 1263 righe). Il training set originale è composto da tweet, quindi l'idea dietro questa scelta è stata quella di poter valutare un modello sia dentro al dominio che fuori. Io ho deciso di importare entrambi i test set in modo da poter avere un riscontro su tutti e due.

Per farlo, ho seguito un procedimento molto simile a quello effettuato per il set originale, con la differenza che, una volta ottenute le rappresentazioni

testuali da parte di Profiling-UD, il numero di features non corrispondeva a quello della rappresentazione originale (124 e 133 contro 136). Ho affrontato dunque questa problematica facendo un confronto, per ognuno dei due test set, con il training set originale:

- Per il test set news con 500 righe, la differenza con il set originale erano solo alcune colonne di features presenti in quest'ultimo, ma non nel test set. Ho deciso, in questo caso, di aggiungere le colonne di features mancanti al test set, riempiendole con 0.0.

- Per il test set tweets con 1263 righe, similmente al caso precedente, la principale differenza con il set originale erano alcune colonne di features presenti solo in quest'ultimo. In questo caso, però, una colonna era presente esclusivamente nel test set, e non nel set originale. Ho deciso dunque di rimuovere quella colonna, e di aggiungere le rimanenti, riempiendole con 0, come per il caso precedente.

Risolto questo problema, ho potuto calcolare le metriche di valutazione sui due test sets, con risultati però abbastanza deludenti: un'accuracy del 63% per il test set news e del 50% per il test set tweets. Per entrambi i test set, ho notato un grosso sbilanciamento verso la classe 0 (recall del 98% e del 91%, contro 2% e 7% della classe 1), mostrando quindi un grosso bias verso la classe maggioritaria. Le accuracy risultano quindi abbastanza ingannevoli, come possiamo notare anche dagli F1 score decisamente non ottimali (41% macro average e 51% weighted average per il test set 1 contro 38% e 39% per il test set 2). I risultati si sono mostrati dunque poco soddisfacenti.

Un'ultima analisi è stata l'estrazione delle 15 features più importanti per la classificazione, estratte attraverso un singolo vettore di coefficienti, in quanto il nostro task riguardava una classificazione binaria, e stampate come bar chart, in figura 1.1, per darne una visualizzazione. Notiamo qui come *lexical_density* risulti essere di gran lunga il parametro più rilevante.

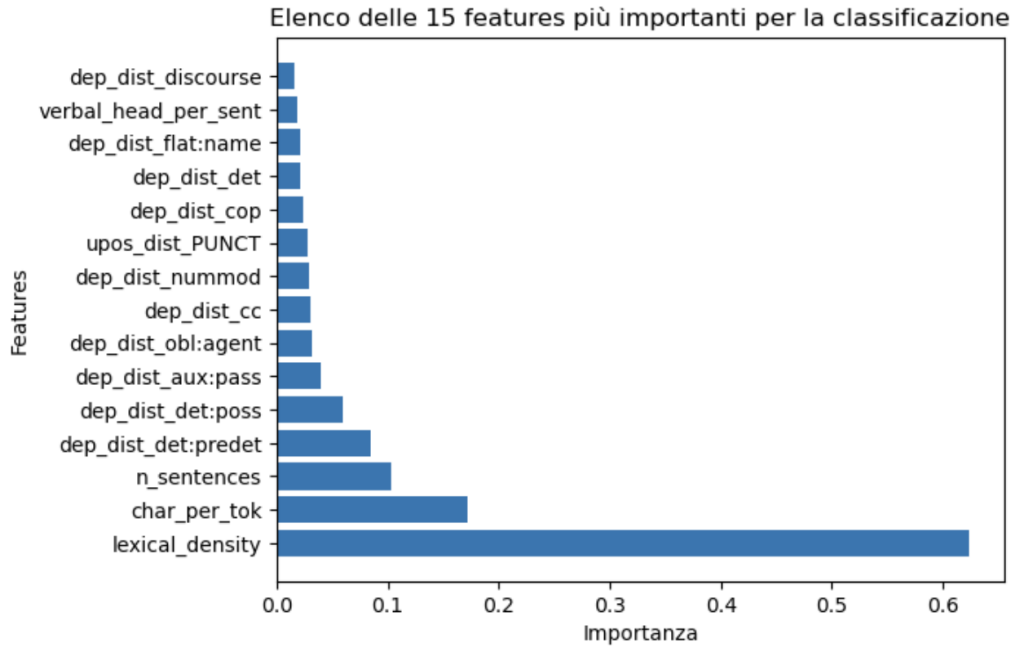

Figura 1.1: Bar chart rappresentante l'importanza delle features.

Capitolo 2

Classificatore basato su SVM lineari con n-grammi

La seconda richiesta richiedeva lo sviluppo di un classificatore basato su SVM lineari, prendendo però in questo caso in input una rappresentazione del testo basata su n-grammi di caratteri, parole e part-of-speech (POS).

Per svilupparlo, mi sono servito dell'output fornito da Profiling-UD per lo sviluppo della prima richiesta, e in particolare della cartella *11925*, al cui interno sono presenti 6383 elementi (file *.conllu*), uno per riga del training set originale (*haspedee_dev_taskAB.tsv*), contenenti i tweet originali e la loro rappresentazione basata su n-grammi.

Ho quindi implementato le classi *Document*, *Sentence* e *Token*, letto e caricato tutti i file dalla cartella e costruito due funzioni, una per estrarre n-grammi di parole, lemmi o POS in una frase (a seconda del parametro *el*) e una per estrarre n-grammi di caratteri. Le due funzioni estraggono n-grammi in base del parametro n inserito.

Ho infine costruito una funzione che estrae n-grammi di parole e/o di caratteri (basandosi sulle due funzioni citate sopra) e li memorizza come attributo features, effettuando anche un'operazione di normalizzazione, dividendo per il numero di parole o di caratteri (a seconda della necessità). Quest'ultima funzione è stata costruita con l'idea di essere quella effettivamente utilizzata per l'implementazione dei vari sistemi.

Ho scelto di analizzare bigrammi di parole per tutti i 4 tipi possibili, quindi per parole, lemmi, POS e caratteri. Per ognuno, ho inoltre filtrato le features, rimuovendo quelle che apparivano in meno di 5 documenti (per farlo, mi sono servito di una funzione *filter_features* che ho applicato per tutte e 4 le analisi).

L'analisi delle prestazioni e dei risultati è stata fatta attraverso una 5-fold cross validation sul training set e i risultati (disponibili in tabella 2.1) hanno mostrato dei risultati migliori nel caso della rappresentazione basata su n-grammi di caratteri, con un punteggio di accuracy del 66% e di F1-Score weighted average del 62%. Come per la scorsa analisi, in tutte le 4 rappresentazioni, ma specialmente in quelle basate su parole e su lemmi, ho notato un bias verso la classe 0, con dei punteggi di recall rispettivamente di 99% e 98%, contro 6& e 11% per la classe 1.

Selezionato infine la rappresentazione di bigrammi basata su caratteri come il miglior sistema tra quelli analizzati, ho addestrato un modello *LinearSVC* sui dati di training e l'ho utilizzato per fare una previsione sui due test set ufficiali (anch'essi passati per lo stesso procedimento di estrazione dei dati di training, ma non di filtraggio delle features), ottenendo un'accuracy del 65% e un F1-score weighted average del 63% nel caso del primo (news),

Tipi di bigrammi	**Accuracy**
Parole	0.61
Lemmi	0.62
POS	0.63
Caratteri	0.66

Tabella 2.1: Accuracy per ciascun tipo di rappresentazione dei bigrammi.

mentre un'accuracy del 58% e un F1-score weighted average del 53% nel caso del secondo (tweets), riportando ancora una volta una discrepanza nei due casi e dei migliori risultati per il test set esterno al dominio. Anche in questo caso è presente un certo bias verso la classe più frequente, 0.

Capitolo 3

Classificatore basato su SVM lineari con word embeddings

La terza richiesta, sempre riguardante lo sviluppo di un classificatore basato su SVM lineari, chiedeva di utilizzare in input una rappresentazione del testo costruita attraverso l'uso dei word embeddings.

Per svilupparlo, ho utilizzato, come nel capitolo 2, la cartella *11925*, fornita da Profiling-UD.

I word embeddings utilizzati sono stati gli Italian Twitter embeddings, forniti dall'ItaliaNLP Lab[2]; questi hanno 128 componenti, e li ho trovati particolarmente adatti per il mio progetto in quanto sono basati sulla piattaforma Twitter, la stessa da cui sono stati estratti i dati del task HaSpeDee.

A partire dal file *.sqlite* disponibile sul sito, ho costruito un file *.txt* in modo da fargli avere una riga per parola, in cui il primo elemento è la parola e i successivi 128 sono le componenti dell'embedding. Da questo, ho poi

costruito una funzione apposita per caricare gli embeddings.

Ho poi caricato il mio dataset, costruendo inoltre alcune funzioni al fine di normalizzare il testo al suo interno secondo le seguenti regole:

- Se una stringa è un url, essa viene sostituita con la stringa *___URL___*.
- Se la stringa ha più di 26 caratteri, essa viene sostituita con la stringa *__LONG-LONG__*.
- Se la stringa ha la prima lettera maiuscola, essa viene capitalizzata (quindi la prima lettera viene resa maiuscola e le restanti minuscole).
- Se la stringa ha la prima lettera minuscola, tutte le sue lettere (inclusa la prima) vengono rese minuscole.
- Se la stringa contiene dei numeri:
  - Se la stringa è composta solo da un numero inferiore a 9999, essa rimane invariata.
  - Se la stringa è composta solo da un numero superiore a 9999, essa viene trasformata nella stringa *DIGLEN_* seguita dal numero di cifre del numero.
  - Se la stringa è composta sia da cifre numeriche che da altri caratteri (punteggiatura inclusa), tutti i numeri vengono sostituiti con un carattere *@Dg*.

Caricato il dataset, ho costruito una funzione che ottenesse la lista dei token da un documento, i quali sono stati rappresentati come dizionari con chiavi *word* e *pos*, rappresentanti ovviamente la parola e la sua part-of-speech.

Nel fare ciò, ho dovuto prestare particolare attenzione alle parole composte, che sono seguite sempre dalle sue parti successive e, oltre a non avere POS, hanno un numero di identificazione composto dai numeri di identificazione delle loro parti, collegate attraverso un trattino.

Iterando la funzione precedente su tutti i documenti della cartella, ho ottenuto la lista di documenti *all_documents*, in cui ogni documento è composto da una lista di dizionari rappresentanti i token.

Per la rappresentazione di ogni documento è necessario aggregare i word embeddings di tutte le sue parole; ho utilizzato la media come funzione di aggregazione comune per tutte le funzioni, e ho sviluppato 3 funzioni che differiscono per il modo in cui questa media è ottenuta:

- Calcolando la media di tutti i word embeddings del documento.
- Calcolando la media dei word embeddings solo degli aggettivi, nomi e verbi del documento.
- Calcolando la media dei word embeddings solo degli aggettivi, nomi e verbi in maniera separata e concatenando i 3 vettori ottenuti.

Creata, per ogni metodo, una lista di features (ovvero i word embeddings) e di labels (ovvero 0 o 1 in base al contenuto o meno di hate speech nel documento, ottenuto dal nome del file), ho normalizzato le features utilizzando un MinMaxScaler e ho applicato una 5-fold cross valdiation per l'ottenimento dei risultati, presenti nella tabella 3.1.

Sotto il punto di vista dell'accuracy e dell'F1-score, il miglior metodo sembra essere quello che semplicemente prende la media di tutti i word embeddings del documento, dunque l'ho utilizzato per fare una predizione sui

Tipi di bigrammi	**Accuracy**	**F1-score weighted avg**
Tutti i WE	0.76	0.76
Solo WE di parole piene	0.73	0.73
Solo WE di parole piene con sep.	0.72	0.72

Tabella 3.1: Accuracy per ciascun modo di ottenere la media.

dati di test ufficiali, processando anch'essi allo stesso modo dei dati di training, compresa la normalizzazione, effettuata con l'utilizzo del *MinMaxScaler* già fittato ai dati di training, e utilizzando per la classificazione un *LinearSVC* anch'esso già fittato sui dati di train. I risultati sono stati alquanto soddisfacenti e, per questa volta, vicinissimi tra loro: sia per il test set interno al dominio che per quello esterno, infatti, l'accuracy del classificatore ha riscontrato un punteggio di 74%, mentre l'F1-score weighted average è risultato del 71% per il test set news e del 74% per il test set tweets. L'unico appunto risulta essere un punteggio basso per il recall della classe 1 nel test set news, del 37%, il quale indica una difficoltà per il modello nel classificare correttamente le news contenenti hate speech. Si potrebbe forse spiegare questo fenomeno considerando che la modalità in cui l'hate speech è erogato sui social network, e quindi su Twitter, sui quali contenuti il modello è stato addestrato, sia diversa dalla modalità in cui l'hate speech è erogato nelle notizie giornalistiche.

Capitolo 4

Fine tuning su Neural Language Model

La quarta e ultima richiesta riguardava i Neural Language Models, e in particolare di condurre un processo di fine-tuning per 5 epoche sul dataset. Il modello da me scelto per l'analisi è stato BERT, in quanto particolarmente adeguato per compiti di classificazione.

A differenza degli altri sistemi, i quali sono stati implementati su un ambiente Jupyter di Anaconda, per questo ho utilizzato un ambiente Google Colab, sopratutto per la possibilità di connettersi a un runtime con GPU, elemento quasi necessario per l'addestramento di un modello Transformer come BERT.

Ho iniziato caricando la cartella *profiling_input*, contenente tutti i documenti di training in file .txt con l'id e 0 o 1 (in base alla presenza o meno di hate speech) nel nome, separati da un #. Una volta salvati i dati di ogni documento in un dizionario e appesi tutti i dizionari generati in una lista

chiamata *all_data*, ho convertito quest'ultima in un dataset e ho effettuato la divisione tra train e test (80/20) e tra train e validation (90/10).

Ho successivamente caricato il modello pre-trained e il tokenizzatore. Avevo inizialmente utilizzato il WordPiece Tokenizer, ma ho poi utilizzato l'AutoTokenizer, a causa di un quasi impercettibile miglioramento delle prestazioni.

Applicata la tokenizzazione ai miei dati e una volta configurati per PyTorch, ho addestrato il modello su 5 epoche, valutandolo sul validation set per ognuna di esse.

Le curve di training e validation (in figura 4.1) mostrano un evidente overfitting, dal momento che la curva di validation va sempre aumentando a partire dalla fine della seconda epoca. Ho cercato di contrastarlo, con scarsi risultati, aumentando il parametro di weight decay a 0.05 (da 0.01), in modo da penalizzare i pesi più grandi.

Le metriche di valutazioni finali riportano 80% di accuracy e 79%/80% di F1-score (79% per la macro average e 80% per la weighted average).

Ho poi caricato e preprocessato i due test set ufficiali alla stessa maniera del training set, in modo da poter fare una prediction anche su di essi; i risultati finali (in tabella 4.1) sono, anche se di poco, i migliori ottenuti considerando tutte le implementazioni di questo progetto. Anche in quest'analisi, però, il dato che salta all'occhio per essere particolarmente basso è quello, per il test set 1 (news), della recall per la classe 1, che raggiunge solo il 47%, aggiungendo credibilità alla teoria formulata per la scorsa analisi per cui la motivazione è la differenza di modalità con cui l'hate speech è erogato sui social network e sulle notizie di giornale.

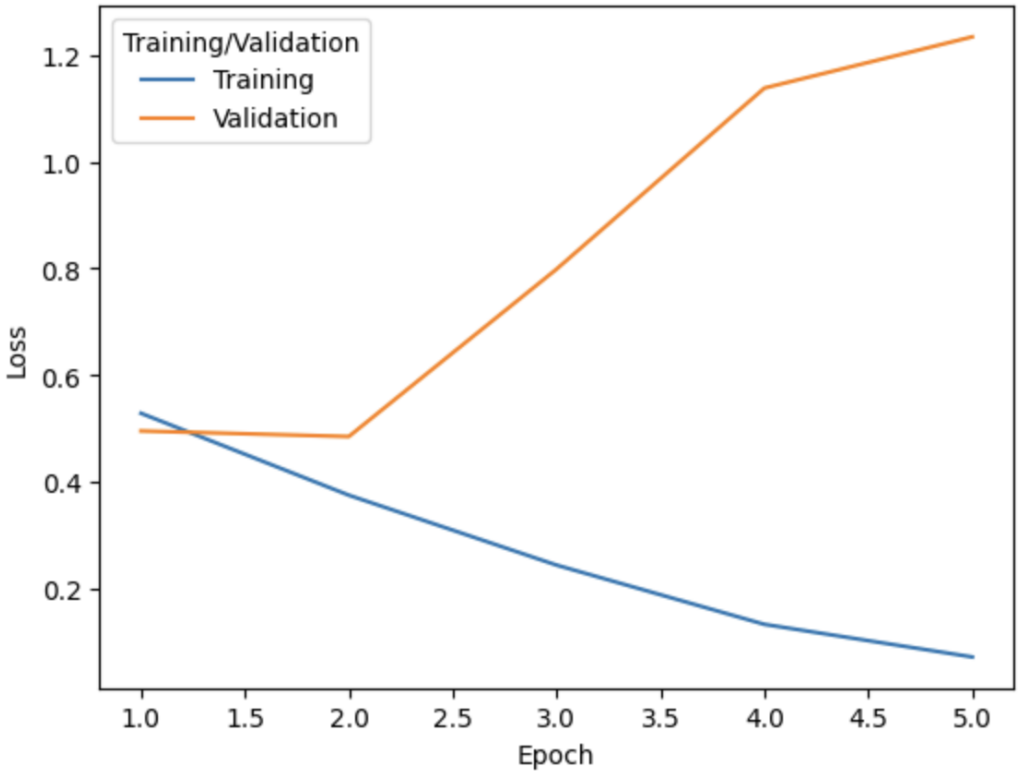

Figura 4.1: Curve di loss per training e validation.

Test set	**Accuracy**	**F1 macro avg**	**F1 weighted avg**
1 (news)	0.77	0.72	0.75
2 (tweets)	0.77	0.76	0.76

Tabella 4.1: Accuracy e F1-Score per i due test set.

Bibliografia

[1] Dominique Brunato, Andrea Cimino, Felice Dell'Orletta, Simonetta Montemagni, and Giulia Venturi. "profiling-ud: a tool for linguistic profiling of texts". *Proceedings of 12th Edition of International Conference on Language Resources and Evaluation*, May 11-16 2020.

[2] Andrea Cimino, Lorenzo De Mattei, and Felice Dell'Orletta. Multi-task learning in deep neural networks at evalita 2018. *Proceedings of EVALITA '18, Evaluation of NLP and Speech Tools for Italian*, December 12-13 2018.

[3] Manuela Sanguinetti, Gloria Comandini, Elisa Di Nuovo, Simona Frenda, Marco Stranisci, Cristina Bosco, Tommaso Caselli, Viviana Patti, and Irene Russo. Haspeede 2@ evalita2020: Overview of the evalita 2020 hate speech detection task. *Proceedings of the 7th evaluation campaign of Natural Language Processing and Speech tools for Italian (EVALITA 2020)*, 2020.