Classificazione di testo con Perceptron e Naive Bayes

Carlo Baronti
Università degli Studi Di Firenze
Email: carlo.baronti@stud.unifi.it

Giulio Baronti
Università degli Studi Di Firenze
Email: giulio.baronti@stud.unifi.it

***Sommario*—Il lavoro consiste nell'implementazione e analisi delle performance di classificatori di testo tramite algoritmi Bernoulli Naive Bayes, Multinomial Naive Bayes e Perceptron. Come dataset di riferimento è stato usato il ModAdpte di Reuters-21578 di cui sono state scelte le 10 categorie con più documenti.**

1. Introduzione

Inizialmente si sono divisi i documenti di training e i documenti di testing, dopo di che si è estrapolato un insieme di parole detto vocabolario,da cui sono state eliminate le stopwords; per fare ciò ci si è avvalsi dello stemmer di Porter. Tali funzioni preliminari sono svolte all'interno del modulo `manager` e vengono eseguite in circa un minuto su Google Colab. In seguito,si continuerà a fare riferimento ai tempi di esecuzione su Google Colab. La struttura del programma comprende, oltre al `main.py`,i moduli `bernoulliNB.py`, `multinomialNB.py` e `perceptron.py` relativi ai tre algoritmi omonimi.

2. ModApte split Reuters-21578 dataset

Delle 90 categorie presenti nel dataset Reuters-21578 sono state selezionate le 10 più popolose in termini di documenti. Si riportano il numero di documenti di train e test per ognuna delle categorie selezionate:

- acq : train 1650,test 719
- corn : train 181 ,test 56
- crude : train 389,test 189
- earn :train 2877,test 1087
- grain:train 433,test 149
- interest:train 347,test 131
- money-fx:train 538,test 179
- ship: train 197,test 89
- trade:train 368,test 117
- wheat: train 212,test 71

3. Definizioni

Nel testo si farà uso delle seguenti definizioni:

- d : documento
- $|D|$: numero di documenti
- C: categoria
- V: vocabolario
- V_c : vocabolario della categoria
- w:parola
- N_{cw}: numero di documenti nella categoria c che contengono la parola w
- N_c: numero documenti nella categoria c

–

4. Bernoulli Naive Bayes

L'algoritmo ha richiesto la suddivisione del vocabolario precedentemente estratto in 10 dizionari contenenti le parole contenute da ciascuna categoria; analogamente si sono divisi i documenti per categoria di appartenenza.
Si è quindi proceduto alla definizione di una probabilità frequentista circa l'appartenenza a priori di un documento a una categoria:

$$P(d \in C) = \frac{|\{d : d \in C\}|}{|D|} \tag{1}$$

.
Si procede contando il numero di documenti per categoria in cui occorre una certa parola e si calcola la probabilità condizionata che data la presenza di una parola *w* nel documento *d* tale documento appartenga alla categoria *C* (sempre in un ottica frequentista):

$$P(d \in C|w \in V_c) = \frac{N_{cw} + 1}{N_c + 2} \tag{2}$$

Il processo di training consiste quindi nel calcolare, considerando l'intera collezione di documenti di train, tali probabilità rispettivamente per ogni categoria e per ogni coppia categoria-parola. Il processo di testing consiste nel calcolare la probabilità condizionata che il documento stia in una o più categorie considerando quali parole compaiono o meno nel documento. Poichè gli eventi di occorrenza delle parole nei documenti sono considerati **indipendenti** gli uni dagli altri, la probabilità condizionata degenera in una produttoria delle probabilità dell'occorrenza delle singole parole.

4.0.1. Risultati e prestazioni. I risultati in termini di predizione ottenuti non risultano particolarmente efficaci, come si può evincere dai grafici relativi alle varie categorie. Il processi di training e testing richiedono complessivamente circa 12 minuti.

5. Multinomial Naive Bayes

Multinomial Naive Bayes differisce dall'algoritmo precedemente presentato per il semplice fatto che non si limita a controllare se una parola è presente o meno nel documento, ma conta anche quante volte essa occorre nei documenti. Quindi, per quanto la probabilità frequentista definita della sezione rimanga valida, la probabilità condizionata risulta:

$$P(d \in C|...) = \frac{n_{w_i,c} + 1}{\sum_{\forall w \in V} n_{w,c} + 1} \tag{3}$$

dove $n_{w_i,c}$ rappresenta il numero di occorrenze della parola w_i nella categoria c e $\sum_{\forall w \in V} n_{w,c}$ tutti i tokens (ovvero le parole, con ripetizioni) nella categoria.

5.1. Risultati e prestazioni

Il conteggio delle occorrenze delle parole nei documenti, di contro alla variante Bernoulli, comporta un aumento deciso dei tempi di esecuzione. Difatti occorrono quasi 8 minuti per il completamento della fase di train, mentre il testing necessita di circa 10 minuti. Tuttavia all'aumento del tempo di esecuzione non sembra essere collegato un aumento della precisione di classificazione:osservando le curve di recall-precision non si osservano miglioramenti sostanziali e in alcuni casi si osserva addirittura un peggioramento (e.g. nelle categorie *interest* e*money-fx*)

6. Perceptron

Perceptron rappresenta una prima rappresentazione del concetto di neurone. Utilizza un approccio totalmente diverso dai precedenti algoritmi presentati in quanto non sfrutta il teorema di Bayes bensì il concetto di regressione lineare. Occorre prima però dare un'adeguata rappresentazione ai documenti.

6.1. Rappresentazione documenti

Dopo aver proceduto all'estrazione del vocabolario, alla suddivisione dei documenti di train e test, e dall'ulteriore divisioni di tali documenti per categorie, per ogni documento viene estratta una sua rappresentazione TF-IDF (*Term Frequency-Inverse Document Frequency*. Tale rappresentazione consiste nel pesare la rilevanza di un dato documento nei confronti una certa parola prendendo in considerazione due fattori:

1) la frequenza di una parola nel documento (TF)
2) l'importanza generale di una parola nell'insieme dei documenti considerati (IDF)

Moltiplicando la IDF relativa a una parola *w* con la corrispondente tf nel documento *d*,si ottiene un vettore di lunghezza pari al numero di parole presenti nel vocabolario, in cui in ogni membro compare il prodotto $tf[w] * idf[w]$.

Term Frequency. Nello scecifico la frequenza della *i-esima* parola nel *j-esimo* documento d_j risulta

$$tf_{i,j} = \frac{n_{i,j}}{|d_j|} \tag{4}$$

dove $n_{i,j}$ rappresenta il numero di occorrenze della parola *i-esima* parola nel *j-esimo* documento.

Inverse Document Frequency. La IDF, come detto, misura l'importanza di una parola nell'insieme dei documenti considerati:

$$idf_i = \log \frac{|D|}{|\{d : w_i \in d\}|} \tag{5}$$

dove $|D|$ rappresenta il numero di documenti di train e il denominatore rappresenta il numero di documenti che contengono la *i-esima* parola w_i del vocabolario *V*.

TF-IDF. Di conseguenza la rappresentazione TF-IDF per la parola *i-esima* w_i e il documento *j-esimo* risulta

$$tfidf_{i,j} = tf_{i.j} \cdot idf_i \tag{6}$$

6.2. Algoritmo

L'algoritmo suddivide lo spazio dei documenti in due parti, positiva e negativa, tramite un iperpiano. Tale iperpiano tramite l'aggiornamento di un vettori dei pesi *weights* e del *bias*. L'aggiornamento dell'iperpiano viene effettuato in base al prodotto vettoriale della rappresentazione TF-IDF del documento considerato *x* per il vettore dei pesi corrente. Per le regole di aggiornamento si fa riferimento agli articoli in bibliografia.

6.3. Risultati e tempi di esecuzione

I tempi di esecuzione variano molto a seconda del numero massimo di iterazioni imposte al processo di appre

7. Conclusion

The conclusion goes here.

Acknowledgments

The authors would like to thank...

Riferimenti bibliografici

[1] H. Kopka and P. W. Daly, *A Guide to LaTeX*, 3rd ed. Harlow, England: Addison-Wesley, 1999.