Naive Bayes for Text Classifications

Luca Maltempo

March 2021

1 Introduzione

Con questo elaborato ho cercato di mostrare come un modello relativamente semplice, il naive Bayes Classifier, riuscisse ad ottenere buoni risultati nella classificazione di documenti. In particolare ho implementato due diversi modelli di classificazione, il modello di Bernoulli ed il modello Multinomiale. Questi due modelli sono stati applicati al Datasets 20 Newsgroup.

2 Estrazione Bag of Words

All'interno del main si può scegliere su quali categorie effettuare le fasi di allenamento e classificazione. A questo punto definiamo due oggetti che conterranno i documenti di train e quelli di test. Questi due oggetti hanno anche il compito di mantenere l'informazione relativa all'appartenenza di ogni documento ad una classe, la tipologia del documento ed un container chiamato MetaData, il quale conterrà:

- righe = Contiene il numero di documento a cui appartiene ogni parola letta.
- colonne = Contiene l'ID col quale la parola letta è stata salvata nel dizionario.
- data = Contiene l'occorrenza della parola "i" nel documento "j".
- dizionario = Contiene il dizionario che verrà usato per le fasi di training e classificazione.

L'estrazione del bags-of-words avviene leggendo un documento alla volta. Per ogni parola nel documento si controlla che questa abbia solo caratteri alfabetici o solo caratteri numerici. Nel primo caso si controlla anche che sia più lunga di 2 caratteri, immaginando che parole di dimensione inferiore non siano molto rilevanti, e che non sia una stopword. A questo punto si effettua lo stemming della parola in modo da considerare solo la parte semanticamente rilevante. Se la parola non è presente nel dizionario la si aggiunge e le si assegna un identificativo, a questo punto sfruttiamo il metodo countWord per tener traccia della

parola letta e poi poter costruire il metaData dei documenti sopra descritto. Se la parola letta fosse stata una stringa numerica allora l'avremmo trasformata in uno '0', questo perchè non ci interessa il valore di un numero per la classificazione di un documento, è invece importante tener traccia dell'occorrenza di numeri all'interno del testo. Non si tiene conto di tutte le stringe che contengono sia caretteri numerici che alfabetici, la stragrande maggioranza di queste stringhe sono errori di battitura e solo una piccola parte corrisponde a modelli di oggetti come dispositivi vari, componentistica elettronica etc. Alla fine di questa fase otteniamo una lista con tutti i documenti letti, la loro classe di appartenenza, un dizionario con le parole incontrate e un container, metaData, che sarà fondamentale per una rapida costruzione dei paramentri necessari alla classificazione. La quasi totalità del tempo di esecuzione del programma è dovuta proprio questa fase di analisi e caricamento dei documenti. In questo programma non sono state utilizzate tecniche per limitare in modo arbitrario la lunghezza del dizionario, dopo aver sperimentato attraverso il calcolo della mutua informazione ho notato che i risultati di classificazione miglioravano all'aumentare delle parole del dizionario. Inoltre il calcolo della mutua informazione non faceva altro che rallentare il programma senza portare alcun beneficio. Ho quindi preferito mantenere nel dizionario tutte le parole incontrate nella lettura dei documenti di training. Come si potrà capire più avanti, ho reso il tempo di esecuzione del programma indipendente dalla lunghezza del dizionario in modo che questa possa essere grande a piacimento.

3 Training

In questa fase il programma si occuperà di creare e calcolare quei parametri necessari per la classificazione. Per fare questo nel main si definiscono due diversi oggetti per calcolare i parametri, uno secondo il modello di Bernoulli e l'altro secondo il modello Multinomiale. Non mi soffermerò sul lato teorico piuttosto proverò a spiegare alcune parti del codice che potrebbero risultare oscure ad una prima lettura. Prima però conviene definire i parametri:

- $\text{classParameter}_{\text{c}}$ = Rappresenta la frazione di documenti appartenenti alla classe 'c'.
- $\text{wordParameter}_{\text{w,c}}$ = Rappresenta la probabilità di leggere la parola 'w' in un documento di classe 'c'.

3.1 binomialTraining

Questo algoritmo sfrutta l'oggetto metaData. Viene fatto un ciclo su 'righe' in modo da identificare il documento che stiamo leggendo, ogni volta che leggiamo un nuovo documento incrementiamo il $\text{classParameter}_{\text{c}}$ corrispondente. L'oggetto 'colonne' indica l'identificativo della parola, quindi per ogni parola letta possiamo incrementare il suo $\text{wordParameter}_{\text{w,c}}$. A questo punto la fase di training sarebbe finita e potremmo procedere con la classificazione. Questa

però, in teoria, prevede che per OGNI documento 'test' si tenga conto di OGNI parola presente all'interno del dizionario e questo per OGNI possibile categoria del documento. Procedere con la versione teorica della classificazione Binomiale porterebbe ad una lunga e spiacevole attesa. Per evitare tutto ciò è sufficiente notare che il $\text{wordParameter}_{w,c}$ di parole diverse è uguale se queste appartengono alla stessa classe ed hanno la stessa occorrenza. Vengono quindi eseguite due diverse funzioni che costruiscono delle tabelle, una per ogni classe. Queste tabelle contengono coppie chiave-valore, dove il valore indica quante parole sono state lette 'X' volte tra tutti i documenti di una stessa classe, ed 'X', l'occorrenza della parola, funge da chiave. Nella parte di classificazione si capirà meglio l'utilità di queste tabelle.

3.2 multinomialTraining

L'algoritmo che si occupa del calcolo dei parametri relativi al modello Multinomiale opera come l'algoritmo sopra descritto ma con una piccola differenza. Ogni volta che viene letto un nuovo valore in 'colonne' il $\text{wordParameter}_{w,c}$ viene incrementato tanto quante volte occorre la parola nel documento, questa informazione la si può trovare nell'attributo data del container metaData. Per il modello multinomiale non sarà necessario il calcolo delle tabelle descritte precedentemente, questo perchè per la classificazione è sufficiente tener conto delle parole che appaiono nel documento e non di tutte le parole del dizionario.

4 Classifier

A questo punto l'unica cosa che rimane da fare è classificare i documenti di test e mostrare in output i risultati. Prima di discutere dei risultati però è opportuno spendere due parole su come ho implementato gli algoritmi di classificazione.

4.1 fastBernoulli

Come si evince dal nome dato all'algoritmo sono stati fatti alcuni accorgimenti per evitare che la classificazione impiegasse svariate ore. La classificazione in teoria prevede la ripetizione di diverse operazioni su ogni parola del dizionario. Questo è un problema per due motivi, il tempo di esecuzione dell'algoritmo è dipendente dalla lunghezza del dizionario ed il dizionario , in generale, contiene molte parole. A questo punto le soluzioni possono essere due, mettere un limite alla dimensione del dizionario andando a scartare le parole che consideriamo meno rilevanti oppure cercare di implementare l'algoritmo in maniera differente. Io ho scelto la seconda opzione. In particolare, sfruttando le tabelle create in 3.1, si può implementare l'algoritmo di classificazione in modo tale che possa eseguire lo stesso identico calcolo teorico senza andare a visionare tutte le parole del dizionario ma solo controllando le parole presenti all'interno del documento. Per ogni parola presente nel documento si controlla la sua occorrenza tra i documenti di train e si va a decrementare il numero di parole con tale occorrenza

dalle tabelle. Fatto ciò per tutte le parole del documento non rimane che tenere conto di tutte le rimanenti parole del dizionario che non erano presenti nel documento, ma grazie alle tabelle possiamo ricostruire il numero di queste parole e il loro $\text{wordParameter}_{\text{w,c}}$ e quindi completare rapidamente il calcolo richiesto per la classificazione.

4.2 fastMultinomial

L'algoritmo implementato è praticamente identico a come è stato descritto a livello teorico in 20 newsgroup. L'unico problema incontrato nel calcolo della funzione di classificazione è legato al modo in cui vengono memorizzati i valori in virgola mobile in Python. Come sappiamo dalla teoria

la $\sum_{d=0}^{len(dictionary)} wordParameter_{d,c} = 1$. Questo significa che in media la probabilità che una parola sia presente nel documento è pari al reciproco della dimensione del dizionario, maggiore è la dimensione del dizionario minore in media sarà la probabilità che la parola sia nel documento. Nel nostro caso il dizionario contiene poco più di 50k parole, questo significa che ogni $\text{wordParameter}_{\text{w,c}}$ $\approx 10^{-5}$. Tentare quindi la classificazione di un documento con 60 o più parole porterebbe il valore della funzione di classificazione a 0 poichè il valore sarebbe minore del più piccolo numero rappresentabile nel formato Double-precision floating-point. Ho quindi implementato un semplice metodo che normalizzi il valore della funzione di classificazione ogni volta che sta per diventare troppo piccolo.

5 Risultati

Dall'output del programma possiamo vedere sia le due matrici di contingenza che i valori che descrivono la precisone raggiunta dai due algoritmi per la classificazione.

Binomial Classifier:

```
Macro-Precision avg:    0.740
Macro-Recall avg:       0.664
Macro-F1Score avg:      0.658
```

Multinomial Classifier:

```
Macro-Precision avg:    0.804
Macro-Recall avg:       0.792
Macro-F1Score avg:      0.784
```

```
134  4    0    19   16   0    21   5    23   6    0    1    4    7    2    64   3    6    3    1
0    256  3    33   18   12   40   0    0    0    0    11   2    1    10   3    0    0    0    0
0    32   154  117  20   12   32   2    1    1    0    13   2    0    4    2    0    0    2    0
0    2    2    322  18   2    23   0    0    0    0    3    20   0    0    0    0    0    0    0
0    3    0    25   306  1    34   2    0    0    0    1    8    0    4    0    0    0    1    0
0    56   8    32   14   240  35   1    0    0    0    3    5    1    0    0    0    0    0    0
0    1    0    21   5    1    347  3    4    0    0    1    5    0    0    0    2    0    0    0
0    2    0    4    5    0    56   303  7    0    0    0    16   0    1    0    1    0    1    0
0    2    0    2    4    1    14   5    366  0    0    0    1    0    0    0    2    0    1    0
0    1    0    3    1    0    36   0    1    346  1    0    3    0    0    0    0    0    5    0
0    0    0    3    4    0    31   0    5    17   333  1    1    0    0    4    0    0    0    0
0    20   0    16   32   3    24   1    5    1    0    269  21   0    0    1    3    0    0    0
0    8    1    43   20   1    43   8    3    0    0    12   249  4    1    0    0    0    0    0
0    22   0    17   26   0    57   8    13   3    0    0    31   205  0    6    1    1    6    0
0    30   0    1    11   1    46   7    1    2    0    2    24   3    261  2    0    0    3    0
2    5    1    11   14   1    37   0    4    4    0    0    3    0    0    316  0    0    0    0
0    2    0    13   7    0    28   26   14   3    0    5    10   1    1    3    250  1    0    0
2    7    2    12   9    0    34   6    18   9    1    1    1    1    1    22   5    244  1    0
2    3    0    8    16   0    21   23   16   17   0    3    8    4    11   13   73   0    92   0
30   14   0    19   8    0    21   7    9    2    0    0    2    3    4    104  20   2    1    5
```

Binomial Classifier

```
232  0    0    1    0    1    0    0    2    1    1    2    1    6    2    42   7    9    4    8
5    295  8    11   13   20   1    0    0    1    1    15   6    2    6    1    1    1    2    0
1    47   119  81   16   76   2    1    0    1    1    23   1    2    8    1    2    0    10   2
0    8    5    309  30   3    4    1    2    1    0    5    23   1    0    0    0    0    0    0
2    5    2    25   314  4    6    4    0    2    0    3    9    3    4    0    1    0    1    0
4    42   0    12   5    311  1    0    0    0    1    10   0    0    4    0    1    2    2    0
4    8    1    38   19   0    273  17   6    1    1    0    11   5    3    0    3    0    0    0
0    2    0    0    0    1    3    372  1    1    2    2    4    0    0    0    3    1    4    0
0    0    0    1    0    0    0    9    377  0    1    0    4    1    0    0    1    1    3    0
1    3    2    0    1    1    0    1    1    360  16   1    0    0    1    2    1    2    4    0
8    3    4    0    0    0    0    0    0    3    373  1    1    0    1    2    1    1    1    0
2    2    1    0    2    1    1    0    1    3    0    369  1    3    0    1    8    0    1    0
0    17   1    24   10   1    2    7    3    0    0    40   276  7    1    0    0    2    1    1
9    6    0    3    0    0    3    1    1    0    0    2    6    334  5    7    3    5    11   0
3    9    0    0    0    2    1    1    0    0    0    1    4    4    356  2    3    0    8    0
24   2    0    0    0    1    0    1    0    0    1    0    0    1    2    357  1    0    3    5
1    0    0    0    0    0    3    0    1    0    0    6    1    0    0    2    339  2    5    4
13   1    0    0    0    0    0    1    2    1    0    4    0    0    0    8    10   324  12   0
12   0    0    0    1    0    0    0    0    1    1    6    0    3    8    2    102  2    172  0
49   3    0    0    0    0    0    0    1    0    1    0    0    0    4    61   26   3    3    100
```

Multinomial Classifier