Elaborato Intelligenza Artificiale

Francesco Pegoraro

1 Introduzione

In questo studio si utilizzano implementazioni disponibili di Naive Bayes per studiare l'andamento dell'errore di generalizzazione con il numero di esempi. Concretamente si usano due datasets di documenti testuali: 20 newsgroups e Reuters-21578. Le learning curves saranno considerate rispetto a due diverse tipologie di classificatori di tipo Naive Bayes: Bernoulli e Multinomiale.

2 Strumenti usati

Il linguaggio di programmazione scelto è Python(version 3.5.2). L'implementazione dei classificatori è quella della libreria di Machine Learning Scikit-Learn.

3 Classificatori

In machine learning, i classificatori naive Bayes sono una famiglia di algoritmi probabilistici basati sull'applicazione del teorema di Bayes con assunzioni di indipendenza (naive) tra le features.

Data una classe y e un vettore di features da x_1 a x_n il teorema di Bayes ci permette di utilizzare la seguente regola di classificazione:

$$y' = arg\,max_y\, P(y) \prod_{y=1}^{n} P(x_i|y)$$

$P(y)$ è la frequenza della classe y nel training set. I vari tipi di classificatori naive Bayes differiscono principalmente nell'assunzione che fanno rispetto alla distribuzione di $P(x_i|y)$.

3.1 Bernoulli Naive Bayes

Nel modello multivariato di Bernoulli le features sono variabili binarie indipendenti e nel nostro caso la feature a per il sample d vale 1 se a è presente in d, 0 altrimenti. Come Multinomial, questo modello è popolare per la classificazione dei documenti. La regola di decisione per Bernoulli Naive Bayes è basata su:

$$P(x_i \mid y) = P(i \mid y)x_i + (1 - P(i \mid y))(1 - x_i)$$

che si differenzia dal Multinomial perchè penalizza esplicitamente la non occorenza di una feature i nella classe y mentre l'altro avrebbe semplicemente ignorato il fatto.

3.2 Multinomial Naive Bayes

Esso implementa naive Bayes per dati distribuiti multinomialmente. La distribuzione è parametrizzata da vettori:

$$\theta_y = (\theta_{y1}, \ldots, \theta_{yn})$$

per ogni classe y, dove n è il numero di features (nel nostro caso la dimensione del vocabolario) e θ_{yi} è la probabilità $P(x_i \mid y)$ della feature i che appare nel sample appartenente alla classe y. Il parametro θ_y è stimato da una versione smussata di massima likelihood, cioè la frequenza relativa nel dataset:

$$\hat{\theta}_{yi} = \frac{N_{yi} + \alpha}{N_y + \alpha n}$$

dove $N_{yi} = \sum_{x \in T} x_i$ è il numero di volte che la feature i appare in un sample della classe y nel training set T, e $N_y = \sum_{i=1}^{|T|} N_{yi}$ è il conteggio totale di tutte le features per la classe y. Il fattore di smussamento $\alpha \geq 0$ serve quando non compaiono features nel learning samples e previene la probabilità zero nelle computazioni successive.

4 Implementazione

Guardando i due datasets ci accorgiamo che gli articoli sono organizzati in modo differente, sia come posizione nelle cartelle, sia per come sono stati salvati in ogni file di testo. Una volta capito come arrivare ai contenuti effettivi bisogna anche operare una piccola selezione per pulire i testi da eventuali impurità, come header e footer che potrebbero falsare i risultati. A questo punto, dopo aver strutturato i dati ottenuti, cioè il testo e la label correlata, si deve procedere a trasformare i contenuti testuali in vettori di feature numeriche. Il modo più intuitivo per farlo è usare la rappresentazione $bag\,of\,word$, che richiede di:

1. assegnare un id intero ad ogni parola che occorre in ogni documento.
2. per ogni testo i, si conta il numero di occorrenze per ogni parola e si salva il risultao in X[i, j] dove j è la feature, in questo caso la parola.

Questo procedimento potrebbe richiedere una grande quantità di memoria RAM, ma per fortuna la $bag\,of\,word$ sarà presumibilmente una matrice sparsa, per il fatto che non tutte le parole occorrono in ogni documento, e quindi si risparmia salvando solo i valori diversi da zero.

Per l'implementazione della $bag\,of\,word$, usiamo la libreria $sklearn.feature\_extraction.text$ da cui importiamo CountVectorizer, che finalizza la scrematura delle stopword, fa un tokenizing per assegnare gli id e crea un dizionario di features trasformando i documenti in vettori di parole. Adesso che abbiamo tutto l'occorrente, possiamo allenare i due classificatori BernoulliNB e MultinomialNB.

Per ottenere risultati validi e indipendenti dal partizionamento in test set e train set si fa uso della shuffle split come Cross-Validation. Nello specifico le learning curve vengono calcolate incrementando ad ogni iterazione la dimensione del DataSet del 10%. Per ognuno di questi DataSet parziali, si sceglie un 10% random come test set ed il restante 90% come train set. Il procedimento viene ripetuto 100 volte; lo score riportato è quindi una media dei vari risultati di questo processo. Si presenta anche la deviazione standard dei risultati ottenuti per rappresentarne la variazione.

Inoltre, per entrambi i DataSets si è aggiunto la possibilità di salvare in un file $.sav$ (con l'utilizzo di pickle) il modello del classificatore e Vectorizer, cosi da permettere future predizioni su nuovi input, senza doverli ricalcolare.

4.1 20NewsGroup

Il DataSet 20 Newsgroups è una collezione di circa 19,000 documenti, raggruppati in 20 differenti categorie. Esso è diventato popolare per certe applicazioni dell'apprendimento automatico come text classification e text clustering.

Effettivamente si è visitato ricorsivamente ogni cartella del dataset, salvandosi il nome della directory che in questo caso indicava la categoria di appartenenza degli articoli e rimosso da essi: intestazioni, citazioni e pié di pagina. Le parole rimanenti rappresentano quelle che sono le nostre features (circa 100000). Nel caso in cui si lascino footer e quotes il numero di features sale a circa 130000.

4.2 Reuters

Reuters-21578 è forse la collezione di testi più usata per la classificazione del testo. Il DataSet consiste di 21,578 documenti includendo articoli senza categoria ed errori tipografici. Per estrarre i dati, questa volta si è implementato il parser SGM ereditando dalla classe HTMLParser della libreria standard di Python, disponibile anche su scikitlearn. Come richiesto, si considerano solo le 10 categorie più frequenti e quindi i samples sono 10000 per un totale di 25000 features circa.

5 Risultati

Nei seguenti paragrafi si illustrano i risultati ottenuti. Sono riportati i grafici delle learning curve per i vari classificatori; in blu è rappresentato lo score sul train set , in verde lo score sul test set.

5.1 20News

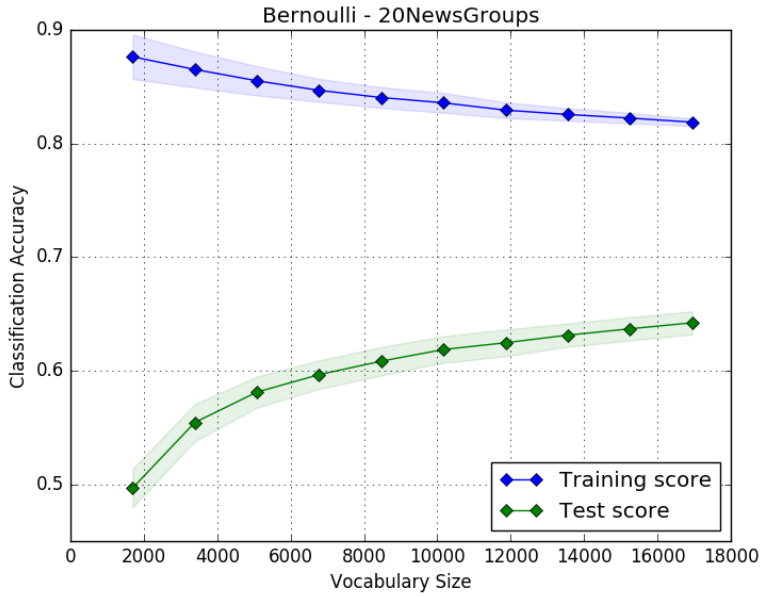

Figure 1: Il grafico mostra i risultati ottenuti dal classificatore Bernoulli NB. In questo caso lo score ottenuto all'ultimo step (90% per il set di Train, 10% per il set di Test) è del 64%.

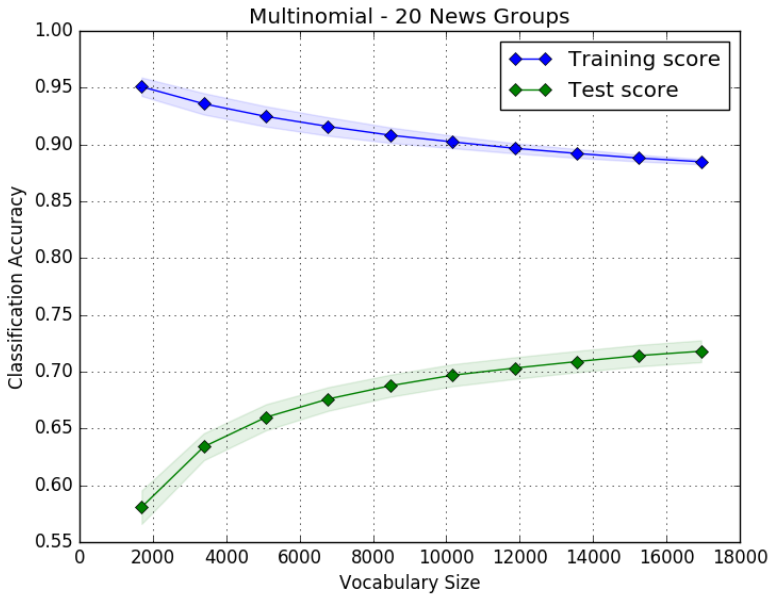

Figure 2: Il grafico mostra i risultati ottenuti dal classificatore Multinomial NB. In questo caso lo score ottenuto all'ultimo step (90% per il set di Train, 10% per il set di Test) è del 72%.

5.2 Reuters

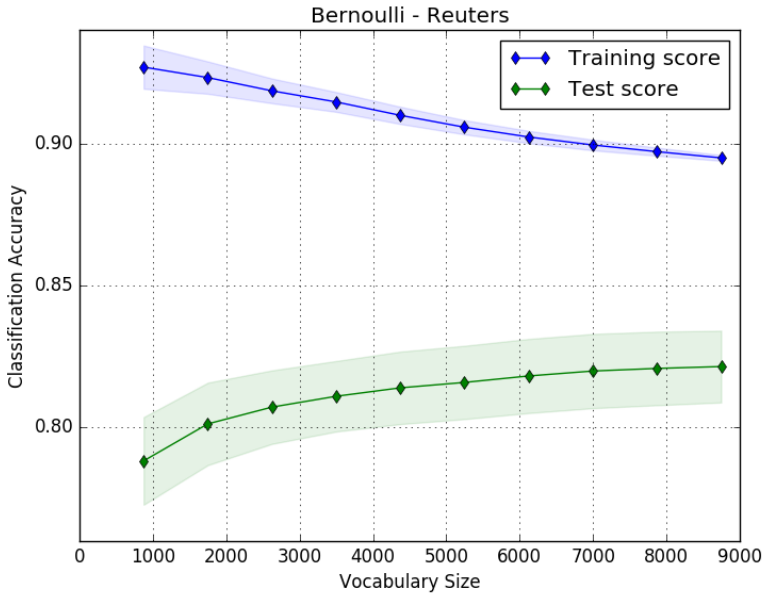

Figure 3: Il grafico mostra i risultati ottenuti dal classificatore Bernoulli NB. In questo caso lo score ottenuto all'ultimo step (90% per il set di Train, 10% per il set di Test) è del 82%.

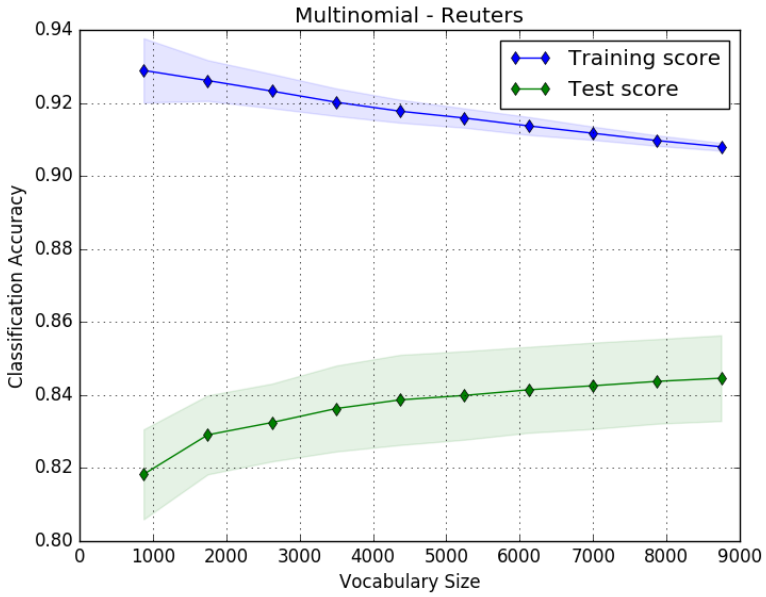

Figure 4: Il grafico mostra i risultati ottenuti dal classificatore Multinomial NB. In questo caso lo score ottenuto all'ultimo step (90% per il set di Train, 10% per il set di Test) è del 84%.

6 Conclusioni

Analizzando i risultati ottenuti si nota, ovviamente, che l'errore di generalizzazione diminuisce (e lo score aumenta) all'aumentare degli esempi utilizzati per fare il train. Inoltre, in entrambi i dataset, si evince che la versione Multinomial dà risultati migliori rispetto a Bernoulli, questo è da attribuire alla diversa considerazione che fanno rispetto a $P(x_i|y)$ e la penalizzazione che Bernoulli applica alla non presenza di alcune parole nel sample.