Confronto tra le prestazioni di voted perceptron e perceptron algorithms

Edoardo Wijaya Grappolini

Università degli Studi di Firenze
Esame di Intelligenza Artificiale A.A. 2018 - 2019

1 Introduzione

Si descive in seguito quanto fatto per confrontare gli algoritmi perceptron e voted perceptron per il training e predizioni tremite percettrone. Essendo un percettrone "isolato", esso è designato a classificazioni di tipo binario. In particolare, non sono state usate tecniche per aumentare il numero di dimensioni dello spazio, oltre a quello definito dalle caratteristiche del dataset (come il kernel trick); dunque è necessario (o comunque auspicabile) l'utilizzo di un dataset "sufficientemente linearmente separabile di default". Il tutto è stato fatto con in mente lo scopo di misurare e confrontare le prestazioni dei due algoritmi, soprattutto in termini di accuratezza; sono tuttavia state misurati anche i tempi di esecuzione per training e predizione, questo perchè per natura dell'algoritmo di voted perceptron, si nota che la fase di predizione è molto più dispendiosa in quanto, di fatto, viene fatta una media pesata dei percettroni ricavati; nel momento in cui i percettroni sono tanti, si hanno anche tanti prodotto scalari da calcolare, cosa che presumibilmente influisce sulle prestazioni. Nel seguente non si analizzano i tempi, per esigenze di brevità, si rimanda tuttavia a [6] per i dati sperimentali raccolti, in forma integrale.

2 Esecuzione

Per quanto concerne l'esecuzione degli esperimenti il processo può essere schematizzato in 3 fasi:

1. Implementazione dei due algoritmi. Per il voted perceptron è stato implementato in python il codice descritto in [1], analogamente è stato fatto per l'implementazione per il perceptron algorithm "classico" (Rosenblatt (1958, 1962)), con l'aiuto della descrizione in [2].

2. Breve verifica del funzionamento degli algoritmi come classificatori. Per fare quanto detto è stato inizialmente dato ai due algoritmi l' iris dataset, reperibile nelle repositories linkate dell'UCI. Il dataset è piccolo ma ha il pregio di essere linearmente separabile, permettendo di verificare se i due algoritmi riuscissero effettivamente a trovare il separatore lineare dei punti. Questa verifica non viene riproposta nel codice in quanto poco interessante.

3. Ricerca di dataset adatti e passaggio all'esecuzione per la raccolta dei dati sperimentali. Segue la descrizione dei datasets e brevemente il preprocessamento attuato:

   - Occupancy detection [3]: dati sull'uccupazione di una stanza utilizzando dati di sensori (e.g. umidità, luminosità, etc.) Su questo dataset è stato fatto poco preprocessing, i dati hanno un range sufficientemente contenuto. La parte di preprocessing è stata fatta solo per l'adattamento degli esiti che devono possono essere {-1, +1} per essere compatibili con i due algoritmi, per come sono stati strutturati nel progetto.
   - Banknote authentication [4]: dataset che in base a informazioni sulla banconote (e.g. spettrografia, misure, etc.) stabilisce se una banconota è autentica o contraffatta. Anche in questo caso il preprocessing è stato limitato, i livelli di accuratezza sono alti per i due algoritmi ed i risultati rispecchiano quanto atteso.
   - Adult dataset [5]: dataset che stabilisce, date informazioni su alcuni individui, come stato sociale, istruzione, provenienza, etc. stabilisce se una persona ha uno stipendio annuale >= 50.000$ o meno. In questo caso il preprocessing è stato più intensivo. I dati di forma

testuale sono stati convertiti in variabili indicatrici tramite la libreria pandas e la funzione get_dummies(). Sono state tolte colonne ridondanti, poi è stato posto l'esito nell'insieme {-1, +1} anzichè {0, 1}, per le ragioni indicate al punto 1. Inoltre sono state scalate le colonne che avevano valori in range fuori dall'intervallo [-1,1]; questo è stato fatto con la formula $X' = (X - X\_min)/X\_max - X\_min$ (con x_min max il minimo e massimo di ogni colonna) su ogni riga del dataset, per quelle colonne che ne avevano bisogno. Per quanto poco interessante ai fini dello scopo del progetto in sè, tutti i files utilizzati per il preprocessing sono disponibili al link https://github.com/Edrid/preprocessing_IA.git Gli esperimenti e la raccolta dei dati sperimentali viene descritta in seguito.

3 Esperimenti eseguiti

Per ogni dataset è stata fatta una funzione dedicata della forma "dataset$n_nome_dataset()$". Queste funzioni fanno tutte le stesse operazioni: caricano il csv del dataset in memoria primaria e lo passano come argomento agli oggetti creati, di tipo Perceptron e VotedPerceptronV2 utilizzati per incapsulare logica e dati dei due algoritmi. Pur non essendo una buona pratica di programmazione, questo metodo permette di richiamare in modo immediato i metodi una volta definiti. Per la riproduzione degli esperimenti, e la raccolta dei dati sperimentali è dunque sufficiente chiamare tali funzioni. Questi metodi ritornano la n-upla della seguente forma:
⟨ unvoted_perf, voted_perf, unvoted_training_time, unvoted_pred_time,
voted_training_time, voted_pred_time ⟩ Con: 1. unvoted_perf := performance del percettrone non votato; 2. analogo a 1; 3. unvoted_training_time := tempo di training del percettrone non votato; 4. unvoted_pred_time := tempo di predizione del percettrone non votato; 5., 6. analoghi a 3.,4. .

Per i test consecutivi è stato creata una funzione che permettesse di iterare il processo più volte, incrementando il numero di epoche per il training. Per riprodurre i risultati sperimentali è possibile far eseguire i test consecutivi tramite la funzione run_tests_and_save_results(). Il ciclo fa n iterazioni, che rappresentano un numero crescente di epoche (EPOCHE=1,...,n), dentro il ciclo che incrementa il numero di epoche è inoltre presente un ciclo innestato che ripete l'esperimento con un numero $i \in 1, ..., n$ di epoche, un numero k di volte, questo per permettere di stabilizzare i risultati ed avere risultati possibilmente obiettivi, attenuando eventuali "singolarità statistiche". La variazione del numero di epoche è fatto tramite l'incremento della variabile globale EPOCHE. La stampa in console delle matrici di confusione viene fatta dai metodi measureVotedPerformance() e measureUnvotedPerformance(). Per riprodurre cambiare il dataset utilizzato è sufficiente chiamare la funzione corretta del dataset a cui vogliamo riferirci, all'interno del ciclo in run_tests_and_save_results().

4 Risultati sperimentali e interpretazioni

I risultati sperimentali sono riportati nella Tabella 1. Per tutti i dati sperimentali raccolti si rimanda alla repository github del progetto, e nella directory Files. I files utili sono, per ciascun dataset preso in considerazione, i files "[nome_dataset]_results.csv" e "[nome_dataset]_mdc" [7] Nella tabella sono riportate alcune matrici di confusione come campioni dei risultati. Gli esperimenti sono stati ripetuti più volte con un numero di epoche crescente, per fare in modo da verificare che le performance dei due percettroni non fosse solamente diretta causa di un "undertraining". Le matrici di confusione -M- sono della forma M(0,0): Classe 1 correttamente predetti, M(0, 1): Classe 1 erratamente predetti, M(1,0): Classe 2 erratamente predetti, M(1,1): Classe 2 correttamente predetti. Al termine delle righe sono riportati gli "score" dei percettroni non votati, e dei percettroni votati; questi numeri rappresentano un diretto confronto tra le prestazioni dei due algoritmi: ad uno dei due algoritmi è attribuito un punto ogni volta che l'accuratezza di uno è superiore all'altro. Le somme dei due numeri rappresentano il numero totale di prove fatte (max#epoche * iterazioni_fatte nei test). E' inoltre riportata la media totale dell'accuratezza per ciascun dataset, e per ciascun algoritmo. Date questi due ultimi dati (ma d'altra parte è una cosa che salta all'occhio guardando i file con i dati integrali) è possibile notare il seguente: il voted perceptron ha una performance, il più delle volte, migliore rispetto alla performance del percettrone non votato, ma questo non si verifica nella totalità delle volte. Dunque è possibile che il voted perceptron venga "battuto"

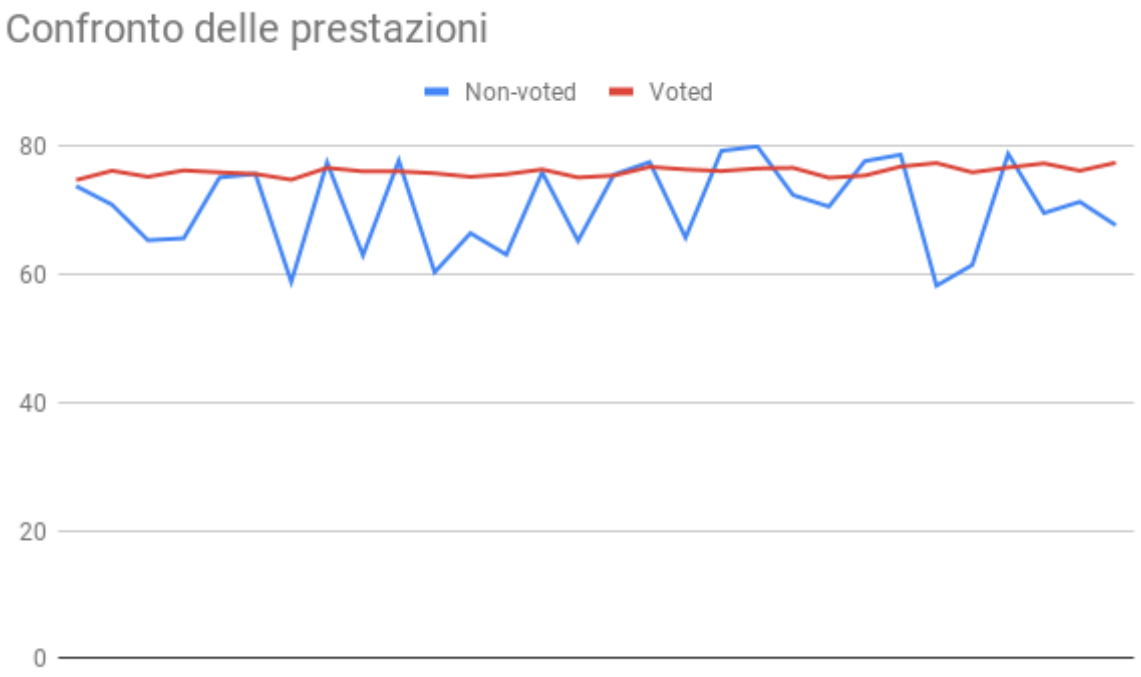

Figure 1: Confronto delle prestazioni su dataset "adult"

dal percettrone non votato (tranne nel dataset occupancy, che si è notevolmente prestato agli esperimenti), ma si tratta per lo più di "singolarità" sparse: la media dei risultati parla chiaro. Questo fatto si nota bene plottando il grafico di tutti i dati raccolti per il dataset "adult" (a titolo d'esempio); in questo si nota che, pur essendo possibile che il percettrone non votato abbia performance, alle volte, migliori, spesso ha prestazioni notevolmente peggiori. Ed è interessante l'andamento del grafico, che mostra, almeno in via sperimentale, come l'algoritmo voted perceptron sia un algoritmo molto più *consistente* e *robusto* rispetto al semplice percettrone non votato; si vede infatti che l'andamento del voted perceptron è per lo più costante, confrontato all'andamento del percettrone non votato. Il vantaggio si nota soprattutto per dataset modestamente rumorosi. Nota: nella figura, l'asse delle ascisse cresce al crescere del numero di epoche, nel caso specifico da 1 a 6.

References

[1] YOAV FREUND, ROBERT E. SCHAPIRE. *Large Margin Classification Using the Perceptron Algorithm* at: https://link.springer.com/content/pdf/10.1023/A:1007662407062.pdf

[2] https://en.wikipedia.org/wiki/Perceptron

[3] https://archive.ics.uci.edu/ml/datasets/Occupancy+Detection+

[4] https://archive.ics.uci.edu/ml/datasets/banknote+authentication

[5] http://archive.ics.uci.edu/ml/datasets/adult

[6] https://github.com/Edrid/voted-vs-unvoted-perceptron/tree/master/Files

[7] https://github.com/Edrid/voted-vs-unvoted-perceptron/tree/master/Files

Occupancy:	#Epoche: 1 [UNVOTED] [[745 227] [54 1639]] [VOTED] [[972 0] [56 1637]]	#Epoche: 2 [UNVOTED] [[972 0] [148 1545]] [VOTED] [[972 0] [56 1637]]	#Epoche: 3 [UNVOTED] [[972 0] [77 1616]] [VOTED] [[972 0] [56 1637]]	#Epoche: 4 [UNVOTED] [[65 907] [0 1693]] [VOTED] [[971 1] [56 1637]]	#Epoche: 5 [UNVOTED] [[971 1] [77 1616]] [VOTED] [[971 1] [56 1637]]	#Epoche: 6 [UNVOTED] [[969 3] [55 1638]] [VOTED] [[971 1] [56 1637]]	#Epoche: 7 [UNVOTED] [[972 0] [113 1580]] [VOTED] [[971 1] [56 1637]]
Banknotes:	[UNVOTED] [[96 54] [0 150]] [VOTED] [[95 55] [0 150]]	[UNVOTED] [[88 62] [0 150]] [VOTED] [[94 56] [0 150]]	[UNVOTED] [[96 54] [0 150]] [VOTED] [[98 52] [0 150]]	[UNVOTED] [[76 74] [0 150]] [VOTED] [[86 64] [0 150]]	[UNVOTED] [[84 66] [0 150]] [VOTED] [[85 65] [0 150]]	[UNVOTED] [[79 71] [0 150]] [VOTED] [[90 60] [0 150]]	[UNVOTED] [[83 67] [0 150]] [VOTED] [[80 70] [0 150]]
Adults:	[UNVOTED] [[908 92] [490 510]] [VOTED] [[922 78] [398 602]]	[UNVOTED] [[995 5] [818 182]] [VOTED] [[923 77] [427 573]]	[UNVOTED] [[964 36] [702 298]] [VOTED] [[915 85] [402 598]]	[UNVOTED] [[807 193] [256 744]] [VOTED] [[908 92] [372 628]]	[UNVOTED] [[789 211] [215 785]] [VOTED] [[898 102] [361 639]]	[UNVOTED] [[941 59] [549 451]] [VOTED] [[912 88] [365 635]]	

Occupancy:	#Epoche: 8 [UNVOTED] [[970 2] [74 1619]] [VOTED] [[971 1] [56 1637]]	#Epoche: 9 [UNVOTED] [[972 0] [77 1616]] [VOTED] [[971 1] [56 1637]]	#Epoche: 10 [UNVOTED] [[972 0] [131 1562]] [VOTED] [[971 1] [56 1637]]	Voted score: 67 Unvoted score: 3 Voted average accuracy: 97,87% Unvoted average accuracy: 93,96%
Banknotes:	[UNVOTED] [[87 63] [0 150]] [VOTED] [[84 66] [0 150]]	[UNVOTED] [[89 61] [0 150]] [VOTED] [[97 53] [0 150]]	[UNVOTED] [[86 64] [0 150]] [VOTED] [[96 54] [0 150]]	Voted score: 56 Unvoted score: 14 Voted average accuracy: 81,18% Unvoted average accuracy: 78,78%
Adults:				Voted score: 20 Unvoted score: 10 Voted average accuracy: 76,08% Unvoted average accuracy: 70,75%

Table 1: Tabella 1