Voted Perceptron

Edoardo Bonanni

Abstract

Implementazione dell'algoritmo voted perceptron descritto in Freund e Schapire 1999 e riproduzione di risultati analoghi a quelli riportati nella sezione 5 dell'articolo (in particolare i grafici per d = 1 e d = 2 della figura 2) ma utilizzando il dataset Zalando al posto di MNIST.

1 Introduzione

L'algoritmo descritto sfrutta i dati che sono separabili linearmente con ampi margini. Se i dati sono linearmente separabili l'algoritmo alla fine convergerà e il vettore di predizione sarà corretto in tutti gli esempi di apprendimento.
Quindi se i dati sono separabili linearmente e si ripete molte volte la procedura di apprendimento (ripetendola un numero di volte pari al numero di epoche T), il voted perceptron convergerà al normale uso dell'algoritmo perceptron che è quello di predire usando il vettore di predizione finale.

2 Dataset Zalando

In questo esperimento a differenza del dataset MNist utilizzato negli esperimenti descritti nell'articolo Freund e Schapire 1999 utilizzeremo il dataset Zalando.
Il dataset Zalando come quello MNist è formato da 60000 esempi di training e 10000 esempi di test.
Dalla seguente tabella è possibile scaricare i file compressi che contengono gli esempi di training e di testing del dataset Zalando.

Name	Content	Examples	Size	Link
train-images-idx3-ubyte.gz	training set images	60000	26 MBytes	Download
train-labels-idx1-ubyte.gz	training set labels	60000	29 KBytes	Download
t10k-images-idx3-ubyte.gz	test set images	10000	4.3 MBytes	Download
t10k-labels-idx1-ubyte.gz	test set labels	10000	5.1 KBytes	Download

Attraverso alcune funzioni in Python si può decomprimere i file e visualizzarne il contenuto attraverso una immagine (funzioni presenti nel programma python nel file "helper.py" nella cartella "utils").
Un esempio del contenuto dei file train-images e test-images è il seguente.

Figure 1: contenuto ridotto del file train-images

3 Kernel-based classification and multiclass data

Come descritto nella sezione 4 di Freund e Schapire 1999 la funzione del kernel che si utilizza è l'espansione polinomiale.

$$\mathrm{K(x,\ y)} = (1 + x \cdot y)^d$$

Quindi per ogni istanza x del dataset per calcolare il vettore predizione v_k si utilizza la seguente formula:

$$v_k \cdot x = \sum_{j=1}^{k-1} y_{ij} \cdot K(x_{ij}, x)$$

Invece come descritto nella sezione 5 di Freund e Schapire 1999 per gestire i dati multiclasse si riduce il problema generale a 10 problemi binari.
Nel dataset Zalando le etichette (labels) che assumono i valori nell'insieme {0,1, ... ,9} in realtà sono dei vestiti. Ogni classe infatti rappresenta una classe di vestiti. L'insieme è formato da 10 classi di vestiti che sono le seguenti:
{t-shirt-top, trouser, pullover, dress, coat, sandal, shirt, sneaker, bag, ankle-boots}
Durante il training sulla classe l, si sostituisce ogni esempio (x_i, y_i)
(dove $y_i \in$ {0,1, ... ,9}) con (x_i, +1) se $y_i = l$ e da (x_i, -1) se y_i != l.
Per fare previsioni su una nuova istanza x, si utilizzano quattro metodi diversi (last-unnormalized, vote, average-unnormalized e random-unnormalized).
In ogni metodo, calcoliamo prima un punteggio s_l per ogni $l \in$ {0,1, ... ,9} e quindi

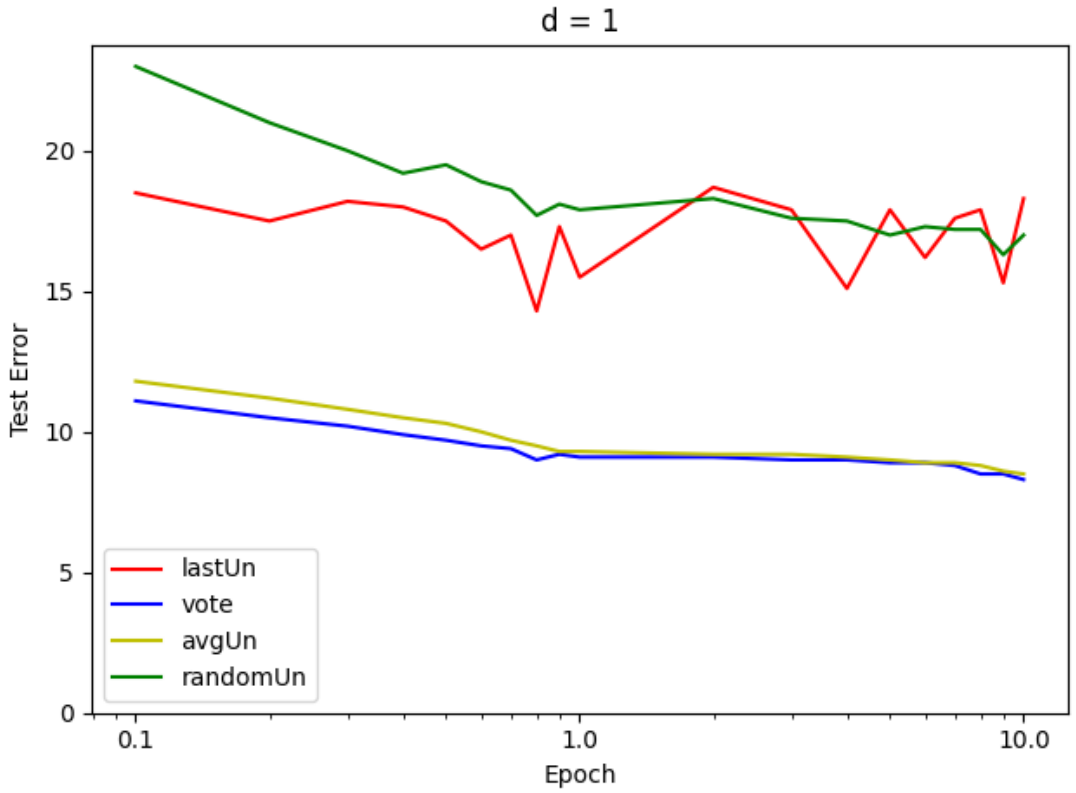
andiamo a prevedere utilizzando l'etichetta che ha il punteggio più alto, essa viene calcolata utilizzando la seguente funzione:

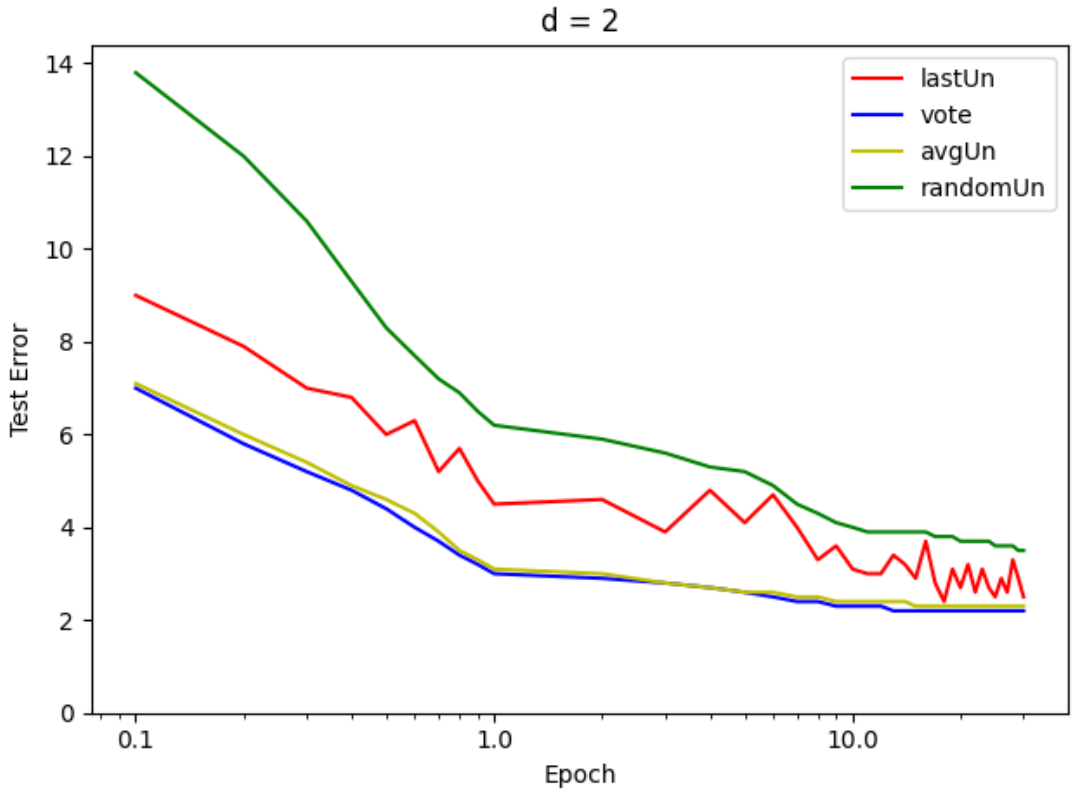
$$\hat{y} = \underset{l}{\operatorname{argmax}}\, s_l$$

4 Implementazione

L'algoritmo di training è implementato nella classe "VotedPerceptron" del file "VotedPerceptron.py".
Per fare le predizioni si utilizza la funzione "predLabelWithPolExp" che si trova nel file "usefulFunctions.py" (la funzione considerata si trova in un file a parte perché serve sia per fare la predizione sulle etichette nel training sia nel testing, allora ho preferito inserirla in un file contenente tutte le funzioni utili del programma per non ripeterla 2 volte).
Tutte le funzioni di predizione sulle istanze x del dataset (last-unnormalized, vote, average-unnormalized e random-unnormalized) sono implementate nel file "predictions.py".
Le funzioni che si occupano di effettuare il training e il testing sulle diverse classi e sui diversi kernel degree si trovano rispettivamente nei file "training.py" e "test.py".
Per informazioni dettagliate sul funzionamento delle varie funzioni implementate nel progetto si rimanda il lettore a consultare il file README del progetto su github.

5 Risultati Sperimentali





Come descritto nell'articolo per $d = 1$ eseguiamo solo 10 epoche perché i dati non sono separabili linearmente, il che significa che l'algoritmo tende a fare molti errori e quindi l'esecuzione viene rallentata in modo significativo.
Come si vede dai grafici nel passaggio da $d = 1$ a $d = 2$ si ha un incremento significativo delle prestazioni perché il numero di errori di predizione è molto minore e quindi il tempo di esecuzione cala drasticamente (nonostante per $d = 2$ si svolgono 30 epoche a differenza di 10 del caso $d = 1$ il tempo di esecuzione è di poco superiore).

6 Conclusioni

I risultati sono in linea con quelli dell'articolo anche se i miei risultati hanno un error rate leggermente superiore.
Questo può essere dato dal fatto che sto utilizzando un dataset diverso cioè il dataset Zalando invece che MNist oppure da una diversa implementazione di qualche funzione in Python che genera un error rate leggermente maggiore.