B003725 - Artificial Intelligence
Voted & Average Perceptron

Relazione del Progetto per il corso di Artificial Intelligence

Autore: Arturo Viti

Gennaio 2024

Indice

Sommario

Implementazione dell'algoritmo Voted Perceptron e della sua variante Average in Python descritto in Freund e Schapire (1999) e riproduzione di risultati analoghi a quelli riportati nella sezione 5 dell'articolo (in particolare i grafici per $d = 1$ della *Figura 2* utilizzando il Data Set MNIST

1 Introduzione

La classificazione binaria implementata permette all'algoritmo di capire se un record del Test Set corrisponde a un numero maggiore o uguale a 5.

2 Data Set

Il Data Set MNIST contiene **70.000 record** di cifre scritte a mano con la opportuna label decimale e ciascun record ha **784 feature**, che corrispondono ai pixel di un'immagine 28×28 in scala di grigi.

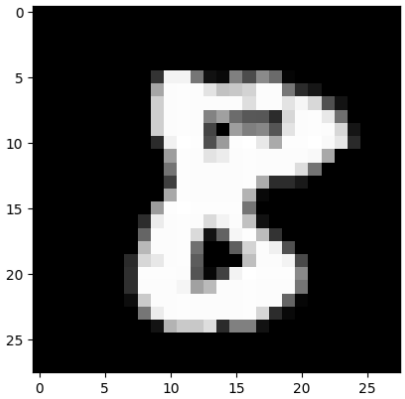

Figura 1: Esempio di un record del Dataset MNIST

Ciascun record ha anche la sua label decimale corrispondente.

3 Implementazione

3.1 Elaborazione del Data Set

Dopo l'importazione del Data Set fornito in file .arff, vengono effettuate alcune manipolazioni all'interno:

- **loadDataset(dsPath)**: carica il Data Set in formato .arff come un NumPy Array a elementi interi

- **splitDataset(mnistData, SPLIT_RATIO)**: il Data Set viene mescolato casualmente e diviso secondo lo *SPLIT_RATIO* passato come parametro.
  Es: 0.9, indica il 90% di Training Set e 10% di Test Set
- **classifyDataset(trainData, CLASS_INDEX)**: ciascun record del Training Set viene classificato in maniera binaria:

$$\forall r_i \in TrainingSet,\ r_i[CLASS\_INDEX] = \begin{cases} 1 & \text{se } r_i \geq 5 \\ -1 & \text{altrimenti} \end{cases}$$

Dove *CLASS_INDEX* è la posizione della colonna contenente la label nel Training Set.

3.2 Training

L'implementazione Voted & Average Perceptron in fase di training è la stessa Hal Daume III A Course in Machine Learning (2017), quindi è stato deciso di creare una classe **VotedPerceptron** con gli stessi attributi e metodi.

3.3 Test

La fase di *predizione* è diversa, quindi i metodi **multiplePredict(X, avgPerceptronModel=False)** e **predict(X, avgPerceptronModel=False)** sono parametrizzati in modo che, l'utilizzatore della classe, possa scegliere quale variante implementare.
Attraverso il profiler, durante lo sviluppo mi sono accorto di un "collo di bottiglia" dovuto a prodotti numpy nella fase di predict. Un'ottimizzazione fatta è quella di utilizzare gli array *trasposti*, perché ciò permette un allineamento delle dimensioni in memoria NumPy Broadcasting

3.4 Documentazione del Codice

La documentazione più dettagliata sul sorgente, l'utilizzo dello script e i requisiti per lanciare lo script da terminale è fornita nel file **README** corrispondente e all'interno dei commenti **PyDoc**.

4 Risultati Sperimentali

Figura 2: Test Error per Numero di Epoche

Come ci si può aspettare, al crescere del numero delle epoche, il Test Error diminuisce a patto di qualche oscillazione, in quanto il Data Set non è linearmente separabile.

5 Conclusioni

I risultati sono in linea con quelli dell'articolo anche se i miei risultati hanno un Test Error leggermente superiore. Ciò può dipendere da un'implementazione diversa, dalla diversa gestione dei dati, diversa versione del Data Set.