Analisi dell'algoritmo "AdaBoost"

Elaborato di Intelligenza Artificiale

Tommaso Botarelli

UNIVERSITÀ
DEGLI STUDI
FIRENZE

1 Introduzione

L'elaborato tratta l'applicazione dell'algoritmo "AdaBoost" a 4 differenti Dataset divisi casualmente in 80% in training set e 20% testing set. In questa breve relazione viene illustrato come è stato svolto il lavoro, i risultati ottenuti, come riprodurli e il confronto con i risultati aspettati discussi nel corso di Intelligenza Artificiale.

2 Adaboost

L'algoritmo Adaboost è un algoritmo di boosting cioè un algoritmo di apprendimento che mette insieme più ipotesi $h_1, ..., h_n$ e combina le loro predizioni mediante un meccanismo di voto che in questo caso avviene come una somma pesata delle predizioni. Il peso di ogni ipotesi è dato nel seguente modo: partendo da un vettore di pesi uguali per ogni esempio ad ogni boosting round, generata una ipotesi, si esegue la predizione e si diminuisce il peso di quegli esempi classificati correttamente e si aumenta a quegli esempi classificati in modo scorretto. In questo modo nel boosting round successivo l'ipotesi generata sarà più portata a classificare nel modo corretto anche gli esempi che sono stati classificati erroneamente nei round precedenti e che quindi hanno un peso maggiore.

Gli algoritmi di boosting quindi utilizzano uno specifico base learner, cioè un algoritmo di apprendimento in grado di rendere una ipotesi. Una particolarità di Adaboost è che il base learner può essere anche un "weak learner" e cioè un algoritmo di apprendimento che fa anche di poco meglio di una classificazione casuale. Per esempio l'algoritmo di apprendimento "Decision stump" divide il dataset con un singolo nodo di un albero di decisione scelto fra quell'attributo e quel valore soglia tale che rende massima la percentuale di successo nella predizione (sempre come somma pesata con il peso degli esempi).

Una particolarità dell'algoritmo Adaboost è il fatto che anche usando un weak learner come Decision stump (quindi un algoritmo estremamente semplice) si possono raggiungere risultati molto promettenti come vedremo nel seguito di questa relazione. Inoltre altra cosa positiva di questo algoritmo di boosting è che l'aggiunta di boosting round fa solo migliorare la predizione sui dati di training e anche quando questa arriva ad un successo del 100%, la percentuale di successo sul testing continua ad aumentare.

In questa relazione l'intento è quindi quello di ritrovare sperimentalmente queste caratteristiche.

3 Datasets

Sono stati utilizzati i seguenti datasets:

1. MAGIC Gamma Telescope Data Set (dataset di classificazione binaria).
2. Anuran Calls (MFCCs) Data Set. Dal momento che questo è un dataset multiclasse è stata utilizzata come classe per la classificazione la categoria 'Family' e all'interno di questa è stata scelta come classificazione positiva 'Leptodactylidae' e come negativa 'Hylidae'.

3. Avila Data Set. Dal momento che questo non è un dataset di classificazione binaria è stata scelta come classe positiva gli esempi classificati con 'A' e con classe negativa gli esempi classificati con 'F'. Per avere un dataset più grande è stata utilizzata l'unione dei dataset di training e testing (in cui il dataset è originariamente reperibile dal sito)

4. Skin Segmentation Data Set (dataset di classificazione binaria)

Nei datasets adattati per la classificazione binaria sono state scelte le classi positive e negative in base a quelle che comparivano più spesso nel dataset stesso.

Per informazioni riguardo alla distribuzione dei dati si rimanda direttamente al link di "UCI machine learning repository".

4 Codice

Il lavoro è stato effettuato nel linguaggio "Python". Questo linguaggio di programmazione è molto adeguato al lavoro da svolgere, perché assieme a Pandas e a NumPy permette una gestione facilitata dei dataset e di tutte le operazioni matematiche su array e non. Oltre a Pandas e NumPy è stato utilizzato anche la funzione *train_test_split()* della libreria sklearn. Questa ultima funzione è stata utilizzata per la divisione dei dataset in dati per il training e dati per il testing.

All'interno del codice sorgente si possono trovare 3 file:

- ***main.py***, contiene le funzioni che modificano i dataset per renderli compatibili con l'esecuzione dell'algoritmo Adaboost (per esempio sostituendo il nome della classe con "1" oppure "-1"). Inoltre la funzione *input_value_for_test()* permette l'introduzione da tastiera del numero di ensambles per il test in questione e il numero di valori scelti casualmente per la scelta del valore di soglia (vedere sezione "Risultati"). Da qui è possibile implementare funzioni ad hoc per nuovi datasets da testare. Per fare ciò basta conformare il datasets alla classificazione binaria e richiamare la classe "Test".

- ***Test.py***, contiene la classe "Test". Questa classe espone il metodo *run_test_get_result()* che viene richiamato dalle funzioni del *main* per eseguire i test. Questa funzione prima suddivide il dataset in training set (80%) e testing set (20%) e poi lancia l'algoritmo Adaboost.

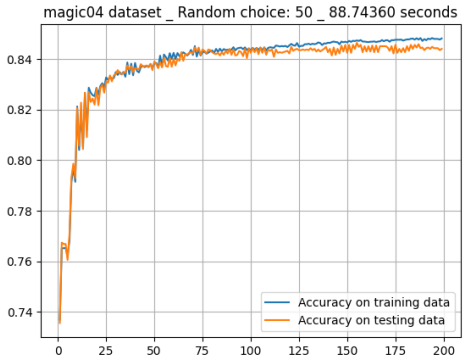
- ***AdaBoost.py***, file che contiene la classe Adaboost (che implementa l'algoritmo) e la classe DecisionStump. La classe DecisionStump è necessaria perché l'algoritmo AdaBoost trattato in questo lavoro utilizza come weak learner proprio Decision stump.

Per ulteriori informazioni sulla struttura del codice sorgente il file README.txt allegato al progetto contiene maggiori approfondimenti.

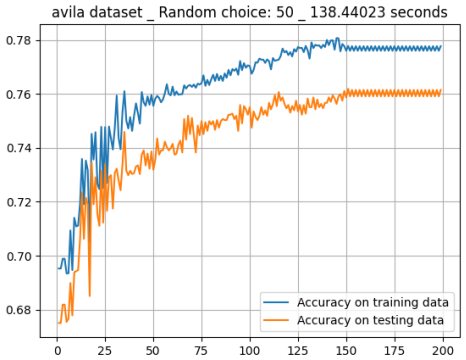
5 Risultati

I risultati seguenti sono stati ottenuti fornendo per ogni dataset un numero massimo di boosting round di 200 e una scelta random di 50 valori per definire il decision stump.

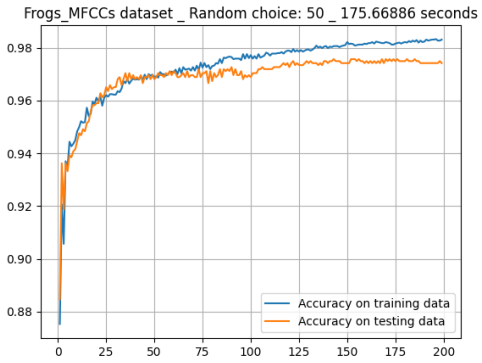
Questi risultati sono riproducibili perfettamente introducendo gli stessi valori nella console all'avvio del programma. Infatti anche laddove vengono eseguite scelte random queste vengono effettuate con un seed che si basa sui parametri in ingresso dell'utente.



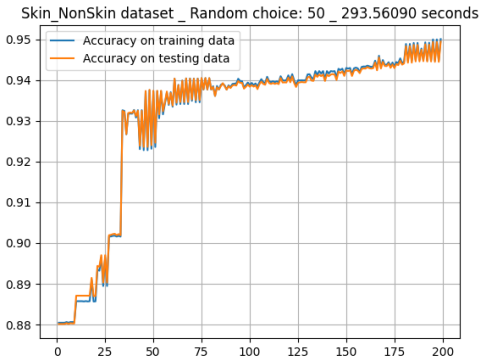
(a) magic04



(b) avila



(c) frog_MFCCs

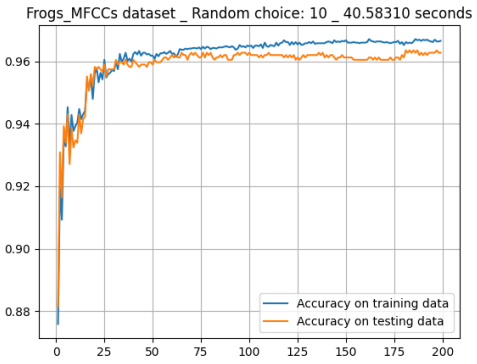


(d) Skin_NonSkin

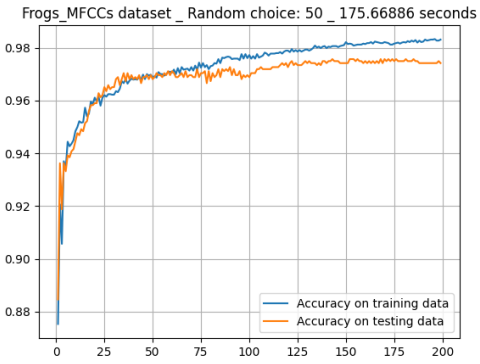
Figure 1: Risultati ottenuti con l'applicazione di 200 boosting rounds e 50 random choice

Il numero di boosting round è il numero di ipotesi per ogni test da cui effettuare l'ensambling.

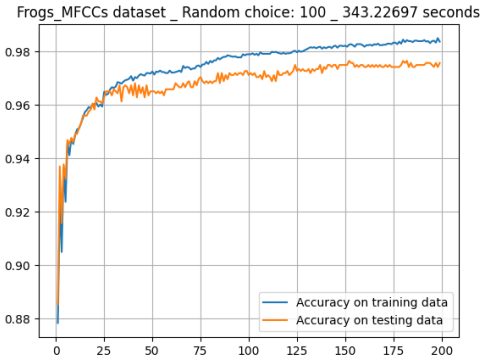
Dal momento che i datasets utilizzati hanno valori numerici non interi è stato utile introdurre il numero di valori considerati per la definizione dei Decision Stump. In questo modo i Decision stump vengono definiti selezionando per ciascun attributo n valori randomici e poi osservando fra questi quale è quello che massimizza la divisione dei dati. Questo procedimento permette una esecuzione più rapida del codice che altrimenti impiegherebbe molto più tempo. Inoltre in termini di accuratezza una scelta randomica per il valore di soglia dei Decision stump non porta a peggioramenti considerevoli della potenza dell'algoritmo. Di seguito tre grafici dell'applicazione di Adaboost al dataset 'Frogs_MFCCs' che spiegano bene le differenze in tempo di esecuzione e prestazioni:



(a) Random choice: 10



(b) Random choice: 50



(c) Random choice: 100

Figure 2: Confronto dell'esecuzione dell'algoritmo con numero di valori scelti casualmente diverso

6 Coclusioni

In conclusione dai grafici ottenuti si vede come l'algoritmo performi molto bene per i datasets "Frog_MFCCs" e "Skin_NonSkin". In quest'ultimo dataset dal grafico si può evincere una stretta correlazione fra i dati, infatti le curve sul training set e sul testing set sono praticamente sovrapponibili. Sugli altri due dataset l'algoritmo ha delle prestazioni peggiori in termini di accuratezza, ma comunque si vede come al crescere dei boosting round l'accuratezza aumenti confermando i risultati teorici.

Nel grafico del dataset "avila" a partire dal boosting round 150 fino all'ultimo boosting round si è evidenziata una sostanziale non crescita dell'accuratezza. Probabilmente questo è dovuto alla conformazione del dataset (l'analisi dei dataset non è compreso nel lavoro svolto) dal momento che questa anomalia non appare negli altri insiemi di dati.