Stanford, 2016 fall, A. Ng, J. Duchi, HW2, pr. 6.d25

Apachitei Maria-Luisa MOC1, Bumbu Ana-Maria MISS1

February 2021

1 Cerinta

(d) [11 points] Now you will implement boosting on data developed from a physics-based simulation of a high-energy particle accelerator. We provide two datasets, `boosting-train.csv` and `boosting-test.csv`, which consist of training data and test data for a binary classification problem on which you will apply boosting techniques. (For those not using **Matlab**, the files are comma-separated files, the first column of which consists of binary ± 1-labels $y^{(i)}$, the remaining 18 columns are the raw attribtues.) The file `load_data.m`, which we provide, loads the datasets into memory, storing training data and labels in appropriate vectors and matrices, and then performs boosting using *your* implemented code, and plots the results.

i. [5 points] Implement a method that finds the optimal thresholded decision stump for a training set $\{x^{(i)}, y^{(i)}\}_{i=1}^m$ and distribution $p \in \mathbb{R}_+^m$ on the training set. In particular, fill out the code in the method `find_best_threshold.m`. Include your code in your solution.

ii. [2 points] Implement boosted decision stumps by filling out the code in the method `stump_booster.m`. Your code should implement the weight updating at each iteration $t = 1, 2, \dots$ to find the optimal value θ_t given the feature index and threshold. Include your code in your solution.

iii. [2 points] Implement *random* boosting, where at each step the choice of decision stump is made completely randomly. In particular, at iteration t random boosting chooses a random index $j \in \{1, 2, \dots, n\}$, then chooses a random threshold s from among the data values $\{x_j^{(i)}\}_{i=1}^m$, and then chooses the tth weight θ_t optimally for this (random) classifier $\phi_{s,+}(x) = \mathrm{sign}(x_j - s)$. Implement this by filling out the code in `random_booster.m`.

iv. [2 points] Run the method `load_data.m` with your implemented boosting methods. Include the plots this method displays, which show the training and test error for boosting at each iteration $t = 1, 2, \dots$. Which method is better?

1.1 Notatii

- Algoritmul *RDS_B* = algoritmul bazat pe alegerea random a compasilor de decizie

- Algoritmul *BDS_B* = algoritmul bazat pe alegerea compasilor de decizie in mod greedy

2 Detaliile de implementare

2.1 load_dataset.py

1. Incarcarea datelor de antrenament si test
2. Rularea algoritmilor *RDS_B* si *BDS_B*
3. Generarea rapoartelor bazate pe erorile gasite pe setul de test si setul de antrenament
   - Se preia eroarea minima rezultata la rularea algoritmului *RDS_B* pe setul de antrenament si setul de test
   - Se compara cu erorile rezultate din rularea algoritmului *BDS_B*, astfel incat sa se determine cel mai apropiat vecin de minimul gasit de *RDS_B*
   - Se creaza un fisier *.json* cu rezultatele obtinute - astfel vom putea trage concluziile pentru rezolvarea subpunctului *iv.*

2.2 find_best_threshold.py

- Alegerea celui mai bun compas de decizie in mod greedy

2.3 stump_booster.py

- Implementarea Algoritmului *BDS_B*
- Alegerea compasului de decizie se va face cu ajutorul functiei implementate in modulul *find_best_threshold.py*
- La fiecare iteratie, se vor afisa riscul/pierderea empiric/a si eroarea empirica
- Valorile returnate: *theta, feature_inds, thresholds*

2.4 random_booster.py

- Implementarea Algoritmului *RDS_B*
- Alegerea compasului de decizie se va face aleator
- La fiecare iteratie, se vor afisa riscul/pierderea empiric/a si eroarea empirica
- Valorile returnate *theta, feature_inds, thresholds*

3 Concluzii bazate pe rezultate

Exercitiul a avut ca scop evidentierea diferentelor dintre comportamentul algoritmilor *RDS_B* si *BDS_B*, prin analizarea graficelor ce arata eroarea de clasificare a algoritmilor de-a lungul iteratiilor - pe seturile de test si antrenament.

```
REPORT PLOT 1
{
    "MIN_TRAIN_ERROR_RND": 0.2206058190514149,
    "MIN_TRAIN_ERROR_STP": 0.19230769230769232,
    "MIN_TEST_ERROR_RND": 0.22501998401278978,
    "MIN_TEST_ERROR_STP": 0.2090327737809752,
    "ITERATION_RND_TRAIN": 181,
    "ITERATION_STP_TRAIN": 17,
    "ITERATION_RND_TEST": 196,
    "ITERATION_STP_TEST": 10
}

REPORT PLOT 2
{
    "MIN_TRAIN_ERROR_RND": 0.22518931845356716,
    "MIN_TRAIN_ERROR_STP": 0.19230769230769232,
    "MIN_TEST_ERROR_RND": 0.2322142286171063,
    "MIN_TEST_ERROR_STP": 0.2090327737809752,
    "ITERATION_RND_TRAIN": 194,
    "ITERATION_STP_TRAIN": 15,
    "ITERATION_RND_TEST": 197,
    "ITERATION_STP_TEST": 9
}
```

3.1 Rezolvarea subpunctului *iv.*

Din graficele si rapoartele prezentate mai sus, putem afirma cu certitudine ca algoritmul *BDS_B* ajunge la o eroare de antrenare si testare in semnificativ mai putini pasi decat algoritmul *RDS_B*. *Motivatie* : Raportul de la Plot 1 arata ca *RDS_B* ajunge la o eroare de 22.06% peste setul de antrenament dupa 181 iteratii, in comparatie cu *BDS_B*, care ajunge la o eroare aproximativ egala in doar 17 iteratii. Pe setul de test, rezultatele stau asemanator - in 10 iteratii, *BDS_B* ajunge la o eroare pe care *RDS_B* o poate atinge in 196 iteratii.
Asadar, algoritmul *BDS_B* demonstreaza acuratete si eficienta de rulare mai bune decat cele ale algoritmului *RDS_B*.

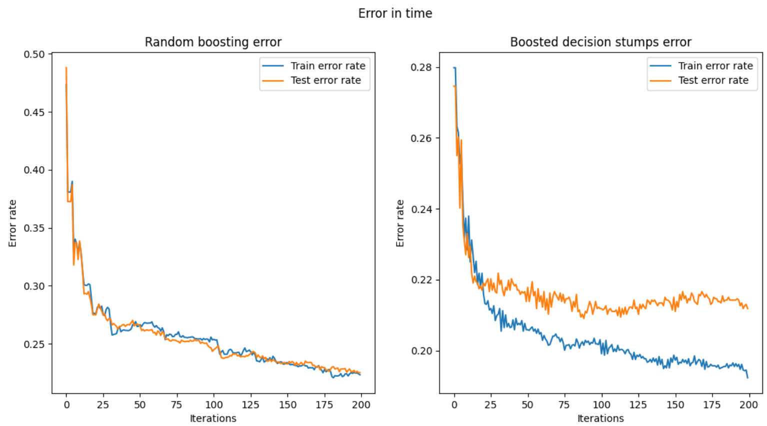

Figure 1: Plot 1

Figure 2: Plot 2

References

[1] https://profs.info.uaic.ro/ ciortuz/ML.ex-book/implementation-exercises/Stanford.2016f.ANg+JDuchi.HW2.pr6.AdaBoost+HighEnergyPhysics.sol.data.Matlab-code/ps2$_{k}ey.pdf$

[2] http://cs229.stanford.edu/extra-notes/boosting.pdf