INSA
ROUEN NORMANDIE

Informatique et Technologie de l'Information 4
Année universitaire 2024-2025

Practical Work

Boosting

simon.bernard@univ-rouen.fr

1 Multiclass classification with Boosting

For this first part, we're going to use Scikit-learn to test the Adaboost and Gradient Boosting methods. Scikit-learn offers two implementations of Gradient Boosting : `GradientBoostingClassifier` and `HistGradientBoostingClassifier`. The former is a classic implementation, while the latter is a faster implementation inspired by the LightGBM implementation. We will use the second implementation, as training times with the first are of the order of several minutes with reasonably large datasets.

1. We propose to use the covtype dataset available in the Scikit-learn datasets. You can load this dataset using the `fetch_covtype` function from scikit-learn :

```
from sklearn.datasets import fetch_covtype
X, y = fetch_covtype(return_X_y=True, as_frame=False)
```

2. Split the dataset into training and test sets with 50% of the data for training.
3. Train an AdaBoost classifier with 100 decision trees of maximum depths equal to 5 :

```
from sklearn.ensemble import AdaBoostClassifier
clf = AdaBoostClassifier(DecisionTreeClassifier(max_depth=3), n_estimators=100)
```

4. Train a Gradient Boosting classifier with 100 decision trees of maximum depths equal to 5 :

```
from sklearn.ensemble import HistGradientBoostingClassifier
hgb = HistGradientBoostingClassifier(max_iter=100, max_depth=5)
```

5. For comparison purposes, train a Random Forest classifier on the same training set with 100 random trees.
6. For each of the classifiers, evaluate their performance on the test set and measure the time it takes to train and test each classifier. To measure training and test times you can use the `time` package like this :

```
import time
start = time.time()
...
stop = time.time()
print(f"time: {stop - start}s")
```

7. Give a brief analysis of the results.

Note : The `covtype` dataset is a large dataset with 581,012 samples and 54 features. It may take some time to train the `AdaBoostClassifier` classifier. Write and test your code with a smaller training subset and only a few trees to make sure it works well before running the final experiment.

2 Depth of decision trees in Adaboost and Gradient Boosting

In this exercise, we propose to analyse the impact of decision tree depth on the performance of boosting methods. To do this, we propose to use two classification problems from the OpenML repository : `car` and `wdbc`. You can load these datasets using the `fetch_openml` function from scikit-learn.

1. Let's start with the `car` dataset. You can read the description of this dataset here. For loading the dataset :

```
from sklearn.datasets import fetch_openml
X, y = fetch_openml("car", return_X_y=True, as_frame=False)
```

2. Split the dataset into training/test subsets with 50% of the data for training.
3. Train an AdaBoost classifier on the training set with 100 decision trees of maximum depths equal to every integer value between 1 and 15. Evaluate the performance of each classifier on the test set.

4. Do the same with the `HistGradientBoostingClassifier` method.
5. Plot the accuracy with respect to the depth of the decision trees for both methods (on the same plot).
6. Repeat the same steps for the `wdbc` dataset. You can read the description of this dataset here.
7. Give a brief analysis of the results.

3 Gradient Boosting : learning rate versus number of trees

In this last experiment, we propose to monitor the evolution of the performance of gradient boosting over the iterations (i.e. as we add trees to the ensemble) for different values of the learning rate.

1. Load the California housing dataset from Scikit-learn. You can load this dataset using the `fetch_california_housing` function from scikit-learn :

```
X, y = fetch_california_housing(return_X_y=True, as_frame=True)
n_samples, n_features = X.shape
print(n_samples, n_features)
```

   Note that this is a regression problem.

2. Train a `HistGradientBoostingRegressor` on the entire dataset with 1000 iterations, maximum depth of trees equal to 5 and learning rates equal to 0.1, 0.01, and 0.001 :

```
lr_range = ...
for lr in lr_range:
   print(f"Learning rate: {lr}")
   gbm = HistGradientBoostingRegressor(... learning_rate=lr)
   ...
```

   Note that the whole experiment should take about 3 minutes to run.

3. A fitted `HistGradientBoostingRegressor` model can give access to the `train_score_` and `validation_score_` attributes that contains the evaluations of performances on training and validation sets at each iteration. To be able to access these attributes, one needs to enable *early stopping*, to set the fraction of instances to use for validation and to set the scoring function to monitor. To do so, look at the documentation. Once all of these parameters are set, use these attributes to plot the evolution of the training and validation performance of the model over the iterations for each learning rate.
4. Give a brief analysis of the results.