2EL1730: MACHINE LEARNING

CENTRALESUPÉLEC

Lab 6: Implementation of the AdaBoost Method

Instructors: Fragkiskos Malliaros and Maria Vakalopoulou
TAs: Yunshi Huang, Yoann Pradat, Mohamed El Amine Seddik, Jun Zhu

December 20, 2019

1 Description

The goal of this lab is to study ensemble learning methods and in particular the one of AdaBoost algorithm. Initially, we discuss the basic characteristics the AdaBoost algorithm, and then we will examine how it can be applied in a classification problem.

2 Boosting

Boosting is a powerful technique for combining multiple base classifiers to produce an ensemble classification model that can significantly outperform any of the base classifiers. The most widely used method is the one of *AdaBoost* (adaptive boosting). One basic charactering of boosting learning methods is that they can give good results even if the base classifiers have performance that is only slightly better than random (this is the reason why the base classifiers are also known as *weak learners*).

One characteristic of the boosting method is that the base classifiers are trained in sequence, and each base classifier is trained using a weighted form of the dataset – in which the weighting coefficient associated with each data point depends on the performance of the previous classifiers. When all the classifiers have been trained, their predictions are combined based on a weighted majority voting rule as shown in Fig. 1.

Consider a two-class classification problem, in which the training data comprises input vectors $\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_n$ along with the corresponding binary target variables $t_1, \ldots, t_n$ where $t_n \in \{-1, 1\}$. Each data point is given an associated weighting parameter w_n, which is initially set $1/n$ for all data points. We shall suppose here that we have a procedure available for training a base classifier using weighted data, to give a function $y(\mathbf{x}) \in \{-1, 1\}$. At each stage of the algorithm, AdaBoost trains a new classifier using a dataset in which the weighting coefficients are adjusted according to the performance of the previously trained classifier, so as to give higher weight to the misclassified data points. Finally, when the desired number of base classifiers have been trained, they are combined to form a committee using coefficients that give different weight to different base classifiers. The precise form of the AdaBoost method is given by Algorithm 1.

We see that the first base classifier $y_1(\mathbf{x})$ is trained using weighting coefficients $\mathbf{w}^{(1)}$ that are all equal, which therefore corresponds to the usual procedure for training a single classifier. In step 7 of the algorithm, we can see that in subsequent iterations, the weighting coefficients $w^{(t)}$ increase for data points

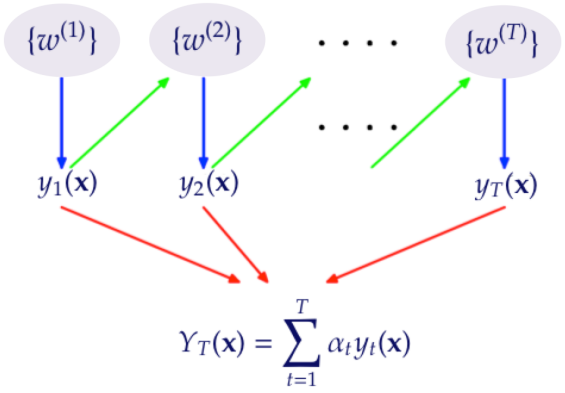

Figure 1: Schematic illustration of the boosting framework. Each base classifier $y_t(\mathbf{x})$ is trained on a weighted form of the training set (blue arrows) in which the weights $w^{(t)}$ depend on the performance of the previous base classifier $y_{(t-1)}(\mathbf{x})$ (green arrows). Once all base classifiers have been trained, they are combined to give the final classifier $Y_T(\mathbf{x})$ (red arrows).

Algorithm 1 AdaBoost $(D_n = \{(\mathbf{x_i}, t_i)\}_{i=1}^n)$

Input: Training dataset $(D_n = \{(\mathbf{x_i}, t_i)\}_{i=1}^n)$
Base classifier $\text{BASE}(\cdot, \cdot)$
Number of base classifiers T
Output: Class label t of $\mathbf{x}$

1: $\mathbf{w}^{(1)} \leftarrow (1/n, \ldots, 1/n)$ {Initial weights}
2: **for** $t \leftarrow 1$ to T **do**
3: $y_t \leftarrow \text{BASE}(D_n, \mathbf{w}^{(t)})$ {Calling the base learner}
4: $\gamma_t \leftarrow \dfrac{\sum_{i=1}^n w_i^{(t)} I(y_t(\mathbf{x_i}) \neq t_i)}{\sum_{i=1}^n w_i^{(t)}}$ {$I(y_t(\mathbf{x_i}) \neq t_i)$ is the indicator variable, gives 1 when $y_i^{(t)} \neq t_i$}
5: $\alpha_t \leftarrow \ln\left\{\dfrac{1-\gamma_t}{\gamma_t}\right\}$ {coefficient of y_t}
6: **for** $i \leftarrow 1$ to n **do**
7: $w_i^{(t+1)} \leftarrow w_i^{(t)} \exp\{\alpha_t I(y_t(\mathbf{x_i}) \neq t_i\}$ {Re-weighting the points}
8: **end for**
9: **end for**
10: **return** $Y_T(\mathbf{x}) = \text{sign}\left(\sum_{t=1}^T \alpha_t y_t(\mathbf{x})\right)$ {Make predicitons using the final model}

that are misclassified, while they remain the same for data points that are correctly classified. Successive classifiers are therefore forced to place greater emphasis on points that have been misclassified by previous classifiers, and data points that continue to be misclassified by successive classifiers receive ever greater weight. The quantities γ_t represent weighted measures of the error rates of each of the base classifiers on the data set. We therefore see that the weighting coefficients α_t defined in step 5 of the algorithm give greater weight to the more accurate classifiers, while computing the overall output given by equation of step 10.

3 Pipeline of the Task

In the lab, we will implement and apply AdaBoost to predict the forest cover type (the predominant kind of tree cover) from strictly cartographic variables[1]. Although the dataset contains multiple classes, we focus on instances that belong to classes "1" and "2" – thus, we have to deal with a binary classification problem.

In the code, we also make use of two widely used data analytics modules of Python, namely the `pandas` and `scikit-learn` packages. The first one offers functionalities for data manipulation, while the second one is the most widely used Python modules for machine learning.

For the classification task, we will use the decision tree classifier implemented at `scikit-learn`. After loading and preprocessing the dataset, we train the classifier and compute the performance using Python built-in functions.

```
from sklearn.tree import DecisionTreeClassifier

clf = DecisionTreeClassifier(max_depth=8)

# Train the classifier and print training time
clf.fit(X_train, y_train)

#%%
# Do classification on the test dataset and print classification results
from sklearn.metrics import classification_report
target_names = data['Cover_Type'].unique().astype(str).sort()
y_pred = clf.predict(X_test)
print(classification_report(y_test, y_pred, target_names=target_names))
```

3.1 Tasks to be done

Implementation of AdaBoost. In the lab, you will need to fill in the code that implements the AdaBoost classifier given in Algorithm 1. As base classifiers, we will use the decision tree offered by `scikit-learn`. The input to the algorithm is the number of decision trees T that will be used by the ensemble method and the depth of the decision trees (variable D). At the end of the process, we compute the training and test errors, so the last section of the code can plot the learning curves.

```
D = 2 # tree depth
T = 1000 # number of trees
w = np.ones(X_train.shape[0]) / X_train.shape[0]
training_scores = np.zeros(X_train.shape[0])
test_scores = np.zeros(X_test.shape[0])

ts = plt.arange(len(training_scores))
training_errors = []
test_errors = []

#================================
for t in range(T):

# Your code should go here


#================================
```

[1] See `https://archive.ics.uci.edu/ml/datasets/Covertype` for details about the dataset.

```
# Plot training and test error
plt.plot(training_errors, label="training error")
plt.plot(test_errors, label="test error")
plt.legend()
```

(Optional) Optimization of AdaBoost parameters. Then, you need to optimize the tree depth of AdaBoost. To do that, create a Python function that implements AdaBoost (adding your code from the previous question) and call it with different tree depths D (for simplicity, set $T = 100$ number of trees). Plot the final test error vs. the tree depth and discuss the plot.

References

[1] Bishop, Christopher M. Pattern Recognition and Machine Learning (Information Science and Statistics). Springer-Verlag New York, Inc., 2006.