Lectures on machine learning II —Boosted Decision Tree

Yu Zhang

IHEP, Beijing

Apr. 6th, 2020

Outline

Algorithms

- Pre-requirement : training and test dataset with given labels
- Root node
- Splitting
  - Scan the threshold of each variables and split the sample into two parts
  - Calculate the improvement of the figure of merit : $\Delta I = I - (\frac{n_1}{n_{int}} I_1 + \frac{n_2}{n_{int}} I_2)$
    - Entropy : $H = -plnp - (1-p)ln(1-p)$
    - Gini impurity : $G = p(1-p)$
    - Significance : $\sigma = n_s/\sqrt{n_s + n_b}$
    - where $p = n_s/(n_s + n_b)$
  - A set of (variable + threshold) with maximum ΔI determins the splitting
- Terminal node
- Prunning
  - (1),maximum number of nodes(depth) (2),minimum ΔI (3),minimum number of events

BDT-Adaptive(BDTA)

What is boosting?

> Boosting is a way of enhancing the classification and regression performance (and increasing the stability with respect to statistical fluctuations in the training sample) of typically weak MVA methods by sequentially applying an MVA algorithm to reweighted (*boosted*) versions of the training data and then taking a weighted majority vote of the sequence of MVA algorithms thus produced. It has been introduced to classification techniques in the early '90s [25] and in many cases this simple strategy results in dramatic performance increases.

AdaBoost

What is AdaBoost?

- (1)change the weight of mis-identified event and do normalization

$$\alpha = \frac{1 - \epsilon_{err}}{\epsilon_{err}}$$
$$w* = w \times \alpha^{\beta}, \ \textit{beta is learning rate.Default } \beta = 1 \tag{1}$$

Renormalize $w*$ *to make sum of* $w*$ *is consistent with sum of* w

- (2)re-do the construction of decition tree
- (3)repeat (1) and (2) until the number of trees is same as the pre-defined number
- (4)the average of the output of all the decision trees is the output of BDT

BDT-Gradient(BDTG)

What is gradient?It is a gradient descent method. We try to use the method of induction to explain this.
Define a loss function

$$L(F,x) = \sum_{i=1}^{N_{events}} (F(x_i) - y_i)^2 \tag{2}$$

F(x) is the predicted value from and y is the real value, Considering the first decision tree :

$$F(x) = \beta_0 F_0(x) \tag{3}$$

$F_0(x)$ is the direct output value from the firt decision tree. where

$$\beta_0 = \arg\min_{\beta_0} \sum_{i=1}^{N_{events}} [\beta_0 F_0(x_i) - y]^2 \tag{4}$$

Note : decision tree is built to achieve best S/B seperation and β_0 is to minimize the loss function.

Gradient

Suppose we finish building No. K tree, then

$$F(x) = \sum_{k=1}^{K} \beta_k F_k(x) \tag{5}$$

To build No. (K+1) tree, let $y - \sum \beta_k F_k(x)$ be the target of the tree and $F_{K+1}(x)$ is obtained as described previously.

$$\beta_{K+1} = \arg\min_{\beta_{K+1}} [\beta_{K+1} F_{K+1}(x) + \Sigma_{k=1}^{K} \beta_k F_k(x) - y]^2 \tag{6}$$

then $F(x) = \sum_{k=1}^{K+1} \beta_k F_k(x)$
By the method described above, a set of $(\beta_m, F_m)_{m=1}^{N_{trees}}$ is defined.
Note : the principle is to minimize the residual of each event by decision trees.

BDT-Bagging(BDTB)

What is bagging?

- The term Bagging denotes a resampling technique
- where a classfier is repeatedly trained using resampled training events
- such that the combined decision tree represents an average of the individual decision tree.
- A priori, bagging does not aim at enhancing a weak classi-fier in the way adaptive or gradient boosting does, and is thus not a boosting algorithm in a strict sense. Instead it effectively smears over statistical representations of the training data and is hence suited to stabilise the response of a decision tree.

BDT-Decorrelation(BDTD)

Just de-correlate the input variables

XGBoost BDT(original paper)

It is developed by Tian-Qi Chen from CS@UWashington.
XGBoost is similar to gradient.
Define the loss function by add a penalty term of the complexity of the model.

$$\begin{aligned}\mathcal{L}_K &= \sum_{i=1}^{N_{events}} l(y_i, \hat{y}_i) + \sum_{k=1}^{K} \Omega(f_k) \\ &= \sum_{i=1}^{N_{events}} l(y_i, y_i^{\hat{K}-1} + f_K(x_i)) + \sum_{k=1}^{K} \Omega(f_k)\end{aligned} \tag{7}$$

where l is the square loss defined before, called training loss here to measures how well model fit on training data. Ω is called regularization term to measure the complexity of the model.

XGBoost - regularization term

Define the regularizaton term

$$\Omega(f_k) = \gamma\mathcal{T} + \frac{1}{2}\lambda\sum_{j=1}^{\mathcal{T}} w_j \tag{8}$$

Where f_k means the structure of the k-th tree. $\mathcal{T}$ means the number of terminal node. w_j means the output value of the node j. γ and λ is the pre-defined parameter.
In the following pages:

- k is the index of decision tree
- j is the index of node
- i is the index of event

XGBoost(2)

Suppose we already build (K-1) bunch of trees,
and consider the loss function from the K-th tree.
Approximate $\mathcal{L}_K$ using second order Taylor expansion:

$$\mathcal{L}_K \approx \mathcal{L}_{K-1}(\textit{this is constant!}) + \sum g_i f_k(x_i) + \frac{1}{2}\sum h_i f_k(x_i)^2 + \Omega(f_k)$$

$$\frac{\partial \mathcal{L}_K}{\partial w_i} = \sum_{j=1}^{T}[(\sum_{i\in I_j} g_i)w_j + (\sum_{i\in I_j} h_i + \lambda)w_j]$$

$$g_i = \frac{\partial}{\partial \hat{y}_i} l(y_i, \hat{y}_i), h_i = \frac{\partial^2}{\partial^2 \hat{y}_i} l(y_i, \hat{y}_i) \tag{9}$$

To minimize the loss function $\mathcal{L}$, when $\frac{\partial \mathcal{L}}{\partial w_j} = 0$

$$w_j = -\frac{G_j}{H_j + \lambda}$$

$$\textit{where } G = \sum g_i, H = \sum h_i \tag{10}$$

How do we build the decision tree?

- Scan the threshold and calculate the gain in loss function

$$threshold = \arg\max_{a} |\frac{1}{2}[\frac{G_L^2}{H_L+\lambda} + \frac{G_R^2}{H_R+\lambda} - \frac{(G_R+G_L)^2}{H_R+H_L+\lambda}] - \gamma| \quad (11)$$
$$L : x < a, R : x > a$$

- Until the gain is smaller than the pre-defined value.

XGBoost —configuration in TMVA

N_{Trees}	number of the decision trees
MaxDepth	max depth of the decision trees
MinNodeSize	minimum percentage of training events required in a leaf node
N_{cuts}	number of grid points in variable range to find the optimal threshold
BoostType	AdaBoost:Gradient:XGBoost
Shrinkage	Learning rate ν: $F(x) = \nu \sum_k^{N_{trees}} \beta_k F_k(x)$. Usually a low learning rate and high N_{trees} are used.
SeparationType	CrossEntropy:GiniIndex:SDivSqrtSPlusB
NegWeightTreatment	Igore:Pair:Inverse

Notes:

- Shrinkage : kind of step size of regression. It is recommended to use small shrinkage and large number of decision trees.
- NegWeightTreatment : negative weights will destroy the loss function and the boosting of weights.
  - "pair" means pair the negative and positive event to balance
  - Inverse : Boost With inverse boostweight(only work in AdaBoost).$w = w/boostFactor$

Scikit-learn : Preparation

Load dataset : "X" is array of variables and "y" is the target.

```
from sklearn.datasets import load_breast_cancer
X,y = load_breast_cancer(return_X_y=True)
```

Split the dataset into training and test

```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=123)
```

Scikit-learn : Decision Tree

```
from sklearn import tree
clf = tree.DecisionTreeClassifier()
clf.fit(X_train,y_train)
print(y_test)
print(clf.predict(X_test))
```

Scikit-learn : AdaBoost

```
from sklearn.ensemble import AdaBoostRegressor
clf = AdaBoostRegressor(n_estimators=100)
clf.fit(X_train,y_train)
print(y_test)
print(clf.predict(X_test))
```

Scikit-learn : Gradient

```
from sklearn.ensemble import GradientBoostingRegressor
clf = GradientBoostingRegressor(n_estimators=100)
clf.fit(X_train, y_train)
print(y_test)
print(clf.predict(X_test))
```

Scikit-learn : XGBoost

```
import xgboost as xgb
clf = xgb.XGBRegressor()
clf.fit(X_train, y_train)
print(y_test)
print(clf.predict(X_test))
```

This is not enough!

- Plot the input and output
- Optimize the Hyper-parameters
- Compare the different methods
- Check the overtraining
- Ranking of variables

Summary

- What is in this lecture?
  - Show you the detail of algorithms : in general, you can use regression tree and minimize the squared-error loss function.
  - Show the very simple examples to setup the tool with scikit-learn
- What is next?
  - Show you more items
    - plotting, optimization of hyper-parameters, comparison, overtraining and ranking before we go to the next topic—neural network!
    - Welcome comments/questions!
    - I will upload the slides/codes/documentation to github!

backup

Reference

- 实验数据多元统计分析 朱永生
- 统计学习方法 李航
- TMVA User Guide
- Scikit-learn tutorial
- Original paper of XGBoost
- github of XGBoost
- presentation from Tianqi Chen