Essentials of XGBoost Machine Learning Algorithm

Yunfan Li PhD

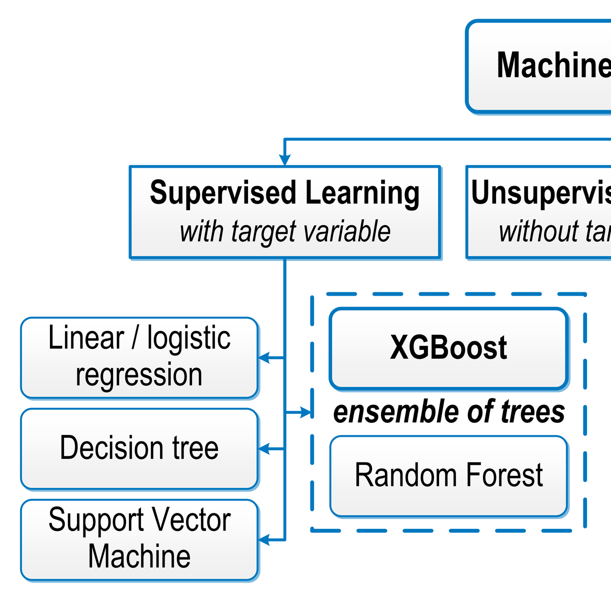
Tsinghua University, University of California at Riverside

About This Poster

This poster is about the XGBoost, a supervised machine learning algorithm, that builds a sequence of decision tress to predict continuous and target variables. Background, model, training steps and mathematics are presented.

Background and a demo model are presented firstly. Model building, including flowchart, underlying mathematics and a demo training is illustrated. Missing value processing, feature importance and advantage summary are also presented.

Background



Ensemble method A machine learning technique that combines multiple base models to obtain better performance.

XGBoost and Random Forest Two tree-based ensemble learning algorithms that builds trees sequentially and in parallel. Final outputs of both are weighted sums of individual tree outputs.

XGBoost stands for e**X**treme **G**radient **B**oosting. Its core algorithm is gradient boosting that builds a sequence of decision trees based on Gradients of cost function. The sum of scaled outputs of the trees is the final output.

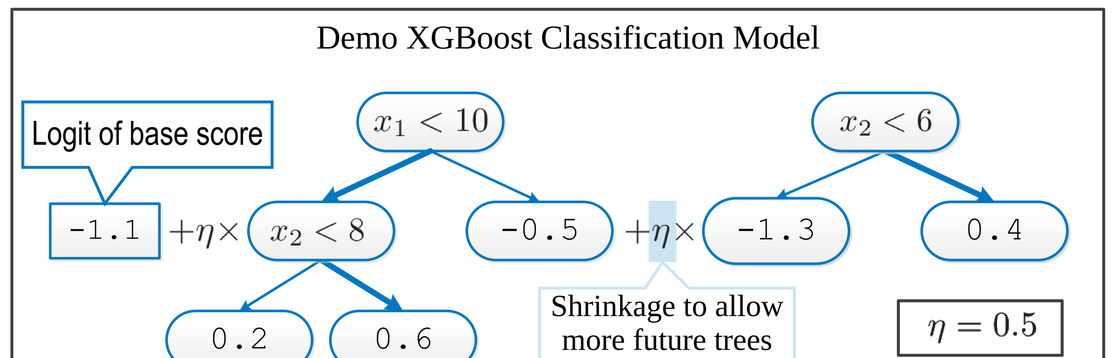
XGBoost Classification and Regression

XGBoost classification for binary target and regression for continuous target are under the same framework that sequentially builds decision trees and sum of outputs of the trees is the final output.

Classification model outputs the log odds of binary target $\ln\dfrac{\Pr[y=1]}{\Pr[y=0]}$.

Regression model outputs the expectation of continuous target $\hat{y}$.

Demo XGBoost Classification Model



Log odds of $(x_1, x_2) = (8.5, 11)$

$$\hat{\omega} = -1.1 + 0.5 \times 0.6 + 0.5 \times 0.4$$

Probability

$$\text{Prob}[Y=1] = \frac{1}{1+e^{-\hat{\omega}}}$$

Model Building Steps

Initialization Set a initial score to all points.

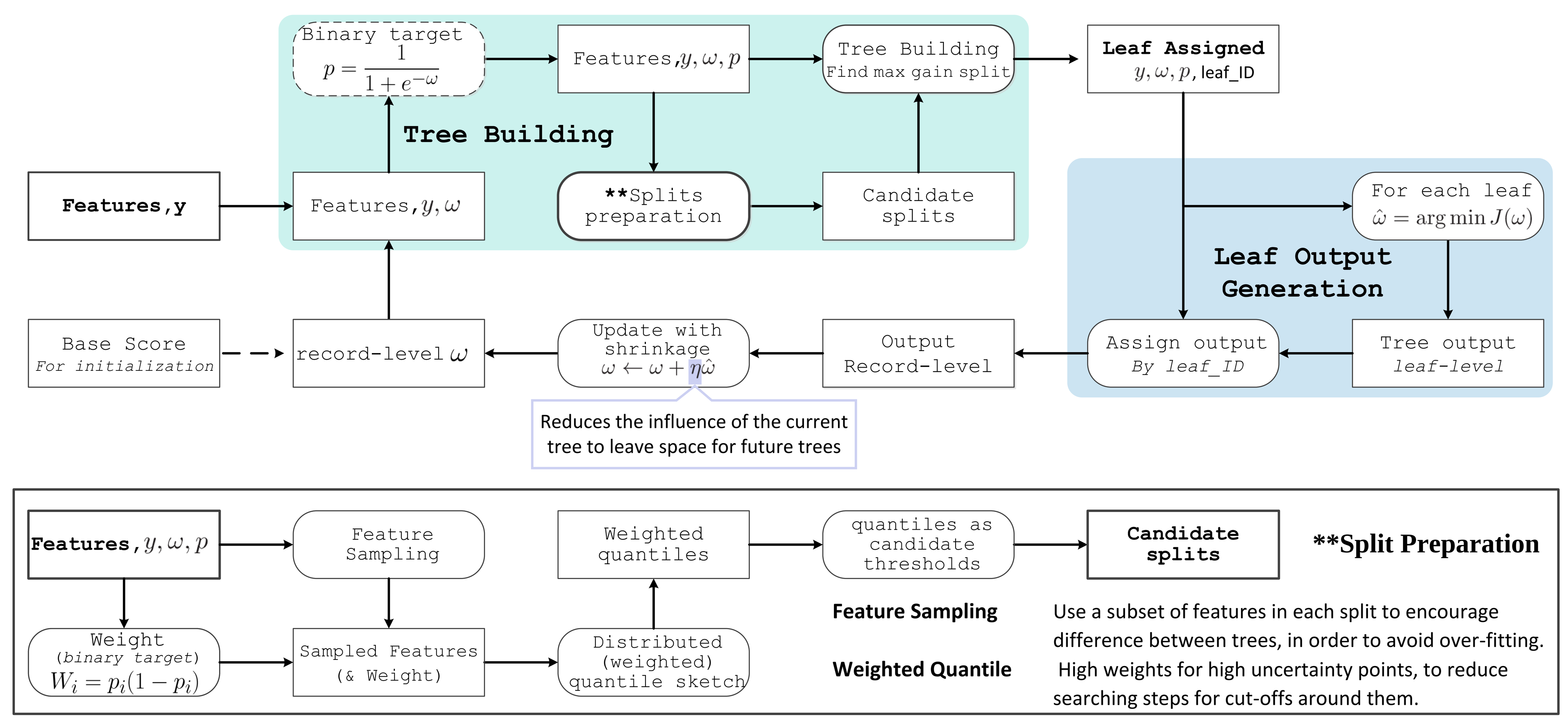
Regression $\bar{y}$

Classification $\ln\dfrac{\bar{y}}{1-\bar{y}}$

Iteration Generate tree output based on previous score.

Step 1: build a tree based on previous score, target and features.
Step 2: generate leaf outputs based on previous score and target.

Model Building Flowchart



Mathematics of XGBoost Classification and Regression

Essential Tasks

- Determining the leaf outputs of a given tree.
- Defining the gain of a split to find the best split of each node to build the tree.

Classification for Binary Target

Cost Function

$$J_i(\omega) \stackrel{\text{def}}{=} -\ln L(p_i; y_i) = -y_i(\omega_i + \omega) + \ln(1 + e^{\omega_i + \omega})$$

$$J(\omega) = \sum_{i \in \text{leaf}} J_i(\omega) + \frac{1}{2}\lambda\omega^2$$

Regularization to avoid overfitting

Likelihood

$$L(p; y) = p^y(1-p)^{1-y}$$

Probability by log odds

$$p_i = \frac{e^{\omega_i+\omega}}{1+e^{\omega_i+\omega}}$$

Cost Function Gradients

$$J'(0) = \sum_{i \in \text{leaf}} g_i \qquad g_i = J_i'(0) = p_i - y_i$$

$$J''(0) = \sum_{i \in \text{leaf}} h_i + \lambda \qquad h_i = J_i''(0) = p_i(1-p_i)$$

Tree Building and Leaf Output

Leaf output

$$\hat{\omega} = -\frac{\Sigma_i g_i}{\Sigma_i h_i + \lambda}$$

Split gain (to identify the best split)

$$\mathcal{L} = \frac{1}{2}\left[\frac{(\sum_{i\in \text{L}} g_i)^2}{\sum_{i\in \text{L}} h_i + \lambda} + \frac{(\sum_{i\in \text{R}} g_i)^2}{\sum_{i\in \text{R}} h_i + \lambda} - \frac{(\sum_i g_i)^2}{\sum_i h_i + \lambda}\right]$$

Newton's Method

$$J(\omega) \approx J(0) + J'(0)\omega + \frac{1}{2}J''(0)\omega^2$$

$$\arg\min J(\omega) = -\frac{J'(0)}{J''(0)}$$

$$\min J(\omega) = J(0) - \frac{J'(0)^2}{2J''(0)}$$

Leaf Output $\hat{\omega} = -\dfrac{J'(0)}{J''(0)}$

Split Gain

$$\begin{aligned}\mathcal{L} &= \min J - (\min J_L + \min J_R) \\ &= \frac{1}{2}\left[\frac{J_L'(0)^2}{J_L''(0)} + \frac{J_R'(0)^2}{J_R''(0)} - \frac{J'(0)^2}{J''(0)}\right]\end{aligned}$$

Regression for Continuous Target

Cost Function

$$J(\omega) = \frac{1}{2}\sum_{i \in \text{leaf}} (y_i - \omega_i - \omega)^2 + \frac{1}{2}\lambda\omega^2$$

Regularization to avoid overfitting

Cost Function Gradients

$$J'(0) = \sum_{i \in \text{leaf}} \omega_i - y_i \qquad J''(0) = N_{\text{leaf}} + \lambda$$

Tree Building and Leaf Output

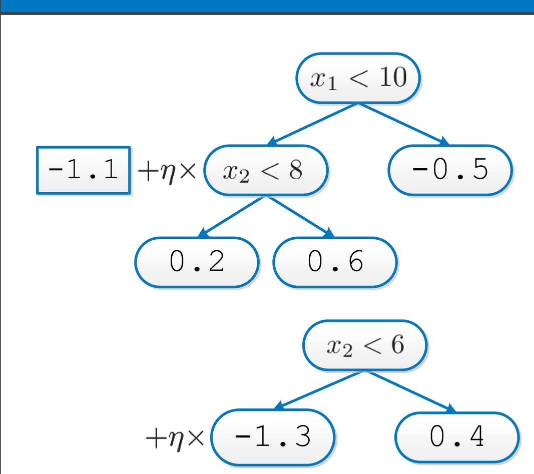
Leaf output

$$\hat{\omega} = \frac{\sum_i y_i - p_i}{N_{\text{leaf}} + \lambda}$$

Split gain (to identify the best split)

$$\mathcal{L} = \frac{1}{2}\left[\frac{(\sum_{i\in \text{L}} y_i - \omega_i)^2}{N_{\text{L}} + \lambda} + \frac{(\sum_{i\in \text{R}} y_i - \omega_i)^2}{N_{\text{R}} + \lambda} - \frac{(\sum_i y_i - \omega_i)^2}{N_{\text{L}} + N_{\text{R}} + \lambda}\right]$$

Three Types of Feature Importance



Denote the probability of th record with base score, one and two trees as $p_{0,i}, p_{1_i}, p_{2,i}$.

Type Weight # of splits with the feature $x_1: 1, x_2: 2$.

Type Cover

Cover of a split Total $p(1-p)$ of the node samples.

First Tree:

$$x_1 < 10: \quad \sum p_0(1-p_0)$$

$$x_2 < 8: \quad \sum_{x_{1,i}<10} p_{1,i}(1-p_{1,i})$$

Second Tree $\sum p_{1,i}(1-p_{1,i})$

Cover of feature Average cover of all its splits.

$$x_1: \quad \sum p_0(1-p_0)$$

$$x_2: \quad \frac{1}{2}\left[\sum_{x_{1,i}<10} p_{1,i}(1-p_{1,i}) + \sum p_{1,i}(1-p_{1,i})\right]$$

Type Gain

Cost function of a node $J(Node) = -\dfrac{1}{2}\dfrac{(\sum_{i\in Node} y_i - p_i)^2}{\sum_{i \in Node} p_i(1-p_i) + \lambda}$

Gain of splitting the node into L and R

$$\text{Gain} = J(Node) - J(L) - J(R)$$

First Tree

$$x_1: \quad \frac{1}{2}\left[\frac{\sum_{x_{1,i}<10}(y_i - p_{0,i})^2}{\sum_{x_{1,i}<10} p_{0,i}(1-p_{0,i}) + \lambda} + \frac{\sum_{x_{1,i}\geq 10}(y_i - p_{0,i})^2}{\sum_{x_{1,i}\geq 10} p_{0,i}(1-p_{0,i}) + \lambda} - \frac{\sum (y_i - p_{0,i})^2}{\sum p_{0,i}(1-p_{0,i}) + \lambda}\right]$$

$$x_2: \quad p'_{0,i} = \text{expit}\left(-1.1 + \eta\frac{\sum_{x_{1,i}<10} y_i - p_{0,i}}{\sum_{x_{1,i}<10} p_{0,i}(1-p_{0,i}) + \lambda}\right)$$

$$\frac{1}{2}\left[\frac{\sum_{x_{2,i}<8}(y_i - p'_{0,i})^2}{\sum_{x_{2,i}<8} p'_{0,i}(1-p'_{0,i}) + \lambda} + \frac{\sum_{x_{2,i}\geq 8}(y_i - p'_{0,i})^2}{\sum_{x_{2,i}\geq 8} p'_{0,i}(1-p'_{0,i}) + \lambda} - \frac{\sum (y_i - p'_{0,i})^2}{\sum p'_{0,i}(1-p'_{0,i}) + \lambda}\right]$$

Cover of feature Average gain of all splits with the feature.

XGBoost in Python — model building

```
df=pd.DataFrame({'y'  :[0,0,0,0,0,1,1,1,0,1],
                'x1':['H','H','H','H','H','H','L','L',np.nan,'H'],
                'x2':[2.8,3,2,2.6,3,3,2.7,4.5,1.3,4.2],
                'x3':[np.nan,2,4.2,2.6,3,2.7,6,2,2,4] })
X=df[['x1','x2','x3']].replace(r'^\s+$',np.nan,regex=True)
cat_cols,num_cols,y=['x1'],['x3', 'x4'],df['y']
```

	y	x2	x3	x4
0	0	H	2.8	NaN
1	0	H	3.0	2.0
2	0	H	2.0	4.2
3	0	H	2.6	2.6
4	0	H	3.0	3.0
5	1	H	3.0	2.7
6	1	L	2.7	6.0
7	1	L	4.5	2.0
8	0		1.3	2.0
9	1	H	4.2	4.0

Feature preprocessor

- Preprocessor: impute missings, one-hot encode.

```
from sklearn.impute import SimpleImputer
from sklearn.preprocessing import OneHotEncoder
from sklearn.compose import ColumnTransformer
cat_transformer=Pipeline(steps=
            [("imputer",SimpleImputer(strategy='constant', fill_value='NA')),
             ("enc",OneHotEncoder(sparse=False, handle_unknown="ignore"))])
preprocessor = ColumnTransformer([("numerical", "passthrough", num_cols),
                                  ("categorical", cat_transformer,cat_cols)])
```

- Demo

```
preprocessor.fit(X)
imputer=preprocessor.named_transformers_["categorical"].named_steps['imputer']
enc=preprocessor.named_transformers_["categorical"].named_steps['enc']
X_imputed=imputer.transform(X[['x1']])
```

```
enc.categories_
```

[array(['H', 'L', 'NA'], dtype=object)]

```
enc.get_feature_names(cat_cols)
```

['x1_H' 'x1_L' 'x1_NA']

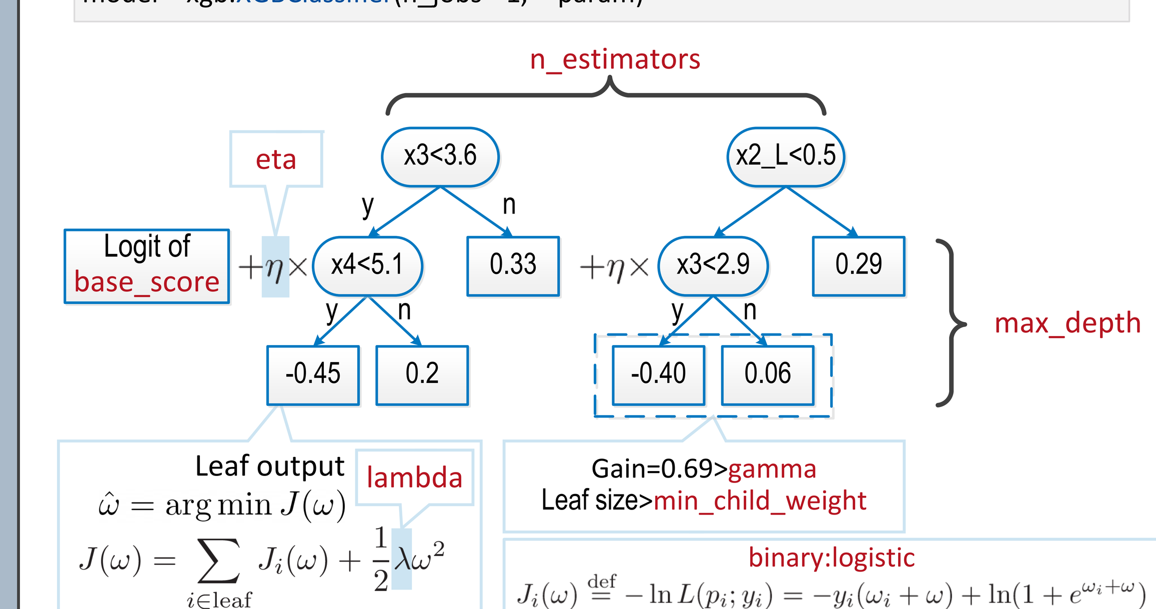
```
enc.inverse_transform(np.array([[1,0,0]]))
```

[['H']]

X_imputed	enc.transform(X_imputed)
[['H']	[[1. 0. 0.]
['H']	[0. 0. 1.]
['H']	[0. 1. 0.]
['H']	[0. 1. 0.]
['H']	[0. 1. 0.]
['H']	[1. 0. 0.]
['L']	[1. 0. 0.]
['L']	[1. 0. 0.]
['NA']	[1. 0. 0.]
['H']]	[0. 1. 0.]]

XGBoost Model

```
param = { 'objective':'binary:logistic','base_score':0.5,'n_estimators':2,'eta':0.5,
         'max_depth':2, 'gamma':0.6,'min_child_weight':0,'lambda':1}
model = xgb.XGBClassifier(n_jobs=-1,**param)
```



Create Pipeline and Fit Model

```
xgb_model = Pipeline([("preprocessor", preprocessor),("model",model)])
xgb_model.fit(X,y)
booster = xgb_model.named_steps["model"].get_booster()
preprocessor = xgb_model.named_steps["preprocessor"]
```

XGBoost in Python — modeling results

Model Definition

```
trees=booster.get_dump()
```

Model Output

```
proba=xgb_model.predict_proba(X)
```

Feature Importance

```
cat_ohe_names = preprocessor._transformers_[1][1]['enc'].get_feature_names(['x1'])
f_map = {"f"+str(i):name for i,name in enumerate(num_cols+list(cat_ohe_names))}
scores = booster.get_score(importance_type='weight')   # or 'total_gain','total_cover'
{f_map[f]:s for (f,s) in scores.items()}
```

{'x2': 4.40581346, 'x3': 2.0, 'x1_L': 2.39651227}