Xgboost – Practical Review

DOMINIC BERTSCHI - MSE

Vorstellung und Motivation

Studium BEP Vertiefung im Bereich Datenanalyse

MSE | MASTER OF SCIENCE IN ENGINEERING

Wieso xgboost?[1]

“As the winner of an increasing amount of Kaggle competitions, XGBoost showed us again to be a great all-round algorithm worth having in your toolbox.” - Dato Winners’ Interview: 1st place, Mad Professors

“When in doubt, use xgboost.” - Avito Winner’s Interview: 1st place, Owen Zhang

Versprechungen[2]: Scalibility, fast learning and handling sparse data

[1]HTTPS://MACHINELEARNINGMASTERY.COM/GENTLE-INTRODUCTION-XGBOOST-APPLIED-MACHINE-LEARNING/[STAND: 14.11.2017]
[2]XGBOOST: A SCALABLE TREE BOOSTING SYSTEM, TIANQI CHEN AND CARLOS GUESTRIN, 2016-06-10, S.5

Boosting

Boosting Verfahren kombinieren viele “weak learner” zu einem einzelnen starken learner

Algorithm 10.1 *AdaBoost.M1.* *(Freund and Schapire 1997)* [2]

1. Initialize the observation weights $w_i = 1/N,\ i = 1, 2, \ldots, N$.
2. For $m = 1$ to M:
   (a) Fit a classifier $G_m(x)$ to the training data using weights w_i.
   (b) Compute
   $$\text{err}_m = \frac{\sum_{i=1}^{N} w_i I(y_i \neq G_m(x_i))}{\sum_{i=1}^{N} w_i}.$$
   (c) Compute $\alpha_m = \log((1 - \text{err}_m)/\text{err}_m)$.
   (d) Set $w_i \leftarrow w_i \cdot \exp[\alpha_m \cdot I(y_i \neq G_m(x_i))],\ i = 1, 2, \ldots, N$.
3. Output $G(x) = \text{sign}\left[\sum_{m=1}^{M} \alpha_m G_m(x)\right]$.

ω_i: Gewichtung des Datensets

Beobachtungen mit falsch klassifizierten Targets werden stärker gewichtet.

α_m: Gewichtung der weak learner

Klassifikatoren mit kleinem Error werden stärker gewichtet.

[2]HASTIE T., TIBSHIRANI R., FRIEDMAN J., (2011), THE ELEMENTS OF STATISTICAL LEARNING, SECOND EDITION, VERLAG: SPRINGER, S.339

Gradient Boosting

→ Generalisiertes Tree Boosting Verfahren[3]

Algorithmus Pseudocode[4]

start with an initial model F
iterate until converge:
calculate negative gradients $-g(x_i) = -\frac{\partial L(y_i, F(x_i))}{\partial F(x_i)}$

1. fit a regression tree h to negative gradients $-g(x_i)$
2. $F := F + \rho h$

Vorteil: Verschiedene Loss Funktionen können verwendet werden.

TABLE 10.2. *Gradients for commonly used loss functions.*[3]

Setting	Loss Function	$-\partial L(y_i, f(x_i))/\partial f(x_i)$
Regression	$\frac{1}{2}[y_i - f(x_i)]^2$	$y_i - f(x_i)$
Regression	$\lvert y_i - f(x_i)\rvert$	$\text{sign}[y_i - f(x_i)]$
Regression	Huber	$y_i - f(x_i)$ for $\lvert y_i - f(x_i)\rvert \le \delta_m$ $\delta_m \text{sign}[y_i - f(x_i)]$ for $\lvert y_i - f(x_i)\rvert > \delta_m$ where $\delta_m = \alpha\text{th-quantile}\{\lvert y_i - f(x_i)\rvert\}$
Classification	Deviance	kth component: $I(y_i = \mathcal{G}_k) - p_k(x_i)$

1. Weak Learners h werden auf «pseudo Residuen» von einem existierenden Model trainiert.

2. Kombinieren des existierenden Models F mit weak learner h

[3]HASTIE T., TIBSHIRANI R., FRIEDMAN J., (2011), THE ELEMENTS OF STATISTICAL LEARNING, SECOND EDITION, VERLAG: SPRINGER, S.352/360
[4]CHENG L., A GENTLE INTRODUCTION TO GRADIENT BOOSTING, URL: HTTP://WWW.CCS.NEU.EDU/HOME/VIP/TEACH/MLCOURSE/4_BOOSTING/SLIDES/GRADIENT_BOOSTING.PDF#PAGE=55, [STAND: 15.11.2017]

Beziehungen: xgboost – gbm

Xgboost ist grundsätzlich Gradient Boosting Verfahren.

Unterschied (Modelsicht) :

Tianqi Chen: Xgboost wird stärker regularisiert um Overfitting zu vermeiden «Regularized gradient boosting[5]»

→ Diverse Tuning Parameter: gamma (complexity control), max_depth, subsample, …

Unterschied (Systemsicht):

Parallel und distributed computing Technologien werden genutzt.

[5]URL: HTTPS://WWW.QUORA.COM/WHAT-IS-THE-DIFFERENCE-BETWEEN-THE-R-GBM-GRADIENT-BOOSTING-MACHINE-AND-XGBOOST-EXTREME-GRADIENT-BOOSTING [STAND: 10.11.2017]

Modelierung - Split finding

Um den besten Split im Node zu finden: **Exact Greedy Algorithm** (standard in tree boosting implementation z.B. scikit-learn oder R-Package “gbm”)

Algorithm 1: Exact Greedy Algorithm for Split Finding [6]

Input: I, instance set of current node
Input: d, feature dimension
$gain \leftarrow 0$
$G \leftarrow \sum_{i \in I} g_i$, $H \leftarrow \sum_{i \in I} h_i$
for $k = 1$ *to* m **do**
  $G_L \leftarrow 0$, $H_L \leftarrow 0$
  for j *in* $sorted(I, by\ \mathbf{x}_{jk})$ **do**
    $G_L \leftarrow G_L + g_j$, $H_L \leftarrow H_L + h_j$
    $G_R \leftarrow G - G_L$, $H_R \leftarrow H - H_L$
    $score \leftarrow \max(score, \frac{G_L^2}{H_L+\lambda} + \frac{G_R^2}{H_R+\lambda} - \frac{G^2}{H+\lambda})$
  end
end
Output: Split with max score

- where gi und hi are first and second order gradient statistics on the loss function.
- sum of gradients over all data points in the j-th instance set.

loss reduction ILR after the split: IL and IR are the instance sets of left and right nodes after the split.

Problem dabei: Hoher Rechenaufwand für Sortierung, vorallem bei stetigen Features.
→ Es werden alle möglichen Splits durchgegangen

[6]XGBOOST: A SCALABLE TREE BOOSTING SYSTEM, TIANQI CHEN AND CARLOS GUESTRIN, 2016-06-10, S.5
SIEHE DAZU AUCH: HASTIE T., TIBSHIRANI R., FRIEDMAN J., (2011), THE ELEMENTS OF STATISTICAL LEARNING, SECOND EDITION, VERLAG: SPRINGER, S.307

Split finding - xgboost

Approximate Algorithmus

Idee: Split Points anhand Perzentilen der

Verteilung eines Features k (S_k) bestimmen

Algorithm 2: Approximate Algorithm for Split Finding [7]

for $k = 1$ *to* m do
  Propose $S_k = \{s_{k1}, s_{k2}, \cdots s_{kl}\}$ by percentiles on feature k.
  Proposal can be done per tree (global), or per split(local).
end
for $k = 1$ *to* m do
  $G_{kv} \leftarrow= \sum_{j \in \{j | s_{k,v} \geq \mathbf{x}_{jk} > s_{k,v-1}\}} g_j$
  $H_{kv} \leftarrow= \sum_{j \in \{j | s_{k,v} \geq \mathbf{x}_{jk} > s_{k,v-1}\}} h_j$
end
Follow same step as in previous section to find max score only among proposed splits.

sum of gradients over all data points in the j-th instance set.

Für xgboost wurde der **weighted quantile sketch**[8] Algorithmus entwickelt, der splitting points in einer Rangfolge ordnet. (solve the ranking problem)

Weitere Anpassungen für Sparsity awared ranking (for handling sparse data)

[7]XGBOOST: A SCALABLE TREE BOOSTING SYSTEM, TIANQI CHEN AND CARLOS GUESTRIN, 2016-06-10, S.3
[8]XGBOOST: A SCALABLE TREE BOOSTING SYSTEM, TIANQI CHEN AND CARLOS GUESTRIN, 2016-06-10, S.4

WEITE MÖGLICHKEIT: HIERARCHICAL MIXTURE MODELL
(VGL. HASTIE T., TIBSHIRANI R., FRIEDMAN J., (2011), THE ELEMENTS OF STATISTICAL LEARNING, SECOND EDITION, VERLAG: SPRINGER, S.329)

Systemdesign

Ranking ist immer noch Rechenintensiv, weitere Anpassungen in der Implementierung:

- Sortierte Features werden in In-memory units (= Block) im Compressed sparse row (CSR) Format abgelegt.[9]
- Auf CPU Zwischenspeicher optimierte Ablage der Blocks . (Cache-aware implementation) [9]

Abruf der gradient statistics kann parallelisiert werden → parallelen split finding Algorithmus

[9]XGBOOST: A SCALABLE TREE BOOSTING SYSTEM, TIANQI CHEN AND CARLOS GUESTRIN, 2016-06-10, S.5/6
[10]XGBOOST: A SCALABLE TREE BOOSTING SYSTEM, TIANQI CHEN AND CARLOS GUESTRIN, PRESENTED BY SCOTT SIEVERT AND ZHENYU ZHANG, 2016-11-15, S.14

Systemdesign

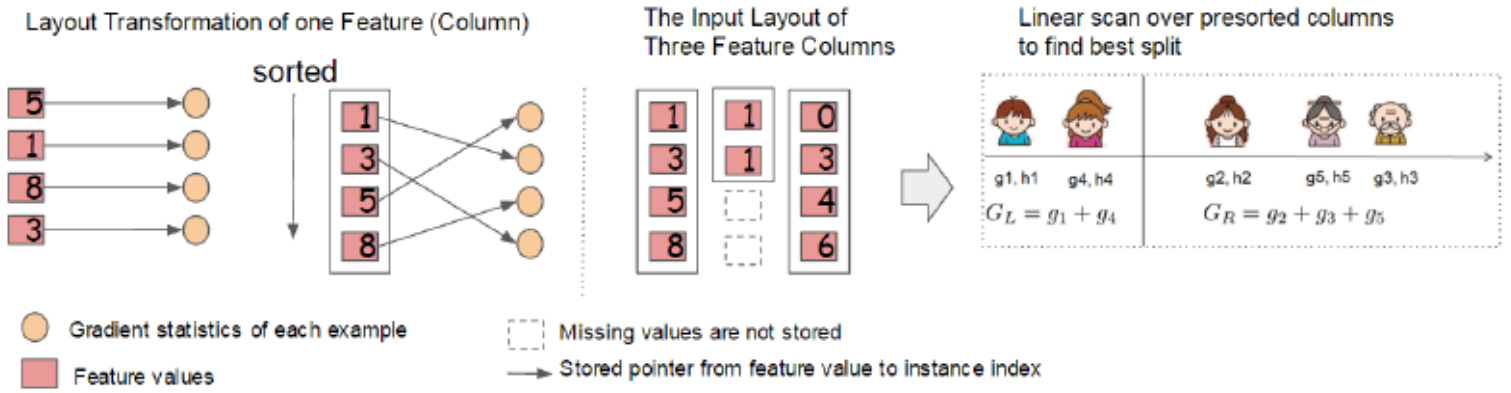

Figure 6: Block structure for parallel learning. Each column in a block is sorted by the corresponding feature value. A linear scan over one column in the block is sufficient to enumerate all the split points.[11]

"Finally, it is even more exciting to combine these techniques to make an end-to-end system that scales to even larger data with the least amount of cluster resources. "

(Chen and Guestrin, 2016, S.2)[12]

[11]XGBOOST: A SCALABLE TREE BOOSTING SYSTEM, TIANQI CHEN AND CARLOS GUESTRIN, 2016-06-10, S.5
[12]XGBOOST: A SCALABLE TREE BOOSTING SYSTEM, TIANQI CHEN AND CARLOS GUESTRIN, 2016-06-10, S.2

Implementierungen und Test

Verfügbar für: C++, Python, R, Java, Scala, Julia

Distributed training on multiple machines z.B. AWS, Azure, Yarn clusters (Hadoop) und mehr

Test mit R-Package «xgboost»

Datenset «winequality-white»[13]; ca. 4900 Records, Klassifikations Aufgabe

→ 11 Input-variablen (z.B. pH, Alkohol, Säuren,…)

→ Zielvariable Quality (Score 1-10)

[13]HTTPS://ARCHIVE.ICS.UCI.EDU/ML/DATASETS/WINE+QUALITY, [STAND: 20.11.2017]
LINK ZUM FILE: HTTPS://ARCHIVE.ICS.UCI.EDU/ML/MACHINE-LEARNING-DATABASES/WINE-QUALITY/WINEQUALITY-WHITE.CSV

Zitat

→ R Code in xgboost.nb.html

Zitat

Anwendung Supervised learning bei grossen Datenmengen mit eigenem Rechner (ohne Server oder Analyse Framework wie z.B. H20)

Anhang

Algorithm 10.3 *Gradient Tree Boosting Algorithm.*

1. Initialize $f_0(x) = \arg\min_\gamma \sum_{i=1}^{N} L(y_i, \gamma)$.
2. For $m = 1$ to M:
   (a) For $i = 1, 2, \ldots, N$ compute
   $$r_{im} = -\left[\frac{\partial L(y_i, f(x_i))}{\partial f(x_i)}\right]_{f=f_{m-1}}.$$
   (b) Fit a regression tree to the targets r_{im} giving terminal regions R_{jm}, $j = 1, 2, \ldots, J_m$.
   (c) For $j = 1, 2, \ldots, J_m$ compute
   $$\gamma_{jm} = \arg\min_\gamma \sum_{x_i \in R_{jm}} L\left(y_i, f_{m-1}(x_i) + \gamma\right).$$
   (d) Update $f_m(x) = f_{m-1}(x) + \sum_{j=1}^{J_m} \gamma_{jm} I(x \in R_{jm})$.
3. Output $\hat{f}(x) = f_M(x)$.

Optimierung Grundsätzlich

Gradient boosting always uses trees that minimize squared error. The trees are fit to the gradient of the loss, in your case the exponential loss. The trees themselves are *not* fit to minimize the exponential loss

According to user feedback, using column sub-sampling (vgl. RF) prevents overfitting even more so than the traditional row sub-sampling (which is also supported). The usage of column sub-samples also speeds up computations of the parallel algorithm described later

https://github.com/dmlc/xgboost/issues/2489

https://xgboost.readthedocs.io/en/latest/model.html

loss functions

Table 10.2 summarizes the gradients for commonly used loss functions. For squared error loss, the negative gradient is just the ordinary residual $-g_{im} = y_i - f_{m-1}(x_i)$, so that (10.37) on its own is equivalent to standard least-squares boosting. With absolute error loss, the negative gradient is the *sign* of the residual, so at each iteration (10.37) fits the tree to the sign of the current residuals by least squares. For Huber M-regression, the negative gradient is a compromise between these two (see the table).

For classification the loss function is the multinomial deviance (10.22), and K least squares trees are constructed at each iteration. Each tree T_{km} is fit to its respective negative gradient vector $\mathbf{g}_{km}$,

$$\begin{aligned} -g_{ikm} &= \frac{\partial L\left(y_i, f_{1m}(x_i), \ldots, f_{1m}(x_i)\right)}{\partial f_{km}(x_i)} \\ &= I(y_i = \mathcal{G}_k) - p_k(x_i), \end{aligned} \tag{10.38}$$

with $p_k(x)$ given by (10.21). Although K separate trees are built at each iteration, they are related through (10.21). For binary classification ($K = 2$), only one tree is needed (exercise 10.10).

Classification loss functions

With K-class classification, the response Y takes values in the unordered set $\mathcal{G} = \{\mathcal{G}_1, \ldots, \mathcal{G}_k\}$ (see Sections 2.4 and 4.4). We now seek a classifier $G(x)$ taking values in $\mathcal{G}$. It is sufficient to know the class conditional probabilities $p_k(x) = \Pr(Y = \mathcal{G}_k|x), k = 1, 2, \ldots, K$, for then the Bayes classifier is

$$G(x) = \mathcal{G}_k \text{ where } k = \arg\max_{\ell} p_\ell(x). \tag{10.20}$$

In principal, though, we need not learn the $p_k(x)$, but simply which one is largest. However, in data mining applications the interest is often more in the class probabilities $p_\ell(x)$, $\ell = 1, \ldots, K$ themselves, rather than in performing a class assignment. As in Section 4.4, the logistic model generalizes naturally to K classes,

$$p_k(x) = \frac{e^{f_k(x)}}{\sum_{l=1}^{K} e^{f_l(x)}}, \tag{10.21}$$

which ensures that $0 \leq p_k(x) \leq 1$ and that they sum to one. Note that here we have K different functions, one per class. There is a redundancy in the functions $f_k(x)$, since adding an arbitrary $h(x)$ to each leaves the model unchanged. Traditionally one of them is set to zero: for example, $f_K(x) = 0$, as in (4.17). Here we prefer to retain the symmetry, and impose the constraint $\sum_{k=1}^{K} f_k(x) = 0$. The binomial deviance extends naturally to the K-class *multinomial deviance* loss function:

$$\begin{aligned} L(y, p(x)) &= -\sum_{k=1}^{K} I(y = \mathcal{G}_k) \log p_k(x) \\ &= -\sum_{k=1}^{K} I(y = \mathcal{G}_k) f_k(x) + \log\left(\sum_{\ell=1}^{K} e^{f_\ell(x)}\right). \end{aligned} \tag{10.22}$$

As in the two-class case, the criterion (10.22) penalizes incorrect predictions only linearly in their degree of incorrectness.

Zhu et al. (2005) generalize the exponential loss for K-class problems. See Exercise 10.5 for details.

Vgl. cross-entropie $H(X; P; Q) = -\sum_{x \in \Omega} P(X = x) \cdot \log Q(X = x)$.

Sei X eine Zufallsvariable mit Zielmenge Ω, die gemäß P verteilt ist. Es sei weiter Q eine Verteilung auf demselben Ereignisraum.

Dann ist die *Kreuzentropie* definiert durch:

$$H(X; P; Q) = H(X) + D(P\|Q)$$

First and second order gradients

On the t-th iteration, we want to find the tree that minimizes the losses and has a low complexity.

Formally, at iteration t we want to minimize

$$\mathcal{L}^{(t)} = \sum_i \ell(y_i, \hat{y}_i^{t-1} + f_t(\mathbf{x}_i)) + \Omega(f_t)$$

This is a greedy approach; it will converge to a local minima.

When we define

- $g_i = \partial_{\hat{y}_i^{t-1}} \ell(y_i, \hat{y}^{t-1}) \in \mathbb{R}$
- $h_i = \partial^2_{\hat{y}_i^{t-1}} \ell(y_i, \hat{y}^{t-1}) \in \mathbb{R}$

we can use Taylor's thm to expand $\mathcal{L}$ as

$$\begin{aligned}\mathcal{L}^{(t)} &= \sum_i \ell(y_i, \hat{y}_i^{t-1} + f_t(\mathbf{x}_i)) + \Omega(f_t) \\ &\approx \left(\sum_i \ell(y_i, \hat{y}_i^{t-1}) + g_i f_t(\mathbf{x}_i) + \frac{1}{2} h_i f_t^2(\mathbf{x}_i) \right) + \gamma T + \frac{\lambda}{2} \|w\|^2\end{aligned}$$

If we remove the constant terms $\ell(y_i, \hat{y}_i)$, we can say that

$$\mathcal{L}^{(t)} = \left(\sum_i g_i f_t(\mathbf{x}_i) + \frac{1}{2} h_i f_t^2(\mathbf{x}_i) \right) + \gamma T + \frac{\lambda}{2} \|w\|^2$$

Sparsity Aware

Algorithm 3: Sparsity-aware Split Finding

Input: I, instance set of current node
Input: $I_k = \{i \in I | x_{ik} \neq \text{missing}\}$
Input: d, feature dimension
Also applies to the approximate setting, only collect statistics of non-missing entries into buckets
$gain \leftarrow 0$
$G \leftarrow \sum_{i \in I}, g_i, H \leftarrow \sum_{i \in I} h_i$
for $k = 1$ *to* m **do**
  // enumerate missing value goto right
  $G_L \leftarrow 0,\ H_L \leftarrow 0$
  for j *in sorted*$(I_k,$ *ascent order by* $\mathbf{x}_{jk})$ **do**
    $G_L \leftarrow G_L + g_j,\ H_L \leftarrow H_L + h_j$
    $G_R \leftarrow G - G_L,\ H_R \leftarrow H - H_L$
    $score \leftarrow \max(score, \frac{G_L^2}{H_L+\lambda} + \frac{G_R^2}{H_R+\lambda} - \frac{G^2}{H+\lambda})$
  end
  // enumerate missing value goto left
  $G_R \leftarrow 0,\ H_R \leftarrow 0$
  for j *in sorted*$(I_k,$ *descent order by* $\mathbf{x}_{jk})$ **do**
    $G_R \leftarrow G_R + g_j,\ H_R \leftarrow H_R + h_j$
    $G_L \leftarrow G - G_R,\ H_L \leftarrow H - H_R$
    $score \leftarrow \max(score, \frac{G_L^2}{H_L+\lambda} + \frac{G_R^2}{H_R+\lambda} - \frac{G^2}{H+\lambda})$
  end
end
Output: Split and default directions with max gain

Anhang - R-Code

```
#xgboost test with  Wine Testdataset ================================================================
rm(list = ls())
library(xgboost)
library(readr)
wineDat <- read_delim("https://archive.ics.uci.edu/ml/machine-learning-databases/wine-quality/winequality-white.csv",
                      ";", escape_double = FALSE, trim_ws = TRUE)

#wineDat[] <- lapply(wineDat, as.numeric)  # convert to numeric

# one-hot-encoding categorical features!
#80% For Training:
set.seed(1135)
idx <- sample.int(.8*nrow(wineDat), replace = FALSE)

#stores a matrix in compressed sparse row (CSR) format.
wineDat_Train <- xgb.DMatrix(data = data.matrix(wineDat[idx,1:11]),
                             label = data.matrix(wineDat[idx,12] ))

wineDat_Test <- xgb.DMatrix(data = data.matrix(wineDat[-idx,1:11]),
                            label = data.matrix(wineDat[-idx,12] ))

## A simple xgb model example: (Params from help)
t1 <- system.time(bst <- xgboost(data = wineDat_Train, num_class = 10,
                                 max_depth = 2, nrounds= 2,
                                 objective = "multi:softmax"))


pred <- predict(bst, wineDat_Test)
(xtabXG <- table(pred, data.matrix(wineDat[-idx,12] )))
(err1 <- mean(pred != data.matrix(wineDat[-idx,12])))
```

	xgboost	randomForest	gbm
elapsedTime s	0.070	3.820	1.66
OOS Error	0.402	0.453	0.46