Gradient Boosting

Ensemble method for supervised learning

Using an explicit loss function

Ricco RAKOTOMALALA

Université Lumière Lyon 2

Outline

1. Preamble
2. Gradient boosting for regression
3. Gradient boosting for classification
4. Regularization (shrinkage, stochastic gradient boosting)
5. Tools and software
6. Conclusion – Pros and cons
7. References

Preamble

Boosting and Gradient Descent

Boosting

BOOSTING is an ensemble method which aggregates classifiers learned sequentially on a sample for which the weights of individuals are adjusted at each step. The classifiers are weighted according to their performance. [RAK, page 28].

Input: B number of models, ALGO learning algorithm, Ω training set, with size = n, y target attribute, X matrix with p predictive attributes.

MODELES = { }
All the instances have the same weight $\omega^1_i = 1/n$
For b = 1 to B Do
  Fit the model M_b from $\Omega(\omega^b)$ using ALGO (ω^b weighting system at the step b)
  Add M_b into MODELES
  Calculate the weighted error rate for M_b : $\varepsilon_b = \sum_{i=1}^{n} \omega_i^b \times I(y_i \neq \hat{y}_i)$
  If $\varepsilon_b > 0.5$ or $\varepsilon_b = 0$, STOP the process
  Else
    Calculate $\alpha_b = \ln \frac{1-\varepsilon_b}{\varepsilon_b}$
    The weights are updated $\omega_i^{b+1} = \omega_i^b \times \exp[\alpha_b . I(y_i \neq \hat{y}_i)]$
    And normalized so that the sum is equal to 1
End For

A weighted (α_b) vote is used for prediction (this is an additive model)

$$f(x) = sign \sum_{b=1}^{B} \alpha_b \times M_b(x)$$

Gradient descent

Gradient descent is an iterative technique that allows to approach the solution of an optimization problem. In supervised learning, the construction of the model is often to determine the parameters that enable to optimize (max or min) an objective function (ex. Perceptron – Least squares criterion, pages 11 et 12).

$$J(y, f) = \sum_{i=1}^{n} j(y_i, f(x_i))$$

f() is a classifier with some parameters

j() is a cost function comparing the observed value of the target and the prediction of the model for an observation

J() is an overall loss function, additively calculated from all observations

➔ The aim is to minimize **J()** with regard to **f()** i.e. the parameters of f().

$$f_b(x_i) = f_{b-1}(x_i) - \eta \times \nabla j(y_i, f(x_i))$$

$f_b()$ is the version of classifier at step ''b''

η is the learning rate which enables to lead the process

∇ is the gradient i.e. the first order partial derivative of the cost function with regard to the classifier

$$\nabla j(y_i, f(x_i)) = \frac{\partial j(y_i, f(x_i))}{\partial f(x_i)}$$

Boosting = Gradient descent

We can show that ADABOOST consists in to optimize an exponential loss function i.e. each classifier M_t learned from the weighted sample resulting from M_{t-1} allows to minimize an overall loss function [BIS, page 659 ; HAS, page 343]

$$J(f) = \sum_{i=1}^{n} \exp\left(-y_i \times f(x_i)\right)$$

$y \in \{-1, +1\}$

J() is the overall loss function

f() is the aggregate classifier composed of a linear combination of the base classifiers M_b

$$f_b = f_{b-1} + \frac{\alpha_b}{2} \times M_b$$

The aggregate classifier at step ''b'' is corrected with the individual classifier M_b learned from the reweighted sample. M_b is the gradient here i.e. each intermediate model allows to reduce the loss of the global model.

$$\omega_i^b = \omega_i^{b-1} \times \exp\left[\alpha_{b-1}.I\left(y_i \neq M_{b-1}(i)\right)\right]$$

The "gradient" classifier comes from a sample where the weights of individuals depend on the performance of the previous model (idea of iterative corrections)

GRADIENT BOOSTING : generalize the approach with other loss functions

Gradient Boosting for Regression

Gradient Boosting = Gradient Descent + Boosting

Regression

The regression is a supervised learning process which estimates the relationship between a quantitative dependent variable and a set of independent variables.

$$y_i = M_1(x_i) + \varepsilon_{1i}$$

ε is the error term. It represents the inadequacy of the model. M is any kind of model, we use regression tree.

$$e_{i1} = y_i - M_1(x_i)$$

e is the residual. Estimated value of the error. High value (in absolute value) reflects a bad prediction.

The aim is to model this residual with a second classifier M2 and associate it with the previous one for a better prediction.

$$e_{i1} = M_2(x_i) + \varepsilon_{2i}$$

We can proceed in the same way for the residual e_2, etc.

$$\hat{y}_i = M_1(x_i) + M_2(x_i)$$

The role of M_2 is (additively) compensate the inadequacy of M_1, thereafter we can learn M_3, etc.

Overall loss function

Connection with gradient descent

The sum of the squares of errors is a well-known overall indicator of quality in regression

$$j(y_i, f(x_i)) = \frac{1}{2}(y_i - f(x_i))^2$$

$$J(y, f) = \sum_{i=1}^{n} j(y_i, f(x_i))$$

$$\frac{\partial j(y_i, f(x_i))}{\partial f(x_i)} = \frac{\partial\left[\frac{1}{2}(y_i - f(x_i))^2\right]}{\partial f(x_i)} = f(x_i) - y_i$$

Calculation of the gradient. It is actually equal to the residual, but with an opposite sign i.e. residual = negative gradient

$$\begin{aligned} f_b(x_i) &= f_{b-1}(x_i) + M_b(x_i) \\ &= f_{b-1}(x_i) + (y_i - f_{b-1}(x_i)) \\ &= f_{b-1}(x_i) - 1 \times \frac{\partial j(y_i, f(x_i))}{\partial f(x_i)} \\ &= f_{b-1}(x_i) - \eta \times \nabla j(y_i, f(x_i)) \end{aligned}$$

Thus, we have an iterative process for the construction of the additive model. Modeling the residuals in step "b" (regression tree M_b) corresponds to a gradient. Ultimately, we minimize the overall cost function J()

The learning rate η is equal to 1 here.

Gradient Boosting Algorithm for Regression

We have an iterative process where, at each step, we use the negative value of the gradient: $-\nabla j(y,f)$ [WIK]

Or, more simply, FOR m = 1, …, B (B : parameter of the algorithm)

The trivial tree corresponds to a tree with only the root. The prediction is equal to the mean of the target attribute Y.

```
Fit the trivial tree f0()
REPEAT UNTIL CONVERGENCE
  Calculate negative gradient -∇j(y,f)
  Fit a regression tree Mb for -∇j(y,f)
  fb = fb-1 + γb.Mb
```

Must be calculated for all the individuals of the training sample (i = 1, …, n)
j() = square of the error ➔ negative gradient = residual

The depth of the trees is a possible parameter

The models are combined in additive fashion

γ_b is chosen at each step in order to minimize (using a numerical optimization approach)

$$\gamma_b = \arg\min_{\gamma} \sum_{i=1}^{n} j(y_i, f_{b-1}(x_i) + \gamma . M_b(x_i))$$

The advantage of this generic formulation is that one can use other loss functions and the associated gradients.

Gradient Boosting

Other loss functions

Other loss functions
➔ Other gradient formulation
➔ Other behavior and performance of the aggregate model

Loss function	$-\nabla j(y_i, f(x_i))$	Pros / Cons
$\frac{1}{2}(y_i - f(x_i))^2$	$y_i - f(x_i)$	Sensitivity to small differences, but not robust against the outliers
$\lvert y_i - f(x_i) \rvert$	$signe[y_i - f(x_i)]$	Less sensitive to small differences but robust against outliers
Huber	$y_i - f(x_i)$ si $\lvert y_i - f(x_i) \rvert \leq \delta_b$ $\delta_b . sign[yi - f(xi)]$ si $\lvert y_i - f(x_i) \rvert > \delta_b$ Where δ_b is a quantile of $\{\lvert y_i - f(x_i) \rvert\}$	Combine the benefits of the square error (more sensitive to small values of the gradient) and absolute error (more robust against the outliers)

Gradient Boosting for Classification

Working with the indicator variables (dummy variables)

Loss function and gradient for classification

The categorical target variable is K values {1,..., K}
The algorithm remains identical but: we must define a loss function adapted to the classification process, and calculate the appropriate gradient.

y^k is a dummy (K dummy variables are generated)

$$y_i^k = \begin{cases} 1 \text{ si } Y_i = k \\ 0 \text{ sinon} \end{cases}$$

$$\pi^k(x_i) = \frac{e^{f^k(x_i)}}{\sum_{k=1}^{K} e^{f^k(x_i)}}$$

π_k corresponds to the class membership probability for “k” (value “k” de Y)

$$j(y_i, f(x_i)) = -\sum_{k=1}^{K} y_i^k \times \log \pi^k(x_i)$$

Loss function : MULTINOMIAL DEVIANCE
(binomial deviance is a special case for binary target attribute)

$$\nabla j(y_i, f(x_i)) = y_i^k - \pi^k(x_i)$$

Gradient

For the class "k", the gradient is the difference between the associated dummy variable and the class membership probability

We must deal with the dummy variables (y^k), and fit a regression tree on the negative gradient 1 tree for each dummy variable). f^k is the aggregate model for the class “k”, needed for the calculation of π^k

Algorithm for classification

The process is not changed compared with the regression. The overall scheme remains the same, except that we use the dummy variables

Y (target) is coded with K dummy variables y^k
Fit K trivial trees $f^k_0()$ for each y^k
REPEAT UNTIL CONVERGENCE
  Calculate K negative gradients $-\nabla j(y^k, f^k)$
  Fit a **regression tree** M^k_b for each $-\nabla j(y^k, f^k)$
  $f^k_b = f^k_{b-1} + \gamma_b . M^k_b$

We obtain K aggregate models f^k. The class membership probability is calculated with the "softmax" function

$$\pi^k(x_i) = \frac{e^{f^k(x_i)}}{\sum_{k=1}^{K} e^{f^k(x_i)}}$$

Even if we are in the classification context, the internal mechanism is based on a regression tree algorithm. GRADIENT TREE BOOSTING.

!

The assignment rule is $\hat{Y}_i = \arg\max_k \pi^k(x_i)$

Regularization

Approaches to prevent overfitting

Other than the limitation of the depth of the trees

Shrinkage

Include an additional parameter (learning rate) in the update rule

$$f_b = f_{b-1} + \nu.\gamma_b.M_b$$

An additional parameter ($0 < \nu \leq 1$) is used in order to "smooth" the update rule. Empirically, we observe that a low value of ν ($\nu < 0.1$) improves the performance, but the converge is slower (number of needed iterations **B** is higher).

Stochastic Gradient Boosting

Random sampling is introduced. At each step, only a fraction β ($0 < \beta \leq 1$) of the learning sample is used for the construction of the trees M_b [HAS, page 365]

$\beta = 1$, we have the standard algorithm. Typically, $0.5 \leq \beta \leq 0.8$ is suited for a moderate sized dataset [WIK]. Advantages:

1. Reduce the computation time.
2. Prevent overfitting by introducing randomness in the learning process (such as Random forest and Bagging)
3. OOB estimation of the error rate (such as Bagging)

Practice of gradient boosting

Software and packages

Python (scikit-learn)

class `sklearn.ensemble.` **`GradientBoostingClassifier`**(*loss='deviance', learning_rate=0.1, n_estimators=100, subsample=1.0, min_samples_split=2, min_samples_leaf=1, min_weight_fraction_leaf=0.0, max_depth=3, init=None, random_state=None, max_features=None, verbose=0, max_leaf_nodes=None, warm_start=False, presort='auto'*) [source]

Gradient Boosting for classification.

GB builds an additive model in a forward stage-wise fashion; it allows for the optimization of arbitrary differentiable loss functions. In each stage `n_classes_` regression trees are fit on the negative gradient of the binomial or multinomial deviance loss function. Binary classification is a special case where only a single regression tree is induced.

There is also a variable sampling mechanism during the node splitting process, as with Random Forest.

```
#importation of the training set
import pandas
dtrain = pandas.read_table("ionosphere-train.txt",sep="\t",header=0,decimal=".")
print(dtrain.shape)
y_app = dtrain.as_matrix()[:,32] # target attribute
X_app = dtrain.as_matrix()[:,0:32] # input attributes
# importation of the GradientBoostingClassifier class
from sklearn.ensemble import GradientBoostingClassifier
gb = GradientBoostingClassifier()
# display the parameters
print(gb)
# fit on the training set
gb.fit(X_app,y_app)
# importation of the test set
dtest = pandas.read_table("ionosphere-test.txt",sep="\t",header=0,decimal=".")
print(dtest.shape)
y_test = dtest.as_matrix()[:,32]
X_test = dtest.as_matrix()[:,0:32]
# prediction on the test set
y_pred = gb.predict(X_test)
# evaluation : test error rate = 0.085
from sklearn import metrics
err = 1.0 - metrics.accuracy_score(y_test,y_pred)
print(err)
```

Python (scikit-learn)

Paramétrage

Scikit-learn proposes a tool for determining by cross-validation the "optimal" parameters of a machine learning algorithm.

```
# grid search tool : http://scikit-learn.org/stable/modules/grid_search.html
from sklearn.grid_search import GridSearchCV
# Combination of the parameters to evaluate. The tool performs an exhaustive search
# The calculations are intensive in cross-validation
parametres = {"learning_rate":[0.3,0.2,0.1,0.05,0.01],"max_depth":[2,3,4,5,6],"subsample":[1.0,0.8,0.5]}

# The supervised learning algorithm to use: Gradient boosting
gbc = GradientBoostingClassifier()

# Create the objet for searching
grille = GridSearchCV(estimator=gbc,param_grid=parametres,scoring="accuracy")
# Perform the process on the training set
resultats = grille.fit(X_app,y_app)
# best combination of parameters : {'subsample': 0.5, 'learning_rate': 0.2, 'max_depth': 4}
print(resultats.best_params_)
# prediction with the ‘’model’’ identified by cross-validation
ypredc = resultats.predict(X_test)

# performances of the ‘’best’’ model: test error rate = 0.065
err_best = 1.0 - metrics.accuracy_score(y_test,ypredc)
print(err_best)
```

R

(package "gbm")

gbm: Generalized Boosted Regression Models

An implementation of extensions to Freund and Schapire's AdaBoost algorithm and Friedman's gradient boosting machine. Includes regression methods for least squares, absolute loss, t-distribution loss, quantile regression, logistic, multinomial logistic, Poisson, Cox proportional hazards partial likelihood, AdaBoost exponential loss, Huberized hinge loss, and Learning to Rank measures (LambdaMart).

```
# import de data files (train and test)
dtrain <- read.table("ionosphere-train.txt",header=T,sep="\t")
dtest <- read.table("ionosphere-test.txt",header=T,sep="\t")

# package "gbm"
library(gbm)

# fit the model on the training set
gb1 <- gbm(class ~ ., data = dtrain, distribution="multinomial")

# prediction: predict provides a score
# the threshold for class assignment is 0
p1 <- predict(gb1,newdata=dtest,n.trees=gb1$n.trees)
y1 <- factor(ifelse(p1[,1,1] > 0, "b", "g"))

# confusion matrix and error rate
m1 <- table(dtest$class,y1)
err1 <- 1 - sum(diag(m1))/sum(m1)
print(err1)
```

Distribution=« bernoulli » is also possible, but the target attribute must be coded 0/1 in this case.

R

(package "mboost")

mboost: Model-Based Boosting

Functional gradient descent algorithm (boosting) for optimizing general risk functions utilizing component-wise (penalised) least squares estimates or regression trees as base-learners for fitting generalized linear, additive and interaction models to potentially high-dimensional data.

Many proposed functionalities.

```
# package "mboost"
library(mboost)
# fit with the default settings (see documentation online)
gb2 <- blackboost(class ~ ., data = dtrain, family=Multinomial())
# prediction on the test set
y2 <- predict(gb2,newdata=dtest,type="class")
# confusion matrix and test error rate = 11.5%
m2 <- table(dtest$class,y2)
err2 <- 1 - sum(diag(m2))/sum(m2)
print(err2)

# Modifying the settings of the underlying base classifier (deeper regression tree)
library(party)
parametres <- ctree_control(stump=FALSE,maxdepth=10,minsplit=2,minbucket=1)
# fit with the settings
gb3 <- blackboost(class ~ ., data = dtrain, family=Multinomial(),tree_controls=parametres)
# prediction on the test set
y3 <- predict(gb3,newdata=dtest,type="class")
# test error rate = 12.5% (clearly, deeper tree is not suitable here)
m3 <- table(dtest$class,y3)
err3 <- 1 - sum(diag(m3))/sum(m3)
print(err3)
```

Details

This function implements the 'classical' gradient boosting utilizing regression trees as base-learners. Essentially, the same algorithm is implemented in package gbm. The main difference is that arbitrary loss functions to be optimized can be specified via the `family` argument to `blackboost` whereas gbm uses hard-coded loss functions. Moreover, the base-learners (conditional inference trees, see `ctree`) are a little bit more flexible.

R

(package "xgboost")

"xgboost" it proposes a parallel implementation, making the calculation feasible on large datasets (and also other base classifiers than tree)

```
# package "xgboost"
library(xgboost)
# convert the data in a format tractable by xgboost
XTrain <- as.matrix(dtrain[,1:32])
yTrain <- ifelse(dtrain$class=="b",1,0) #codage 1/0 de la cible
# fit with the default settings (eta=0.3, max.depth=6)
gb4 <- xgboost(data=XTrain,label=yTrain,objective="binary:logistic",nrounds=100)
# prediction on the test set
XTest <- as.matrix(dtest[,1:32])
p4 <- predict(gb4,newdata=XTest)
# we obtain PI("b") – we convert in class prediction
y4 <- factor(ifelse(p4 > 0.5,"b","g"))
# confusion matrix and test error rate = 9.5%
m4 <- table(dtest$class,y4)
err4 <- 1 - sum(diag(m4))/sum(m4)
print(err4)
# fit with other settings
gb5 <- xgboost(data=XTrain,label=yTrain,objective="binary:logistic",eta=0.5,max.depth=10,nrounds=100)
# prediction
p5 <- predict(gb5,newdata=XTest)
y5 <- factor(ifelse(p5 > 0.5, "b","g"))
# confusion matrix and test error rate = 9%
m5 <- table(dtest$class,y5)
err5 <- 1 - sum(diag(m5))/sum(m5))
print(err5)
```

About the settings

Gradient boosting is based on many parameters that influence heavily their performances. They can interact with each other, making their handling difficult. The challenge is to make the right trade-off between fully exploit the available data and to prevent overfitting.

Characteristics of the underlying trees	Maximum depth trees, number of samples required to split a node, number of samples required in leaves. Small tree: prevent overfitting, but risk of under fitting. Conversely for large tree.
Learning rate ν	Too low, slow convergence; too high, oscillation, overfitting. Good value around 0.1. If we decrease η, we must increase the number of trees for offset.
Number of trees	The risk of overfitting is low if we increase the number of trees. But the calculation time increases obviously.
Sampling of the instances β	Stochastic gradient boosting. $\beta = 1$, the algorithm uses all the instances. $\beta < 1$ reduces the overdependence to the training set and prevents overfitting. Possible value is about 0.5
Sampling of the variables	Mechanism analogous to the Random Forest. Allows to "diversify" trees and therefore reduce the variance. Perhaps jointly handled with characteristics of trees (large trees imply less bias). This parameter is available only in some packages (xgboost, scikit-learn)

Conclusion

Gradient Boosting

The "gradient boosting" is an ensemble method that generalizes the boosting by providing the opportunity of use other loss functions.

The global frameworks are identical: underlying algorithm = tree, construction in sequential way of models, "variable importance" measurement allows to assess the relevance of the predictors, similar problems for set the right values of parameters.

But unlike boosting, even in the classification context, the underlying algorithm is a regression tree.

Tools/software exist, but we really need to go into the details of the documentation to understand what is behind implementations and the handling of parameters.

Gradient boosting (GBM : Gradient Boosting Machine)

Pros and cons

Especially in classification process which is the main subject of this course.

Advantages

- Compared with to the "usual" boosting, GBM makes the emphasis on the difference (y-π) during the construction of the regression trees with the deviance loss function
- Lots of flexibility with the choice of loss functions, adaptable to the characteristics of the studied problems
- GBM has shown its effectiveness in several challenges!

Drawbacks

- Non-explicit model (as for all ensemble methods)
- Many parameters which can interact and influence heavily the behavior of the approach (number of iterations, regularization parameters, etc.)
- Overfitting can occur if values of parameters are not suitable

- Computationally intensive (especially when number of trees is high)
- Memory occupation of the trees

References

References

[BIS] Bishop C.M., « Pattern Recognition and Machine Learning », Springer, 2006.

[HAS] Hastie T., Tibshirani R., Friedman J., « The elements of Statistical Learning - Data Mining, Inference and Prediction », Springer, 2009.

[LI] LI C., « A Gentle Introduction to Gradient Boosting », 2014.

[NAT] Natekin A., Knoll A., « Gradient boosting machines, a tutorial », in *Frontiers in NeuroRobotics*, December 2013.

[RAK] Rakotomalala R., « Bagging – Random Forest – Boosting », 2015.

[WIK] Wikipédia, « Gradient boosting ».