Machine Learning - 1100-ML0ENG (Ćwiczenia informatyczne Z-23/24)

XGBoost Model - Extreme Gradient Boosting

XGBoost

XGBoost

The XGBoost model requires that

1. all variables were numeric,
2. target variable - binary 0 and 1 (in case of german credit)
3. matrix as an argument. In R, you can use **Matrix :: sparse.model.matrix** or **caret :: dummyVars**,or other.

We transform the set gc

```
gc<-read.csv("germancredit.csv", stringsAsFactors = T)
summary(gc)
```

1. into numeric form

```
library(caret)
gc.dv <- dummyVars("~ .",gc[-21], fullRank = F)
gc.d = as.data.frame(predict(gc.dv, newdata=gc[-21]))
gc.d=cbind(gc.d,gc[21])
str(gc.d)
summary(gc.d)
```

```
library(caTools)
set.seed(12345)
split = sample.split(gc.d$credit_risk, SplitRatio = 0.7)
gc.d.Train <- subset(gc.d, split == TRUE)
gc.d.Test <- subset(gc.d, split == FALSE)
```

2. and into matrix form:

```
#install.packages("Matrix")
library(Matrix)
```

```
mat.train <- as.matrix(gc.d.Train[,-62])
m.train <- as(mat.train,"dgCMatrix")
mat.test<- as.matrix(gc.d.Test[,-62])
m.test <- as(mat.test,"dgCMatrix")
```

We use the **xgboost function**

```
#install.packages("xgboost")
```

```
library(xgboost)
gc.xgb <- xgboost(data=m.train,label=gc.d.Train$credit_risk,
 nrounds = 500,objective="binary:logistic",
 eval_metric = "logloss")
```

```
xgb.predict <- predict(gc.xgb, m.test)
```

```
xgb.pred.class = ifelse(xgb.predict > 0.5, 1, 0)
```

```
table(xgb.pred.class,gc.d.Test$credit_risk)
```

```
acc(xgb.pred.class,gc.d.Test$credit_risk)
```

```
xgb.roc = roc(xgb.pred.class,gc.d.Test$credit_risk)
x=plot(xgb.roc)
x
coords(xgb.roc, "best")
```

Some selected parameters

- **nrounds**[default=100] - maximum number of iterations.
- **eta**[default=0.3][range: (0,1)] - controls the learning rate, i.e. the speed at which our model learns patterns in the data. After each round, it reduces the feature weights to reach the best optimum. A lower eta leads to slower computation. It must be supported by increasing the number of iterations. It is usually in the range of 0.01 - 0.3.
- **gamma**[default=0][range: (0,Inf)] - controls regularity (or prevents overfitting). The optimal gamma value depends on the dataset and other parameter values. Higher value, higher regularity. default = 0 means no regularity. gamma brings improvement, with low trees (low max_depth).
- **max_depth**[default=6][range: (0,Inf)] - tree depth. Taller trees - more complex model; greater chance of overfitting. There is no standard value for max_depth. **Large datasets require tall trees.**
- **min_child_weight**[default=1][range:(0,Inf)] - in regression refers to the minimum number of observations required in the descendant node. In classification, if a leaf node has a minimum sum of observation weights (calculated by the second-order partial derivative) less than min_child_weight, tree splitting stops.

- **subsample**[default=1][range: (0,1)] - number of observations supplied to the tree. Typically, its values are in the range (0.5-0.8)
- **colsample_bytree**[default=1][range: (0,1)] the number of variables in the tree. Typically, its values are in the range (0.5,0.9)

Model loss and evaluation functions. In addition to the parameters listed below, a custom loss and evaluation function can be used.

- objective[default=reg:linear]
- reg:linear - for linear regression
- binary:logistic - Logistic regression for binary classification. Returns class probabilities.

eval_metric [no default, depends on the chosen objective] - these metrics are used to assess the accuracy of the model on test data. For regression, the default metric is RMSE. For classification, the default metric is error.

The available error functions are as follows:
mae - mean absolute error (used in regression)
Logloss - logit (used in classification)
AUC - area under the curve (used in classification)
RMSE - root mean square error (used in regression)
error - binary classification error rate

Cross-validation

We claimed that the **xgboost package** does not require extra coding for the crossvalidation analysis. The **xgb.cv function** is useful here, and it works with the same arguments as the **xgboost function** with the cross-validation folds specified by the nfold option. Here, we choose **nfold=10**.

```
gc.xgb.cv <- xgb.cv(data=m.train,label=gc.d.Train$credit_risk,
 nfold=10,nrounds = 100,objective="binary:logistic",
 prediction = TRUE,eval_metric = "logloss",
 early_stopping_rounds = 55)
xgb.cv.predict <- gc.xgb.cv$pred
gc.xgb.cv$best_iteration
gc.xgb.cv$best_ntreelimit
```

```
gc.xgb2 <- xgboost(data=m.train,label=gc.d.Train$credit_risk,
 nrounds = 11,objective="binary:logistic",
 eval_metric = "logloss")
```

```
xgb.predict2 <- predict(gc.xgb2, m.test)
```

```
table(gc.d.Test$credit_risk,c(xgb.predict2>0.5))
acc(gc.d.Test$credit_risk,c(xgb.predict2>0.5))
```

```
roc.function(gc.d.Test$credit_risk,c(xgb.predict2>0.5))
```

Importance of variables

```
importance_matrix <- xgb.importance(colnames(m.train),model = gc.xgb2)
```

```
xgb.plot.importance(importance_matrix, top_n = 10,
 measure = "Gain",
 main="waznosc zmiennych")
```

You can also tune the model using the caret package

```
#tuning z caret
```

```
library(caret)
tunegrid <- expand.grid(nrounds = 100,
 max_depth = c(4,6,10),
 eta = seq(0.1,0.4,len=4),
 gamma = 0,
 colsample_bytree = 1,
 min_child_weight = 1,
 subsample = 1)
```

```
trcontrol <- trainControl(method = "repeatedcv",
 number = 10,
 repeats = 2,
 allowParallel = T)
```

```
xg_train = train(credit_risk~.,
 data= gc,
 trControl = trcontrol,
 tuneGrid = tunegrid,
 method = "xgbTree")
```

```
plot(xg_train)
```

```
xg_train$bestTune
xg_train$results
```

Last modified: środa, 17 stycznia 2024, 9:35

nazwa użytkownika i hasło
wykorzystywane podczas logowania
się do systemu **USOSweb**.