How can I perform XGBoost in R using a step-by-step approach?

Authored by

stats writer

April 22, 2024

RECOMMENDED CITATION

stats writer (2024). *How can I perform XGBoost in R using a step-by-step approach?*. PSYCHOLOGICAL SCALES. Retrieved from https://scales.arabpsychology.com/?p=138200

ARABPSYCHOLOGY.COM

XGBoost is a popular machine learning algorithm used for predictive modeling and classification tasks. It stands for "Extreme Gradient Boosting" and is known for its speed and performance in handling large datasets. In order to perform XGBoost in R, a step-by-step approach can be followed to ensure accurate and efficient results.

The first step is to import the necessary packages, including the "xgboost" and "caret" packages. Next, the data needs to be preprocessed, including handling missing values, converting categorical variables, and splitting the data into training and testing sets.

Once the data is prepared, the XGBoost model can be built by specifying the parameters such as the objective function, the number of trees, and the learning rate. This is followed by training the model on the training set and evaluating its performance on the testing set.

After evaluating the model, it can be fine-tuned by adjusting the parameters and performing cross-validation to find the optimal values. Finally, the model can be used to make predictions on new data and the results can be evaluated using performance metrics such as accuracy, precision, and recall.

In summary, performing XGBoost in R requires a systematic approach, including data preprocessing, model building, fine-tuning, and evaluation, to achieve accurate and efficient results.

XGBoost in R: A Step-by-Step Example

Boosting is a technique in machine learning that has been shown to produce models with high predictive accuracy.

One of the most common ways to implement boosting in practice is to use XGBoost, short for "extreme gradient boosting."

This tutorial provides a step-by-step example of how to use XGBoost to fit a boosted model in R.

ARABPSYCHOLOGY.COM

Step 1: Load the Necessary Packages

First, we'll load the necessary libraries.

```
library(xgboost) #for fitting the xgboost modellibrary(caret) #for general data preparation and model fitting
```

Step 2: Load the Data

For this example we'll fit a boosted regression model to the Boston dataset from the MASS package.

This dataset contains 13 predictor variables that we'll use to predict one response variable called mdev, which represents the median value of homes in different census tracts around Boston.

```
#load the data
data = MASS::Boston

#view the structure of the data
str(data)

'data.frame': 506 obs. of 14 variables:
$ crim : num 0.00632 0.02731 0.02729 0.03237 0.06905 ...
$ zn : num 18 0 0 0 0 0 12.5 12.5 12.5 12.5 ...
```

ARABPSYCHOLOGY.COM

$ indus : num 2.31 7.07 7.07 2.18 2.18 2.18 7.87 7.87 7.87 7.87 ...

$ chas : int 0 0 0 0 0 0 0 0 0 0 ...

$ nox : num 0.538 0.469 0.469 0.458 0.458 0.458 0.524 0.524 0.524 0.524 ...

$ rm : num 6.58 6.42 7.18 7 7.15 ...

$ age : num 65.2 78.9 61.1 45.8 54.2 58.7 66.6 96.1 100 85.9 ...

$ dis : num 4.09 4.97 4.97 6.06 6.06 ...

$ rad : int 1 2 2 3 3 3 5 5 5 5 ...

$ tax : num 296 242 242 222 222 222 311 311 311 311 ...

$ ptratio: num 15.3 17.8 17.8 18.7 18.7 18.7 15.2 15.2 15.2 15.2 ...

$ black : num 397 397 393 395 397 ...

$ lstat : num 4.98 9.14 4.03 2.94 5.33 ...

$ medv : num 24 21.6 34.7 33.4 36.2 28.7 22.9 27.1 16.5 18.9 ...

ARABPSYCHOLOGY.COM

We can see that the dataset contains 506 observations and 14 total variables.

Step 3: Prep the Data

Next, we'll use the createDataPartition() function from the caret package to split the original dataset into a

training and testing set.

For this example, we'll choose to use 80% of the original dataset as part of the training set.

Note that the xgboost package also uses matrix data, so we'll use the data.matrix() function to hold our predictor variables.

```
#make this example reproducible
set.seed(0)

#split into training (80%) and testing set (20%)
parts = createDataPartition(data$medv, p = .8, list = F)
train = data
test = data

#define predictor and response variables in training set
train_x = data.matrix(train)
train_y = train

#define predictor and response variables in testing set
test_x = data.matrix(test)
test_y = test

#define final training and testing sets
```

ARABPSYCHOLOGY.COM

```
xgb_train = xgb.DMatrix(data = train_x, label = train_y)
xgb_test = xgb.DMatrix(data = test_x, label = test_y)
```

Step 4: Fit the Model

Note that we chose to use 70 rounds for this example, but for much larger datasets it's not uncommon to use hundreds or even thousands of rounds. Just keep in mind that the more rounds, the longer the run time.

Also note that the max.depth argument specifies how deep to grow the individual decision trees. We typically choose this number to be quite low like 2 or 3 so that smaller trees are grown. It has been shown that this approach tends to produce more accurate models.

```
#define watchlist
watchlist = list(train=xgb_train, test=xgb_test)

#fit XGBoost model and display training and testing data at each round
model = xgb.train(data = xgb_train, max.depth = 3, watchlist=watchlist, nrounds = 70)
```

```
train-rmse:10.167523 test-rmse:10.839775
train-rmse:7.521903 test-rmse:8.329679
```

ARABPSYCHOLOGY.COM

train-rmse:5.702393 test-rmse:6.691415
train-rmse:4.463687 test-rmse:5.631310
train-rmse:3.666278 test-rmse:4.878750
train-rmse:3.159799 test-rmse:4.485698
train-rmse:2.855133 test-rmse:4.230533
train-rmse:2.603367 test-rmse:4.099881
train-rmse:2.445718 test-rmse:4.084360
train-rmse:2.327318 test-rmse:3.993562
train-rmse:2.267629 test-rmse:3.944454
train-rmse:2.189527 test-rmse:3.930808
train-rmse:2.119130 test-rmse:3.865036
train-rmse:2.086450 test-rmse:3.875088
train-rmse:2.038356 test-rmse:3.881442
train-rmse:2.010995 test-rmse:3.883322
train-rmse:1.949505 test-rmse:3.844382
train-rmse:1.911711 test-rmse:3.809830
train-rmse:1.888488 test-rmse:3.809830
train-rmse:1.832443 test-rmse:3.758502
train-rmse:1.816150 test-rmse:3.770216
train-rmse:1.801369 test-rmse:3.770474
train-rmse:1.788891 test-rmse:3.766608
train-rmse:1.751795 test-rmse:3.749583
train-rmse:1.713306 test-rmse:3.720173
train-rmse:1.672227 test-rmse:3.675086

ARABPSYCHOLOGY.COM

train-rmse:1.648323 test-rmse:3.675977
train-rmse:1.609927 test-rmse:3.745338
train-rmse:1.594891 test-rmse:3.756049
train-rmse:1.578573 test-rmse:3.760104
train-rmse:1.559810 test-rmse:3.727940
train-rmse:1.547852 test-rmse:3.731702
train-rmse:1.534589 test-rmse:3.729761
train-rmse:1.520566 test-rmse:3.742681
train-rmse:1.495155 test-rmse:3.732993
train-rmse:1.467939 test-rmse:3.738329
train-rmse:1.446343 test-rmse:3.713748
train-rmse:1.435368 test-rmse:3.709469
train-rmse:1.401356 test-rmse:3.710637
train-rmse:1.390318 test-rmse:3.709461
train-rmse:1.372635 test-rmse:3.708049
train-rmse:1.367977 test-rmse:3.707429
train-rmse:1.359531 test-rmse:3.711663
train-rmse:1.335347 test-rmse:3.709101
train-rmse:1.331750 test-rmse:3.712490
train-rmse:1.313087 test-rmse:3.722981
train-rmse:1.284392 test-rmse:3.712840
train-rmse:1.257714 test-rmse:3.697482
train-rmse:1.248218 test-rmse:3.700167
train-rmse:1.243377 test-rmse:3.697914

ARABPSYCHOLOGY.COM

train-rmse:1.231956 test-rmse:3.695797
train-rmse:1.219341 test-rmse:3.696277
train-rmse:1.207413 test-rmse:3.691465
train-rmse:1.197197 test-rmse:3.692108
train-rmse:1.171748 test-rmse:3.683577
train-rmse:1.156332 test-rmse:3.674458
train-rmse:1.147686 test-rmse:3.686367
train-rmse:1.143572 test-rmse:3.686375
train-rmse:1.129780 test-rmse:3.679791
train-rmse:1.111257 test-rmse:3.679022
train-rmse:1.093541 test-rmse:3.699670
train-rmse:1.083934 test-rmse:3.708187
train-rmse:1.067109 test-rmse:3.712538
train-rmse:1.053887 test-rmse:3.722480
train-rmse:1.042127 test-rmse:3.720720
train-rmse:1.031617 test-rmse:3.721224
train-rmse:1.016274 test-rmse:3.699549
train-rmse:1.008184 test-rmse:3.709522
train-rmse:0.999220 test-rmse:3.708000
train-rmse:0.985907 test-rmse:3.705192

From the output we can see that the minimum testing RMSE is achieved at 56 rounds. Beyond this point, the test RMSE actually begins to increase, which is a sign

ARABPSYCHOLOGY.COM

that we're overfitting the training data.

Thus, we'll define our final XGBoost model to use 56 rounds:

```
#define final model
final = xgboost(data = xgb_train, max.depth = 3, nrounds = 56, verbose = 0)
```

Note: The argument verbose = 0 tells R not to display the training and testing error for each round.

Step 5: Use the Model to Make Predictions

Lastly, we can use the final boosted model to make predictions about the median house value of Boston homes in the testing set.

We will then calculate the following accuracy measures for the model:

MSE: Mean Squared ErrorMAE: Mean Absolute ErrorRMSE: Root Mean Squared Error

```
mean((test_y - pred_y)^2) #mse
caret::MAE(test_y, pred_y) #mae
```

ARABPSYCHOLOGY.COM

```
caret::RMSE(test_y, pred_y) #rmse

13.50164
2.409426
3.674457
```

The root mean squared error turns out to be 3.674457. This represents the average difference between the prediction made for the median house values and the actual observed house values in the test set.

If we want, we could compare this RMSE to other models like multiple linear regression, ridge regression, principal components regression, etc. to see which model produces the most accurate predictions.

You can find the complete R code used in this example here.

ARABPSYCHOLOGY.COM