sparklyr: tuning your hyperparameter

Samuel MacÃªdo

2018-06-21

Welcome, friend :)

This is another tutorial about spark using the [sparklyr](https://spark.rstudio.com/) package. In this way, I am going to present you how tuning your model parameters. It’s not so difficult but there is some details that I have tell you. If you are not confident yet about trainning your models in spark, check [my previous](http://samuelmacedo.netlify.com/2018/05/sparklyr-supervised-learning/) post and come back here later :)

Let’s get to action…

# Pipeline

First of all, you need to create a pipeline. A pipeline is just a sequence of steps that you are going to execute in your data. In sparklyr context, you don’t point a pipeline directly to your data, you have to point to connection where the data is connected. Let’s see.

library(sparklyr)  
  
sc <- spark\_connect(master = "local")  
  
pipeline <- ml\_pipeline(sc)

Now that you create a pipeline object, you can add the steps that you like to use. To add these steps you can use the %>% operator. I’ll perform an example using random forest in our well-known iris dataset.

iris\_tbl <- sdf\_copy\_to(sc, iris, name = "iris\_tbl", overwrite = TRUE)  
  
pipeline <- ml\_pipeline(sc) %>%  
 ft\_r\_formula(Species ~ .) %>%  
 ml\_random\_forest\_classifier()

The ft\_r\_formula() function creates two columns named features and label. You don’t need to worry, it’s just a way to “tell” to spark what are the predictors (features) and the response (label) using the R syntax (Species ~ .). If you want to check, run it into your console ft\_r\_formula(iris\_tbl, Species ~ .).

The ml\_random\_forest(), used in the [previous post](http://samuelmacedo.netlify.com/2018/05/sparklyr-supervised-learning/), is a wrapper and inside it there are two other functions: ml\_random\_forest\_classifier() (for classification) and ml\_random\_forest\_regressor() (for regression). I’d rather be very explicit concerning the steps inside the pipeline, it’s a good practice to avoid a headache when you have to read a very long pipeline with a lot of steps :(  
Let’s see how the object pipeline looks like!

pipeline  
## Pipeline (Estimator) with 2 stages  
## <pipeline\_1c30275a6eff>   
## Stages   
## |--1 RFormula (Estimator)  
## | <r\_formula\_1c3056f063b2>   
## | (Parameters -- Column Names)  
## | features\_col: features  
## | label\_col: label  
## | (Parameters)  
## | force\_index\_label: FALSE  
## | formula: Species ~ .  
## | handle\_invalid: error  
## | stringIndexerOrderType: frequencyDesc  
## |--2 RandomForestClassifier (Estimator)  
## | <random\_forest\_classifier\_1c30537b37fa>   
## | (Parameters -- Column Names)  
## | features\_col: features  
## | label\_col: label  
## | prediction\_col: prediction  
## | probability\_col: probability  
## | raw\_prediction\_col: rawPrediction  
## | (Parameters)  
## | cache\_node\_ids: FALSE  
## | checkpoint\_interval: 10  
## | feature\_subset\_strategy: auto  
## | impurity: gini  
## | max\_bins: 32  
## | max\_depth: 5  
## | max\_memory\_in\_mb: 256  
## | min\_info\_gain: 0  
## | min\_instances\_per\_node: 1  
## | num\_trees: 20  
## | seed: 207336481  
## | subsampling\_rate: 1

As you can see, our pipeline has two stages: RFormula and RandomForestClassifier. In each one the parameters that may be used are specified

# The grid parameters

To control the parameters that you want to check, you need to create a grid object. An example is shown below.

grid <- list(  
 random\_forest = list(  
 num\_trees = c(5,10),  
 impurity = c("entropy", "gini")  
 )  
)

In this grid that I created, I will test what is the best combination of parameters, varying the number of trees, and the impurity. You can change and use any combination that you want to check. There is one thing that you have to keep in mind, the name of the list inside the grid has to be the same name used by the classifier. Let’s look again our pipeline!

pipeline  
## Pipeline (Estimator) with 2 stages  
## <pipeline\_1c30275a6eff>   
## Stages   
## |--1 RFormula (Estimator)  
## | <r\_formula\_1c3056f063b2>   
## | (Parameters -- Column Names)  
## | features\_col: features  
## | label\_col: label  
## | (Parameters)  
## | force\_index\_label: FALSE  
## | formula: Species ~ .  
## | handle\_invalid: error  
## | stringIndexerOrderType: frequencyDesc  
## |--2 RandomForestClassifier (Estimator)  
## | <random\_forest\_classifier\_1c30537b37fa>   
## | (Parameters -- Column Names)  
## | features\_col: features  
## | label\_col: label  
## | prediction\_col: prediction  
## | probability\_col: probability  
## | raw\_prediction\_col: rawPrediction  
## | (Parameters)  
## | cache\_node\_ids: FALSE  
## | checkpoint\_interval: 10  
## | feature\_subset\_strategy: auto  
## | impurity: gini  
## | max\_bins: 32  
## | max\_depth: 5  
## | max\_memory\_in\_mb: 256  
## | min\_info\_gain: 0  
## | min\_instances\_per\_node: 1  
## | num\_trees: 20  
## | seed: 207336481  
## | subsampling\_rate: 1

In our example, the random forest classifier is on the second stage, take a look on the second line, did you see <random\_forest\_classifier\_###########>? That is the name you have to use, ok? You don’t need to use the entire name, note that I just use the beginning, for short.

# Cross validator object

Our final step is to create a cross validator object. Let’s check it out.

cv <- ml\_cross\_validator(  
 sc, estimator = pipeline, estimator\_param\_maps = grid,  
 evaluator = ml\_multiclass\_classification\_evaluator(sc),  
 num\_folds = 5,  
 parallelism = 4  
)

As you can see, you have to pass your connection, pipeline and the grid parameters that you want to test. Make sure you are using the correct evaluator, in our case, is multiclass. The num\_folds is how much folders you want to use in your train-validation split and parallelism is the number of threads to use when running parallel algorithms.

# Train and check the metrics

Now, we arrived to fun part, let’s train and check the metrics out!

# Train the models  
cv\_model <- ml\_fit(cv, iris\_tbl)  
  
# Print the metrics  
ml\_validation\_metrics(cv\_model)  
## f1 impurity\_1 num\_trees\_1  
## 1 0.9563436 entropy 5  
## 2 0.9622804 entropy 10  
## 3 0.9630614 gini 5  
## 4 0.9461037 gini 10

In our toy example, the best parameters were impurity Gini with 5 number of trees, because f1-score was the greater (0.963). Now it is your turn, try different parameters and have some fun :)

## That’s all folks

Liked it? You can share this tutorial using the buttons below. If you want to contribute with my website you can fork me on [github](https://github.com/samuelmacedo83/my_webpage). If you still have any doubts feel free to contact at samuelmacedo@recife.ifpe.edu.br.

See ya!