Developer Network

Developing Analytics Solutions with the Data & AI Global Practice

Validating a model in R Services using the k-fold

★★★★★

July 8, 2016 by Michele Usuelli // 0 Comments

By **Michele Usuelli**, **Data Scientist Consultant**

Why k-fold

Predictive modelling consists in predicting a future outcome based on the data. Starting from data which outcome is already known, the predictive models detect patterns that had an impact on the outcome. Then, in presence of data which outcome is unknown, the model looks for the same patterns and predicts the outcome on their base. Since there are several business scenarios where a prediction will drive a lot of value, predictive modelling is among the most popular machine learning scenarios.

Before deploying a predictive model, it's essential to estimate how accurate it will be. To measure the accuracy of a model, a common approach is the *cross-validation* and it's based on testing the model on the data which outcome is already known. The measures of the accuracy are based on comparing the prediction with the actual value. In order to simulate a real-life scenario, cross-validation consists in splitting the data into two parts: the training set and the test set. The model learns the patterns from the training set and predicts the outcome of the test set.

Splitting the data in two parts as described, the estimation of the accuracy is based on the prediction of the test set only. To extend this approach to the entire data-set, we can split the data multiple times, in such a way that we perform a prediction on each record at least once. This approach is called k-fold and these are the steps:

1. Split the data into k sets
2. For each set, build a predictive model using the remaining data and predict the outcome on the set
3. Validate the model comparing the real outcomes with the predictions

In this article we see how to implement it using SQL Server 16 with Microsoft R Services.

A use-case

To show the k-fold approach, we deal with a business-like scenario. Some customers of a bank have mortgages and the bank wants to know which people are the most likely to have a default. The bank collected some data about its customers. Each record corresponds to a customer and each column to an attribute. The last column is 1 if the customer defaulted and 0 otherwise. This is a sample of the data

creditScore	houseAge	yearsEmploy	ccDebt	year	default
691	16	9	6725	2000	0
691	4	4	5077	2000	0
743	18	3	3080	2000	0
728	22	1	4345	2000	0
745	17	3	2969	2000	0
539	15	3	4588	2000	0

Starting from data about new customers, the target is to predict whether the last column will be 0 or 1.

Facing the use-case using SQL Server 16

In this article, we use a SQL Server 16 VM in the Azure cloud with Microsoft R Services and RStudio. To organize the R code, we use a RStudio project.

The data is initially stored into a xdf (eXternal Data Frame, the Microsoft R Services proprietary file-system). To predict the default, we use a popular model: the logistic regression.

This article is high-level in the sense that we're going through the logic of the steps, without diving into the technical details of the code syntax.

The steps to build the k-fold cross-validation are

1. **Define the data sources and the parameters**
    - Define a xdf specifying where the original data is stored. The data is contained into a sample data directory. To define the data source, we use the function *rxReadXdf*.

```
mortDefaultSmall <- rxReadXdf(file = file.path(rxGetOption("sampleDataDir"),

"mortDefaultSmall.xdf"))
```

- Define a local xdf where we want to build a copy of the original data. The file will be stored within the RStudio project.

```
xdf_in <- RxXdfData("mort.xdf")
```

- Define the number of folds of the algorithm.

```
n_folds <- 5
```

- Define the outcome that we want to predict.

```
target_outcome <- "default"
```

- Define the fields that we want to use to predict the target outcome. In this case, the predictors are the first 5 columns of *mortDefaultSmall*.

```
predictors <- names(mortDefaultSmall)[1:5]
```

- **Import the data and define the fold column**: we define a local copy of the mortgages data, with the addition of a fold column, containing a random number between 1 and 5, that assigns each record to one of the 5 sets. Also, we define a prediction column, containing the predicted score of having a credit default. The default value of the prediction is -1. For these purposes, we use the function *rxDataStep*.

```
rxDataStep(mortDefaultSmall, xdf_in,

transformObjects = list(n_folds = n_folds),

transforms = list(fold = sample(1:n_folds, .rxNumRows, TRUE),

predicted = rep(-1., .rxNumRows)),

overwrite = TRUE)
```

2. **Define the features to include into the predictive model**: to build the prediction model, we need to define a *formula* object describing which column is the outcome we want to predict and which columns are the predictors. The syntax is *outcome ~ predictor1 + predictor2*. To build the string, we use the function *paste*. Then, to define a formula object, we use the function *formula*.

```
string_predictors <- paste(predictors, collapse = " + ")

formula_predict <- formula(paste(target_outcome, string_predictors, sep = " ~ "))

formula_predict
```

default ~ creditScore + houseAge + yearsEmploy + ccDebt + year

3. **For each fold, build and apply the model as per described before**

We run a for loop defining a prediction for each set.

```
for(this_fold in 1:n_folds) {
```

- Build a predictive model on the base of the remaining data. For this purpose, we use rxLogit and we specify that the model is based on the data which fold is different from *this_fold*. To build a logistic regression model, we use *rxLogit*.

```
model_logit <- rxLogit(formula = formula_predict,

data = xdf_in,

transformObjects = list(this_fold = this_fold),

rowSelection = fold != this_fold)
```

- Predict the default on the base of the predictive model. For technical reasons, the easiest way is to predict it on the entire dataset and to select the data about the fold later. We call this column *prediction_fold*. The function predicting the score is *rxPredict*.

```
rxPredict(model_logit, xdf_in,

predVarNames = "prediction_fold",

overwrite = TRUE)
```

- For each record of the fold, set the *predicted* column equal to *prediction_fold*. For this purpose, we use the function *rxDataStep*.

```
rxDataStep(xdf_in, xdf_in,

transformObjects = list(this_fold = this_fold),

transforms = list(

predicted = ifelse(fold == this_fold, prediction_fold, predicted)),

overwrite = TRUE)
```

```
}
```

4. **Measure the accuracy of the model**: to evaluate the performance, we use a popular metric called AUC. For this purpose, we use the functions *rxRoc* and *rxAuc*.

```
roc <- rxRoc(target_outcome, "predicted", xdf_in)

auc <- rxAuc(roc)

auc
```

```
[1] 0.9328864
```

The AUC score measures the predictive power of the model and it can be used to compare different models and optimize their parameters.

Conclusions

Using SQL Server with Microsoft R Server, we have been able to set-up the K-fold cross-validation. The code is written in such a way that it can be scaled across large data volumes.

Popular Tags

#ArtificialIntelligence #DataScience AI AML Analytics Apache Spark APS Artificial Intelligence Azure Data Factory Azure Key Vault AzureML Azure SQL Data Warehouse Basket analysis Center of Excellence Centre of Excellence CoE Data Data Science Data Scientist Role Decision forests Event Hubs k-fold Machine Learning Measurability Model Goodness Modelling Power BI PowerShell ML Scoring R Random Projection R Services Scikit SQL Server 2016 SQL Server R Services Stream Analytics Visualizations Whitepaper

Archives

November 2018 (1)
All of 2018 (4)
All of 2017 (3)
All of 2016 (11)
All of 2015 (11)

Tags k-fold R R Services SQL Server 2016

Join the conversation

Add Comment

Dev centers

Windows

Learning resources

Microsoft Virtual Academy

Channel 9