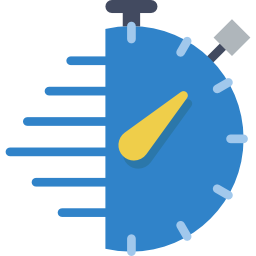
Native scoring is a much overlooked feature in SQL Server 2017 (available only under Windows and only on-prem), that provides scoring and predicting in pre-build and stored machine learning models in near real-time.



**Icons made by** [**Smashicons**](https://www.flaticon.com/authors/smashicons) **from** [**www.flaticon.com**](https://www.flaticon.com/) **is licensed by** [**CC 3.0 BY**](http://creativecommons.org/licenses/by/3.0/)

Depending on the definition of real-time, and what does it mean for your line of business, I will not go into the definition of real-time, but for sure, we can say scoring 10.000 rows in a second from a mediocre client computer (similar to mine) .

Native scoring in SQL Server 2017 comes with couple of limitations, but also with a lot of benefits. Limitations are:

* currently supports only SQL server 2017 and Windows platform
* trained model should not exceed 100 MiB in size
* Native scoring with PREDICT function supports only following algorithms from RevoScaleR library:
  + rxLinMod (linear model as linear regression)
  + rxLogit (logistic regression)
  + rxBTrees (Parallel external memory algorithm for Stochastic Gradient Boosted Decision Trees)
  + rxDtree (External memory algorithm for Classification and Regression Trees
  + rxDForest (External memory algorithm for Classification and Regression Decision Trees)

Benefits of using PREDICT function for native scoring are:

* No configuration of R or ML environment is needed (assuming that the trained models are already stored in the database),
* Code is cleaner, more readable, and no additional R code is needed when performing scoring,
* No R engine is called in the run-time, so tremendous deduction of  CPU and I/O costs as well as, no external calls,
* Client or server running Native scoring with PREDICT function does not need R engine installed, because it uses C++ libraries from Microsoft, that can read serialized model stored in a table and un-serialize it and generate predictions, all without the need of R

Overall, if you are looking for a faster predictions in your enterprise and would love to have a faster code and solution deployment, especially integration with other applications or building API in your ecosystem, native scoring with PREDICT function will surely be advantage to you. Although not all of the predictions/scores are supported, majority of predictions can be done using regression models or decision trees models (it is estimated that both type (with derivatives of regression models and ensemble methods) of algorithms are used in 85% of the predictive analytics).

To put the PREDICT function to the test, I have deliberately taken the semi-larger dataset, available in RevoScaleR package in R – AirlineDemoSmall.csv. Using a simple BULK INSERT, we get the data into the database:

BULK INSERT ArrivalDelay

FROM 'C:\Program Files\Microsoft SQL Server\140\R\_SERVER\library\RevoScaleR\SampleData\AirlineDemoSmall.csv'

WITH

( FIELDTERMINATOR =',', ROWTERMINATOR = '0x0a', FIRSTROW = 2, CODEPAGE = 'RAW');

Once data is in the database, I will split the data into training and test sub-sets.

SELECT TOP 20000 \*

INTO ArrDelay\_Train

FROM ArrDelay ORDER BY NEWID()

-- (20000 rows affected)

SELECT \*

INTO ArrDelay\_Test

FROM ArrDelay AS AR

WHERE NOT EXISTS (SELECT \* FROM ArrDelay\_Train as ATR

WHERE

ATR.arrDelay = AR.arrDelay

AND ATR.[DayOfWeek] = AR.[DayOfWeek]

AND ATR.CRSDepTime = AR.CRSDepTime

)

-- (473567 rows affected

And the outlook of the dataset is relatively simple:

ArrDelay CRSDepTime DayOfWeek

1 9,383332 3

4 18,983334 4

0 13,883333 4

65 21,499998 7

-3 6,416667 1

**Creating models**

So we will create essentially two same models using rxLinMod function with same formula, but one with additional parameter for real-time scoring set to TRUE.

**-- regular model creation**

DECLARE @model VARBINARY(MAX);

EXECUTE sp\_execute\_external\_script

@language = N'R'

,@script = N'

arrDelay.LM <- rxLinMod(ArrDelay ~ DayOfWeek + CRSDepTime,

data = InputDataSet)

model <- rxSerializeModel(arrDelay.LM)'

,@input\_data\_1 = N'SELECT \* FROM ArrDelay\_Train'

,@params = N'@model varbinary(max) OUTPUT'

,@model = @model OUTPUT

INSERT [dbo].arrModels([model\_name], [native\_model])

VALUES('arrDelay.LM.V1', @model) ;

**-- Model for Native scoring**

DECLARE @model VARBINARY(MAX);

EXECUTE sp\_execute\_external\_script

@language = N'R'

,@script = N'

arrDelay.LM <- rxLinMod(ArrDelay ~ DayOfWeek + CRSDepTime,

data = InputDataSet)

model <- rxSerializeModel(arrDelay.LM, **realtimeScoringOnly = TRUE**)'

,@input\_data\_1 = N'SELECT \* FROM ArrDelay\_Train'

,@params = N'@model varbinary(max) OUTPUT'

,@model = @model OUTPUT

INSERT [dbo].arrModels([model\_name], [native\_model])

VALUES('arrDelay.LM.NativeScoring.V1', @model) ;

Both models will have same training set, and will be stored into a table for future scoring. Upon first inspection, we can see there is a difference in the model size:

model_size

**Scoring Models**

Both models  took relatively the same amount of time to train and to store in the table. Both can also be created on R Machine Learning server and stored in the same way (with or without argument realtimeScoringOnly). The model size gives you an idea, why and how the realtime scoring can be achieved -> is to keep your model as small as possible. Both models will give you exact same predictions scores, just that the native scoring will be much faster. Note also, if you are planning to do any text analysis with real-time scoring, keep in mind the 100 MiB limitation, as the text prediction models often exceed this limitation.

Comparing the execution of scoring models, I will compare using “traditional way” of using external procedure sp\_execute\_external\_script and using [PREDICT](https://docs.microsoft.com/en-us/sql/t-sql/queries/predict-transact-sql?view=sql-server-2017) function.

------------------------------------

-- Using sp\_execute\_external\_script

------------------------------------

DECLARE @model VARBINARY(MAX) = (SELECT native\_model FROM arrModels

WHERE model\_name = '**arrDelay.LM.V1**')

EXEC sp\_execute\_external\_script

@language = N'R'

,@script = N'

modelLM <- rxUnserializeModel(model)

OutputDataSet <- rxPredict( model=modelLM,

data = ArrDelay\_Test,

type = "link",

predVarNames = "ArrDelay\_Pred",

extraVarsToWrite = c("ArrDelay","CRSDepTime","DayOfWeek")

)'

,@input\_data\_1 = N'SELECT \* FROM dbo.ArrDelay\_Test'

,@input\_data\_1\_name = N'ArrDelay\_Test'

,@params = N'@model VARBINARY(MAX)'

,@model = @model

WITH RESULT SETS

((

AddDelay\_Pred FLOAT

,ArrDelay INT

,CRSDepTime NUMERIC(16,5)

,[DayOfWeek] INT

))

-- (473567 rows affected)

-- Duration 00:00:08

---------------------------

-- Using Real Time Scoring

---------------------------

DECLARE @model varbinary(max) = ( SELECT native\_model FROM arrModels

WHERE model\_name = '**arrDelay.LM.NativeScoring.V1**');

SELECT

NewData.\*

,p.\*

FROM PREDICT(MODEL = @model, DATA = dbo.ArrDelay\_Test as newData)

WITH(ArrDelay\_Pred FLOAT) as p;

GO

-- (473567 rows affected)

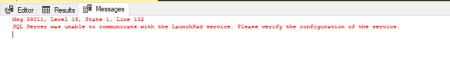
-- Duration 00:00:04

Both examples are different from each other, but PREDICT function looks much more readable and neater. Time performance is also on the PREDICT function side, as the model returns the predictions much faster.

In addition, I have mentioned that PREDICT function does not need R engine or Launchpad Service to be running in the same environment, where the code will be executed. To put this to test, I will simply stop the SQL Server Launchpad Service:

Service_stop

After executing the first set of predictions using sp\_execute\_external\_script, SQL Server or Machine Learning Server will notify you that the service is not running:



whereas, the PREDICT function will work flawlessly.

**Verdict**

For sure, faster predictions are the something that can be very welcoming in gaming industry, in transport, utility and metal industry, financial as well as any other types, where real-time predictions against OLTP systems will be much appreciated. With the light-weight models and good algorithm support, I would for sure give it an additional thought, especially, if you see a good potential in faster and near real-time predictions.