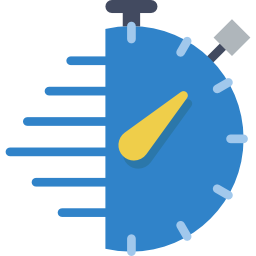
Native scoring is a much overlooked feature in SQL Server 2017 (available only under Windows and only on-prem), that provides scoring and predicting in pre-build and stored machine learning models in near real-time.



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 function.

PREDICT Function

**Syntax**

syntaxsqlCopy

PREDICT

(

MODEL = @model | model\_literal,

DATA = object AS <table\_alias>

[, RUNTIME = ONNX ]

)

WITH ( <result\_set\_definition> )

<result\_set\_definition> ::=

{

{ column\_name

data\_type

[ COLLATE collation\_name ]

[ NULL | NOT NULL ]

}

[,...n ]

}

MODEL = @model | model\_literal

**Arguments**

**MODEL**

The MODEL parameter is used to specify the model used for scoring or prediction. The model is specified as a variable or a literal or a scalar expression.

PREDICT supports models trained using the [RevoScaleR](https://docs.microsoft.com/en-us/sql/machine-learning/r/ref-r-revoscaler?view=sql-server-2017) and [revoscalepy](https://docs.microsoft.com/en-us/sql/machine-learning/python/ref-py-revoscalepy?view=sql-server-2017) packages.

**DATA**

The DATA parameter is used to specify the data used for scoring or prediction. Data is specified in the form of a table source in the query. Table source can be a table, table alias, CTE alias, view, or table-valued function.

**RUNTIME = ONNX**

**Important**

The RUNTIME = ONNX argument is only available in [**Azure SQL Managed Instance**](https://docs.microsoft.com/en-us/azure/azure-sql/managed-instance/machine-learning-services-overview), [**Azure SQL Edge**](https://docs.microsoft.com/en-us/azure/sql-database-edge/onnx-overview), and [**Azure Synapse Analytics**](https://docs.microsoft.com/en-us/azure/synapse-analytics/overview-what-is).

Indicates the machine learning engine used for model execution. The RUNTIME parameter value is always ONNX. The parameter is required for Azure SQL Edge and Azure Synapse Analytics. On Azure SQL Managed Instance, the parameter is optional and only used when using ONNX models.

**WITH ( <result\_set\_definition> )**

The WITH clause is used to specify the schema of the output returned by the PREDICT function.

In addition to the columns returned by the PREDICT function itself, all the columns that are part of the data input are available for use in the query.

**Return values**

No predefined schema is available; the contents of the model is not validated and the returned column values are not validated either.

* The PREDICT function passes through columns as input.
* The PREDICT function also generates new columns, but the number of columns and their data types depends on the type of model that was used for prediction.

Any error messages related to the data, the model, or the column format are returned by the underlying prediction function associated with the model.

**Remarks**

The PREDICT function is supported in all editions of SQL Server 2017 or later, on Windows and Linux. [Machine Learning Services](https://docs.microsoft.com/en-us/sql/machine-learning/sql-server-machine-learning-services?view=sql-server-2017) does not need to be enabled to use PREDICT.

**Supported algorithms**

The model that you use must have been created using one of the supported algorithms from the [RevoScaleR](https://docs.microsoft.com/en-us/sql/machine-learning/r/ref-r-revoscaler?view=sql-server-2017) or [revoscalepy](https://docs.microsoft.com/en-us/sql/machine-learning/python/ref-py-revoscalepy?view=sql-server-2017) packages. For a list of currently supported models, see [Native scoring using the PREDICT T-SQL function](https://docs.microsoft.com/en-us/sql/machine-learning/predictions/native-scoring-predict-transact-sql?view=sql-server-2017).

**Permissions**

No permissions are required for PREDICT; however, the user needs EXECUTE permission on the database, and permission to query any data that is used as inputs. The user must also be able to read the model from a table, if the model has been stored in a table.

**Examples**

The following examples demonstrate the syntax for calling PREDICT.

**Using PREDICT in a FROM clause**

This example references the PREDICT function in the FROM clause of a SELECT statement:

SQLCopy

SELECT d.\*, p.Score

FROM PREDICT(MODEL = @model,

DATA = dbo.mytable AS d) WITH (Score FLOAT) AS p;

The alias **d** specified for table source in the DATA parameter is used to reference the columns belonging to dbo.mytable. The alias **p** specified for the PREDICT function is used to reference the columns returned by the PREDICT function.

* The model is stored as varbinary(max) column in table call **Models**. Additional information such as **ID** and **description** is saved in the table to identify the mode.
* The alias **d** specified for table source in the DATA parameter is used to reference the columns belonging to dbo.mytable. The input data column names should match the name of inputs for the model.
* The alias **p** specified for the PREDICT function is used to reference the predicted column returned by the PREDICT function. The column name should have the same name as the output name for the model.
* All input data columns and the predicted columns are available to display in the SELECT statement.

**Combining PREDICT with an INSERT statement**

A common use case for prediction is to generate a score for input data, and then insert the predicted values into a table. The following example assumes the calling application uses a stored procedure to insert a row containing the predicted value into a table:

SQLCopy

DECLARE @model VARBINARY(max) = (SELECT model FROM scoring\_model WHERE model\_name = 'ScoringModelV1');

INSERT INTO loan\_applications (c1, c2, c3, c4, score)

SELECT d.c1, d.c2, d.c3, d.c4, p.score

FROM PREDICT(MODEL = @model, DATA = dbo.mytable AS d) WITH(score FLOAT) AS p;

* The results of PREDICT are stored in a table called PredictionResults.
* The model is stored as varbinary(max) column in table call **Models**. Additional information such as ID and description can be saved in the table to identify the model.
* The alias **d** specified for table source in the DATA parameter is used to reference the columns in dbo.mytable.The input data column names should match the name of inputs for the model.
* The alias **p** specified for the PREDICT function is used to reference the predicted column returned by the PREDICT function. The column name should have the same name as the output name for the model.
* All input columns and the predicted column are available to display in the SELECT statement.

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

-- 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)

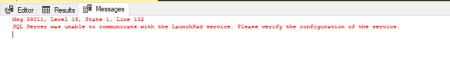
-- Duration 00:00:04

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.