Deploying R Predictive Models

From a business and production standpoint, creating a good predictive model and spending time on this, is only useful if you can deploy such a model in a system where the business can make use of the predictions in their day to day operations. In this tutorial, you will learn how to build end-to-end solution for predictive modeling based on R feature support in SQL Server. This tutorial uses the data of the just concluded Xente fraud detection competition on Zindi Africa.

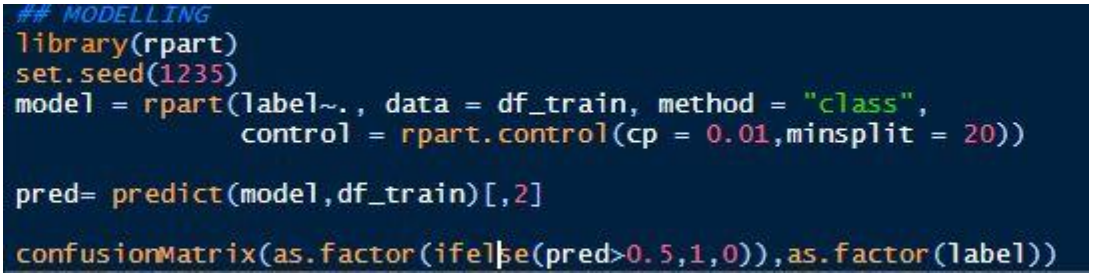
We will use a R code to build a simple baseline decision tree model that indicates the probability that the transaction on a given day and time might be fraudulent. We will also deploy the R model to SQL Server and use competition test data to generate scores based on the model. This tutorial can be extended to all kinds of real-life problems and machine learning algorithms, such as predicting customer churn, or predicting spending patterns. Because the model can be invoked from a stored procedure which can easily be embedded in an application.

Prerequisites: We need to install the following on our local machine: SQL Server machine learning Services with R integration, SQL Server management Studio, RStudio or R GUI, click here to get started on setting up your environment.

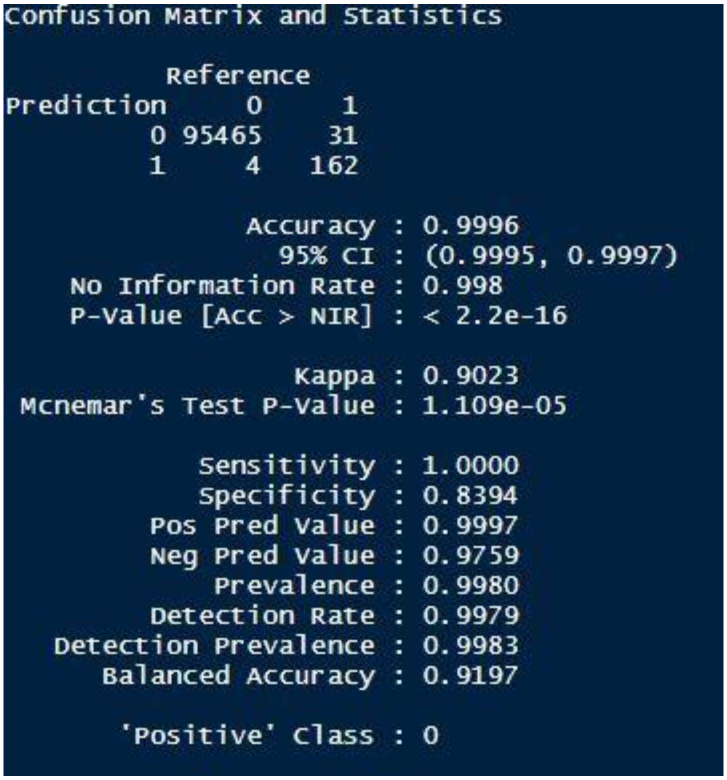
Data cleaning and Feature Engineering: I won't dwell too much on this as there is not much cleansing to do as the data is in a well-structured manner, unnecessary columns are removed and engineered new features such as

- Time features: hour of the day, most and least frequent transacting hours
- Date features: weekday, month, day of the month
- Unique count features: counts of unique value, counts of different groups of productid, hour, weekday, product category, providerid, channelId.

Modelling: we will build a simple decision tree classification model using the R rpart library shown in the screenshot below with the confusion matrix.



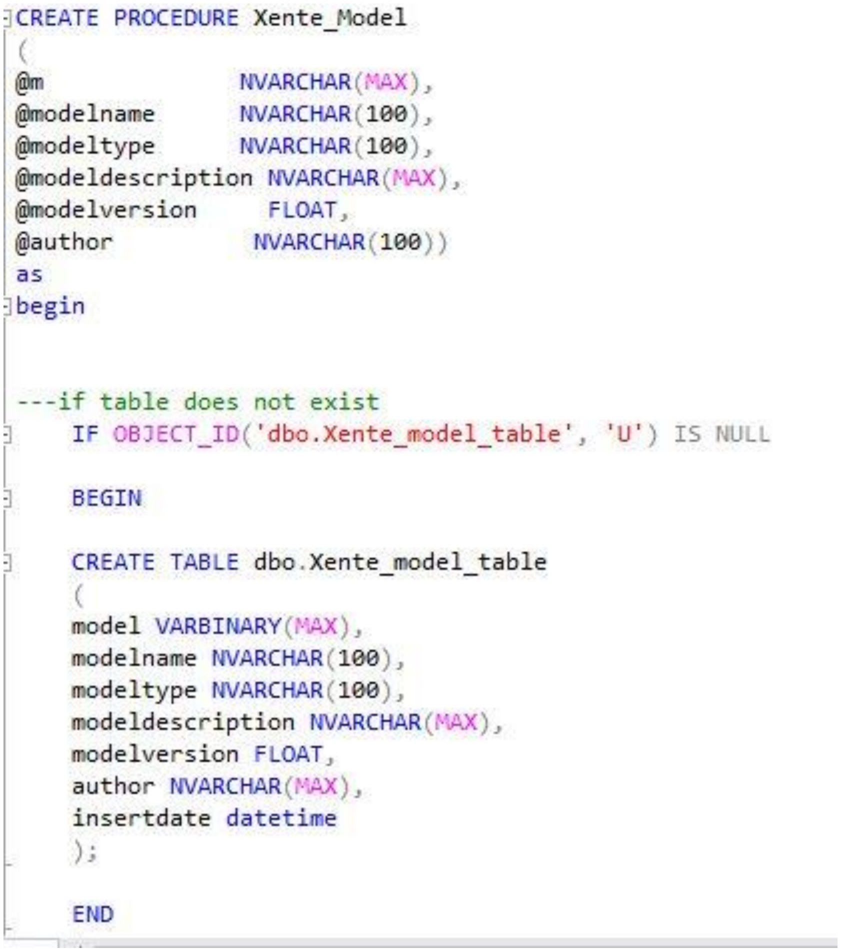
```
## MODELLING
library(rpart)
set.seed(1235)
model = rpart(label~., data = df_train, method = "class",
              control = rpart.control(cp = 0.01,minsplit = 20))

pred= predict(model,df_train)[,2]

confusionMatrix(as.factor(ifelse(pred>0.5,1,0)),as.factor(label))
```



```
Confusion Matrix and Statistics

          Reference
Prediction     0     1
         0 95465    31
         1     4   162
                                          
               Accuracy : 0.9996          
                 95% CI : (0.9995, 0.9997)
    No Information Rate : 0.998           
    P-Value [Acc > NIR] : < 2.2e-16       
                                          
                  Kappa : 0.9023          
 Mcnemar's Test P-Value : 1.109e-05       
                                          
            Sensitivity : 1.0000          
            Specificity : 0.8394          
         Pos Pred Value : 0.9997          
         Neg Pred Value : 0.9759          
             Prevalence : 0.9980          
         Detection Rate : 0.9979          
   Detection Prevalence : 0.9983          
      Balanced Accuracy : 0.9197          
                                          
       'Positive' Class : 0               
```

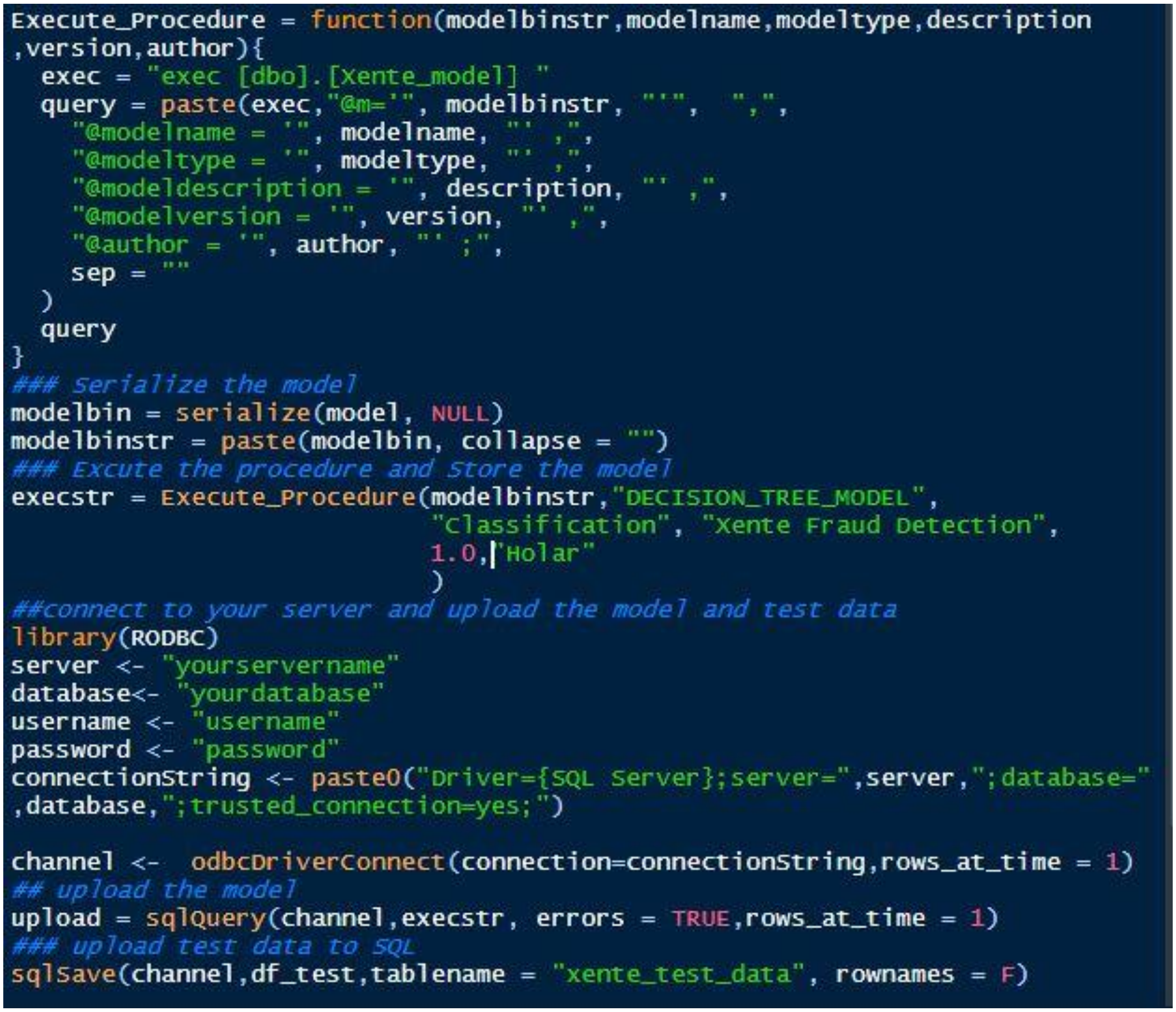
Once you are satisfied with the predictive model, next is to upload the model to SQL Server so that you can use it there. This consists of the following steps:

1. **SQL Code in SQL Server,** create a stored procedure (Xente_model) in SQL server that will accept the predictive R model, some metadata and saves it into a table (dbo.Xente_model_table) in SQL Server. This stored procedure will then be called from R session.



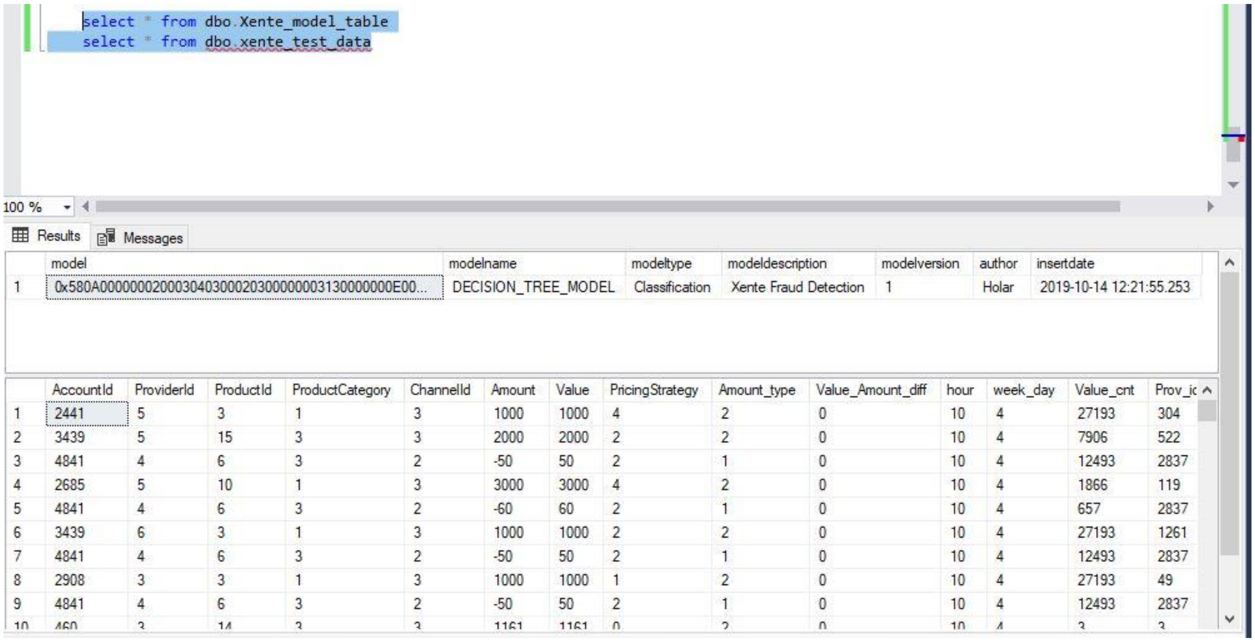
```
CREATE PROCEDURE Xente_Model
(
@m                NVARCHAR(MAX),
@modelname        NVARCHAR(100),
@modeltype        NVARCHAR(100),
@modeldescription NVARCHAR(MAX),
@modelversion       FLOAT,
@author            NVARCHAR(100))
as
begin


---if table does not exist
    IF OBJECT_ID('dbo.Xente_model_table', 'U') IS NULL

    BEGIN

    CREATE TABLE dbo.Xente_model_table
    (
    model VARBINARY(MAX),
    modelname NVARCHAR(100),
    modeltype NVARCHAR(100),
    modeldescription NVARCHAR(MAX),
    modelversion FLOAT,
    author NVARCHAR(MAX),
    insertdate datetime
    );

    END
```

2. **Upload model from R to SQL:** A typical deployment workflow consists of the following 3 steps: Serialize the model into a hexadecimal string, transmit the serialized object to the database and Save the model in a varbinary(max) column. To make it a little bit easier you can write a small function which takes in the model and its metadata and execute the stored procedure created in step 1 and store the model on SQL server.



```r
Execute_Procedure = function(modelbinstr,modelname,modeltype,description
,version,author){
  exec = "exec [dbo].[Xente_model] "
  query = paste(exec,"@m='", modelbinstr, "'",  ",",
    "@modelname = '", modelname, "' ,",
    "@modeltype = '", modeltype, "' ,",
    "@modeldescription = '", description, "' ,",
    "@modelversion = '", version, "' ,",
    "@author = '", author, "' ;",
    sep = ""
  )
  query
}
### Serialize the model
modelbin = serialize(model, NULL)
modelbinstr = paste(modelbin, collapse = "")
### Excute the procedure and Store the model
execstr = Execute_Procedure(modelbinstr,"DECISION_TREE_MODEL",
                            "Classification", "Xente Fraud Detection",
                            1.0,"Holar"
                            )
##connect to your server and upload the model and test data
library(RODBC)
server <- "yourservername"
database<- "yourdatabase"
username <- "username"
password <- "password"
connectionString <- paste0("Driver={SQL Server};server=",server,";database="
,database,";trusted_connection=yes;")

channel <-  odbcDriverConnect(connection=connectionString,rows_at_time = 1)
## upload the model
upload = sqlQuery(channel,execstr, errors = TRUE,rows_at_time = 1)
### upload test data to SQL
sqlSave(channel,df_test,tablename = "xente_test_data", rownames = F)
```

Below is the result? In SQL Server I now have a tables called dbo.Xente_model_table and dbo.xente_test_data with the predictive model and the test data respectively.

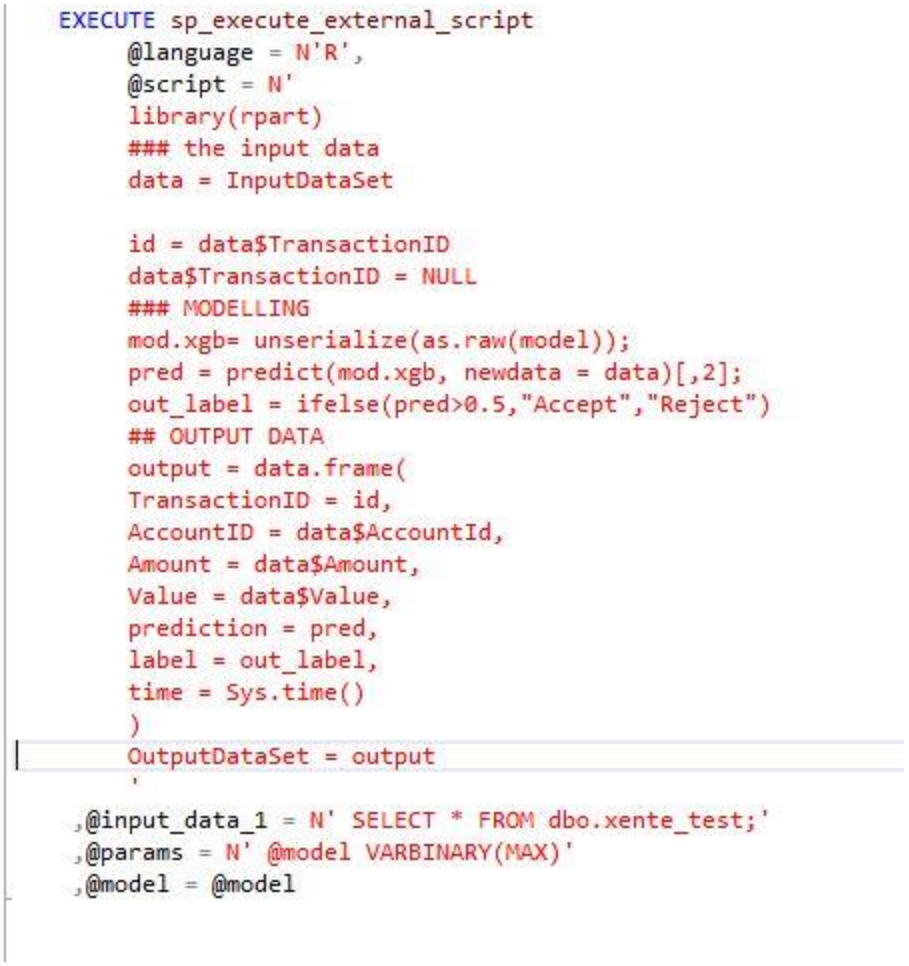


```sql
select * from dbo.Xente_model_table
select * from dbo.xente_test_data
```

	model	modelname	modeltype	modeldescription	modelversion	author	insertdate
1	0x580A000000020003040300020300000003130000000E00...	DECISION_TREE_MODEL	Classification	Xente Fraud Detection	1	Holar	2019-10-14 12:21:55.253

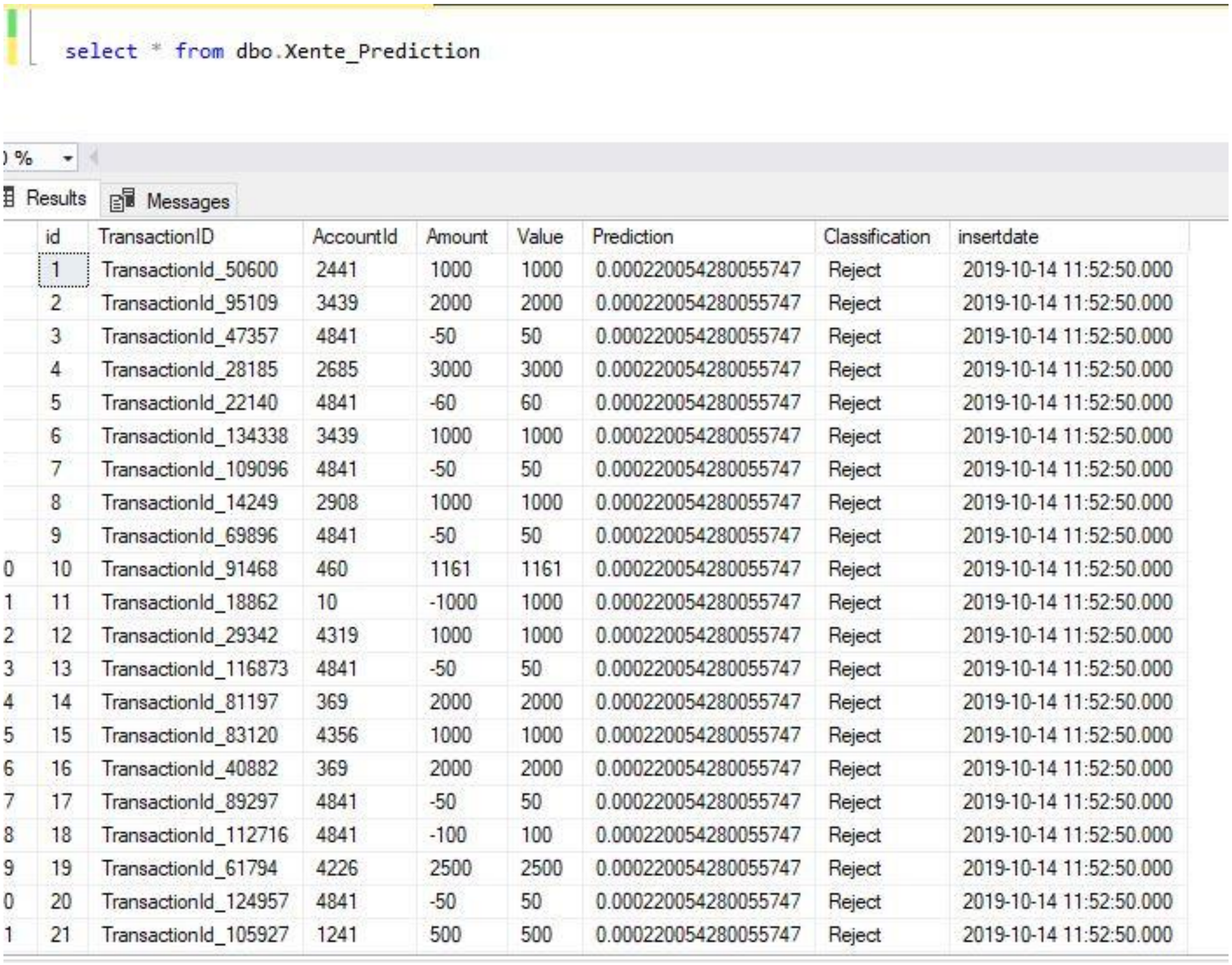
	AccountId	ProviderId	ProductId	ProductCategory	ChannelId	Amount	Value	PricingStrategy	Amount_type	Value_Amount_diff	hour	week_day	Value_cnt	Prov_ic
1	2441	5	3	1	3	1000	1000	4	2	0	10	4	27193	304
2	3439	5	15	3	3	2000	2000	2	2	0	10	4	7906	522
3	4841	4	6	3	2	-50	50	2	1	0	10	4	12493	2837
4	2685	5	10	1	3	3000	3000	4	2	0	10	4	1866	119
5	4841	4	6	3	2	-60	60	2	1	0	10	4	657	2837
6	3439	6	3	1	3	1000	1000	2	2	0	10	4	27193	1261
7	4841	4	6	3	2	-50	50	2	1	0	10	4	12493	2837
8	2908	3	3	1	3	1000	1000	1	2	0	10	4	27193	49
9	4841	4	6	3	2	-50	50	2	1	0	10	4	12493	2837
10	460	3	14	3	3	1161	1161	0	2	0	10	4	3	3

3. **Apply the Predictive model in SQL Server:** Now that we have a model stored on SQL Server, we can use it to calculate each transaction probability scores on the test data in SQL Server. With the new R services in SQL Server there is a function called **sp_exec_external_script** which you can use to call R and execute R scripts to calculate model scores.



```
EXECUTE sp_execute_external_script
     @language = N'R',
     @script = N'
     library(rpart)
     ### the input data
     data = InputDataSet

     id = data$TransactionID
     data$TransactionID = NULL
     ### MODELLING
     mod.xgb= unserialize(as.raw(model));
     pred = predict(mod.xgb, newdata = data)[,2];
     out_label = ifelse(pred>0.5,"Accept","Reject")
     ## OUTPUT DATA
     output = data.frame(
     TransactionID = id,
     AccountID = data$AccountId,
     Amount = data$Amount,
     Value = data$Value,
     prediction = pred,
     label = out_label,
     time = Sys.time()
     )
     OutputDataSet = output
     '
 ,@input_data_1 = N' SELECT * FROM dbo.xente_test;'
 ,@params = N' @model VARBINARY(MAX)'
 ,@model = @model
```

The code is very generic, instead of decision tree models it works for any model if you have the library installed on the SQL Server. The result and the final model output is shown below.



```
select * from dbo.Xente_Prediction
```

	id	TransactionID	AccountId	Amount	Value	Prediction	Classification	insertdate
	1	TransactionId_50600	2441	1000	1000	0.000220054280055747	Reject	2019-10-14 11:52:50.000
	2	TransactionId_95109	3439	2000	2000	0.000220054280055747	Reject	2019-10-14 11:52:50.000
	3	TransactionId_47357	4841	-50	50	0.000220054280055747	Reject	2019-10-14 11:52:50.000
	4	TransactionId_28185	2685	3000	3000	0.000220054280055747	Reject	2019-10-14 11:52:50.000
	5	TransactionId_22140	4841	-60	60	0.000220054280055747	Reject	2019-10-14 11:52:50.000
	6	TransactionId_134338	3439	1000	1000	0.000220054280055747	Reject	2019-10-14 11:52:50.000
	7	TransactionId_109096	4841	-50	50	0.000220054280055747	Reject	2019-10-14 11:52:50.000
	8	TransactionId_14249	2908	1000	1000	0.000220054280055747	Reject	2019-10-14 11:52:50.000
	9	TransactionId_69896	4841	-50	50	0.000220054280055747	Reject	2019-10-14 11:52:50.000
0	10	TransactionId_91468	460	1161	1161	0.000220054280055747	Reject	2019-10-14 11:52:50.000
1	11	TransactionId_18862	10	-1000	1000	0.000220054280055747	Reject	2019-10-14 11:52:50.000
2	12	TransactionId_29342	4319	1000	1000	0.000220054280055747	Reject	2019-10-14 11:52:50.000
3	13	TransactionId_116873	4841	-50	50	0.000220054280055747	Reject	2019-10-14 11:52:50.000
4	14	TransactionId_81197	369	2000	2000	0.000220054280055747	Reject	2019-10-14 11:52:50.000
5	15	TransactionId_83120	4356	1000	1000	0.000220054280055747	Reject	2019-10-14 11:52:50.000
6	16	TransactionId_40882	369	2000	2000	0.000220054280055747	Reject	2019-10-14 11:52:50.000
7	17	TransactionId_89297	4841	-50	50	0.000220054280055747	Reject	2019-10-14 11:52:50.000
8	18	TransactionId_112716	4841	-100	100	0.000220054280055747	Reject	2019-10-14 11:52:50.000
9	19	TransactionId_61794	4226	2500	2500	0.000220054280055747	Reject	2019-10-14 11:52:50.000
0	20	TransactionId_124957	4841	-50	50	0.000220054280055747	Reject	2019-10-14 11:52:50.000
1	21	TransactionId_105927	1241	500	500	0.000220054280055747	Reject	2019-10-14 11:52:50.000

Conclusion

Deploying predictive models that are created in R in SQL Server has become easy with the new SQL R services. It does not require new technology. If your company is already making use of SQL Server like mine, then integrated R services are something to look at if you want to deploy predictive models. The full code for this tutorial is on my github page.