World Academy of Science, Engineering and Technology
International Journal of Computer and Information Engineering
Vol:11, No:3, 2017

Open Science Index, Computer and Information Engineering Vol:11, No:3, 2017 publications.waset.org/10006640.pdf

Building a Scalable Telemetry Based Multiclass Predictive Maintenance Model in R

Jaya Mathew

***Abstract*—**Many organizations are faced with the challenge of how to analyze and build Machine Learning models using their sensitive telemetry data. In this paper, we discuss how users can leverage the power of R without having to move their big data around as well as a cloud based solution for organizations willing to host their data in the cloud. By using ScaleR technology to benefit from parallelization and remote computing or R Services on premise or in the cloud, users can leverage the power of R at scale without having to move their data around.

***Keywords*—**Predictive maintenance, machine learning, big data, cloud, on premise SQL, R.

I. Introduction

COMPANIES routinely own and operate various machinery as part of their daily business operations. During the lifecycle of these machines, they will inevitably run into some form of mechanical failures which leads to downtime in normal operations of the business. To ensure that these machines are highly reliable, businesses are now turning to Machine Learning (ML) techniques to help them predict these failures ahead of time so that they can proactively send their maintenance teams to address these issues [1].

II. Methodology

A. Data Collection

Machines are now equipped with sophisticated sensors that generate telemetry data. These sensors based on design can transmit data in real-time or in batches into a central data repository. Based on privacy policies within the company, the data are either stored on-premise or can be hosted in the public cloud.

Depending on the scale of their operations, some companies are now amassing large amount of data which could prove to be useful in predicting mechanical failures of their machinery.

B. Problem Formulation

Depending on the business need, the ML problem can be formulated in different ways. If the primary goal of the business is to determine if a machine is likely to fail or not, then the problem can be formulated as a simple binary classifier. In this case, the business will be able to get an overview of how many of their machines are functional at any given point in time. However, if the business is interested in whether a specific component in a machine is likely to fail, the business problem can be reformulated as a multi-class classifier. In this formulation, the business is able to plan on scheduling time with their maintenance teams to address the issue and possibly pre-order spare parts needed for the repair, ahead of time so as to minimize delays due to a possible downtime.

In the rest of the paper, the steps taken from feature engineering to building a model for the multi-class classifier business use case are outlined. The process essentially is the same for building a binary classifier with a minor change in the way the labels are generated.

C. Feature Engineering Process

The raw data that get collected from various inbuilt machine sensors are not very useful except when used in monitoring dashboards. These dashboards however do not help answer business questions like ‘when is a machine likely to fail due to a certain component?’ or ‘when does the maintenance team need to order specific spare parts?’ To build an ML model to address these business questions, data need to be collated from various available sources (purchase order or invoice, telemetry, maintenance records, failures etc.) and additional feature engineering needs to be done to best describe the health of the machine at a given point in time. Static machine attribute features like make, model, type can be easily acquired from purchase orders or invoices. Additionally, date of purchase can be used to determine the age of the machine at any given point in time. Routine or ad hoc maintenance records can be used to determine when a repair or replacement was done for a specific component within a machine. Most of these static machine features and maintenance data can be used as is, with minimal additional feature engineering. However, sensors typically generate data at a regular interval in time, thus the data typically consists of multiple time series. Usually these series of data show a lot of fluctuation and volatility, hence rolling/tumbling aggregates [13] of these sensor values with varying window sizes (width= 6, 12, 24, etc.) are generated to create additional features that are less volatile in nature. To determine the optimal window size, it is recommended that the data scientist consult with a business domain expert.

After the data collation and feature engineering exercise, the resultant dataset typically consists of many features (over hundreds depending on the number of sensors). Parsimonious models are always preferred over models with all possible features. The rule of thumb is to retain only the top 50-100 of the most relevant features.

The next step is to build a label for the ML supervised learner; this information is usually retrieved from

Jaya Mathew is with Microsoft, 1 Memorial Drive, Suite 100, Cambridge, MA 02142, USA (phone: (857) 453-6000; fax: (857) 453-6013; e-mail: jaymathe@microsoft.com).

maintenance/repair logs which record an actual failure event for the machine due to a specific component. In the event, the business is only interested in a binary classifier, the label takes only 2 values, no-failure (0) and failure (1), whereas if the business is interested in a multi-class classifier, the label can take the values, no-failure (0) and the other labels indicating failure due to other components. Typically, most models do not perform very well when the business need is to predict a failure weeks/months ahead of time.

D. Modeling: Training, Validation and Evaluation

After generating the final dataset with the labels for the supervised learner, the data are now ready for building the ML model. In this paper, the assumption is that the business is interested in predicting which component of their machinery is likely to fail; hence, a multi-class classifier is built. This problem can be easily restated as a two-class problem as well if the business is interested in only predicting whether a failure occurs irrespective of the components. In a two-class problem, the ML model will only be able to determine whether a machine is likely to fail or is not likely to run into a failure in the near future.

In the sample use case, the data need to be first sorted and then split based on time into training and testing datasets. The training data are used to build and tune the performance of the classifier prior to evaluating the multi-class classifier on the testing dataset which was withheld from the entire training process. An alternate approach would be to split the machines into training/testing datasets, where the machines that are used to build the trained model are not used for testing the model. In such cases, the team should be certain that all the machines are identical in nature and are likely to function in a similar manner.

Based on the nature of the dataset, the optimal classifier can vary. It is recommended to build multiple classifiers using Decision Forest, Decision Jungle, Logistic Regression, Neural Network algorithms and then evaluate each of these models on the withheld testing dataset. Multiple classifiers can be easily tested within Microsoft Azure ML Studio's drag and drop environment as shown in Fig. 8 [6]. Often, this ends up being an iterative process where the model is tuned and evaluated multiple times before the business finds a satisfactory model based on relevant evaluation metrics like precision, recall, F-1 score or accuracy.

The methodology so far can be summarized as shown in Fig. 1.

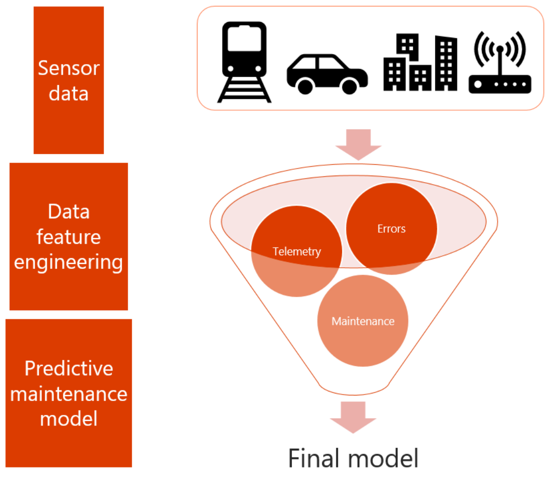


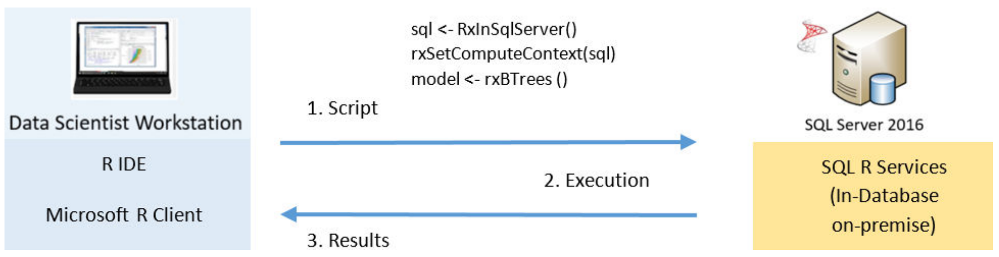
Fig. 1 Methodology

E. Operationalization

After the model is finalized, it is vital that the business can use this model in their production environment on a regular basis to determine whether a machine is likely to fail in the near future. Typically, most models do not get operationalized due to the difficulty in integrating it with the existing productional environment. However, with Microsoft's Cortana Intelligence suite of products [4], [6], [7], these ML models can be easily converted into a web service based on whether it is an on-premise (Fig. 2) or a cloud based solution (Figs. 3, 4) often with a single click deployment. Once these models are deployed, they can be operationalized into existing production systems. Such an end to end system can help streamline and prioritize the maintenance schedule for their machines and reduce the downtime to normal business operations.

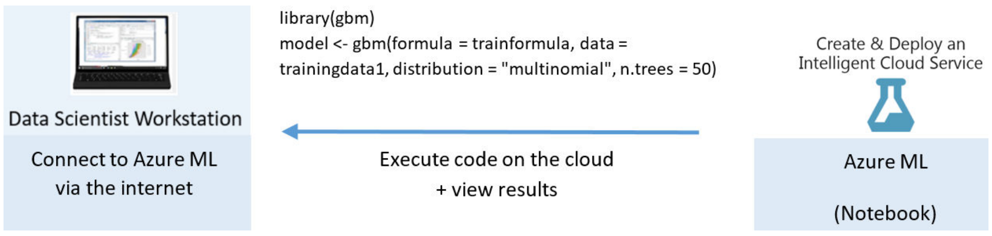
III. Scaling the Solution

The final solution options vary depending on size of the business data and the scale of their operations as well as their data storage and privacy policies. This section addresses the common issues that need to be considered before picking either an on-premise solution as shown in Fig. 2 [2], [4] or build a cloud based solution as shown in Figs. 3 [3] or 4. The solution framework described below gives broad guidelines to small, medium/large business owners as they determine their future business road map.



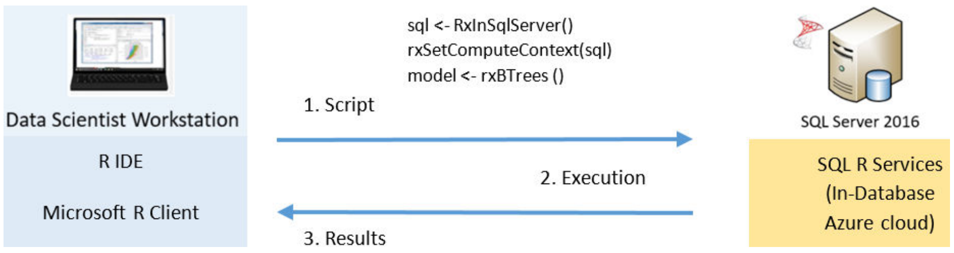
```
sql <- RxInSqlServer()
rxSetComputeContext(sql)
model <- rxBTrees ()
```

Fig. 2 On premise solution framework



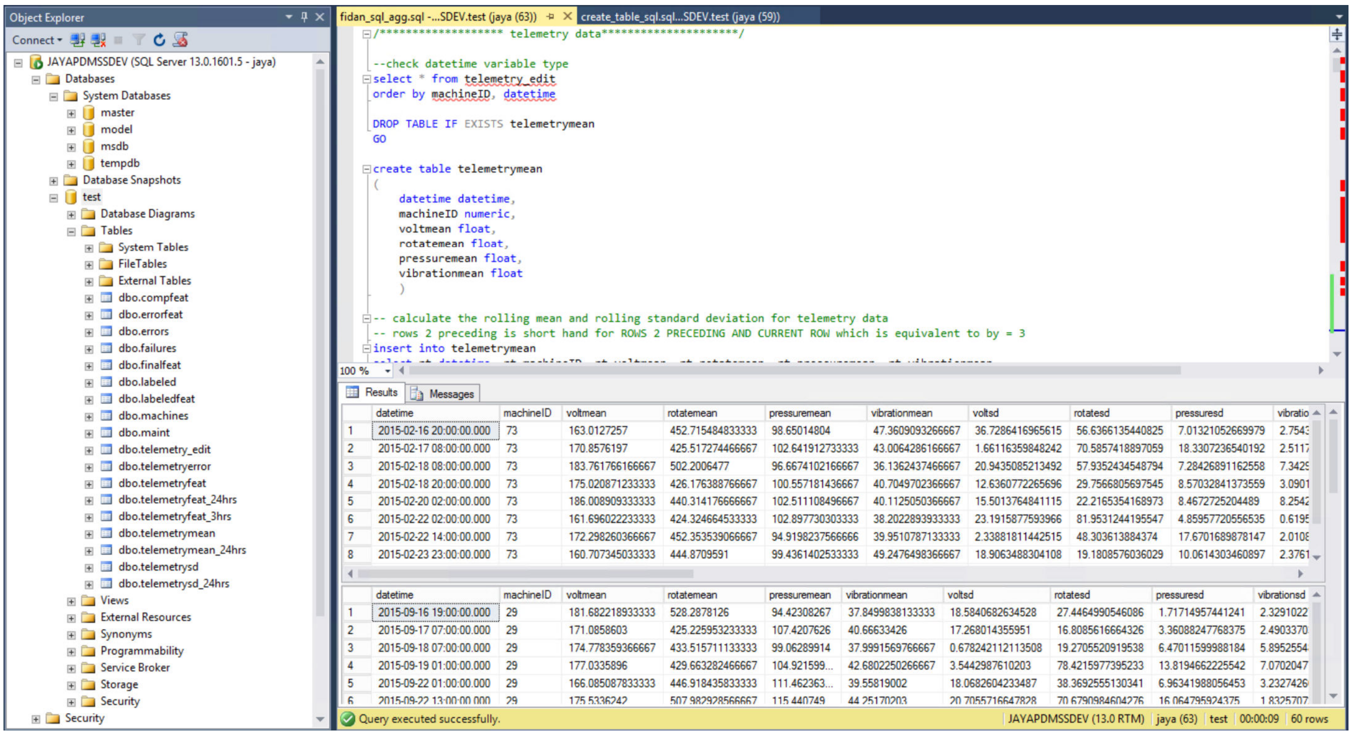
```
library(gbm)
model <- gbm(formula = trainformula, data =
trainingdata1, distribution = "multinomial", n.trees = 50)
```

Fig. 3 Cloud based solution framework-1



```
sql <- RxInSqlServer()
rxSetComputeContext(sql)
model <- rxBTrees ()
```

Fig. 4 Cloud based solution framework-2



```
JAYAPDMSSDEV (SQL Server 13.0.1601.5 - jaya)
  Databases
    System Databases
      master
      model
      msdb
      tempdb
    Database Snapshots
    test
      Database Diagrams
      Tables
        System Tables
        FileTables
        External Tables
        dbo.compfeat
        dbo.errorfeat
        dbo.errors
        dbo.failures
        dbo.finalfeat
        dbo.labeled
        dbo.labeledfeat
        dbo.machines
        dbo.maint
        dbo.telemetry_edit
        dbo.telemetryerror
        dbo.telemetryfeat
        dbo.telemetryfeat_24hrs
        dbo.telemetryfeat_3hrs
        dbo.telemetrymean
        dbo.telemetrymean_24hrs
        dbo.telemetrysd
        dbo.telemetrysd_24hrs
      Views
      External Resources
      Synonyms
      Programmability
      Service Broker
      Storage
      Security
  Security
```

```
/****************** telemetry data********************/

--check datetime variable type
select * from telemetry_edit
order by machineID, datetime

DROP TABLE IF EXISTS telemetrymean
GO

create table telemetrymean
(
    datetime datetime,
    machineID numeric,
    voltmean float,
    rotatemean float,
    pressuremean float,
    vibrationmean float
    )

-- calculate the rolling mean and rolling standard deviation for telemetry data
-- rows 2 preceding is short hand for ROWS 2 PRECEDING AND CURRENT ROW which is equivalent to by = 3
insert into telemetrymean
```

	datetime	machineID	voltmean	rotatemean	pressuremean	vibrationmean	voltsd	rotatesd	pressuresd	vibratio
1	2015-02-16 20:00:00.000	73	163.0127257	452.715484833333	98.65014804	47.3609093266667	36.7286416965615	56.6366135440825	7.01321052669979	2.7543
2	2015-02-17 08:00:00.000	73	170.8576197	425.517274466667	102.641912733333	43.0064286166667	1.66116359848242	70.5857418897059	18.3307236540192	2.5117
3	2015-02-18 08:00:00.000	73	183.761766166667	502.2006477	96.6674102166667	36.1362437466667	20.9435085213492	57.9352434548794	7.28426891162558	7.3429
4	2015-02-18 20:00:00.000	73	175.020871233333	426.176388766667	100.557181436667	40.7049702366667	12.6360772265696	29.7566805697545	8.57032841373559	3.0901
5	2015-02-20 02:00:00.000	73	186.008909333333	440.314176666667	102.511108496667	40.1125050366667	15.5013764841115	22.2165354168973	8.4672725204489	8.2542
6	2015-02-22 02:00:00.000	73	161.696022233333	424.324664533333	102.897730303333	38.2022893933333	23.1915877593966	81.9531244195547	4.85957720556535	0.6195
7	2015-02-22 14:00:00.000	73	172.298260366667	452.353539066667	94.9198237566666	39.9510787133333	2.33881811442515	48.303613884374	17.6701689878147	2.0108
8	2015-02-23 23:00:00.000	73	160.707345033333	444.8709591	99.4361402533333	49.2476498366667	18.9063488304108	19.1808576036029	10.0614303460897	2.3761

	datetime	machineID	voltmean	rotatemean	pressuremean	vibrationmean	voltsd	rotatesd	pressuresd	vibrationsd
1	2015-09-16 19:00:00.000	29	181.682218933333	528.2878126	94.42308267	37.8499838133333	18.5840682634528	27.4464990546086	1.71714957441241	2.3291022
2	2015-09-17 07:00:00.000	29	171.0858603	425.225953233333	107.4207626	40.66633426	17.268014355951	16.8085616664326	3.36088247768375	2.4903370
3	2015-09-18 07:00:00.000	29	174.778359366667	433.515711133333	99.06289914	37.9991569766667	0.678242112113508	19.2705520919538	6.47011599988184	5.8952554
4	2015-09-19 01:00:00.000	29	177.0335896	429.663282466667	104.921599...	42.6802250266667	3.5442987610203	78.4215977395233	13.8194662225542	7.0702047
5	2015-09-22 01:00:00.000	29	166.085087833333	446.918435833333	111.462363...	39.55819002	18.0682604233487	38.3692555130341	6.96341988056453	3.2327426
6	2015-09-22 13:00:00.000	29	175.5336242	507.982928566667	115.440749	44.25170203	20.7055716647828	70.6790984604276	16.064795924375	1.8325707

```
Query executed successfully.    JAYAPDMSSDEV (13.0 RTM) | jaya (63) | test | 00:00:09 | 60 rows
```

Object Explorer
Connect
fidan_sql_agg.sql -...SDEV.test (jaya (63))
create_table_sql.sql...SDEV.test (jaya (59))
100 %
Results
Messages

Fig. 5 Data processing in SQL

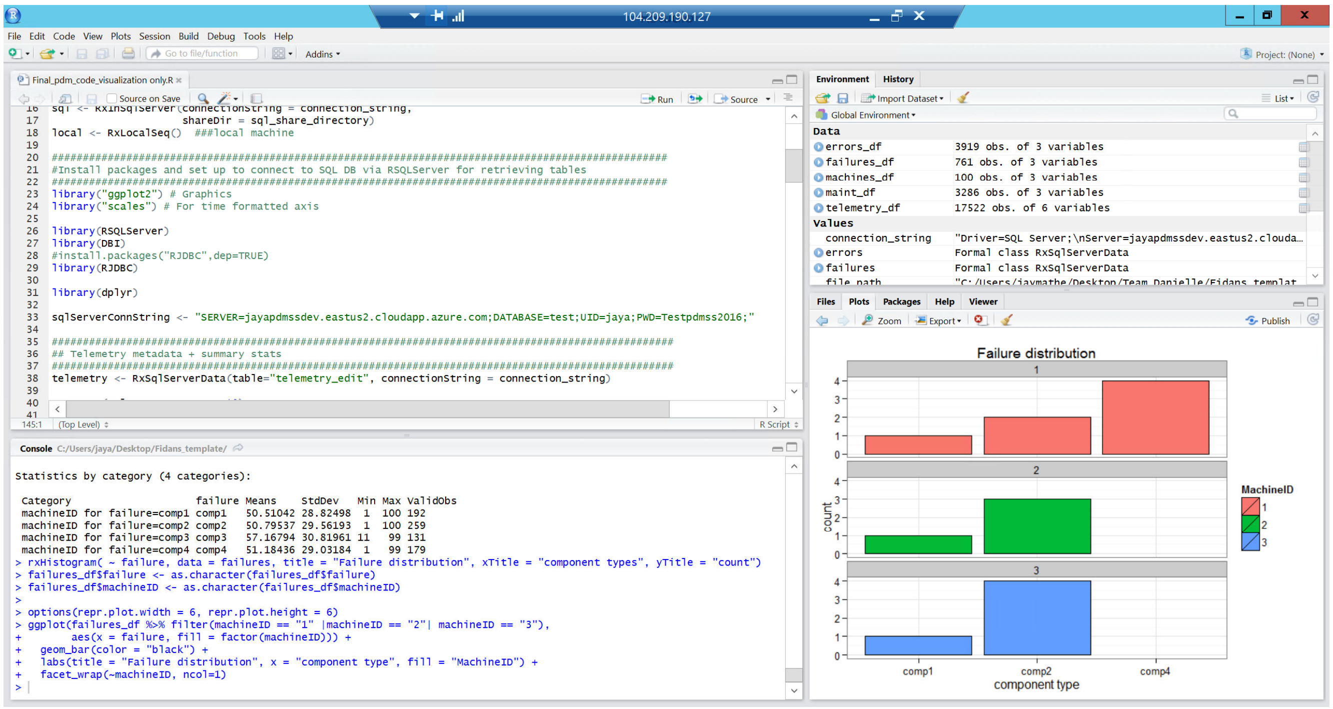
The feature engineering and model building code is available in R at the GitHub repository [4] along with a sample dataset. R [5] is a popular open source statistical programming language that is frequently used for data manipulation, feature engineering and building customized ML models. However, open source R is often constrained by the amount of data it can hold in memory as well as expensive data movement. This is where Microsoft's R can help overcome this limitation by using various scalable parallel processing algorithms or by using R within SQL Server [12]. Microsoft's R [11] functions typically have a prefix 'rx' to help distinguish them from native R functions.

In the next section, a few scenarios are discussed in more detail to help address the business needs of a small/medium/ large business with varying data requirements.

A. Scenario: Small Business – On Premise Solution

Consider a small business who currently stores their entire data on-premise in either CSV or any other text file format, a possible solution for such a business would be to use the R based solution using any R IDE (integrated data environment) on-premise. However, as their data size grows over time, they would need to consider alternative options to store and organize their data along with their data security and privacy team to determine whether they would need to continue to store their data on premise as text files or if they prefer to move their data onto any cloud. If the business prefers to store their data on premise, it is advisable to start organizing their data into a relational database like SQL. In this scenario, as shown in Fig. 2, the data can be stored in SQL Server on-premise. In such a setup, it is optimal to do most of the feature engineering directly on SQL Server as shown using SQL queries in Fig. 5.

After feature engineering, then the model can be built in-database with the scale R codes in SQL context with any R IDE environment like R Studio as shown in Fig. 6. Hence the models are not constrained by available local memory limits.



```
sql <- RxInSqlServer(connectionString = connection_string,
                     shareDir = sql_share_directory)
local <- RxLocalSeq()  ###local machine

##########################################################################################
#Install packages and set up to connect to SQL DB via RSQLServer for retrieving tables
##########################################################################################
library("ggplot2") # Graphics
library("scales") # For time formatted axis

library(RSQLServer)
library(DBI)
#install.packages("RJDBC",dep=TRUE)
library(RJDBC)

library(dplyr)

sqlServerConnString <- "SERVER=jayapdmssdev.eastus2.cloudapp.azure.com;DATABASE=test;UID=jaya;PWD=Testpdmss2016;"

##########################################################################################
## Telemetry metadata + summary stats
##########################################################################################
telemetry <- RxSqlServerData(table="telemetry_edit", connectionString = connection_string)
```

Console C:/Users/jaya/Desktop/Fidans_template/

```
Statistics by category (4 categories):

 Category                     failure Means    StdDev   Min Max ValidObs
 machineID for failure=comp1 comp1   50.51042 28.82498  1  100 192
 machineID for failure=comp2 comp2   50.79537 29.56193  1  100 259
 machineID for failure=comp3 comp3   57.16794 30.81961 11   99 131
 machineID for failure=comp4 comp4   51.18436 29.03184  1   99 179
> rxHistogram( ~ failure, data = failures, title = "Failure distribution", xTitle = "component types", yTitle = "count")
> failures_df$failure <- as.character(failures_df$failure)
> failures_df$machineID <- as.character(failures_df$machineID)
>
> options(repr.plot.width = 6, repr.plot.height = 6)
> ggplot(failures_df %>% filter(machineID == "1" |machineID == "2"| machineID == "3"),
+        aes(x = failure, fill = factor(machineID))) +
+   geom_bar(color = "black") +
+   labs(title = "Failure distribution", x = "component type", fill = "MachineID") +
+   facet_wrap(~machineID, ncol=1)
> |
```

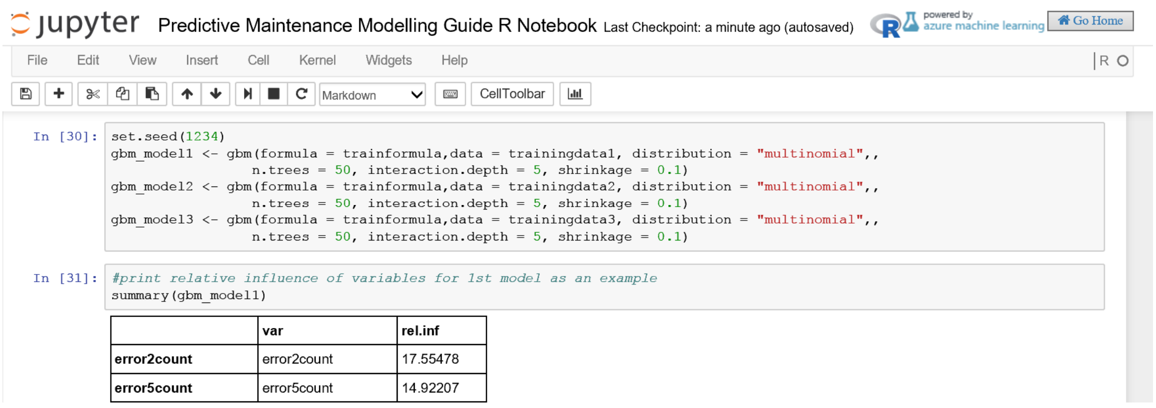
```
Data
errors_df               3919 obs. of 3 variables
failures_df             761 obs. of 3 variables
machines_df             100 obs. of 3 variables
maint_df                3286 obs. of 3 variables
telemetry_df            17522 obs. of 6 variables
Values
  connection_string     "Driver=SQL Server;\nServer=jayapdmssdev.eastus2.clouda…
errors                  Formal class RxSqlServerData
failures                Formal class RxSqlServerData
  file_path             "C:/Users/jaymathe/Desktop/Team_Danielle/Fidans_templat
```

Fig. 6 Using an R IDE to access SQL Server

After the model is built and finalized, it can be operationalized [8] by creating a T-SQL stored procedure which can be invoked on a pre-defined schedule.

B. Scenario: Small Business – Cloud Based Solution

If the small business however decides to avoid all the costs incurred in maintaining an on-premise data repository, an option would be to move their data to a public cloud. In this scenario, as shown in Fig. 3, the data are available on the cloud and by using Microsoft's Azure ML [6], [7] notebook environment, the feature engineering and modeling can be done via a jupyter notebook as shown in Fig. 7. Once the model is deployed as a web service, it can be invoked from any web application. If on the other hand, the business prefers to continue to use a SQL Server, now the data can be stored in SQL Server on the cloud [9] instead and the model can be built in-database with scale R code in SQL context as recommended in the cloud-based solution shown in Fig. 4.


jupyter
Predictive Maintenance Modelling Guide R Notebook Last Checkpoint: a minute ago (autosaved)
powered by azure machine learning
Go Home
File
Edit
View
Insert
Cell
Kernel
Widgets
Help
R
Markdown
CellToolbar

```
In [30]: set.seed(1234)
         gbm_model1 <- gbm(formula = trainformula,data = trainingdata1, distribution = "multinomial",,
                           n.trees = 50, interaction.depth = 5, shrinkage = 0.1)
         gbm_model2 <- gbm(formula = trainformula,data = trainingdata2, distribution = "multinomial",,
                           n.trees = 50, interaction.depth = 5, shrinkage = 0.1)
         gbm_model3 <- gbm(formula = trainformula,data = trainingdata3, distribution = "multinomial",,
                           n.trees = 50, interaction.depth = 5, shrinkage = 0.1)

In [31]: #print relative influence of variables for 1st model as an example
         summary(gbm_model1)
```

	var	rel.inf
error2count	error2count	17.55478
error5count	error5count	14.92207

Fig. 7 Using jupyter notebook in Azure

C. Scenario: Medium/Large Business – On Premise Solution

Consider a medium/large business with large scale operations but has very stringent data privacy guidelines which prevents the business from migrating any of their data to any cloud. In this scenario, it is advisable that the business organizes their data into a relational database like a SQL Server. Then as shown in Fig. 2, the data can be systematically organized in SQL Server where all the feature engineering and the model can be built in-database with the scale R code in SQL context. Here the models are not constrained by any available local memory limits. In the end, after the model is built, it can be operationalized [8] by creating a T-SQL stored procedure which can be invoked on a predefined schedule.

D. Scenario: Medium/Large Business – Cloud Based Solution

If the medium/large business however decides to avoid all the costs and hassle involved in maintaining their data on-premise, an option would be to move their data to a public cloud. Along with data storage, cloud providers like Microsoft Azure [10], offers additional services to manage the data in a database as well as use additional tools to build models on the cloud. In this scenario, as shown in Fig. 4, the data can be stored in SQL Server in the cloud [9] and the model can be built in-database with the same scale R code in SQL context or the model can be built using Azure ML as shown in Fig. 3. Hence, the models are not constrained by any available local memory limits. Another option would be to host the data on the cloud in any other storage option and then read the data into the Azure ML Studio environment and then build additional features, create the label, build the model and operationalize it via the easy to use drag and drop environment as shown in Fig. 8.

Once the model is deployed as a web service, it can be invoked from any web application or can be invoked using a T-SQL script on a predefined schedule.

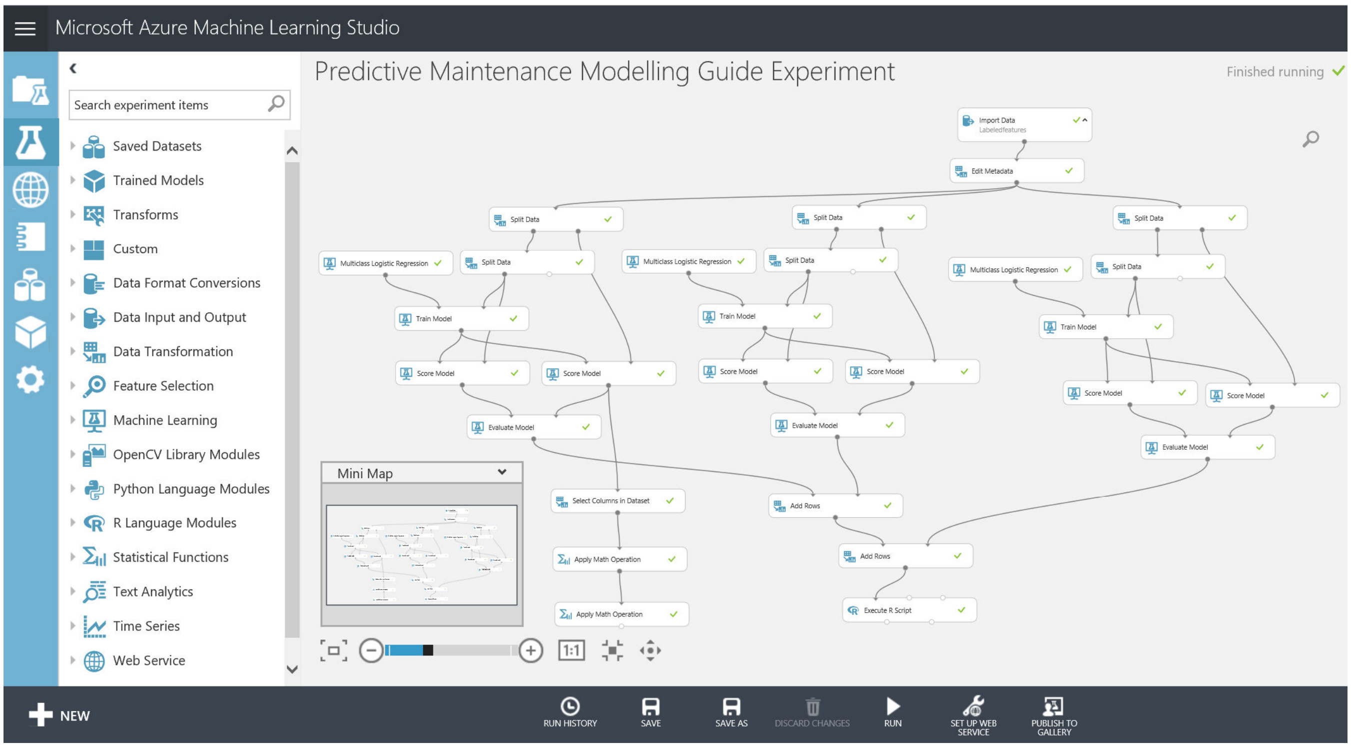


Fig. 8 Azure ML Studio environment

Size / Privacy	Small business	Medium/Large business
Strict with no option to store the data in the cloud	On premise	On premise
Liberal data storage options	On premise or Cloud based	On premise or Cloud based

Fig. 9 Guidelines to picking a solution

E. Recommendation

In summary, the various options discussed can be tabulated as shown in Fig. 9 with the options based on the size and privacy concerns of the business.

IV. Conclusion

At the end of the paper, the user would be able to formulate their business problem, build features and the dataset for building a relevant classifier to solve their data science problems when working with their data.

1) This use case will help users who have small on-premise data and some expertise in R build a basic ML predictive maintenance model.
2) This use case will also help users scale their on-premise solutions using sensitive big data using R.
3) This use case also gives an overview of how the same solution can be ported over as a cloud based solution with minor edits in the R code.
4) The sample code in R that is available on GitHub can be a good starting point for more advanced business specific use cases.
5) This use case also gives recommendations on how to pick a solution for a business.

ACKNOWLEDGMENT

The author would like to acknowledge and thank Danielle Dean from the Microsoft Analytics and Data Science team for her input and guidance.

REFERENCES

[1] http://www.forbes.com/sites/louiscolumbus/2016/12/03/industrial-analytics-based-on-internet-of-things-will-revolutionize-manufacturing/#689fc90149ac (Accessed on 10/02/2017)
[2] http://blog.revolutionanalytics.com/2016/09/r-services-maintenance.html (Accessed on 10/02/2017)
[3] https://gallery.cortanaintelligence.com/Notebook/Predictive-Maintenance-Modelling-Guide-R-Notebook-1 (Accessed on 10/02/2017)
[4] https://gallery.cortanaintelligence.com/Tutorial/Predictive-Maintenance-Modeling-Guide-using-SQL-R-Services-1 (Accessed on 10/02/2017)
[5] https://www.r-project.org/about.html (Accessed on 10/02/2017)
[6] https://studio.azureml.net/ (Accessed on 10/02/2017)
[7] https://notebooks.azure.com/ (Accessed on 10/02/2017)
[8] https://msdn.microsoft.com/en-us/library/mt590864.aspx (Accessed on 10/02/2017)
[9] https://azure.microsoft.com/en-us/services/sql-database/ (Accessed on 10/02/2017)
[10] https://azure.microsoft.com/en-us/services/cloud-services/ (Accessed on 10/02/2017)
[11] https://msdn.microsoft.com/en-us/library/mt652103.aspx (Accessed on 10/02/2017)
[12] https://blogs.msdn.microsoft.com/microsoft_press/2016/10/19/free-ebook-data-science-with-microsoft-sql-server-2016/ (Accessed on 10/02/2017)
[13] https://docs.microsoft.com/en-us/azure/machine-learning/cortana-analytics-playbook-predictive-maintenance (Accessed on 10/02/2017)