A Stateless, Query Based API for Model Risk Management

Patrick Harned

3/7/2022

The following document is an example of a stateless, streaming, query based API for model risk management. It is a proof of concept application designed to illustrate the feasibility of implementing MRM algorithms in-database. It has the following capabilities.

1. Disparate Impact Ratio evaluations for monitoring deployed ML models for bias (otherwise known as fairness)
2. Generation of local and global surrogate models for explaining predictions of black box models (explainability)
3. Accuracy Drift - Analyze and track the change in probability distribution of model features with respect to model prediction
4. Data Drift - Analyze and track the change in univariate distribution of model features

The Application operates on numerically encoded model transactions stored in a DB2 database. It can be extended to analyze transactions stored in other database by leveraging IBM data fabric offerings like Data Virtualizaton and Big SQL, or simple data federation and performatively analyzes model transactions and offer useful insights on the smallest possible amount of data, without requiring external dependencies or API calls.

Stateless

The Application is stateless because it requires no user configuration or any external dependencies other than a database connection and data to operate on. This means that The Application can provide useful insights into the behavior of a deployed ML model without access to training data or external model APIS.

Query-based

The Application is query-based because it pushes all computations, including ML modeling, into the database. The Application is thus a horizontally and vertically scalable solution to MRM, and can operate on massive amounts of data, since IBM DB2 and compatible products are horizontally and vertically scalable through SMP and MMP deployments, and are proven capable of processing extremely high volume workloads.

Streaming

The Application is streaming because it requires no blocking ETL in-memory transformations on data in order to provide analytics results. It is written entirely in Scala, and has only two external dependencies: Akka HTTP which provides the web server for the API, and the necessary database drivers from IBM.

The remainder of this document contains descriptions of the implementations of the various features as well as example requests and responses from the API, including time taken to complete the full request to give insight into the performance of the API. Note that as stated above, besides a table in a database containing the features and predictions of a machine learning model there are no other requirements to produce analytics results in this document.

Disparate Impact Ratio

Disparate Impact Ratio is the differential ratio of model outcomes with respect to membership in some protected class.

$$DI = P(y|k)$$

Where

$$k$$

is the probability of membership in a protected class, and

$$y$$

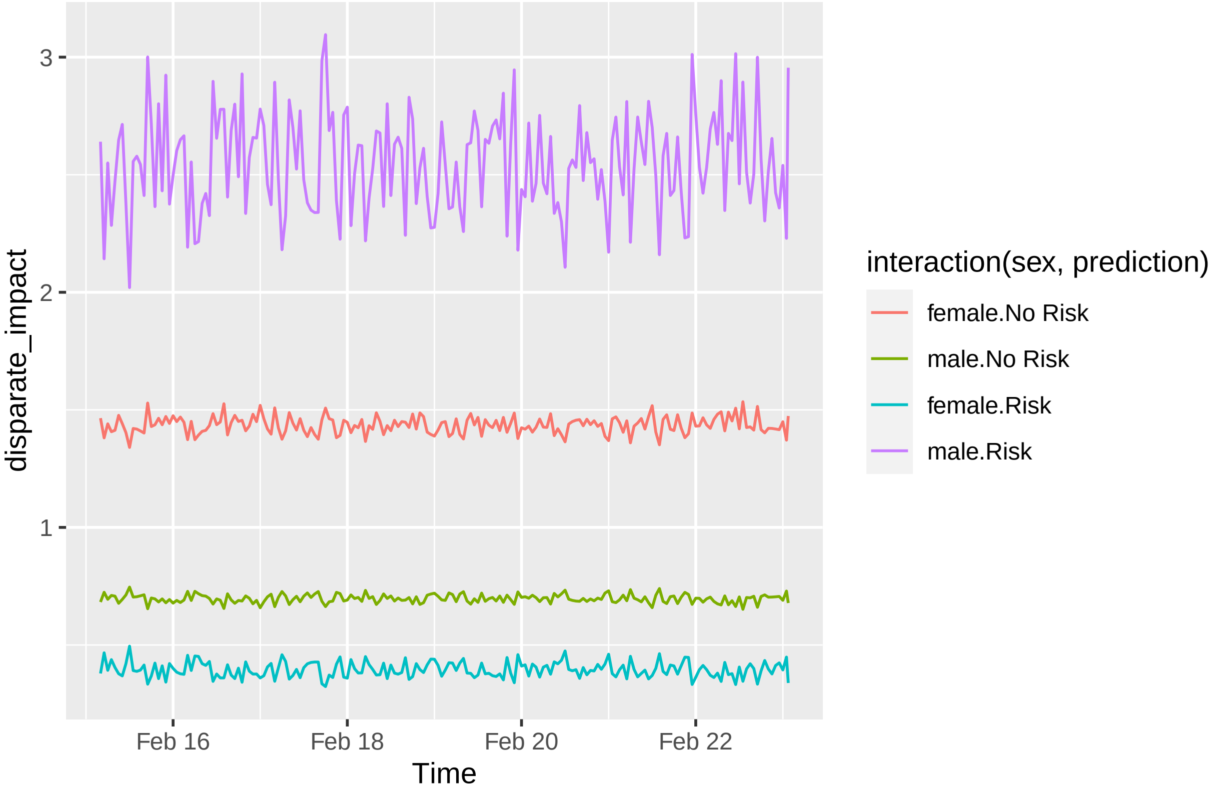
is the probability of the model outcome.

The application will calculate the respective impact ratios for any combination of outcomes and group membership in the data, given a prediction column and a column encoding the protected attribute (such as race or sex). Therefore, The application does not need to be configured to know which outcomes are considered favorable and which groups are considered protected.

```
url = "http://localhost:8080/api/disparate_impact"
body = '{"prediction":"risk", "table":"test_data2", "protected_column":"sex", "scoring_timestamp":"times
res = POST(url = url, content_type_json(), body = body)
data = content(res)

df = tibble(prediction = lapply(data, `[[`, 1), sex = lapply(data, `[[`, 2 ),group = lapply(data, `[[`,
df = as_tibble(lapply(df, unlist))%>% mutate(time = as.POSIXct(time))
df %>% ggplot(aes(x = time, group = interaction(sex, prediction), colour = interaction(sex,prediction),
```

FIG 1: Difference in Outcomes for Males versus Females



Time to execute this request 3.88958978652954 seconds

FIG1 Contains the result of an API call to an The application instances which shows the rates in which Males and Females experienced different outcomes of the mode over a week long period. The purple line is the rate at which men were assigned to the category of “Risk” by the model, while the blue line indicates the rate at which Females received the same assignation. Conversely, The red line indicates the rate at which Females were assigned to the “No Risk” category relative to all other groups, and the green line indicates the same for Men.

```
WITH t1 as (
SELECT
risk as prediction,
max(timestamp) OVER( PARTITION BY hour(timestamp),day(timestamp) ) as time,
sex FROM test_data2) ,
t2 as (
  SELECT  prediction,
          COUNT(sex) as group,
          sex,
          time
          FROM t1
          GROUP BY sex ,
          prediction ,
          time) ,
t3 as (
      SELECT
      prediction,
      sex,
      group as group,
      time,
      (cast(group as float)/sum(group) OVER( PARTITION BY sex,time )) as ratios
      FROM t2) SELECT prediction,sex,
      group as group,
      case when   (sum(ratios) OVER( PARTITION BY prediction,time )-ratios) = 0 then 1 else (cast(ratios
      end as disparate_impact,time FROM t3
```

Explainability

We model an explanation as the prediction of a interpretable machine learning model trained on the model feature set to predict the black box model’s outcome. Because the transactions produced by the machine learning model provide us with a training data set and predictions produced by the black box model whose behavior we are trying to infer, we do not require access to the model itself in order to predict or explain the models outcome. This approach should be contrasted with the LIME algorithm, which builds a perturbed dataset from the original model training distribution and then fits a model from the perturbed training dataset onto the black box models prediction.

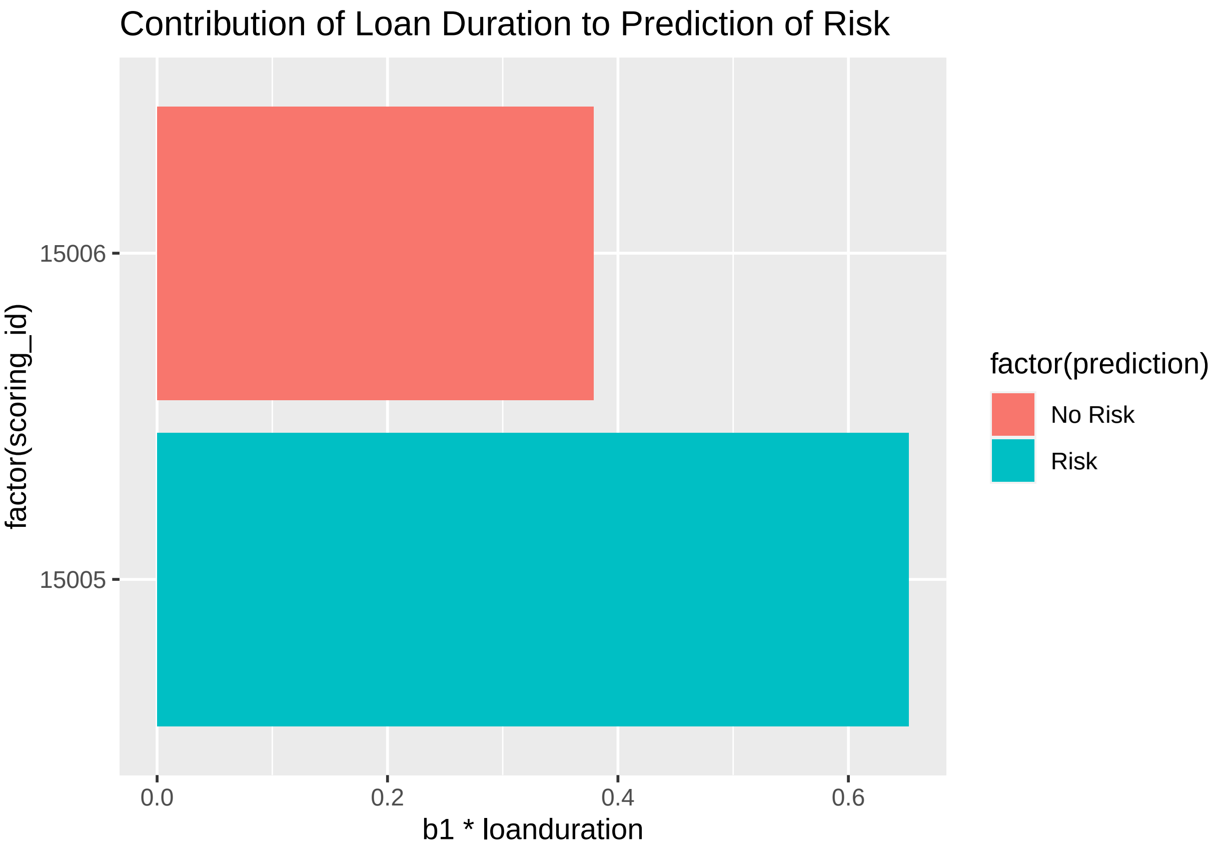
Since LIME builds a perturbed dataset in memory, instead of using transactional data for which labels are available, it requires access to the model itself in order to explain the model since the perturbed dataset must be scored in order to build the local linear model. These steps are simplified in order to remove the requirement of access to a black box model. Since no IO is required, The Application builds the model in the database and returns the coefficients of a local linear model and all transactions for which an explanation is requested, meaning it can efficiently generate local linear models to predict the black box models behavior at scale.

You can see in the API request we can control the features with which to build the model, the number of iterations used to fit the model using the gradient descent algorithm, the step size or learning rate, the target on which to train the model as well as a list of scoring ids for which we desire a prediction.

```
url = "http://localhost:8080/api/explainability"
body = '{"table_name":"scored_credit", "target":"prediction", "features":["loanduration"], "id_column":
res = POST(url = url, content_type_json(), body = body)
data = lapply(content(res), `[[`, 1)

df = tibble(scoring_id = lapply(data, `[[`, 1), b1 = lapply(data, `[[`, 2 ),mse = lapply(data, `[[`, 3
df = as_tibble(lapply(df, unlist))%>%mutate(prediction = case_when(round(b1*loanduration)==0 ~"No Risk"

df %>% ggplot(aes(x =factor(scoring_id), y = b1*loanduration, fill= factor(prediction))) + geom_bar(pos
```



Time to execute this request 3.38150691986084 seconds

Query Example

```
WITH model as (WITH
 TRAINING AS (SELECT t.*, t.prediction as target, ROW_NUMBER() OVER() AS ROW FROM scored_credit  AS T o
RATES (LEARN_RATE, COUNT) AS (SELECT .001 AS LEARN_RATE, (SELECT COUNT(*) FROM TRAINING) AS COUNT FROM
LEARNING (ITERATION, B1, INTERCEPT, MSE, M1, C) AS (SELECT 1, CAST(0.0 AS DOUBLE),CAST(0.0 AS DOUBLE),C
 UNION ALL SELECT A.ITERATION +1, A.M1, A.C, T.MSE, T.M1, T.C FROM LEARNING A,
TABLE
 (
 SELECT A.ITERATION,
AVG(g.differences*g.differences) AS MSE,
A.M1-(SUM(G.DIFFERENCES*G.loanduration)/(SELECT COUNT FROM RATES))*(SELECT LEARN_RATE from RATES) AS M1
A.C-(SUM(g.differences)/(SELECT COUNT from RATES))*(SELECT LEARN_RATE FROM RATES) AS C
 FROM
TABLE ( SELECT
((((Z.loanduration*A.B1))+A.INTERCEPT)-z.target) as differences,
  z.loanduration
, z.target
 from training as z
)
as g) t
 WHERE A.ITERATION <= 10000)
  SELECT * FROM LEARNING
 WHERE ITERATION >1 ORDER BY MSE LIMIT 1 ),result as (SELECT loanduration,scoring_id FROM scored_credit
```

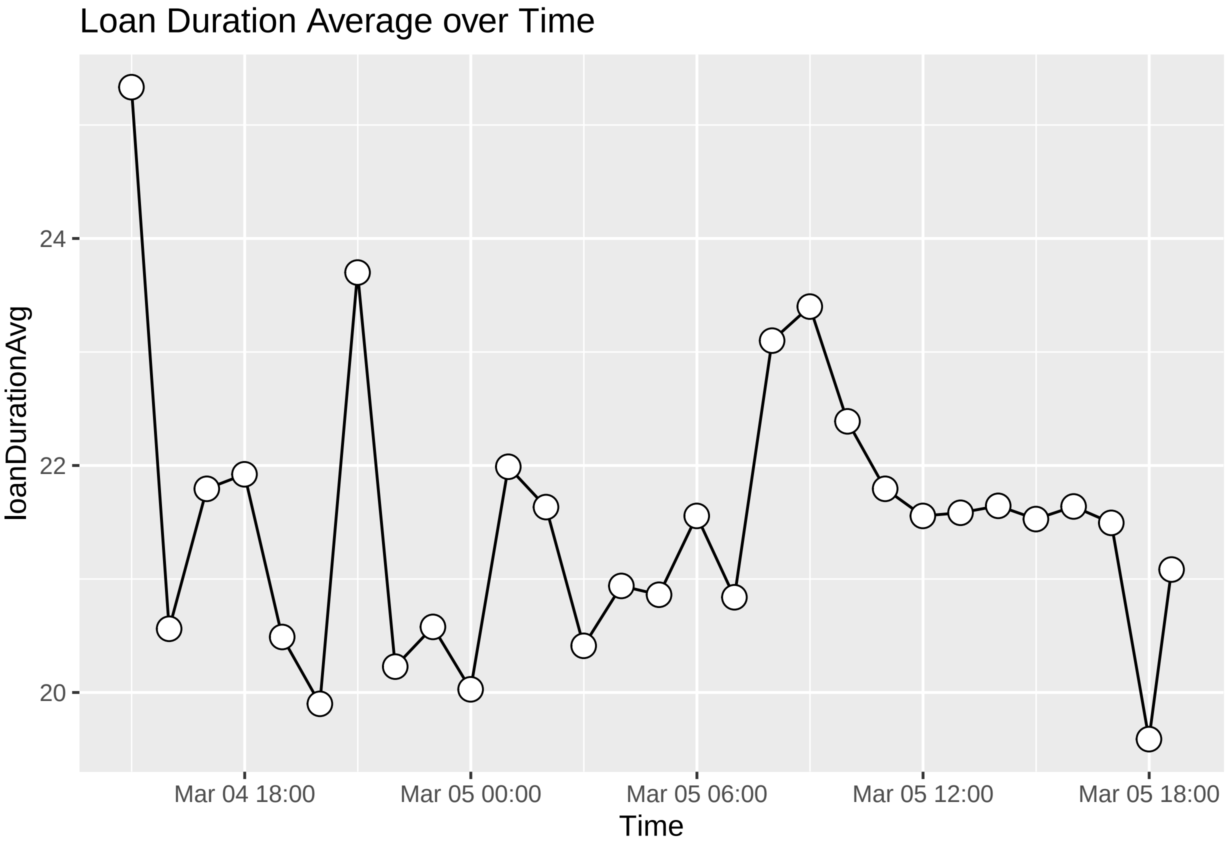
Drift

The Application models Data Drift in a simple manner: by measuring running averages of the change in various measures of spread over time. Currently we can support mean and standard deviation and support other forms of aggregation as well. Notice that these queries take seconds to execute. Following show the ability to monitor changes in the distribution of features over time.

```
url = "http://localhost:8080/api/drift"
body = '{"table_name":"scored_credit",  "features":["loanduration"], "scoring_timestamp":"scoring_timest
res = POST(url = url, content_type_json(), body = body)
data = lapply(content(res), `[[`, 1)

df = tibble(loanDurationAvg = lapply(data, `[[`, 1), time = lapply(data, `[[`, 2 ))
df = as_tibble(lapply(df, unlist))%>% mutate(time = as.POSIXct(time))

df %>% ggplot(aes(x = time, y = loanDurationAvg)) + geom_line() + xlab("Time") +  geom_point( size=4, sh
```

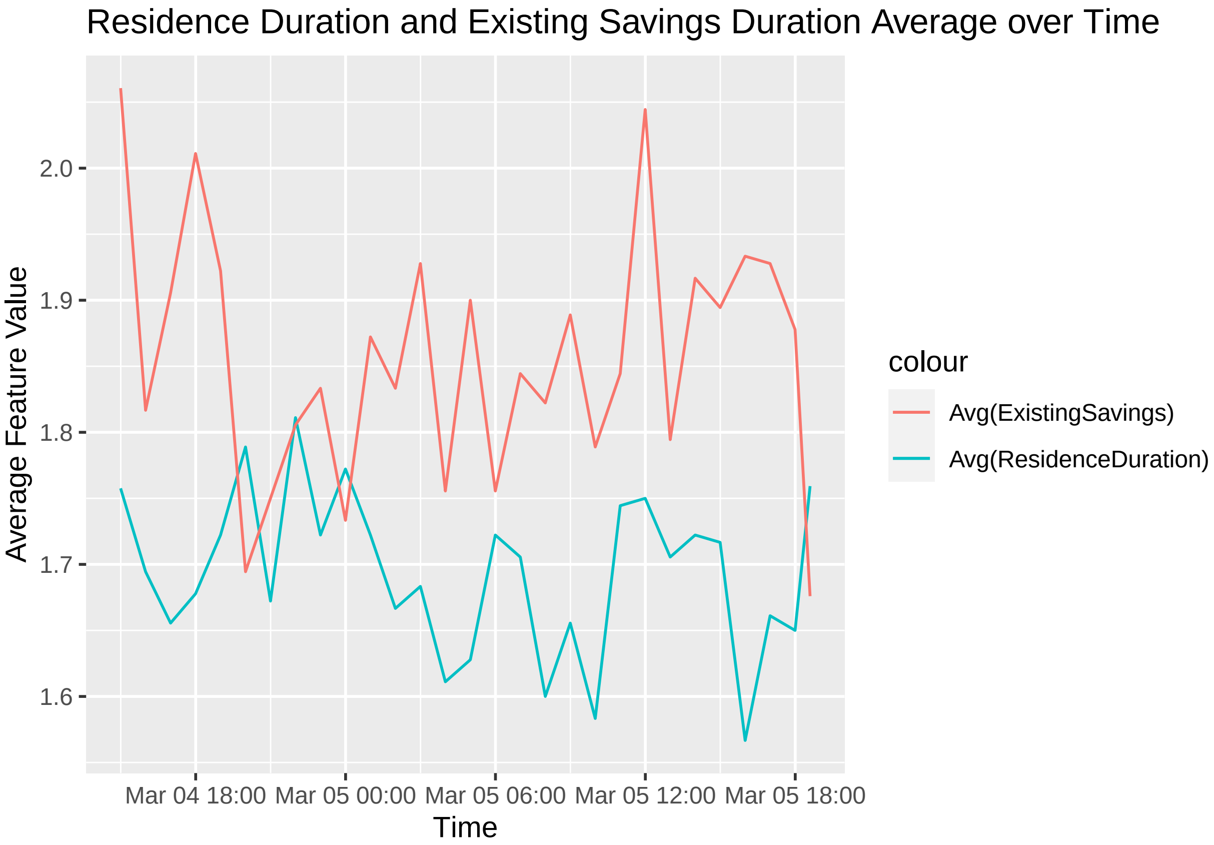


Time to execute this request 0.180912017822266 seconds

```
url = "http://localhost:8080/api/drift"
body = '{"table_name":"scored_credit",  "features":["currentresidenceduration", "existingsavings"], "sc
res = POST(url = url, content_type_json(), body = body)
data = lapply(content(res), `[[`, 1)

df = tibble(currentresidenceduration = lapply(data, `[[`, 1),existingsavings = lapply(data, `[[`, 2), t
df = as_tibble(lapply(df, unlist))%>% mutate(time = as.POSIXct(time))

df %>% ggplot(aes(time)) + geom_line(aes(y = currentresidenceduration, color = "Avg(ResidenceDuration)"
```



Time to execute this request 0.191365003585815 seconds

Accuracy Drift

Accuracy Drift is

$$DI = P(y|K)$$

Where K is the multivariate probability distribution of the features space and y is the model outcome. The Application trains an in-database surrogate multivariate model to predict the models outcome given a feature array, and then tracks the MSE of the surrogate model - a decrease in the MSE of the surrogate model over time indicates a drift in the relationship of

$$P(y|K)$$

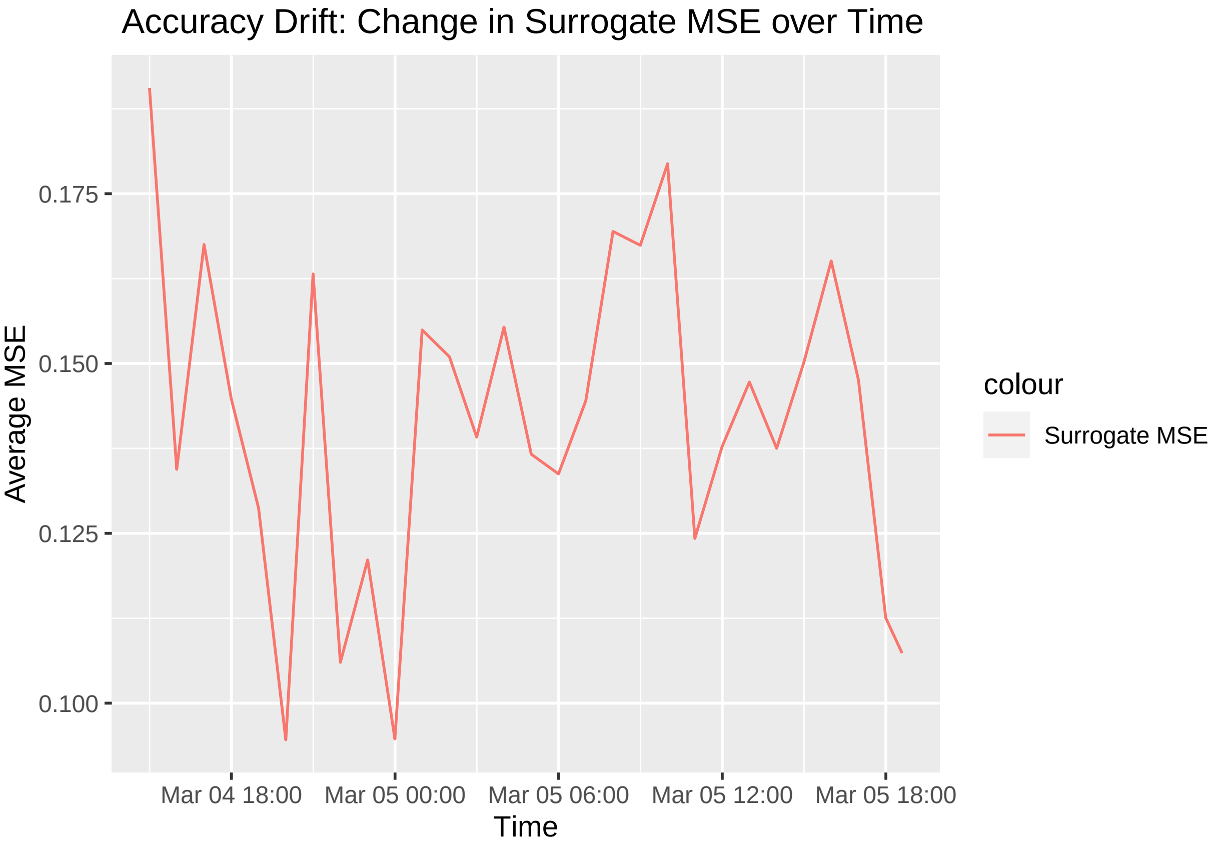
The following graph shows the MSE of the surrogate model staying below where it was at train time, indicating that there is no significant accuracy drift in this model.

```
url = "http://localhost:8080/api/drift/accuracy"
body = '{"table_name":"scored_credit",  "features":["loanduration", "existingsavings", "currentresidenc
res = POST(url = url, content_type_json(), body = body)
```

```
data = lapply(content(res), `[[`, 1)

df = tibble(time = lapply(data, `[[`, 1),mse = lapply(data, `[[`, 2))
df = as_tibble(lapply(df, unlist))%>% mutate(time = as.POSIXct(time))%>%mutate(mse = mse)

df %>% ggplot(aes(time)) + geom_line(aes(y = mse, color = "Surrogate MSE")) + xlab("Time")  + ylab("Aver
```



Time to execute this request 4.30452799797058 seconds

Query Example

```
WITH model as (WITH
 TRAINING AS (SELECT t.*, t.prediction as target, ROW_NUMBER() OVER() AS ROW FROM scored_credit  AS T o
RATES (LEARN_RATE, COUNT) AS (SELECT .001 AS LEARN_RATE, (SELECT COUNT(*) FROM TRAINING) AS COUNT FROM S
LEARNING (ITERATION, B1,B2,B3, INTERCEPT, MSE, M1,M2,M3, C) AS (SELECT 1, CAST(0.0 AS DOUBLE),CAST(0.0 A
 UNION ALL SELECT A.ITERATION +1, A.M1,A.M2,A.M3, A.C, T.MSE, T.M1,T.M2,T.M3, T.C FROM LEARNING A,
TABLE
 (
 SELECT A.ITERATION,
AVG(g.differences*g.differences) AS MSE,
A.M1-(SUM(G.DIFFERENCES*G.loanduration)/(SELECT COUNT FROM RATES))*(SELECT LEARN_RATE from RATES) AS M1
A.M2-(SUM(G.DIFFERENCES*G.existingsavings)/(SELECT COUNT FROM RATES))*(SELECT LEARN_RATE from RATES) AS
A.M3-(SUM(G.DIFFERENCES*G.currentresidenceduration)/(SELECT COUNT FROM RATES))*(SELECT LEARN_RATE from R
A.C-(SUM(g.differences)/(SELECT COUNT from RATES))*(SELECT LEARN_RATE FROM RATES) AS C
 FROM
TABLE ( SELECT
((((Z.loanduration*A.B1)+(Z.existingsavings*A.B2)+(Z.currentresidenceduration*A.B3))+A.INTERCEPT)-z.targ
  z.loanduration,
z.existingsavings,
z.currentresidenceduration
, z.target
 from training as z
)
as g) t
 WHERE A.ITERATION <= 10000)
  SELECT * FROM LEARNING
 WHERE ITERATION >1 ORDER BY MSE LIMIT 1 ),result as (SELECT loanduration,existingsavings,currentresiden
```

Conclusion

Model transactions provide a rich data set from which analytics applications can generate insights into the behavior of machine learning models, even without direct access to training data or model APIs. Moreover, the algorithms used to generate MRM insights can be implemented in-database, which provides significant advantages from a usability, availability and performance perspective.