A Stateless, Query Based API for Model Risk Management (Aletheutes)

Patrick Harned

3/7/2022

Aletheutes ("True" in Greek) is a stateless, streaming, query based API for model risk management. 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

Aletheutes 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. It leverages IBM DB2s advanced in-memory analytics capabilities to performatively analyze model transactions and offer useful insights on the smallest possible amount of data.

Stateless

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

Query-based

Aletheutes is query-based because it pushes all computations, including ML modeling, into the database. Aletheutes is also 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

Aletheutues 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.

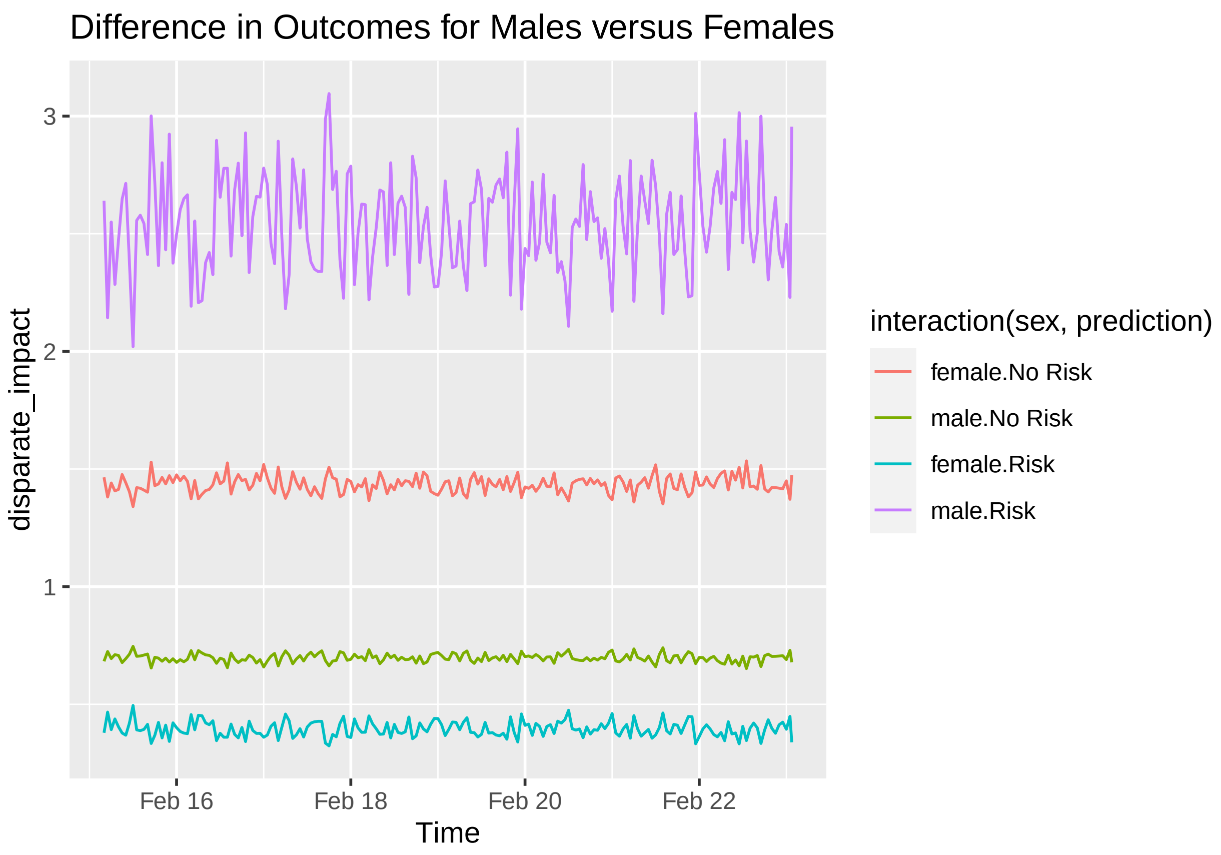
Disparate Impact Implementation

Disparate Impact Ratio is the differential ratio of model outcomes with respect to membership in some protected class. Disparate Impact ratio can be defined as the ratio of positive outcomes for a member of a protected class over the ratio of positive outcomes for a member of a non-protected class, but this is a more constrained and specific definition that would require a user to explicitly state what outcomes of a model are considered "favorable" and what classes are considered "protected". By contrast to this approach, Aletheutes will calculate the respective impact ratios for any combination of outcomes and group membership in the data, and therefore does not to be given any prior information about the data. Aletheutues will also calculate these ratios over any partition of time to produce a running average.

The following graph 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. Its clear from the graph then that Males are assigned the "Risk" Category at a far higher rate than Females, suggesting the model contains implicit bias. Again, the benefit of this approach is that we can see all possible outcomes for all possible groups relative to one another, and the apis do not need to be aware of what an end user or model owner might consider to be a favorable or unfavorable outcome or a protected or reference groups in order to access analytic insights from the api.

```r
url = "http://localhost:8080/api/disparate_impact"
body = '{"prediction":"risk", "table":"test_data2", "protected_column":"sex", "scoring_timestamp":"time
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),
```



Time to execute this request 2.49603891372681 seconds

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, Aletheutes 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.

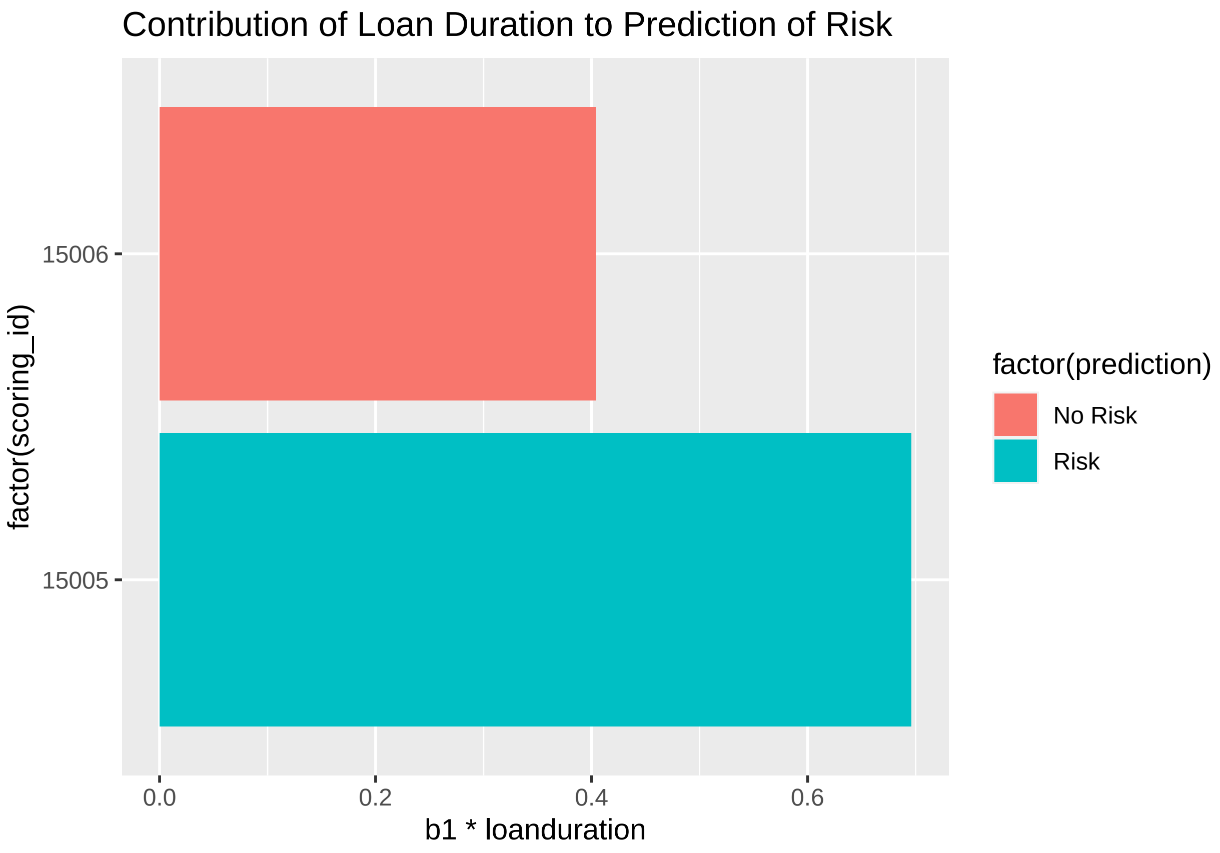
In fact, since we can specify any feature set and target column in an adhoc manner, we can also use this part of the api to detect covariance and dependence between feature columns by building models between feature columns.

```
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 3.63413190841675 seconds

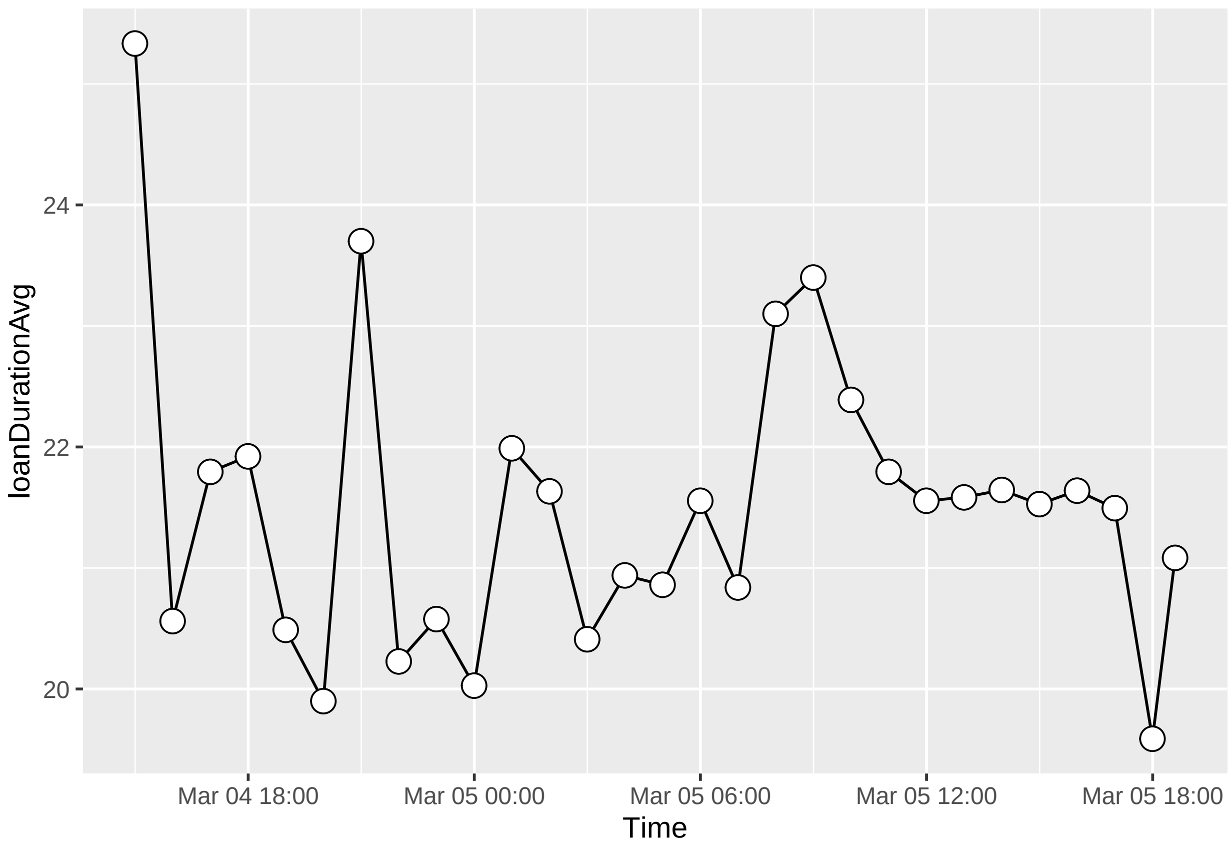
Drift

Aletheutes 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.

```
url = "http://localhost:8080/api/drift"
body = '{"table_name":"scored_credit",  "features":["loanduration"], "scoring_timestamp":"scoring_times
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, s
```

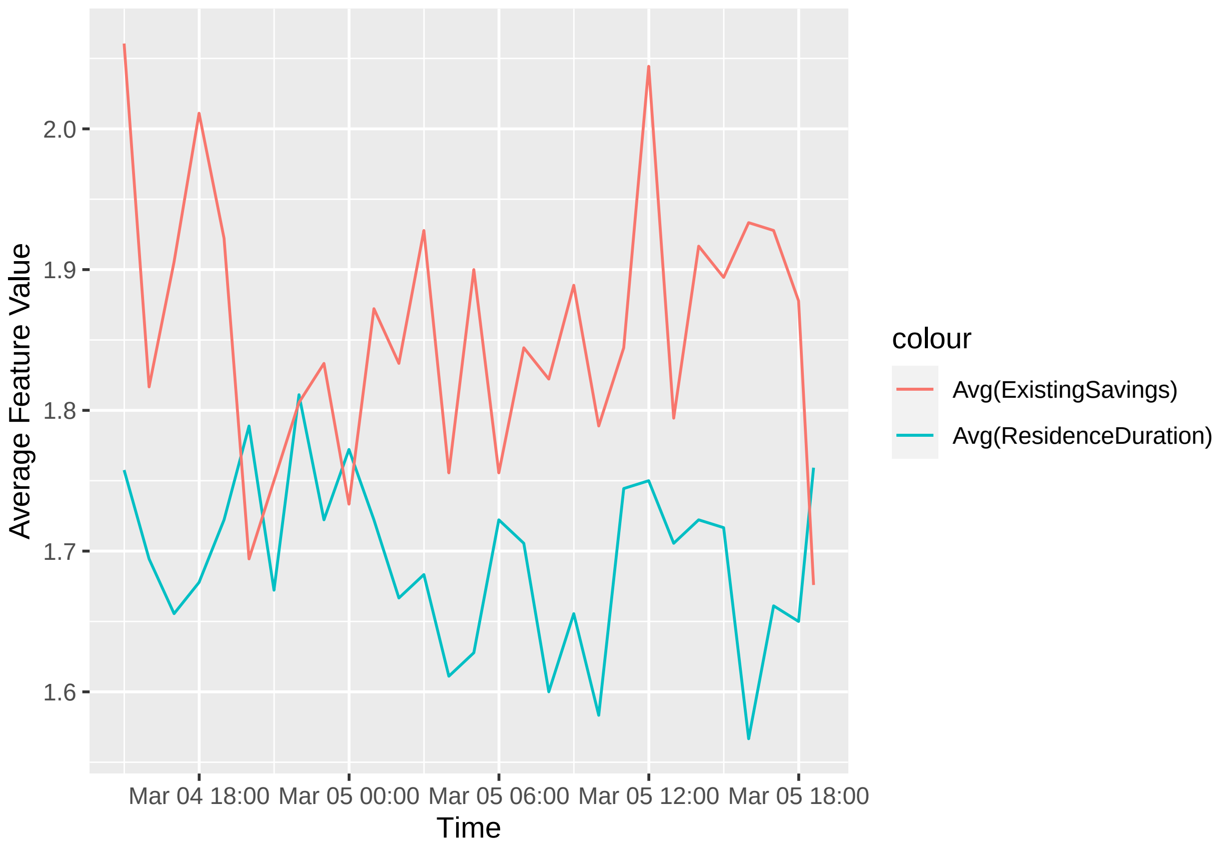


Time to execute this request 0.163724184036255 seconds

```
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 0.179212093353271 seconds