# **Steps in processing request-based factdata**

Contents

[**Steps in processing request-based factdata** 1](#_Toc106309144)

[1. Raw Data Characteristics 3](#_Toc106309145)

[2. Data Cleaning & Uckey Reformat 3](#_Toc106309146)

[2.1 Unstable Slot\_Id Uckey Removal 3](#_Toc106309147)

[2.2 Bucket-Id Calculation 4](#_Toc106309148)

[3. Ucdoc Preprocessing 4](#_Toc106309149)

[3.1 Concept Definition – Popularity, Popularity\_Norm, Nz\_Cnt\_Ratio 4](#_Toc106309150)

[3.2 Bad Ucdoc Removal 4](#_Toc106309151)

[3.2 Sparse Uckeys Selection 5](#_Toc106309152)

[3.3 Group Sparse Uckeys Into Virtual Dense Uckey 6](#_Toc106309153)

[3.4 Accumulation Of Traffic From Sparse Uckeys 6](#_Toc106309154)

[3.5 Generation Of Virtual Uckey(s) 7](#_Toc106309155)

[3.6 denoise each uckey’s time series 7](#_Toc106309156)

[3.7 each uckey’s time series global filtering (global outlier correction) 7](#_Toc106309157)

[3.8 each uckey’s time series local outlier correction 8](#_Toc106309158)

[3.9 Generating Input Data for Trainer 8](#_Toc106309159)

[3.10 DLPredictor Pre-Processing Execution 8](#_Toc106309160)

[3.11 DLPredictor Post-Processing Flowchart 10](#_Toc106309161)

[4. Training Parameters and Tuning 11](#_Toc106309162)

[4.1 Training Parameters 11](#_Toc106309163)

[4.2 Tuning Parameters 12](#_Toc106309164)

[5. Key Guidelines in The Process 12](#_Toc106309165)

[5.1 Daily Traffic Instead of Hourly Traffic to Be Used 12](#_Toc106309166)

[5.2 Unstable Slot-Id Removal 12](#_Toc106309167)

[5.3 Controlling the Ratio of Total Traffic Between Dense Uckey and Virtual Uckey 13](#_Toc106309168)

[6. Problems With Current Dataset 13](#_Toc106309169)

[6.1 Majority UCKEYs from Unstable Slot-Id 13](#_Toc106309170)

[6.3 Traffic Pattern from Stable Slot-Id May Not Be Stable 13](#_Toc106309171)

[7. Test Reports 14](#_Toc106309172)

[8. Q & A 15](#_Toc106309173)

[8.1 Is price\_cat part of uckey formula? 15](#_Toc106309174)

[8.2 how to increase the prediction window in DL 16](#_Toc106309175)

[8.3 Parameter Calculation in Config File 16](#_Toc106309176)

[8.4 What is the criteria/condition of slotids selection? 18](#_Toc106309177)

[8.5 Unable to load saved model for prediction 18](#_Toc106309178)

[8.6 Can current DL model predict more than 29 day? 18](#_Toc106309179)

## Raw Data Characteristics

There are some different characteristics of this set of factdata from previous factdata.

* Request-based instead of impression-based -> more stable.
* Impression count is daily instead of hourly
* Impression count is for all price categories

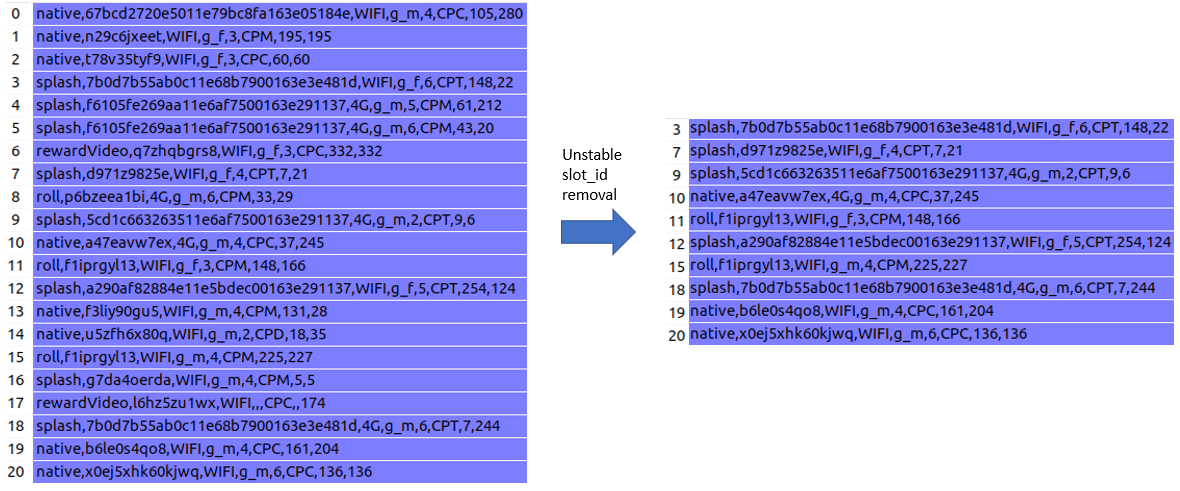
## Data Cleaning & Uckey Reformat

### 2.1 Unstable Slot\_Id Uckey Removal

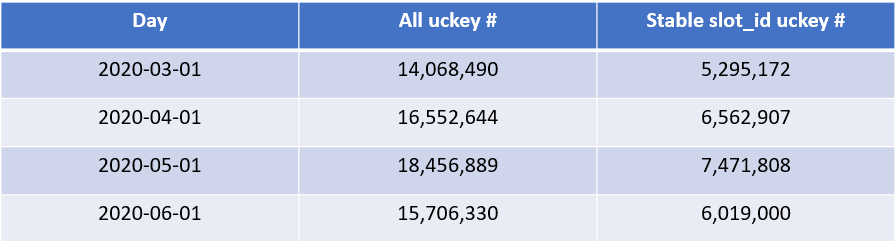
Only data from uckeys with stable slot\_ids are used for training and testing purpose. Here’s the list of stable slot\_ids.

|  |  |  |  |
| --- | --- | --- | --- |
| 5cd1c663263511e6af7500163e291137 | e351de37263311e6af7500163e291137 | f1iprgyl13 | w3wx3nv9ow5i97 |
| 66bcd2720e5011e79bc8fa163e05184e | a47eavw7ex | j1430itab9wj3b | w9fmyd5r0i |
| 68bcd2720e5011e79bc8fa163e05184e | a8syykhszz | k4werqx13k | x0ej5xhk60kjwq |
| 71bcd2720e5011e79bc8fa163e05184e | b6le0s4qo8 | l03493p0r3 | x2fpfbm8rt |
| 7b0d7b55ab0c11e68b7900163e3e481d | d971z9825e | l2d4ec6csv | z041bf6g4s |
| a290af82884e11e5bdec00163e291137 | d9jucwkpr3 | s4z85pd1h8 |  |

As shown below, as an example, the number of rows after “stable slot\_id filtering” changed from 21 in raw data to 10.



Based on data analysis result, as shown in the table below, 60% of distinct uckeys are from unstable slot\_ids, thus this step will eliminate more than half of the dataset from further processing.



### 2.2 Bucket-Id Calculation

Since UCKeys have been changed through previous phases, it is necessary to recalculate the bucket-ids. Bucket-ids are used to group data based on UCKeys. The maximum number for bucket-id is determined by the size of data. Our experience with the latest data shows that 10 is a reasonable size for bucket-id.

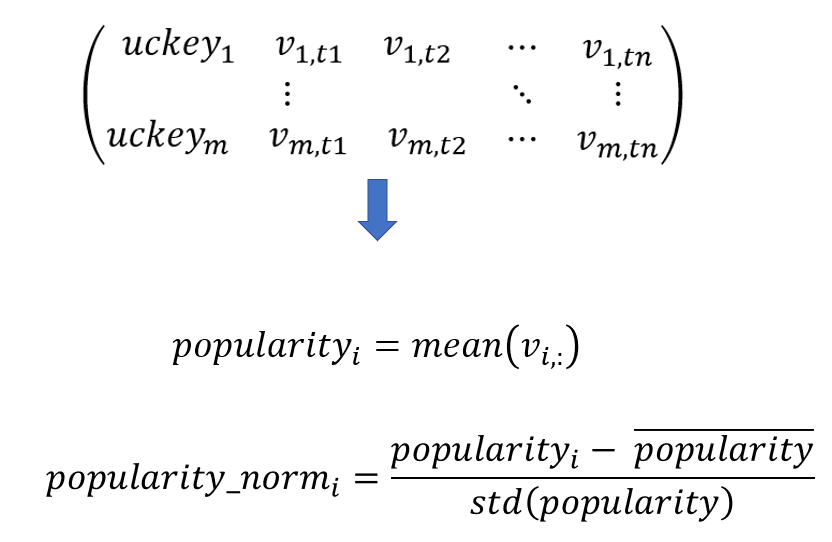
The hash function to calculate bucket-id is sha256.

Bucket-id recalculation is part of first step of pipeline and the bucket size can be configured in config.yaml.

## Ucdoc Preprocessing

### 3.1 Concept Definition – Popularity, Popularity\_Norm, Nz\_Cnt\_Ratio

Uckey’s popularity and normalized popularity can be defined as follows.



Where *t1*, *t2*, …, *tn* are time points, i.e. *day1*, *day2*, …, *dayn*. *v1,t1* is the daily traffic for *uckey1* at *t1* (day1), and so on for *v1,t2*, *v1,tn*, *vm,t1*, *vm,t2*, *vm,tn*

### 3.2 Bad Ucdoc Removal

Even after data cleaning, further preprocessing steps are needed for a few reasons

1. Most of the uckeys are rarely used and their daily traffic is so sparse that can be treated as noise in model training.
2. Due to some unexpected reason (mainly unpredicted human being behavior), some uckey has regular popularity value, however, they are pretty sparse in time domain, i.e. they have pretty high traffic for very few days and they have 0 traffic most of the days.

Uckeys meet the above 2 criteria will be removed since they will have negative impact to model training. However, there is an important guideline we need to keep in mind when determine how many bad uckeys need to be removed.

**Guideline: do not exclude more than 2% of total traffic for bad uckeys (outliers)**

Based on data analysis, the following 2 steps were used to remove bad ucdocs.

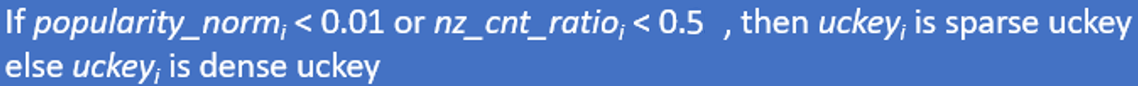
1. popularity < th=5

|  |  |  |
| --- | --- | --- |
| Th | # of bad uckeys | % excluded traffic |
| 1 | 1,104,782 | 0.08% |
| 2 | 1,228,407 | 0.15% |
| 3 | 1,296,609 | 0.21% |
| 4 | 1,342,545 | 0.28% |
| 5 | 1,377,419 | 0.34% |
| 6 | 1,404,868 | 0.40% |
| 7 | 1,427,323 | 0.46% |
| 8 | 1,446,736 | 0.51% |
| 9 | 1,463,611 | 0.57% |
| 10 | 1,478,315 | 0.62% |
| 11 | 1,491,401 | 0.68% |
| 12 | 1,503,144 | 0.73% |
| 13 | 1,513,851 | 0.78% |
| 14 | 1,523,538 | 0.84% |
| 15 | 1,532,659 | 0.89% |
| 16 | 1,541,107 | 0.94% |
| 17 | 1,548,899 | 0.99% |

1. and ratio of non-zero data point < th = 0.5

### 3.2 Sparse Uckeys Selection

Based on analysis on January Factdata got from HQ, the threshold of *popularity\_norm* is 0.01 and the threshold of *nz\_cnt\_ratio* is 0.5, i.e.



**Guideline: the total traffic from sparse uckey should not exceed 20% of total traffic**

### 3.3 Group Sparse Uckeys Into Virtual Dense Uckey

Based on the step defined in 3.2, all uckey will be marked as either dense uckey or sparse uckey, we then need to group sparse uckeys into virtual dense uckeys. Grouping all sparse uckeys into single virtual dense uckey will generate one extremely dense uckey and may not be a good idea, we want to group them into “normal dense uckey” as the major of those marked as dense uckeys.

There’s no change needed to dense uckeys.

To simplify the process of grouping sparse uckeys into virtual dense uckey, a fixed number, *m* (sparse uckey number in each virtual dense uckey), of sparse uckeys are grouped into one virtual uckey, this number is estimated as

Here are the steps of grouping sparse uckeys into virtual dense uckey

1. randomly shuffle the order of sparse uckeys in order to evenly distribute various traffic (large and small) across all sparse uckeys
2. starting from the beginning of shuffled sparse uckey list and group the every *m* (defined above) sparse uckeys into one virtual dense uckey.

### 3.4 Accumulation Of Traffic From Sparse Uckeys

Suppose there are *m* sparse uckeys to generate a virtual dense uckey, their n time points (days for example) traffic data can be expressed as the matrix below.

The virtual uckey’s data will be

And their corresponding distribution probability

Where

### 3.5 Generation Of Virtual Uckey(s)

Since features defined in uckey are mostly with one-hot encoding, the preprocessing (statistic terms) for uckey is different from that of traffic (impress count).

Suppose there are *m* sparse uckeys to generate a virtual uckey, each of them is composed of *n* components (age, gender, etc.) all sparse uckeys can be expressed as:

The generated *virtual\_uckey* can be expressed as

The above formula uses mean of each one-hot variable as the value of virtual\_uckey’s corresponding field which may result in multi-hot value. Since the model does not check whether a uckey is in the format of one-hot or not and these one-hot or multi-hot variables will be normalized with zero-mean normalization anyway, it should work.

### 3.6 denoise each uckey’s time series

In robust signal processing, time series signal, for example, 95% CI (confidence interval) is always used to get rid of outlier noise. In this use case, we are more concerning about lower value outliers than high value outlier, the concept of 10-percentile is used as a rough estimation of threshold to remove lower value noise.

### 3.7 each uckey’s time series global filtering (global outlier correction)

Due to the noisy nature of uckey time series and uncertainty in traffic generation, there usually exist a certain percentage of uckeys that have large number of 0-value days which does not like a normal time series signal. A filter step is then designed to correct these 0-values to some other statistically reasonable values.

The purpose of this filter is to correct signal in problematic uckeys that have a lot of 0 values. Usually it only affect a small portion (say: 10%) of uckeys. It composes of 2 steps as follows.

1. Find all 0 values indices in each uckey’s time series
2. Replace 0 values on these indices with the median value of the median value of all values with a window of width w that centered at the index

Suppose vi,j == 0, then corrected

Where

Where m is the total number of 0s in

Due to the noisy nature of uckey time series and uncertainty in traffic generation, there usually exist a certain percentage of uckeys that have large number of 0-value days which does not like a normal time series signal. A filter step is then designed to

### 3.8 each uckey’s time series local outlier correction

In 3.7, all global outliers (0-value points) have been correction. In this step, a further moving window-based outlier correction is used to detect & correct outliers within a local moving window sliding from the beginning to the end of each uckey’s time series.

For each , if , then is detected as an outlier, its value will be replaced by

### 3.9 Generating Input Data for Trainer

After normalization, the data would be converted to TFRecords format. The TFRecord format is a simple format for storing a sequence of binary records. Then the tfrecords would be read and store into two variables called tensor and plain. Tensor is a dictionary which contains all features like pf\_age, pf\_price\_cat, etc. and plain is a dictionary which contains data stats like number of uckeys, number of days, etc.

At this point the module called feeder would build a temporary TF graph, injects variables into, and saves variables to TF checkpoint.

These checkpoints would be used as an input for the trainer.

### 3.10 DLPredictor Pre-Processing Execution

Data pre-processing to train DL-Predictor is pipeline of several processes. Each process is a spark base job and is initiated by spark script. The pipeline can be also run by Airflow (DAGs are provided).

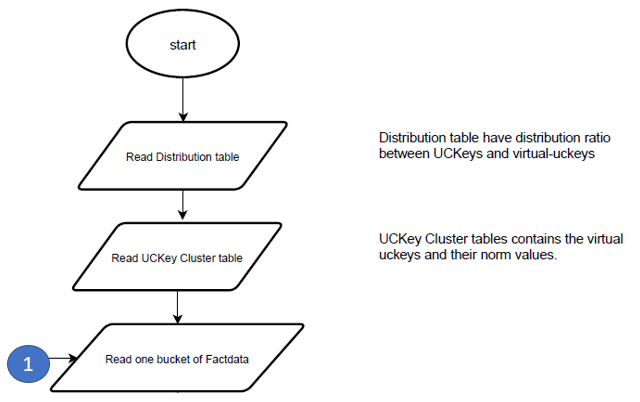
Here is the sequence of the individual processes that are required to run pre-processing.

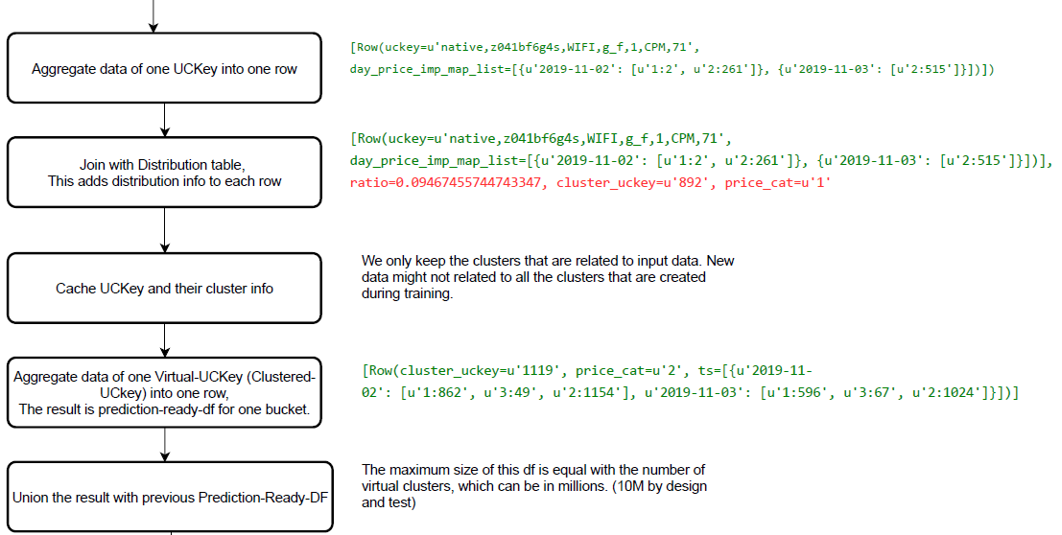
The scripts and the sequence to run the Pre-Processing are found in run.sh

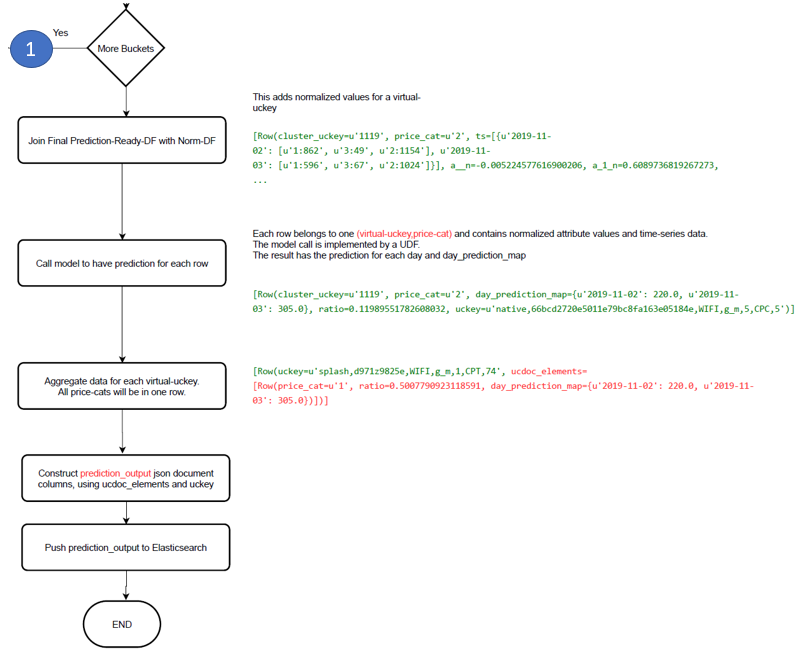
### 3.11 DLPredictor Post-Processing Flowchart

The DLPredictor Post-Process is the prediction process. This process uses model-client and deployed model to predict future inventories for UCKEYs (audience-segments).

The following is the flowchart of this process.







## Training Parameters and Tuning

### 4.1 Training Parameters

There are 2 configuration files that define parameters used in training, i.e. config.xml and hparams.py.

#### 4.1.1 Config.xml

There’s one section related to trainer.py defined in config.xml. However, the list of holidays usually needs to be specified when change from one dataset to the other. Since the period in this dataset is 2020-03-01 ~ 2020-06-30, the holiday list under “pipeline” section is defined as:

holidays: ['2020-03-06', '2020-03-07', '2020-03-08', '2020-03-09', '2020-03-14', '2020-03-15','2020-03-16', '2020-03-28','2020-03-29', '2020-03-30','2020-03-31', '2020-04-01', '2020-04-17', '2020-04-18','2020-04-19', '2020-04-20', '2020-04-21', '2020-04-22', '2020-04-23', '2020-04-26', '2020-04-27', '2020-04-28', '2020-04-29', '2020-04-30', '2020-06-16', '2020-06-17', '2020-06-18', '2020-06-19', '2020-06-20', '2020-06-21', '2020-06-23', '2020-06-24', '2020-06-25', '2020-06-26', '2020-06-27', '2020-06-28', '2020-06-29']

One parameter changed in “trainer” section is: max\_epoch: 345

#### 4.1.2 Hparams.py

Under params\_s32 section

|  |  |
| --- | --- |
| **parameter** | **value** |
| batch\_size | 300 |
| train\_window | 60 |
| rnn\_depth | 500 |

### 4.2 Tuning Parameters

Some parameters may worth tune to have different performance

1. USE\_ATTENTION option in model.py may be turned on or off to include or exclude attention block in the model
2. batch\_size in hparams.py may be changed to a larger value to have more stable training convergence
3. train\_window in hparams.py may be changed to larger value to include more time series data in training
4. use\_attn may be changed accordingly with “USE\_ATTENTION” in model.py
5. rnn\_depth may also be tuned to either larger or smaller depending on data properties.

## Key Guidelines in The Process

### 5.1 Daily Traffic Instead of Hourly Traffic to Be Used

Modern machine learning models are built on top of statistics, it is thus beneficial to have some statistics features (values) included in the training. Hourly traffic is quite sparse, and the traffic pattern is kind of random (noisy), while daily traffic – aggregation of hourly traffic is more stable, and pattern is more sustainable.

As we all know, no matter what machine learning used, garbage in and garbage out, it is then quite critical to grasp meaningful information from noisy & sparse data to build a meaningful model.

### 5.2 Unstable Slot-Id Removal

The purpose for this step is like that of 5.1 -> to avoid of noisy traffic data in model training. traffic data from unstable slot\_id is random by nature. If they are included in model training, the built model will turn to fit noise instead of real signals.

### 5.3 Controlling the Ratio of Total Traffic Between Dense Uckey and Virtual Uckey

The purpose of building current model is to predict traffic and it needs to focus on the major of total traffic.

Sparse uckey data will cause problems in training since model is trained in batches. If sparse UCKEY data is fed into training directly, most of the batches will be from sparse uckeys which will make the training shaking and hard to converge.

Aggregating sparse uckeys into virtual dense uckeys can solve the problem above, however, we know that the model’s accuracy on sparse uckey will be limited, the model is built mainly for dense uckeys, thus it is quite important to keep in mind that the total traffic from sparse uckeys will not exceed a certain percentage (say 10%~20%) of the total traffic, so that the model built can accurately predict the major traffic and functionally work for traffic from sparse uckeys.

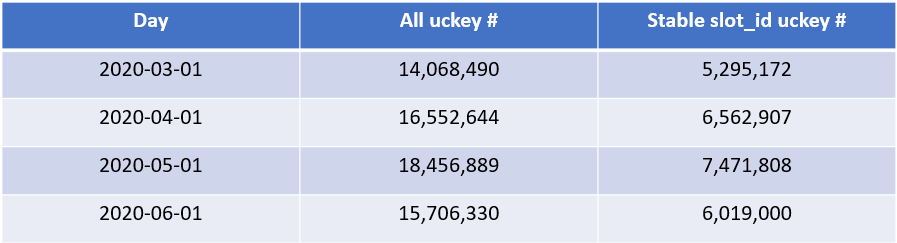
## Problems With Current Dataset

There are a few problems we found with dataset.

NOTE: The following numbers are based on one specific dataset, numbers and statistics might be different for different datasets from different resources (impression logs vs request logs)

### 6.1 Majority UCKEYs from Unstable Slot-Id

As shown in the table below, more than 60% of distinct uckeys are from unstable slot\_id and can not be used in training and testing.



### 6.3 Traffic Pattern from Stable Slot-Id May Not Be Stable

As can be seen in the figures below, the traffic pattern of some uckeys are quite unstable even through they are from stable slot-ids.

## Test Reports

Input data spec:

Days: 87 days from 2022-01-01 until 2022-03-28,

Days used for training: 2022-01-01 until 2022-03-20 (79 days)

Days used for verification: 2022-03-21 until 2022-03-28 (8 days)

Use-Case 1:

Use-Case spec:

Using outlier-1

Pipeline 06092022 based on 05172022-ts

Number-of-Uckeys (dense + non-dense): 369,308

Trainer error rate: 4.1%

System prediction error rate (SPER): 7.05%

Note: SPER is aggregated at slot-id level and weighted by traffic for each day. The script is si\_traffic\_prediction\_check\_4.

Use-Case 2:

Use-Case spec:

Using outlier-2

Pipeline 06132022 based on 05172022-ts

Trainer error rate: 3.4%

System prediction error rate: 6.78%

## Q & A

Questions from issues section on Github (<https://github.com/Futurewei-io/blue-marlin/issues>) so far are summarized and answered in this section.

### 8.1 Is price\_cat part of uckey formula?

Answer:

In section 2.4, uckey formula is defined as:

Uckey = adv\_type, slot\_id, net\_type, gender, age, pricing\_type, IP location, price\_cat

In the implementation code, uckey formula does not include “price\_cat” which raised the code compatibility concern.

For each uckey, the model will train&predict 4 time-series (ideally) corresponding to 4 “price\_cat” separately. “price\_cat” is needed to differentiate the 4 time-series. Instead of append “price\_cat” to uckey formula, in implementation codes, the (uckey, price\_cat) tuple is used to implement the discriminant.

The purpose of the 2 approaches (adding price\_cat in uckey vs using (uckey, price\_cat)) is the same. There is no need to make any code change as long as the convention is consistent throughout the pipeline.

### 8.2 how to increase the prediction window in DL

Answer:

By default, the full dataset is split into 3 parts: training, testing, and validation. The number of days in both testing and validation equal to predict\_window. Due to the condition defined in input\_pipe.py, if the full dataset is 90 days, the training dataset is 60 days, the maximum number of days of testing (prediction) is 14 days.

The assertion of the 3 parts data splitting is defined in input\_pipe.py as:

asset inp.data\_days - predict\_window > predict\_window + train\_window, "Predict+train window length is larger than the total number of days in dataset"

In order to make use of reserved validation data as testing data (increase prediction window), the above code can be modified to:

asset inp.data\_days > predict\_window + train\_window, "Predict+train window length is larger than the total number of days in dataset"

so that the maximum predict\_window can be extended to 29.

### 8.3 Parameter Calculation in Config File

*Question*: <https://github.com/Futurewei-io/blue-marlin/issues/55>

I have few questions regarding the config parameters. For prediction as well as for training, there exists the following parameters:  
cluster\_size:

* datapoints\_min\_th: 0.15
* datapoints\_th\_uckeys: 0.5
* datapoints\_th\_clusters: 0.5
* popularity\_norm: 0.01
* popularity\_th: 5
* median\_popularity\_of\_dense: 1856.2833251953125

We would like to get access to the script which calculates the above mentioned parameters. Alternately, a document explaining how to calculate these parameters would also be helpful.

*Answer*:

**Please refer to section 3 for the meaning as well as how to calculate these parameters**.

A brief description is as follows:

Assuming all data is in a 2D matrix of m x n (m rows and n columns), each row is a time-series signal.  
Basically, there are 3 types of rows, i.e.  
1). bad row (will not be used)  
2). sparse row (will be aggregated)  
3). dense row (good)

1. Bad row  
   there are 2 criteria to determine if a row is a bad row, they are:  
   1). the row's popularity (mean value) < popularity\_th (defined as 5)  
   2). the row's popularity > mean\_popularity\_of\_dense (defined as 1856.2833251953125)  
   and  
   the row's percentage of non-zero columns < datapoints\_min\_th (defined as 0.15)  
   These 2 numbers are determined based on an analysis of the percentage of the sum of removed rows in total values of the matrix.  
   According to our analysis, the 1) condition will remove about 400k rows whose total values contribute to only 0.12% of the matrix's total value; the 2) condition will remove about 500 rows whose total values contribute to only 0.52% of the matrix's total value. In other words, both conditions will add up to 0.64% of the matrix's total value which is trivial and will not affect the final result.
2. Sparse row  
   Once bad rows have been removed from the matrix, left-over rows will be marked as either dense row or sparse row. The criteria to mark if a row is a sparse row or not is as follows:

the row's normalized popularity < popularity\_norm (defined as 0.01)  
and  
the row's percentage of non-zero columns < datapoints\_th\_uckeys (defined as 0.5)

there are 2 concepts need to be defined in order to understand the criteria above  
popularity of row\_i= mean value of row\_i  
normalized popularity of row\_i = (popularity of row\_i - mean of all rows' popularity) / std of all row's popularity  
According to our analysis, the sum of all values in a sparse row contributes to ~5% of the matrix's total value.

1. aggregate sparse rows into a virtual dense row (sparse rows cluster)  
   All sparse rows will be grouped into a few clusters, all sparse rows belong to a certain cluster will be aggregated and virtually be treated as a dense row.  
   After clustering, each virtual dense row will be dense in value, we still need to check whether they are dense in columns, the criteria to determine whether a virtual dense row is good or not is  
   the virtual dense row's percentage of non-zero columns > datapoints\_th\_clusters (defined as 0.5)

After all the 3 steps mentioned above, all rows are either removed or dense enough and are ready for further processing.

### 8.4 What is the criteria/condition of slotids selection?

Question: <https://github.com/Futurewei-io/blue-marlin/issues/55>

Answer:

The criteria for slot\_ids selection is stable. Please find in section 2.1 for a full list of stable slot\_id used for selection/filtering.

### 8.5 Unable to load saved model for prediction

Question: <https://github.com/Futurewei-io/blue-marlin/issues/36>

Answer:

GRUBlockCell needs to be imported in order to load the model, please add:

from tensorflow.contrib.rnn import GRUBlockCell

### 8.6 Can current DL model predict more than 29 day?

Question: <https://github.com/Futurewei-io/blue-marlin/issues/64>

Answer:

The DL model can predict more than 40 day.  
The default total period of days is 90 days, the 90 days will cover both training and testing, i.e. training + testing = 90 days. If training = 60 days (default), the maximum testing days is 30 days.  
If the total period of days is 120 days and training is 70 days, the maximum testing days will be 120 - 70 = 50 days.

8.7 Daily basis prediction

Question: <https://github.com/Futurewei-io/blue-marlin/issues/64>

Answer:

The prediction from the DL model is daily basis. There are some steps in pipeline to distribute daily basis prediction to hourly basis, these steps can be skipped if daily basis is the only choice.