

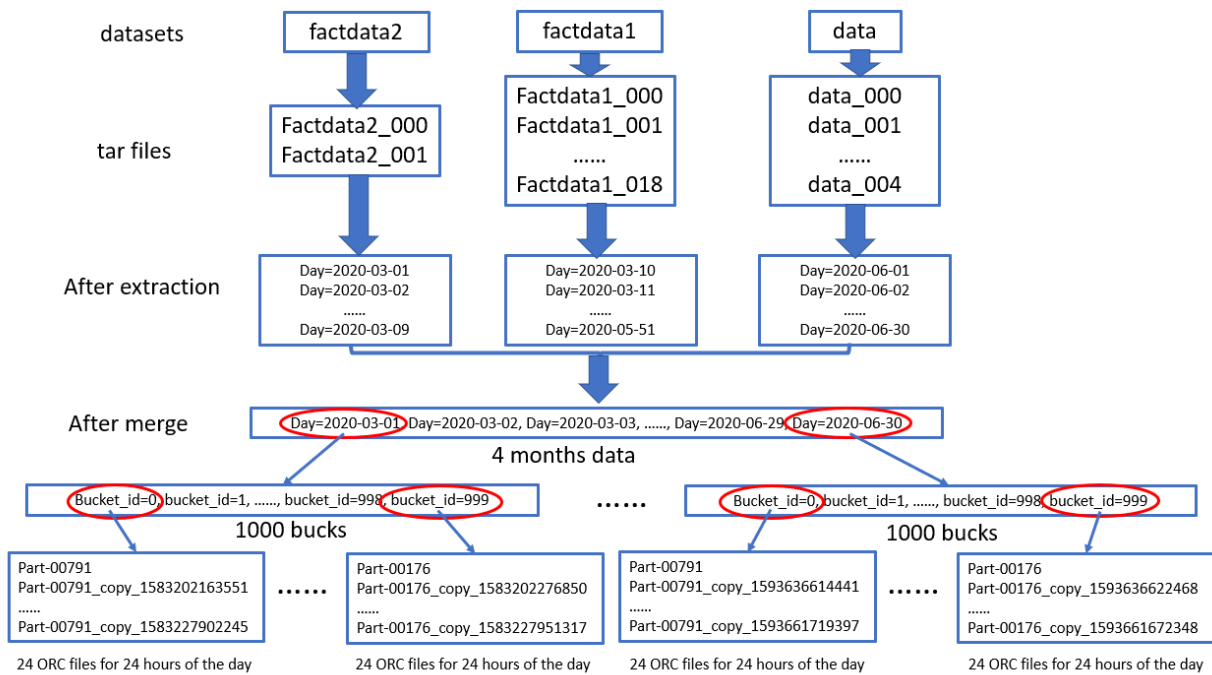
Steps in processing new factdata

Contents

Steps in processing new factdata	1
1. Raw Data Structure	2
2. Data Cleaning And Uckey Reformat	3
2.1 Unstable Slot_Id Uckey Removal	3
2.2 Dropping City_Index In Uckey	4
2.3 IPL Remapping	5
2.4 Daily Traffic Generation And “Price Category” Based Uckey Splitting	7
2.5 Bucket-Id Calculation	8
3. Ucdoc Preprocessing	9
3.1 Concept Definition – Popularity, Popularity_Norm, Nz_Cnt_Ratio	9
3.2 Bad Ucdoc Removal	9
3.2 Sparse Uckey Selection	10
3.3 Group Sparse Uckey Into Virtual Dense Uckey	10
3.4 Accumulation Of Traffic From Sparse Uckey	11
3.5 Generation Of Virtual Uckey(s)	11
3.6 Generating Input Data for Trainer	12
3.7 DL-Predictor Pre-Processing Execution	12
3.8 DL-Predictor Post-Processing Flowchart	13
4. Training Parameters And Tuning	14
4.1 Training Parameters	14
4.2 Tuning Parameters	15
5. Key Guidelines In The Process	15
5.1 Daily Traffic Instead Of Hourly Traffic To Be Used	15
5.2 Unstable Slot_Id Removal	16
5.3 Controlling The Ratio Of Total Traffic Between Dense Uckey And Virtual Uckey	16
6. Problems With Current Dataset	16
6.1 One Ucdoc Maps To Multiple Bucket_Id	16
6.2 Majority Uckey From Unstable Slot_Id	17
6.3 Traffic Pattern From Stable Slot_Id May Not Be Stable	17

7. Q & A.....	17
7.1 Is price_cat part of uckey formula?	17
7.2 how to increase the prediction window in DL	18
7.3 Parameter Calculation in Config File.....	18
7.4 What is the criteria/condition of slotids selection?.....	20
7.5 Unable to load saved model for prediction	20

1. Raw Data Structure



The figure shown above is the structure of raw data.

Due to transmission limitation, the whole dataset was split into 3 sub-datasets:

- *factdata1*
- *factdata2*
- *data*

Each sub-dataset comes along with various number of archived files (.tar format).

- *factdat2* has 2 files (*factdata2_000* and *factdata2_001*)
- *factdat1* has 19 files (*factdata1_000* ~ *factdata2_018*)
- *data* has 5 files (*data_000* ~ *data_004*)

After extraction, each dataset has raw factdata covering different time period.

- Factdata1 covers 2020-03-10 ~ 2020-05-31
- Factdata2 covers 2020-03-01 ~ 2020-03-09
- Data covers 2020-06-01 ~ 2020-06-30

Altogether, the whole dataset covers 4 months (2020-03-01 ~ 2020-06-30).

There are 24,000 files for each day which are organized into 2 levels, i.e. bucket-level (1000 buckets) and hour-level (24 hours). Each bucket is a sub-directory named as bucket_id=0 ~ bucket_id=999; each hour is a single ORC file with the pattern of “part-xxxxx_copy_xxxxxxxxxxxxxx”.

Each ORC file has count_array information with specific hour-of-the-day and bucket_id it belongs to, as shown below. The “count_array” has impression count information for 4 different price_category.

uckey	count_array	hour	bucket_id
banner,d3lxq4l6rn,WIFI,g_f,5,CPC,194,194	['1:6']	4	0
native,68bcd2720e5011e79bc8fa163e05184e,4G,g_m,5,CPM,294,294	['3:2' '2:1' '1:2']	4	0
native,a47eavw7ex,4G,g_m,3,CPC,150,298	['1:1']	4	0
native,a47eavw7ex,WIFI,g_m,4,CPC,138,43	['1:5']	4	0
native,b6le0s4qo8,3G,g_m,4,CPC,,	['1:24']	4	0
native,d9su9ng8m1rgte,WIFI,g_m,5,CPC,62,61	['1:2']	4	0
native,g7m2zuits8,4G,g_m,5,CPM,46,276	['1:2']	4	0
native,g7m2zuits8,WIFI,g_m,4,CPM,22,218	['1:2']	4	0
native,i7ift1wtxgwb0y,4G,,,CPC,,130	['1:2']	4	0
native,s4zd02tbbt,WIFI,,,CPC,,190	['1:3']	4	0
native,t78v35tyf9,4G,,,CPC,,1	['1:31']	4	0
native,t95j80t2lk,WIFI,,,CPM,,176	['1:2' '2:1' '3:3']	4	0
native,u5zfh6x80q,WIFI,g_m,5,CPD,14,129	['1:4']	4	0
rewardVideo,e1te7375ku,WIFI,,,CPC,,177	['1:1']	4	0
rewardVideo,j659nqmtci,WIFI,g_f,4,CPC,178,36	['1:1']	4	0
splash,7b0d7b55ab0c11e68b7900163e3e481d,WIFI,g_f,2,CPM,8,128	['3:1']	4	0
splash,c38adivea9,WIFI,q_f,2,CPM,12,	['2:1']	4	0

2. Data Cleaning And Uckey Reformat

2.1 Unstable Slot_Id Uckey Removal

Only data from uckey with stable slot_ids are used for training and testing purpose. Here's the list of stable slot_ids.

'a290af82884e11e5bdec00163e291137'	'11'	'05'	'04'
'e351de37263311e6af7500163e291137'	'02'	'01'	'06'
'5cd1c663263511e6af7500163e291137'	'03'	'd9jucwkpr3'	'x2fpfbm8rt'
'17dd6d8098bf11e5bdec00163e291137'	'x0ej5xhk60kjqwq'	'j1430itab9wj3b'	'w9fmyd5r0i'
'd4d7362e879511e5bdec00163e291137'	'g9iv6p4sjy'	'k4werqx13k'	'l03493p0r3'
'15e9ddce941b11e5bdec00163e291137'	'g7m2zuits8'	'w3wx3nv9ow5i97'	'a1nvkxk62q'
'71bcd2720e5011e79bc8fa163e05184e'	'd47737w664'	'c4n08ku47t'	'b6le0s4qo8'
'68bcd2720e5011e79bc8fa163e05184e'	'p7gsrebd4m'	'a8syykhszz'	'l2d4ec6csv'
'66bcd2720e5011e79bc8fa163e05184e'	'h034y5sp0i'	's4z85pd1h8'	'z041bf6g4s'
'72bcd2720e5011e79bc8fa163e05184e'	'a47eavw7ex'	'f1iprgyl13'	'q4jtehrqn2'

'7b0d7b55ab0c11e68b7900163e3e481d'	'm1040xexan'	'd971z9825e'	'a83jryvehg'
------------------------------------	--------------	--------------	--------------

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

0	native,67bcd2720e5011e79bc8fa163e05184e,WIFI,g_m,4,CPC,105,280		
1	native,n29c6jxeet,WIFI,g_f,3,CPM,195,195		
2	native,t78v35tyf9,WIFI,g_f,3,CPC,60,60		
3	splash,7b0d7b55ab0c11e68b7900163e3e481d,WIFI,g_f,6,CPT,148,22		
4	splash,f6105fe269aa11e6af7500163e291137,4G,g_m,5,CPM,61,212		
5	splash,f6105fe269aa11e6af7500163e291137,4G,g_m,6,CPM,43,20		
6	rewardVideo,q7zhqbg8r,WIFI,g_f,3,CPC,332,332		
7	splash,d971z9825e,WIFI,g_f,4,CPT,7,21		
8	roll,p6bzeea1bi,4G,g_m,6,CPM,33,29		
9	splash,5cd1c663263511e6af7500163e291137,4G,g_m,2,CPT,9,6		
10	native,a47eavw7ex,4G,g_m,4,CPC,37,245		
11	roll,f1iprgyl13,WIFI,g_f,3,CPM,148,166		
12	splash,a290af82884e11e5bdec00163e291137,WIFI,g_f,5,CPT,254,124		
13	native,f3liy90gu5,WIFI,g_m,4,CPM,131,28		
14	native,u5zfh6x80q,WIFI,g_m,2,CPD,18,35		
15	roll,f1iprgyl13,WIFI,g_m,4,CPM,225,227		
16	splash,g7da4oerda,WIFI,g_m,4,CPM,5,5		
17	rewardVideo,l6hz5zu1wx,WIFI,,,CPC,,174		
18	splash,7b0d7b55ab0c11e68b7900163e3e481d,4G,g_m,6,CPT,7,244		
19	native,b6le0s4qo8,WIFI,g_m,4,CPC,161,204		
20	native,x0ej5xhk60kjqw,WIFI,g_m,6,CPC,136,136		

Unstable slot_id removal

3	splash,7b0d7b55ab0c11e68b7900163e3e481d,WIFI,g_f,6,CPT,148,22
7	splash,d971z9825e,WIFI,g_f,4,CPT,7,21
9	splash,5cd1c663263511e6af7500163e291137,4G,g_m,2,CPT,9,6
10	native,a47eavw7ex,4G,g_m,4,CPC,37,245
11	roll,f1iprgyl13,WIFI,g_f,3,CPM,148,166
12	splash,a290af82884e11e5bdec00163e291137,WIFI,g_f,5,CPT,254,124
15	roll,f1iprgyl13,WIFI,g_m,4,CPM,225,227
18	splash,7b0d7b55ab0c11e68b7900163e3e481d,4G,g_m,6,CPT,7,244
19	native,b6le0s4qo8,WIFI,g_m,4,CPC,161,204
20	native,x0ej5xhk60kjqw,WIFI,g_m,6,CPC,136,136

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

Day	All uckey #	Stable slot_id uckey #
2020-03-01	14,068,490	5,295,172
2020-04-01	16,552,644	6,562,907
2020-05-01	18,456,889	7,471,808
2020-06-01	15,706,330	6,019,000

2.2 Dropping City_Index In Uckey

The uckey in raw data was reformatted via dropping the city_index (2nd to the last field in uckey definition) as shown below.

3	splash,7b0d7b55ab0c11e68b7900163e3e481d,WIFI,g_f,6,CPT,148,22
7	splash,d971z9825e,WIFI,g_f,4,CPT,7,21
9	splash,5cd1c663263511e6af7500163e291137,4G,g_m,2,CPT,9,5
10	native,a47eavw7ex,4G,g_m,4,CPC,37,245
11	roll,f1iprgyl13,WIFI,g_f,3,CPM,148,166
12	splash,a290af82884e11e5bdec00163e291137,WIFI,g_f,5,CPT,254,124
15	roll,f1iprgyl13,WIFI,g_m,4,CPM,225,227
18	splash,7b0d7b55ab0c11e68b7900163e3e481d,4G,g_m,6,CPT,7,244
19	native,b6le0s4qo8,WIFI,g_m,4,CPC,161,204
20	native,x0ej5xhk60kjqw,WIFI,g_m,6,CPC,136,136

↓ Dropping city_index

3	splash,7b0d7b55ab0c11e68b7900163e3e481d,WIFI,g_f,6,CPT,22
7	splash,d971z9825e,WIFI,g_f,4,CPT,21
9	splash,5cd1c663263511e6af7500163e291137,4G,g_m,2,CPT,6
10	native,a47eavw7ex,4G,g_m,4,CPC,245
11	roll,f1iprgyl13,WIFI,g_f,3,CPM,166
12	splash,a290af82884e11e5bdec00163e291137,WIFI,g_f,5,CPT,124
15	roll,f1iprgyl13,WIFI,g_m,4,CPM,227
18	splash,7b0d7b55ab0c11e68b7900163e3e481d,4G,g_m,6,CPT,244
19	native,b6le0s4qo8,WIFI,g_m,4,CPC,204
20	native,x0ej5xhk60kjqw,WIFI,g_m,6,CPC,136

The step further greatly reduced the number of distinct uckey. As shown in the table below, for example, the total distinct uckey numbers for 2020-03-01 changed from ~14 million in raw data to 348k after 2.1 & 2.2 combined together – 40 times reduction.

Day	All uckey #	Stable slot_id uckey #	Uckey # after dropping city_index
2020-03-01	14,068,490	5,295,172	347,908

2.3 IPL Remapping

The uckey after 2.2 will further be further be processed.

The IP location (IPL) field in uckey (last field) has 360+ distinct values and introduces sparsity in traffic for each distinct uckey. The IPL value in each uckey is mapped to a value between 1~82 based on the following mapping table – mapping “new” column value to “old” column value.

New	old	new	old	new	old	New	old	New	old
1	1	80	70	159	74	238	78	317	81
2	2	81	70	160	74	239	78	318	81
3	3	82	70	161	74	240	78	319	81
4	4	83	70	162	74	241	78	320	81
5	5	84	70	163	74	242	79	321	81
6	6	85	70	164	74	243	79	322	81
7	7	86	70	165	74	244	79	323	81
8	8	87	70	166	74	245	80	324	81
9	9	88	70	167	74	246	80	325	81
10	10	89	70	168	74	247	80	326	81
11	11	90	70	169	74	248	80	327	81
12	12	91	70	170	74	249	80	328	81
13	13	92	70	171	74	250	80	329	81
14	14	93	70	172	74	251	80	330	81

15	15	94	70	173	74	252	80	331	81
16	16	95	70	174	74	253	80	332	81
17	17	96	70	175	74	254	80	333	81
18	18	97	70	176	74	255	80	334	81
19	19	98	70	177	74	256	80	335	81
20	20	99	70	178	75	257	80	336	81
21	21	100	70	179	75	258	80	337	81
22	22	101	71	180	75	259	80	338	81
23	23	102	71	181	75	260	80	339	81
24	24	103	71	182	75	261	80	340	82
25	25	104	71	183	75	262	80	341	82
26	26	105	71	184	75	263	80	342	82
27	27	106	71	185	76	264	80	343	82
28	28	107	71	186	76	265	80	344	82
29	29	108	71	187	76	266	80	345	82
30	30	109	71	188	76	267	80	346	82
31	31	110	71	189	76	268	80	347	82
32	32	111	71	190	76	269	80	348	82
33	33	112	71	191	76	270	80	349	82
34	34	113	71	192	76	271	80	350	82
35	35	114	71	193	76	272	80	351	82
36	36	115	71	194	76	273	80	352	82
37	37	116	71	195	76	274	80	353	82
38	38	117	71	196	76	275	80	354	82
39	39	118	71	197	76	276	80	355	82
40	40	119	71	198	76	277	80	356	82
41	41	120	71	199	76	278	80	357	82
42	42	121	71	200	76	279	80	358	82
43	43	122	71	201	76	280	80	359	82
44	44	123	71	202	76	281	80	360	82
45	45	124	72	203	76	282	80	361	82
46	46	125	72	204	76	283	80	362	82
47	47	126	72	205	76	284	80	363	82
48	48	127	72	206	76	285	80	364	82
49	49	128	72	207	76	286	80	365	82
50	50	129	72	208	76	287	80	366	82
51	51	130	72	209	76	288	80	367	82
52	52	131	72	210	76	289	80	368	82
53	53	132	72	211	76	290	80		
54	54	133	72	212	76	291	80		
55	55	134	72	213	76	292	80		
56	56	135	72	214	77	293	80		
57	57	136	72	215	77	294	80		
58	58	137	72	216	77	295	81		
59	59	138	72	217	77	296	81		
60	60	139	72	218	77	297	81		

61	61	140	72	219	77	298	81		
62	62	141	72	220	77	299	81		
63	63	142	72	221	77	300	81		
64	64	143	72	222	78	301	81		
65	65	144	72	223	78	302	81		
66	66	145	72	224	78	303	81		
67	67	146	72	225	78	304	81		
68	68	147	72	226	78	305	81		
69	69	148	72	227	78	306	81		
70	69	149	72	228	78	307	81		
71	69	150	72	229	78	308	81		
72	69	151	73	230	78	309	81		
73	69	152	73	231	78	310	81		
74	69	153	73	232	78	311	81		
75	69	154	73	233	78	312	81		
76	69	155	74	234	78	313	81		
77	69	156	74	235	78	314	81		
78	70	157	74	236	78	315	81		
79	70	158	74	237	78	316	81		

2.4 Daily Traffic Generation And “Price Category” Based Uckey Splitting

Up to 2.3, traffic measured is hourly.

uckey	count_array	hour
splash,15e9ddce941b11e5bdec00163e291137,WIFI,g_m,6,CPM,184	['1:1']	6
splash,15e9ddce941b11e5bdec00163e291137,WIFI,g_m,6,CPM,184	['1:1']	8
splash,15e9ddce941b11e5bdec00163e291137,WIFI,g_m,6,CPM,184	['2:1']	9
splash,15e9ddce941b11e5bdec00163e291137,WIFI,g_m,6,CPM,184	['2:2','1:1']	10
splash,15e9ddce941b11e5bdec00163e291137,WIFI,g_m,6,CPM,184	['2:2']	11
splash,15e9ddce941b11e5bdec00163e291137,WIFI,g_m,6,CPM,184	['2:1']	14
splash,15e9ddce941b11e5bdec00163e291137,WIFI,g_m,6,CPM,184	['1:1']	14
splash,15e9ddce941b11e5bdec00163e291137,WIFI,g_m,6,CPM,184	['1:1','2:1']	15
splash,15e9ddce941b11e5bdec00163e291137,WIFI,g_m,6,CPM,184	['2:1']	18
splash,15e9ddce941b11e5bdec00163e291137,WIFI,g_m,6,CPM,184	['2:2']	19
splash,15e9ddce941b11e5bdec00163e291137,WIFI,g_m,6,CPM,184	['1:1','2:1']	21

1. Hourly traffic -> daily traffic
2. Each uckey split into 4 uckey based on price category

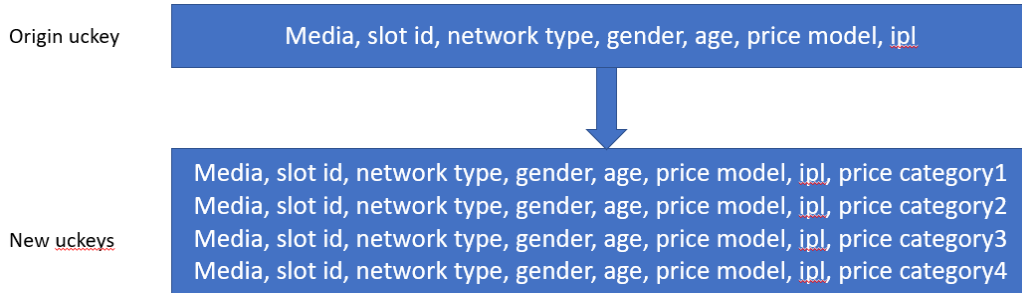


uckey	2020-06-01
splash,15e9ddce941b11e5bdec00163e291137,WIFI,g_m,6,CPM,184,0	0
splash,15e9ddce941b11e5bdec00163e291137,WIFI,g_m,6,CPM,184,1	6
splash,15e9ddce941b11e5bdec00163e291137,WIFI,g_m,6,CPM,184,2	11
splash,15e9ddce941b11e5bdec00163e291137,WIFI,g_m,6,CPM,184,3	0

In this step, as indicated in the figure above, there are 2 steps to be finished

1. Price category is appended as an additional field at the end of each uckey, since there are 4 price categories, i.e. each uckey from 2.3 is then split into 4 different uckey.

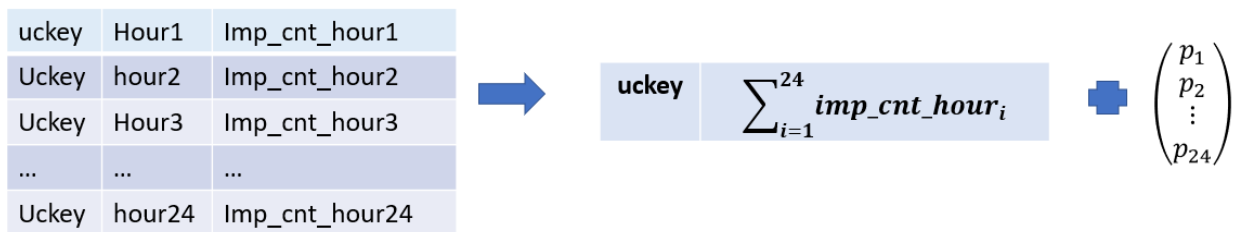
Due to the requirement that UI console needs to query impression count at price category level. It is then necessary to split each distinct uckey defined in factdata into its price category-based version. The naming convention of uckey is changed from uckey to uckey + price_categories, as shown below.



For example,



2. For each (uckey, price category), hourly traffic is aggregated into daily traffic. The % of each hour's traffic contributes to aggregated daily traffic is also computed.



2.5 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.

3. Ucdoc Preprocessing

3.1 Concept Definition – Popularity, Popularity_Norm, Nz_Cnt_Ratio

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

$$\begin{pmatrix} uckey_1 & v_{1,t1} & v_{1,t2} & \cdots & v_{1,tn} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ uckey_m & v_{m,t1} & v_{m,t2} & \cdots & v_{m,tn} \end{pmatrix}$$


$$popularity_i = mean(v_{i,:})$$

$$popularity_norm_i = \frac{popularity_i - \overline{popularity}}{std(popularity)}$$

Where $t1, t2, \dots, tn$ are time points, i.e. *day1, day2, ..., dayn*. $v_{1,t1}$ is the daily traffic for *uckey1* at *t1* (day1), and so on for $v_{1,t2}, v_{1,tn}, v_{m,t1}, v_{m,t2}, v_{m,tn}$

$$nz_cnt_ratio_i = \frac{\sum_{j=t1}^{tn} f(v_{i,j})}{n}, \text{ where } f(x) = \begin{cases} 1 & |x| > 0 \\ 0 & \text{else} \end{cases}$$

3.2 Bad Ucdoc Removal

Even after data cleaning, aggregation steps defined in 1.2, further preprocessing steps are needed for a few reasons

1. Most of the uckey are rarely used and their daily traffic is so sparse that can be treated as noise in model training.
2. in 2.4, each uckey was split into 4 uckey based on price category, however, some price category has none or very little traffic for all uckey, their corresponding ucdoc need to be eliminated
3. 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.

Uckey meet the above 3 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 uckey need to be removed.

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

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

- a. popularity < th=5

Th	# of bad ukeys	% 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%

b. $popularity > \overline{popularity}$ and ratio of non-zero data point $< th = 0.15$

Th	# of bad ukeys	% of excluded traffic
0.15	0	0%

3.2 Sparse Ukeys 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 0.15, i.e.

If $popularity_norm_i < 0.01$ or $nz_cnt_ratio_i < 0.15$, then $ukey_i$ is sparse ukey
else $ukey_i$ is dense ukey

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

3.3 Group Sparse Ukeys Into Virtual Dense Ukey

Based on the step defined in 1.3.2, all ukey will be marked as either dense ukey or sparse ukey, we then need to group sparse ukeys into virtual dense ukeys. Grouping all sparse ukeys into single virtual dense ukey will generate one extremely dense ukey and may not be a good idea, we want to group them into “normal dense ukey” as the major of those marked as dense ukeys.

There’s no change needed to dense ukeys.

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

$$m = \frac{\text{median}(\text{popularity}_{\text{dense_uckey}})}{\text{mean}(\text{popularity}_{\text{sparse_uckey}})}$$

Here are the steps of grouping sparse ukeys into virtual dense ukey

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

3.4 Accumulation Of Traffic From Sparse Ukeys

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

$$\begin{pmatrix} \text{uckey}_1 & v_{1,t1} & v_{1,t2} & \cdots & v_{1,tn} \\ & \vdots & & \ddots & \vdots \\ \text{uckey}_m & v_{m,t1} & v_{m,t2} & \cdots & v_{m,tn} \end{pmatrix} \quad (6.1.5.1)$$

The virtual ukey's data will be

$$\left(\text{virtual_uckey} \quad \sum_{i=1}^m v_{i,t1} \quad \sum_{i=1}^m v_{i,t2} \quad \cdots \quad \sum_{i=1}^m v_{i,tn} \right) \quad (6.1.5.2)$$

And their corresponding distribution probability

$$P = \begin{pmatrix} p_1 \\ p_2 \\ \vdots \\ p_m \end{pmatrix} \quad (6.1.5.3)$$

Where

$$p_i = \frac{\sum_{t=t1}^{t=tn} v_{i,t}}{\sum_{i=1}^m \sum_{t=t1}^{t=tn} v_{i,t}} \quad (6.1.5.4)$$

3.5 Generation Of Virtual Ukey(s)

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

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

$$\begin{pmatrix} f_{1,1} & f_{1,2} & f_{1,3} & \cdots & f_{1,n} \\ & \vdots & & \ddots & \vdots \\ f_{m,1} & f_{m,2} & f_{m,3} & \cdots & f_{m,n} \end{pmatrix} \quad (6.1.6.1)$$

The generated *virtual_uckey* can be expressed as

$$\left(\left\lfloor \frac{\sum_{i=1}^m f_{i,1}}{m} \right\rfloor \left\lfloor \frac{\sum_{i=1}^m f_{i,2}}{m} \right\rfloor \left\lfloor \frac{\sum_{i=1}^m f_{i,3}}{m} \right\rfloor \dots \left\lfloor \frac{\sum_{i=1}^m f_{i,n}}{m} \right\rfloor \right) \quad (6.1.6.2)$$

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 ukey 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 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 ukeys, 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.

3.7 DL-Predictor Pre-Processing Execution

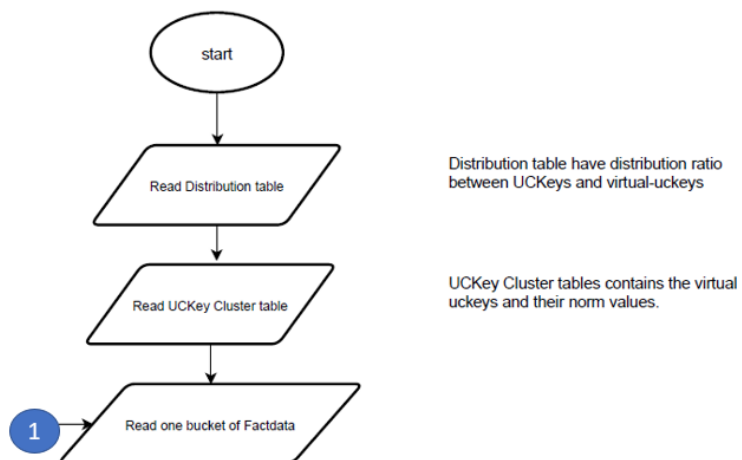
Data pre-processing to train DL-Predictor is pipeline of several processes. Each process is a spark base job and is initiated by spark script. The pipeline can be also run by Airflow.

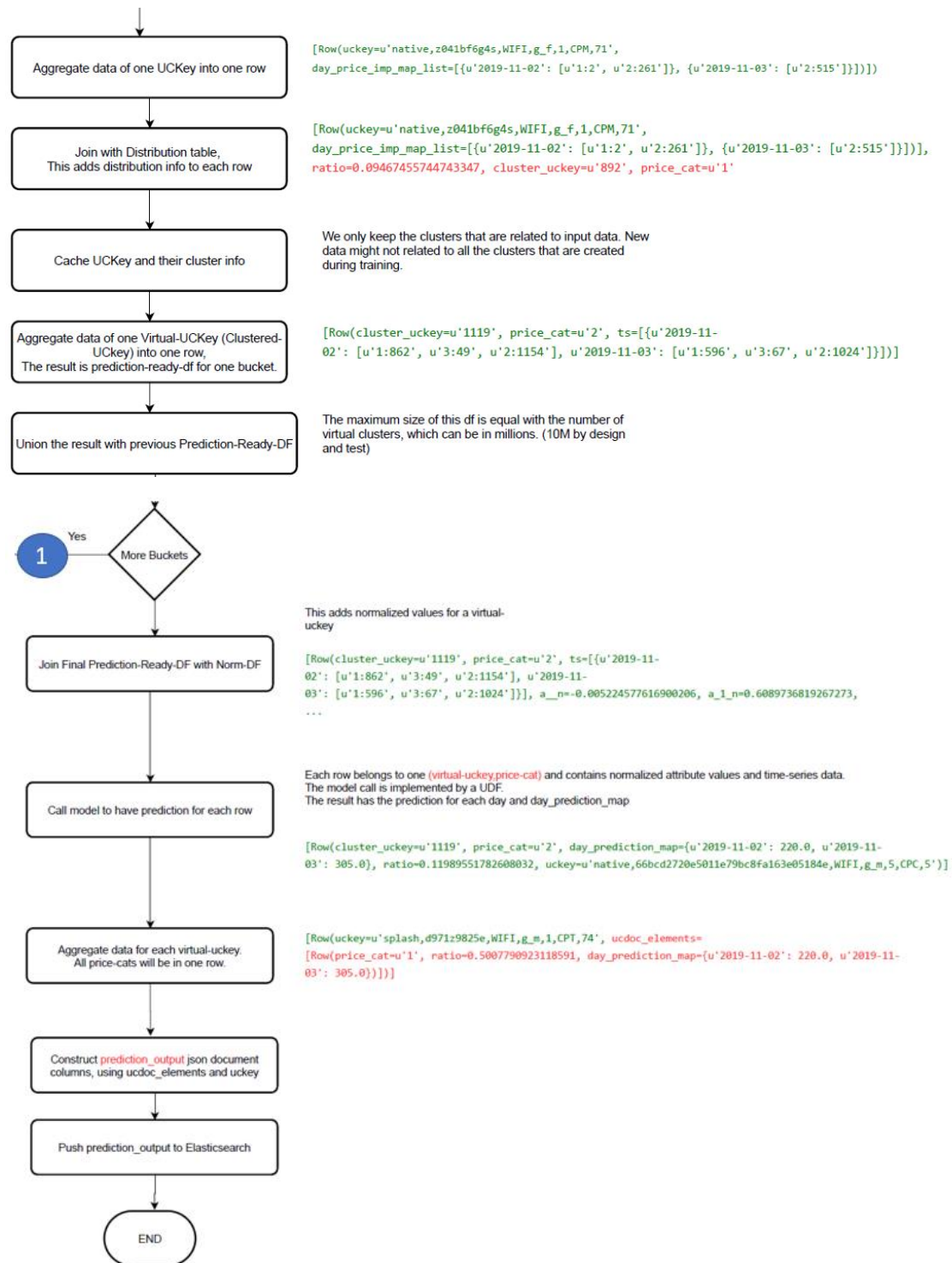
Here is the sequence of the individual processes that are required to run pre-processing.

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



3.8 DL-Predictor Post-Processing Flowchart





4. 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.

5. 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 similar to 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 feeded into training directly, most of the batches will be from sparse uckey which will make the training shaking and hard to converge.

Aggregating sparse uckey into virtual dense uckey 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 uckey, thus it is quite important to keep in mind that the total traffic from sparse uckey 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 uckey.

6. Problems With Current Dataset

There are a few problems we found for current set of factdata

6.1 One Ucdoc Maps To Multiple Bucket_Id

uckey	bucket count	Bucket_id list:
banner,b6gm3hgaux,4G,,,CPM,,	17	122, 143, 157, 293, 368,
banner,b6gm3hgaux,4G,,,CPM,,1	17	37, 380, 389, 404, 408,
banner,b6gm3hgaux,4G,,,CPM,,264	17	422, 54, 607, 630, 709,
banner,b6gm3hgaux,4G,,,CPM,,60	17	734, 931
banner,b6gm3hgaux,4G,,,CPM,,271	16	277,283,303,396,460,
banner,b6gm3hgaux,4G,,,CPM,,28	16	582,587,638,67,689,
banner,b6gm3hgaux,4G,,,CPM,,29	16	697,700,787,844,861,
banner,b6gm3hgaux,4G,,,CPM,,298	16	914,966
banner,b6gm3hgaux,4G,,,CPM,,3	16	
banner,b6gm3hgaux,4G,,,CPM,,4	16	
banner,b6gm3hgaux,4G,,,CPM,,40	16	
banner,b6gm3hgaux,4G,,,CPM,,45	16	
banner,b6gm3hgaux,4G,,,CPM,,8	16	

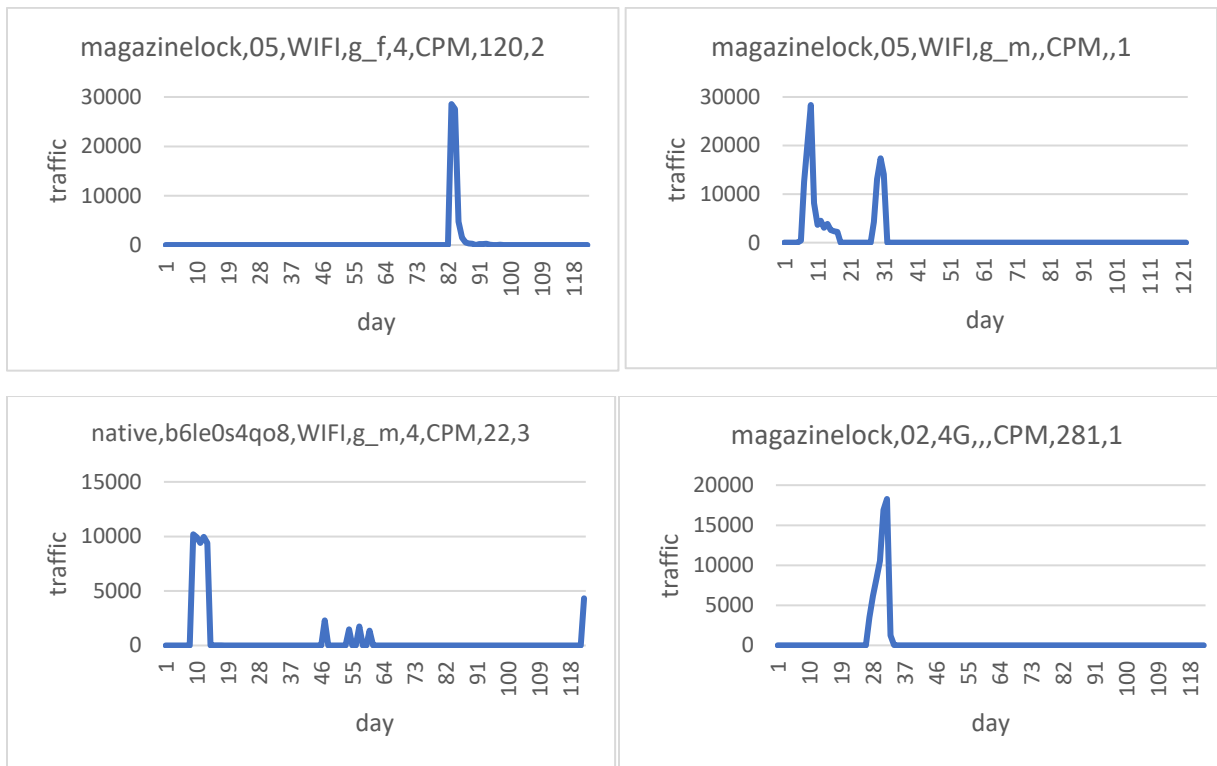
6.2 Majority Uckey From Unstable Slot_Id

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

Day	All uckey #	Stable slot_id uckey #
2020-03-01	14,068,490	5,295,172
2020-04-01	16,552,644	6,562,907
2020-05-01	18,456,889	7,471,808
2020-06-01	15,706,330	6,019,000

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 uckey are quite unstable even through they are from stable slot_ids.



7. 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.

7.1 Is price_cat part of uckey formula?

Answer:

In section 2.4, ukey formula is defined as:

Ukey = adv_type, slot_id, net_type, gender, age, pricing_type, IP location, price_cat

In the implementation code, ukey formula does not include "price_cat" which raised the code compatibility concern.

For each ukey, 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 ukey formula, in implementation codes, the (ukey, price_cat) tuple is used to implement the discriminant.

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

7.2 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:

```
assert 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:

```
assert 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.

7.3 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_ukeys: 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 $m \times n$ (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.

3. 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.

7.4 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.

7.5 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
```