Measuring and Identifying Non-diligence

We first describe the ideas in our method and also point to the code.

What is a non-diligence probability?

Our first main idea is to get a probability of non-diligence for each rule, given the data from an ANM. We use the 18 rules given by Khushibaby. **These rules are of two types** (1) rules that apply to one health camp and (2) rules that track longer term phenomenon. All rules specify a percentage of something and we know which extreme (0% or 100%) corresponds to non-diligence. As a running example we will use two rules:

Example healthcamp rule: % of Blood Pressure readings that are 120/80 or 110/70. We know that higher percentage corresponds to non-diligence.

Example long-term rule: % of child deaths. We know that towards 0% is non-diligence.

Handling of health-camp rule:

I describe here how we extract non-diligence probability using the BP rule – rest for other rules are similar or flipped where 0% is non-diligence. In BP rule 100% is non-diligent behavior with probability 1. We look at the data for each ANM for each heath camp to obtain the percentages for any rule. We filter out percentage that are exactly 0% or 100% - as these are for sure not diligent (or dilgent). For the remaining percentage we plot a density distribution using KDE (kded1d library in R). The location of this code is:

Code filename: diligenceprediction/rfunc/helpers.r

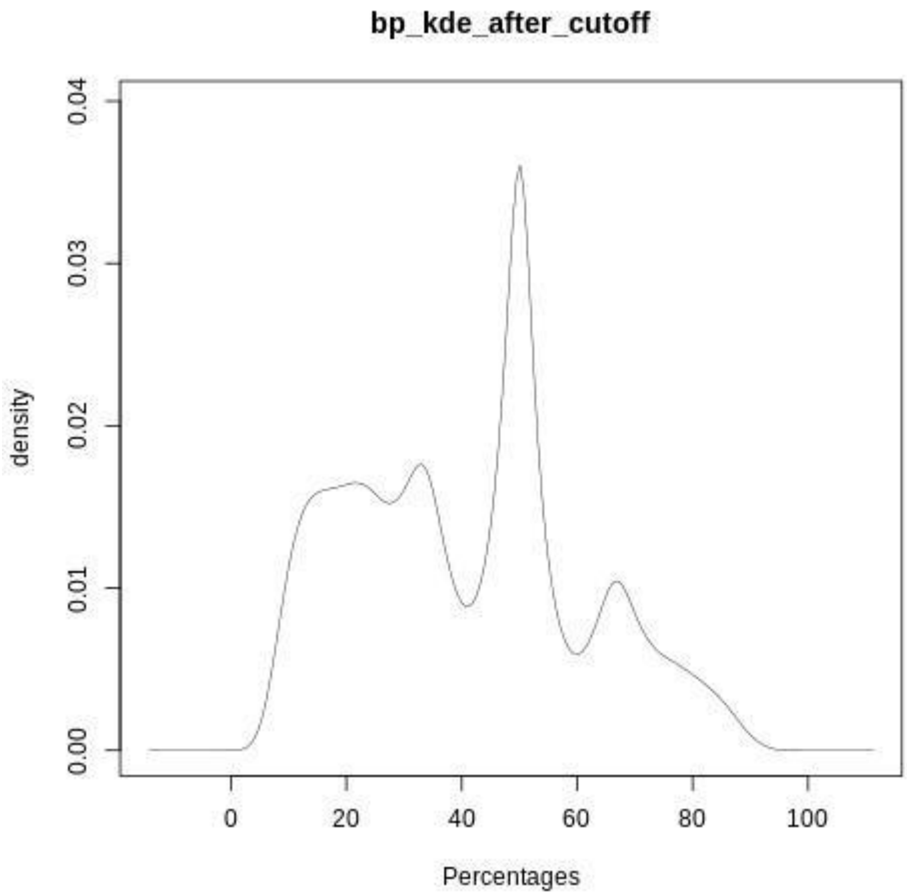
function name: func_cut_off_clusters

Then, given the percentage, say x, for a heath camp for an ANM – the probability of non-diligence p is the probability mass between (0,x). Clearly as x increases, the probability of non-diligence is increasing and is exactly 1 when percentage is 100. The location of this code is:

Code filename: diligenceprediction/rfunc/helpers.r

function name: func_get_prob_mass_trans

See figure below for example for the BP rule:



Aggregate over one month: the health camps have lot of variability so instead of looking at one health camp we consider the average of probability of non-diligence p over all health camps in 4 weeks. **This is the short-term rule non-diligence probability**. The aggregation code is at:

Code filename: diligenceprediction/ruleskde/shortrangerules.py

function name: getBPsdRuleData

Handling of long-term rule:

These are evaluated on data for past 6 months (1 month = 4 weeks) since these rules track events that happen in a longer term. I describe the child death rule here – it is actually very similar to BP rule except the non-diligence extreme is flipped. We look at the 6 month data for each ANM to get the percentages for any long term rule. We filter out percentage that are exactly 0% or 100% - as these are for sure non-diligent (or not). For the remaining percentage we plot a density distribution using KDE (kded1d library in R). The location of this code is:

Code filename: diligenceprediction/rfunc/helpers.r

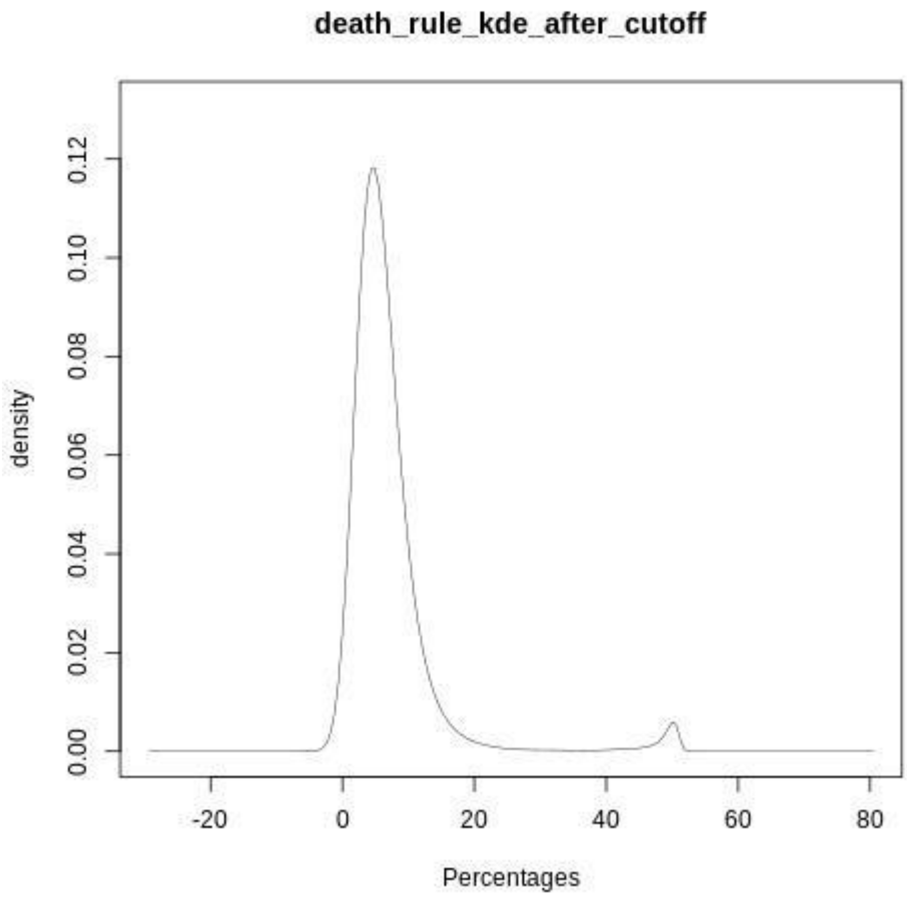
function name: func_cut_off_clusters

Then, given the percentage, say x, for a heath camp for an ANM – the probability of non-diligence p is the probability mass between (x,100). Clearly as x increases, the probability of non-diligence is decreasing and is exactly 0 when percentage is 100 (actually for this rule it is zero much before 100 as the max percentage is low). The location of this code is:

Code filename: diligenceprediction/rfunc/helpers.r

function name: func_get_prob_mass_trans

See figure below for example for child death rule:



The location of the code of processing death rule is:

Code filename: diligenceprediction/ruleskde/longrangerules.py

function name: deathRule

The behaviors of ANM over time:

We compute the vector of non-diligence probabilities (short-term and long-term rules, 18 in total) for an ANM at multiple time points that are separated by one month (sliding window of one month). For each ANM we get 35 such vectors of non-diligence probabilities. There are 85 ANMS. So, we should get 35x85 vectors of probabilities, where each vector is 18-dimensional but due to some missing data we get only 1455 such vectors. The location of this code is

Code filename: diligenceprediction/features/features.py

function name: get_fraud_probabilities

Clustering:

We perform clustering on the 35x85 vectors into two clusters. We tried 3 clusters first, but two seems better**.** We tried two clustering techniques: kernel k-means and fuzzy c-means**. Look at the two cluster centers for k-means (see below)**, we can clearly see that one cluster appears to be non-diligent. K-means gives us binary {0,1} output about each data point (a data point is probability vector) whether it belong to one cluster or other. C-means gives us [0,1] output about each data point indicating the probability of whether a data point belongs to one cluster or another. **We treat these clusters as the ground truth for either classification {0,1} or regression [0,1] (see appendix for comparison with ANM rank provided by Khushibaby)**.

Kernel k-means		
Rule number	cluster1	cluster2
0	0.279056	0.279538
1	0.938181	0.850939
2	0.97323	0.976695
3	0.815536	0.860463
4	0.993748	0.994306
5	0.01093	0.015201
6	0.00256	0.004474
7	0.008157	0.010944
8	0.034295	0.047189
9	0.007529	0.006947
10	0.030473	0.037413
11	0.030534	0.037461
12	0.258926	0.900015
13	0.613204	0.855198
14	0.339806	0.924078
15	0.00389	0.01652
16	0.878486	0.869055
17	0.797263	0.812241
Average of column	**0.389767**	**0.472149**

Cmeans		
Rule number	cluster1	cluster2
0	0.286711	0.259713
1	0.899121	0.880162
2	0.974468	0.97788
3	0.825135	0.865867
4	0.994514	0.994218
5	0.012445	0.013637
6	0.003431	0.003693
7	0.009842	0.009056
8	0.040303	0.04167
9	0.007088	0.006759
10	0.035159	0.031454
11	0.035219	0.031496
12	0.457557	0.86247
13	0.669221	0.860004
14	0.518352	0.88803

15	0.009751	0.0123
16	0.874455	0.881757
17	0.796217	0.829739
Average of column	**0.413833**	**0.469439**

Note that the clustering is done for once, the clusters are not changed as new data comes in. The balance of clusters (points in each cluster) is good. We want the cluster reference to be fixed. For example, if in new data all ANMs behave good then all new data points will lie in one cluster according to the fixed cluster we have but if we add these new data points and re-cluster that would make imbalanced clusters.

The code for this part is at: https://colab.research.google.com/drive/1-u6cpVu06awivaWUMCXcTjZMK3gzfA6_?usp=sharing
K-means: In the classification section
C-means: In the regression section under clustering

Predicting Non-diligence

We tried the following features

1. The number of patients expected in the prediction window
2. Number of camps expected to be conducted by the ANM in the window (assuming if there are more camps -> high workload -> more chance for non-diligence?)
3. Expected average time duration between 2 camps (similar to the above assumption, frequent camps -> high chance to cheat and ANM might replicate data from other camps?).
4. The expected number of camp locations in the window
5. Non-diligent probability vectors of past 6 months of the ANM (6 vectors in all)

Our final implementation only used feature 5 above as that is good enough by itself.

- **Our classification prediction is 0/1 which is whether the next month** (1 month = 4 weeks) **non-diligence vector is in the non-diligence cluster or not.**
- **Our regression prediction is 0-1 which is probability of whether the next month** (1 month = 4 weeks) **non-diligence vector is in the non- diligence cluster or not.**

We split data into training and test by time – note that time series data is split by time, not randomly. Note below that training data starts much after the actual data available from Feb 2017. This is because we need 6 month (1 month = 4 weeks) fraud vectors and also for any fraud vector we further needs 6 months data from past.

Training:	2017-12-27	2019-06-11
Testing:	2019-12-25	2020-03-17

This training/test code is not in the package for Khusibaby, as we do not retrain in production. The production uses the already trained model. But, for reference the code for training/testing is at:

Colab workspace for all code: https://colab.research.google.com/drive/1-u6cpVu06awivaWUMCXcTjZMK3gzfA6_?usp=sharing

The production just does inference with the pre-trained model. The inference code is at:

Github repo for code: https://github.com/RameshaKaru/diligence-prediction

Code filename: diligenceprediction/hdps.py

function name: predict_scores_next

The results:

Classification results are below (with kkm – kernel k-means)

	Jan	Feb	Mar	July	August
Accuracy	0.96	0.96	0.86	0.95	0.91
Class 0 recall	0.9	0.88	0.82	0.87	0.78
Class 0 precision	1	1	0.82	0.93	0.78
Class 1 recall	1	1	0.88	0.98	0.95
Class 1 precision	0.95	0.94	0.88	0.96	0.95

Regression results are below (with fcm – fuzzy c-means)

	Jan	Feb	Mar	July	August
MSE	0.0112	0.008	0.0152	0.0111	0.0115
R2 score	0.7644	0.8355	0.6359	0.7048	0.5836

Appendix:

% of ANM behaviors that are in each cluster. Cluster 2 is non-diligence cluster. Many ANMs have behaviors that lie in both clusters. The ranks of ANM are also shown – there is no clear correlation with cluster.

	sub_center_id	rank	cluster1	cluster2
0	4	rank1	42.85714	57.14286
1	5	rank1	91.42857	8.571429
2	7	no rank	20	80
3	14	rank1	97.14286	2.857143
4	20	rank1	100	0

5	24	rank3	22.85714	77.14286
6	25	rank3	77.14286	22.85714
7	26	rank3	62.85714	37.14286
8	33	no rank	0	100
9	34	rank3	0	100
10	36	rank1	88.57143	11.42857
11	37	rank1	88.57143	11.42857
12	38	rank3	100	0
13	39	rank1	100	0
14	45	rank3	68.57143	31.42857
15	48	rank3	31.42857	68.57143
16	53	rank3	22.85714	77.14286
17	55	rank1	2.857143	97.14286
18	58	rank1	40	60
19	65	rank3	0	100
20	67	no rank	25.71429	74.28571
21	69	rank2	0	100
22	73	rank2	0	100
23	81	rank3	0	100
24	82	rank1	0	100
25	90	rank3	2.857143	97.14286
26	93	rank3	14.28571	85.71429
27	103	rank1	0	100
28	104	no rank	0	100
29	108	rank3	0	100
30	109	rank3	17.14286	82.85714
31	111	rank2	0	100
32	114	rank3	77.14286	22.85714
33	115	rank3	17.14286	82.85714
34	117	rank3	0	100
35	118	rank2	5.714286	94.28571
36	121	rank2	34.28571	65.71429
37	122	rank1	0	100
38	123	rank3	34.28571	65.71429
39	125	rank3	34.28571	65.71429

40	126	rank3	54.28571	45.71429
41	131	rank2	37.14286	62.85714
42	145	rank2	0	100
43	146	rank2	60	40
44	147	rank2	42.85714	57.14286
45	148	rank1	25.71429	74.28571
46	149	rank1	5.714286	94.28571
47	154	rank2	0	100
48	155	rank3	48.57143	51.42857
49	156	rank3	20	80
50	159	rank1	20	80
51	161	no rank	0	100
52	165	rank2	100	0
53	167	rank1	0	100
54	169	rank3	14.28571	85.71429
55	170	rank2	42.85714	57.14286
56	177	rank2	100	0
57	178	rank2	71.42857	28.57143
58	181	rank2	51.42857	48.57143
59	195	no rank	40	60
60	201	rank1	0	100
61	203	no rank	22.85714	77.14286
62	207	rank3	0	100
63	210	rank2	45.71429	54.28571
64	211	no rank	37.14286	62.85714
65	212	rank1	0	100
66	216	no rank	17.14286	82.85714
67	217	no rank	0	100
68	226	rank1	45.71429	54.28571
69	228	rank3	62.85714	37.14286
70	231	rank2	0	100
71	232	rank2	5.714286	94.28571
72	233	rank1	14.28571	85.71429
73	235	rank3	14.28571	85.71429
74	237	rank2	22.85714	77.14286

75	239	rank1	31.42857	68.57143
76	240	rank1	31.42857	68.57143
77	241	rank2	22.85714	77.14286
78	243	rank2	37.14286	62.85714
79	247	rank2	20	80
80	252	rank2	40	60
81	256	rank1	0	100
82	257	rank1	8.571429	91.42857
83	259	rank3	0	100
84	308	rank2	91.42857	8.571429