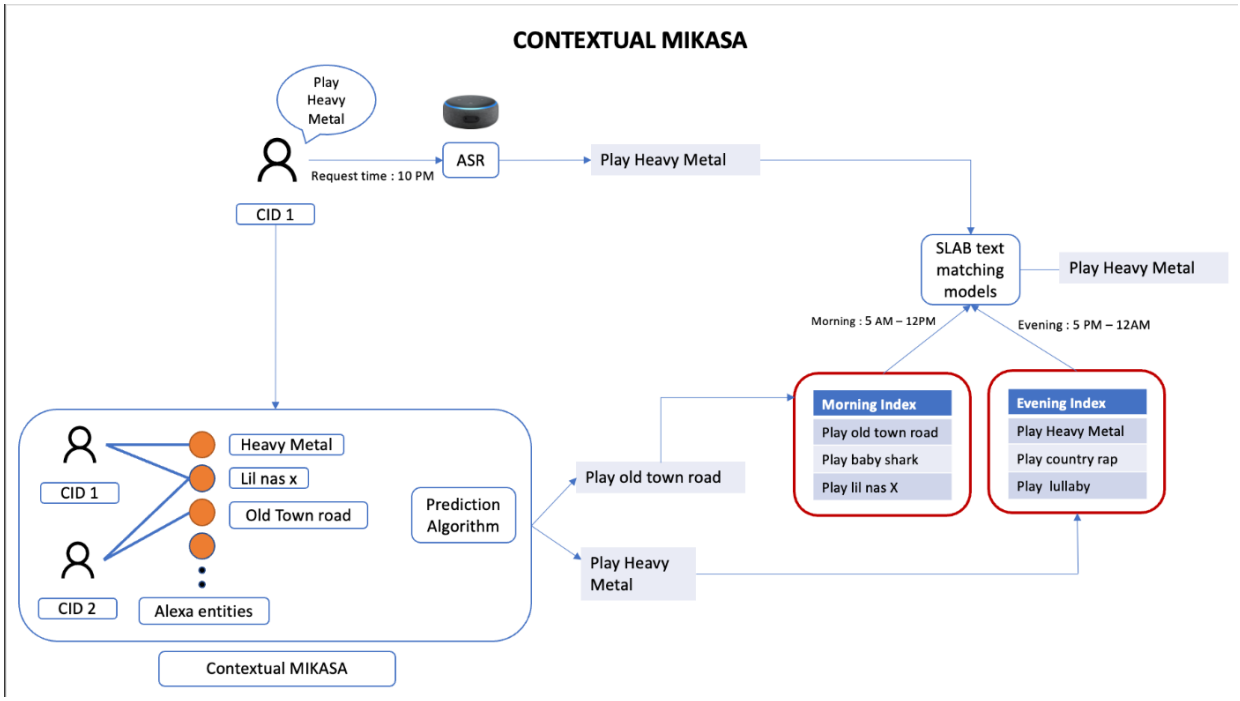
Does Context Matter? : Incorporating personalized contextual information for correcting unseen defects in MIKASA

Members :

Manager : Oleg Poliannikov
Mentor : Manik Bhandari
Intern : Pratik Aher

Summary :



The goal of the project is to identify contextual signals and incorporate those signals to improve customer defects on unseen requests. After initial exploration, we found “user local-time of the day” was most important for us to improve future entity predictions.

For every user, MIKASA generates a list of future predictions as “customer index”, which is a predicted list of entities that a user is likely to request in the future. We propose a framework where instead of having a single index, we have multiple indexes corresponding to context categories. For example, separate customer indexes for morning and evening context. These customer indexes are being populated based on the scoring from a novel random walk based algorithm which takes into account contextual values.

With the same treatment for the training and testing data, we found our framework, which includes context, shows a relative improvement in the number of correct predictions of ~**4%** over current MIKASA. Furthermore, MIKASA seems to have a popularity bias in its results. Our framework has higher recall on torso and tail entities. Beyond top 100 head entities(popular globally), our framework shows ~**18.5**% improvement over current MIKASA. Beyond top **200** head entities, we can see our new algorithm, shows an improvement of ~**37.5%** percent, and so on.

This is an indication that including context helps us predict entities which are more personal/nuanced entities rather than popular entities.

Background:

The goal of project MIKASA is to reduce customer defects by anticipating their future requests. MIKASA uses state-of-the-art collaborative filtering to anticipate a customer's future requests. This is done by modeling a customer's relationship with popular entities (Song, Artist, Genres, etc).

MIKASA instantiates a simplified version of Pixie which essentially runs random walks on the graph and accumulates visit counts for each visited node. MIKASA aims to answer the question, "customers who like this entity also like...?". We would essentially be trying to go one step further and ask the question, "customers who like this entity in this context (**like morning**) also like... in the same context?". Mikasa's prediction algorithm consumes an interaction graph consisting of Alexa customers and entities and applies collaborative filtering to predict future customer requests.

Neither Pixie nor MIKASA take into account additional contextual signals like time of the day, Weekday/Weekend, Device Type, etc between a user and an entity. We believe exploring these features might provide better recommendations that can improve a customer's defects rate for unseen requests.

We are not limited by the number of customer indexes that we can have for a single customer. We are limited by the index size as we have ~10ms to go through a customer index and respond. "Morning Index"/ "Evening Index"/"MIKASA index" all have the same size. The advantage of having multiple indexes comes in because we can choose to use different indexes at different times of the day. So choosing the correct index for the correct context at request time gives us better and more personalized predictions.

Here is a link to current MIKASA's documentation : MIKASA MUSIC

Data :

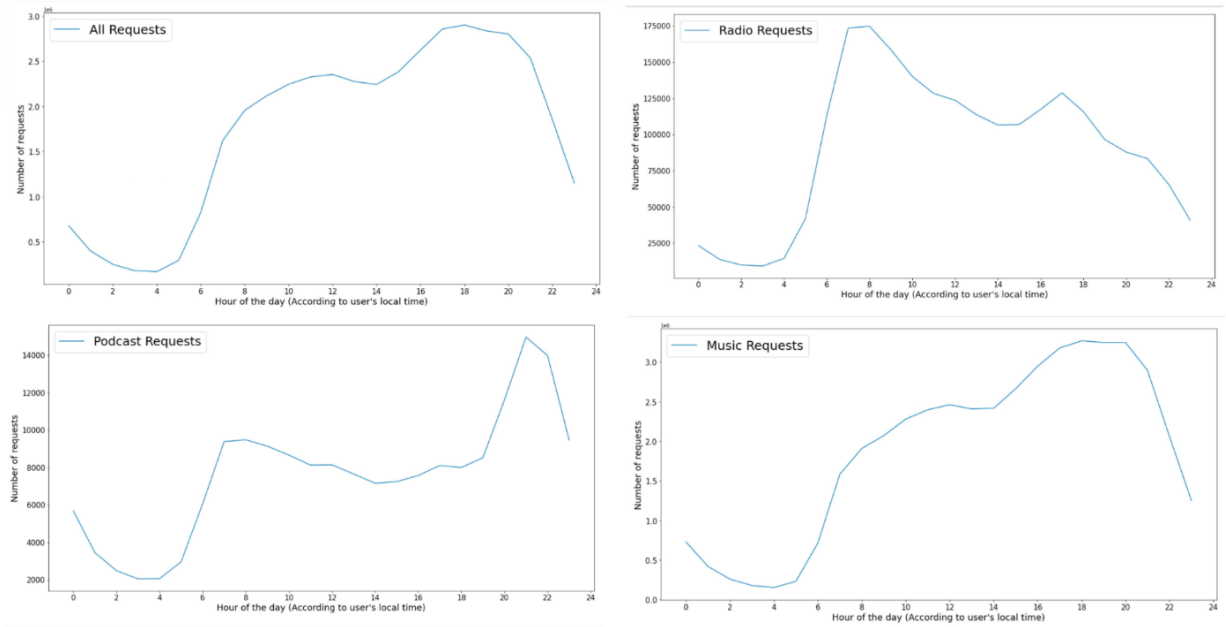
Here are the following features we had planned to explore as context/signals :

- **User local-time of the day** : User behaviors may change depending on the time of day. Example : Users request more radio stations in the morning, and lullabies at night.
- **Day of the Week** : Day of the week. E.g. users are more likely to request long form podcasts channels during weekends.
- **DeviceType** : differentiate based different Alexa devices, like FireTV and Echo, to see if customer request different stuff of different Alexa devices.
- **Active Video Session** : To check if a users request specific items during active a video session.

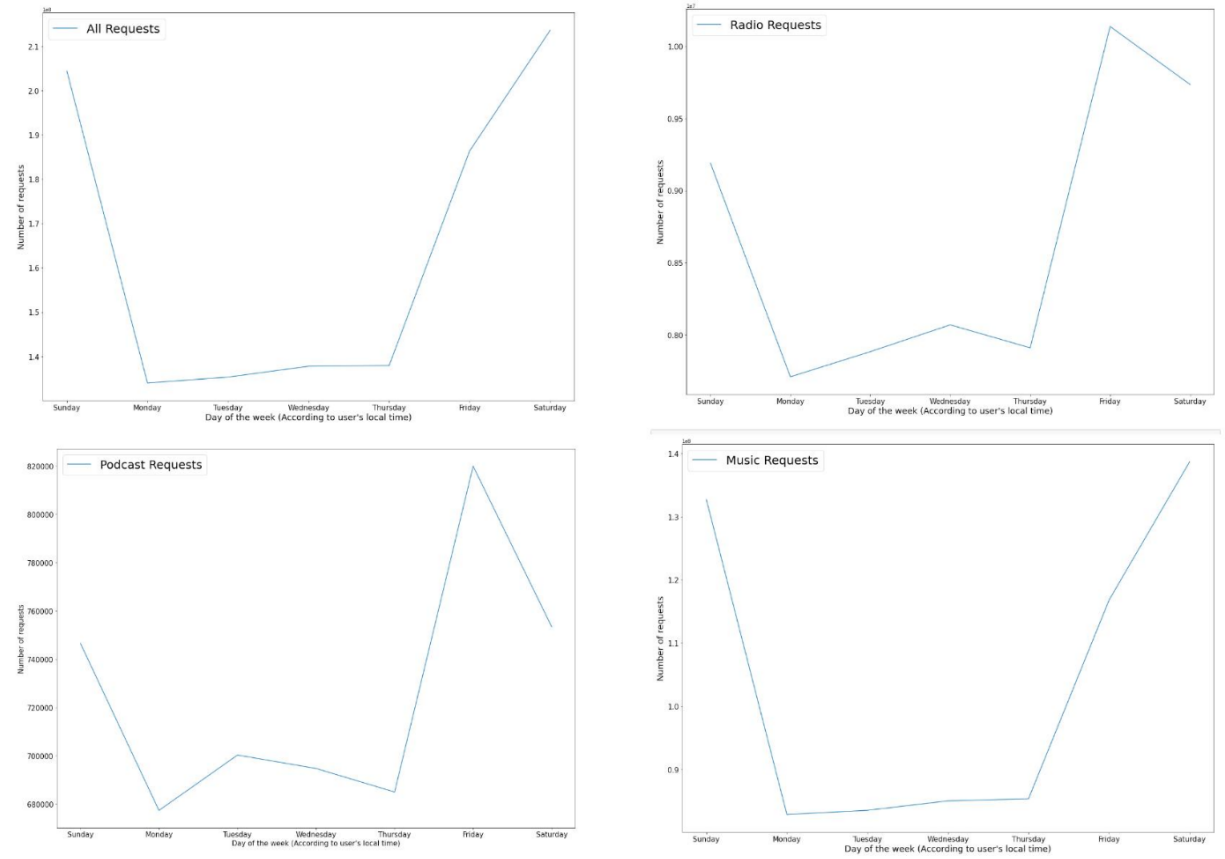
After analysis, we found "**User local-time of the day**" to be the most important for this project.

Analysis and Validation of Contextual Variables :

This is the analysis for the number of requests coming in depending on user time of the day. The data is averaged across weekdays for a month.



We can see users request different entity types depending on the time of the day. For example, we can see a spike in podcast requests around 11pm at night, and radio channels before 8am before the start of a workday.



Here is the analysis for number of requests according to the day of the week. We can see user behavior does not change a lot for this particular context. Requests for all entity types peak around the weekend, and remain fairly stable on weekdays. This is

an indication that this might not be a strong variable to use for our project. This is the reason why we did not explore this context beyond initial analysis.

The context we explored for this project after analysis was “User local-time of the day”, which is the time of the day a user requests for an entity. We divided the user local time of the day into two categories, “**Morning**” and “**Evening**”, where :

- Morning : 5 AM - 12 PM
- Evening : 5 PM - 12 AM

Here is the process to get “User local-time of the day” : Process to get local time of the day

HYPOTHESIS TESTING FOR - MORNING/EVENING CONTEXT

For an interaction between a user and entity, let's say the number of morning edges is 3 and the number of evening edges is 7. Then the probability of the morning edge is 0.3 and the evening edge is 0.7. The difference is then 0.4.
Across all interactions, we found mean difference between Morning/Evening interaction probabilities to be **0.67**, with a standard deviation of **0.332**.

Problem statement :

Do users interact with entities different number of times in the morning and in the evening ? If so, for how many users is the variation of entity requests between morning and evening requests statistically significant ?

When we don’t consider context, we can imagine there are equal number of morning and evening interactions between a user and entity. So, the difference between the probability of an edge being a Morning edge or an Evening edge is approximately 0. (We treat this ground truth for odd and even interactions differently : Detailed explanation)

Again, let’s imagine the number of morning edges is 3 and the number of evening edges is 7. Then the probability of the morning edge is 0.3 and the evening edge is 0.7, and hence, difference is 0.4.

We want to see if this difference is statistically significant, from when we do not use context, for all entities connected to a user.

DATA :

- One months data from 04/15/22 - 05/15/22. (30 days)
- Morning - Hour 5 to 12.
- Evening - Hour 17 to 24.
- Consider only requests that fall in the Morning/Evening category. Every request is either be Morning or Evening depending of time they are requested, all other requests are not considered.

CONDITIONS:

- We consider only those edges where there are **5 or more interactions** between a customer and entity. The reason is , we don’t want to compute significance where there are just few interactions between a customer and entity.
- We consider entities which are requested by 5 or more customers.

TEST :

- Wilcoxon Signed-Rank Test for statistical significance

Null Hypothesis : Users interact with its entities equal number of times in the morning and in the evening (difference = 0).
Alternate Hypothesis : The difference between the number of time a user interacts with its entities in the morning and in the

evening is non-zero.

P-value for significance = 0.05 (if the p-value is less than 0.05 for an entity then we classify the entity as statistically significant).

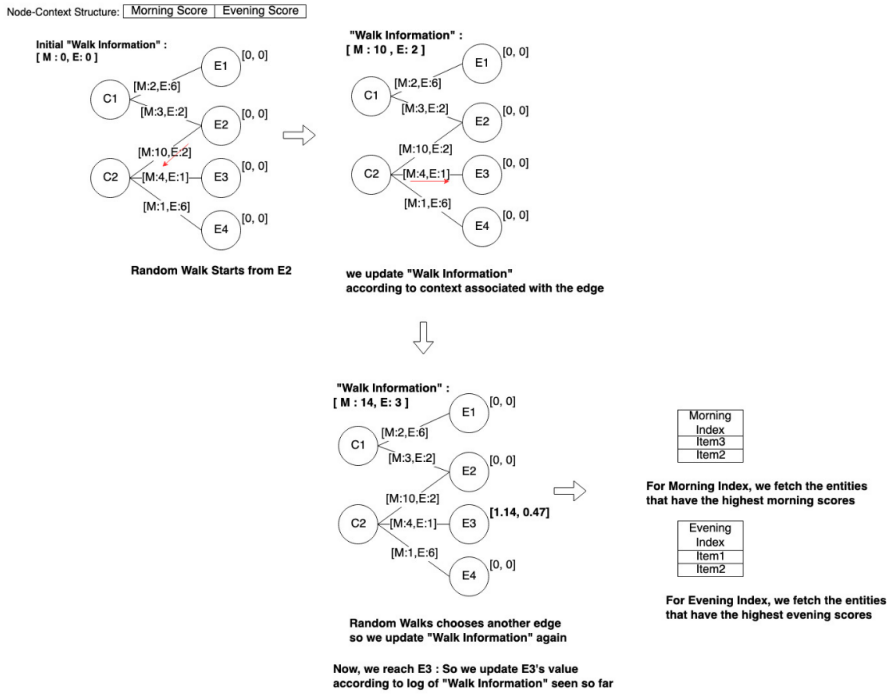
OBSERVATIONS:

Significant_YN	Number of Users
Significant	1720150
Not Significant	117068

> The difference in number of requests a user makes to its entities in morning and evening is statistically significant for 94 percent of users.

More detailed analysis on other contexts, you can view : Context Information - EDA and in the Appendix.

How to use contextual variables in MIKASA : Incorporating walk information for scoring entities using personalized context in MIKASA



The goal is to create multiple indexes for different context types. So, in our case, we have two context types for the context “User local-time of the day”, which are “Morning” and “Evening”. We can use these indexes according to the time of the day a request is coming in. For Instance, for requests coming in, in the morning, then we can use the “Morning Index”, and “Evening Index” when requests coming in, in the evening.

The major difference between the new approach and the current MIKASA is that, we store the edge count of nodes we find on random walks, and use this edge count to create a score for visit count of visited nodes. So, if a random walk to a node only

consists of Morning edges, then the hypothesis is that the node is more relevant for the Morning context.

Approach :

1. Before we start a random walk, we initialize “walk information” of the form [M:0, E:0] to store the context we are exploring.
2. A random walk of length L starts at node N, and selects a neighbor of the node to visit randomly.
3. We then update the “walk information” with the context we see on the edge that goes from current node to the selected node.
4. Instead of just incrementing the count of visited node by 1, as we do in MIKASA, we will update the score of the visited node(Morning/Evening visit count) of the selected node by the “**walk_information”** score we have seen so far. We increment the score by the log of the values, to prevent the “**walk information”** values from exploding to large numbers. For example, if the current walk score when we are at node “**N”** is [M:15, E:5], then increment the “**morning visit count**” of node N by log(15), and “**evening visit count**” of node N by log(5).
5. We then move to the next random neighbor and repeat the process, but keep the walk information we have seen so far for future scoring.
6. We then output “K” nodes with the highest visit count from morning and evening maps. This visit count has contextual information baked into it as the scores are calculated on the basis of context we saw on the random walks.

You can see the pseudocode of the algorithm in the Appendix section : Complete Algorithm

This approach is flexible and can be added to extended to other contextual signals : How to add more contextual variables to the algorithm

Evaluation :

NUMBER OF CORRECTLY PREDICTED ENTITIES :

After augmenting the customer index with entities generated from **contextual MIKASA** for all contextualized indexes**,** how many of the future entities can we anticipate.

Here are the steps for calculating number of correctly anticipated future entities for a single customer:

1. Get future entities for the user, predictions for the the user and historical entities for a particular user.
2. Remove all predicted entities which can be found in history (just like current MIKASA).
3. Fill remaining index space with predicted entities upto available index size (just like current MIKASA).
4. From the future entities, remove entities which do not have any “Morning” or “Evening” interactions. predictions for these entities would not make any sense.
5. Calculate the number of requests from future entities if they are present in regular MIKASA index. This is our “MIKASA Count”.
6. From future entities, create two sets one for “Morning entities” (any entity that has Morning interactions) and one for “Evening entities” (any entity that has Evening interactions).
7. Check how many of the “Morning entities” are in “Morning index”, from those entities consider the number of requests in the morning, this is our “Morning Count”.
8. Similarly, calculate “Evening Count”.
9. Sum up “Morning Count” and “Evening Count” to get “Contextual MIKASA Count”.

Compare percentage difference between “MIKASA Count” and “Contextual MIKASA Count”. This percentage difference is our improvement of one over the other.

ANALYSIS OF CURRENT MIKASA MODEL :

The current MIKASA model seems to have a popularity bias. On an Average, **70%** of the entities in the customer index are entities that are in top 100 overall requested entities.

I took the most popular entities (requested by most number of customers) for one month, and ranked them according to popularity. Here are top few rows of that data :

Entity	Number of Customers requesting the entity	Rank
country|GenreName	2074876	1
we don't talk about bruno|SongName	1053415	2
taylor swift|ArtistName	978767	3
encanto|AlbumName	836910	4
baby shark|SongName	788871	5
lin manuel miranda|ArtistName	771826	6
imagine dragons|ArtistName	754809	7
eighties|GenreName	743724	8
pop|GenreName	722248	9
morgan wallen|ArtistName	698708	10

We define “sum of ranks” of an index as the sum of “Rank”, which is it’s in position in the overall popular table(as we define above), for all entities in a given index. “sum of ranks” of a particular index is good indicator of whether a particular index has popularity bias or not. For an index, that does not have a popularity bias, the sum of the ranks is expected to be huge as this index would contain lowly ranked and personalized entities. This would be in contrast to an index which has a popularity bias, where one would expect “sum of rank” of an index to be low, as the index would have popular entities which have rank closer to 1 as we can see in the table above.

We tried to see the “sum of ranks” for entities in Morning/Evening Index, and the sum of ranks in MIKASA’s Index.
For example, let's assume that the following are the predicted entities in the given indexes :

> Morning Index : [‘***counting starts|SongName***’, ‘***drake|ArtistName***’]
>
> Mikasa Index : [‘***pop|GenreName***’, ‘***baby shark|SongName***’]

For the MIKASA index, ‘*pop*|*GenreName*’ entity has rank 9 and ‘*baby shark*|*SongName*’ entity has rank 5. we sum these ranks. The “sum of ranks” for MIKASA index is , 5+9 = 14.
We do the same for entities in “Morning Index”, we see that “sum of ranks” for entities in “Morning Index” is 25.
We can see that “Mikasa Index”, has a lower “sum of ranks” score than “Morning Index”.

We conducted the “sum of ranks” experiment on a random sample of 1000 users. On an average, we see that the “sum of ranks” for MIKASA is **~17** percent lower than the “sum of ranks” for either of Morning/Evening indexes. This is further proof that MIKASA index has more popular entities compared to Morning/Evening Index.

This information is crucial for our evaluations, as we are testing to see how the model performs on non-popular entities.

Experiments and Results :

The training data for all purposes was T-30 days, and the testing data was T+14 days.

ON SYNTHETIC DATA :

We performed the below experiment as a sanity check to test see whether the algorithm we are using for incorporating this context is correct or not in an ideal scenario.

For this experiment, we imagine a hypothetical situation where entities are purely “Morning” or “Evening”. We assume that every entity is related to all its users with just one kind of edge, “Morning” or “Evening”.

We made all the entities as “Morning entities” until we reached half the total amount of total customers (Note that these entities are shuffled so we are not making the popular entities as one of the category, “Morning” or “Evening”). After that point, we made all the entities as “Evening entities”.

Results:

Sample size : 1000 users

	Contextual MIKASA	Current MIKASA
Number of correct predictions	1133	540

We can see that number of correct predictions of contextual indexes is more than 2 times than that of MIKASA. This was expected as, in this ideal situation playing out, we are taking complete advantage of our contextualized indexes. Due to this experiment, we validated that context in the way of Morning/Evening is works in an a ideal scenario when we use it in our algorithm.

ON REAL USER DATA :

For a sample of 90k users,

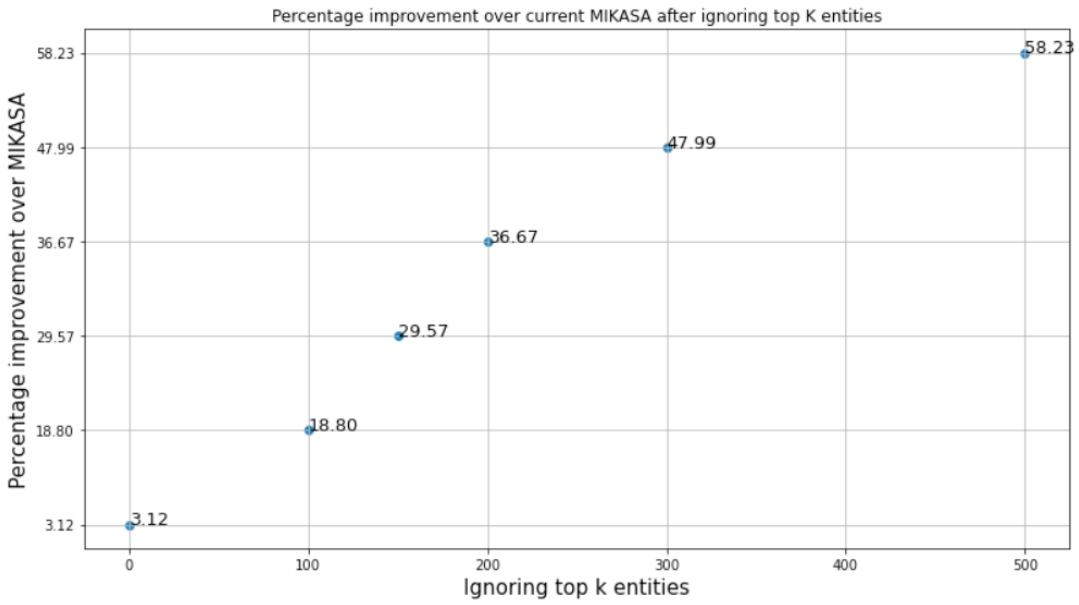
> On direct comparison, we see an improvement of ~4% over current MIKASA.

We took the most popular entities for a month, and ranked them according to popularity. We see that MIKASA has a popularity bias where on an average 70% of the predictions are top 100 entities. This is because top entities are connected by a lot of customers.

> We ignored top 100 most popular entities from future entities and compared number of correctly predicted entities between current MIKASA and Contextual MIKASA, we see an improvement of ~ 18%.

Here is an analysis how much performance improves for non-popular entities (This experiment was conducted for a sample for around 1.1 million users):

Top entities ignored	Number of entities correctly predicted : Contextual MIKASA	Number of entities correctly predicted : MIKASA	Percentage Difference
0	1459189	1413698	3.12%
100	527836	437060	18.80%
150	403606	299611	29.57%
200	324996	221957	36.67%
300	235291	144232	47.99%
300	152172	83539	58.23%



Here are some of the examples of the predictions :

“Past Entities” : historical entities of a user (T-30).
“Future Entities” : entities requested be a user in T+14 days.

- Example 1 :

```
Past Entities:
[('smooth jazz|GenreName', '{"E":0,"M":1}'),
('frank sinatra|ArtistName', '{"E":0,"M":2}'),
('exercise|GenreName', '{"E":0,"M":1}'),
('dean martin|ArtistName', '{"E":0,"M":2}'),
('babbling brook|SongName', '{"E":1,"M":0}'),
('pop rock|GenreName', '{"E":0,"M":0}'),
('jazz|GenreName', '{"E":0,"M":1}')]

Future Entities:  {'nineties hip hop|GenreName': '{"E":1,"M":0}',
'highest in the room|SongName': '{"E":1,"M":0}',
'rain|GenreName': '{"E":1,"M":0}',
'workout|GenreName': '{"E":0,"M":1}',
'cg5|ArtistName': '{"E":2,"M":0}'}

Found in Morning Index: [('workout|GenreName', '{"E":0,"M":1}')]
Found in Evening Index: [('cg5|ArtistName', '{"E":2,"M":0}')]
Found in MIKASA Index: [('rain|GenreName', '{"E":1,"M":0}')]
```

The algorithm is able to correctly predict “workout” genre for Morning because “exercise” genre was requested in the past. while the current MIKASA isn’t able to predict that.

- Example 2 :

```
Past Entities:
('duke|StationName', '{"E":0,"M":1}'),
('g. flip|ArtistName', '{"E":0,"M":1}'),
('teeth|SongName', '{"E":1,"M":0}'),
('joy|StationName', '{"E":0,"M":5}'),
('cheese|SongName', '{"E":1,"M":0}'),
('christian|GenreName', '{"E":2,"M":0}')]

Future Entities: {'florida the joy|StationName': '{"E":0,"M":18}'}

Found in Morning Index : [('florida the joy|StationName', '{"E":0,"M":18}')]
Found in Evening Index : []
Found in MIKASA Index : []
```

The algorithm is able to correctly predict "*florida the joy*" station name for Morning because the entity is same as joy StationName was requested in the morning in the past. The current MIKASA isn't able to predict that.

- Example 3 :

```
[('say my name|SongName', '{"E":3,"M":0}'), ('straight outta compton|SongName', '{"E":
Future Entities: {'saints|SongName': '{"E":1,"M":0}', 'world is mine|SongName': '{"E":

Found in Morning Index : [('slipknot|ArtistName', '{"E":1,"M":1}'), ('girl on fire|Son
Found in Evening Index: [('megadeth|ArtistName', '{"E":1,"M":0}'), ('slipknot|ArtistNa
Found in MIKASA Index: [('megadeth|ArtistName', '{"E":1,"M":0}'), ('slipknot|ArtistNam
```

In some cases, the algorithm seems to be more effective on power user (users have a lot of past entities), this maybe because of lot entities that are related might be requested in the same context, and that helps in better predicting them. For example, if a person listen to lot of podcasts in the morning then it is table to better capture an unseen podcast request that is requested in the morning in the future.

Conclusion :

As a part of this project, we identified and explored various contexts that could give us more information to better predict future defects. We selected "user local-time of the day" of the day and validated it in various ways, and established statistical significance so that we can use this context for scoring.

We proposed a framework where we have different customer indexes according to different time of the day. Another benefit of having multiple indexes for the same user, is that we could then divide historical entities according to context. This would gives more room in the customer index, that could help us populate more diverse entities in the index.

Using contextual MIKASA, we are able to get **~4**% improvement in the number of predicted entities over current MIKASA. Furthermore using contextual MIKASA, we are able to predict nearly 18.5% more entities than MIKASA if we don't consider top 100 popular entities. More importantly, we are able to predict more personalized choices, due to having more nuanced details about user behavior through the use of context. We could also hypothesize that including other contexts would give more personalized predictions.

Future Work :

1. As I mentioned in the conclusion, there is additional benefit to having multiple indexes, as we could divide the historical index according to the index types. This would gives us more room in the index to have personalized entities. It would be worth exploring this as an experiment.

2. Creating a probability distribution at runtime based on the edges connected to a node. Make selection based on this probability distribution. The process currently takes a lot of time due to dense graph and inefficient python functions.
3. If creating and using multiple index for a user is a huge engineering effort, we could combined entities in the morning and evening index, and generate a combined index. It would be interesting to see results for this experiment.
4. I tried an approach where I did not visit backedge (edge we came from), when selecting future edges. The model showed some improvement for a small sample. So, it would be worthwhile to explore this approach.
5. It would be interesting to explore Weekend/Weekday indexes for customers. There is a difference between number of requests made by customers on weekends/weekdays. It may be worth to conduct that experiment.
6. Rather than having a fixed node budget for all the high affinity nodes, we can have a variable budget based on 'popularity' of the node. So a node with a high degree (no of edges connecting to it), would have a lower budget (shorter random walk), while node with low degrees would have a longer random walk.
7. We can also factor in things like type on entities to decide the random walk length. For Music Domain, we have three kinds of entities (Podcasts, Radio, & Music), We know that podcasts consist of less amount of data.
8. Personalized 'Probability Distributions' based on customer history. Look at customer history to bias the random walk. If the customer makes more morning requests, then bias the random walk toward morning requests.

References

CR:

```
prediction algorithm - contextual mikasa : https://code.amazon.com/reviews/CR-74357197
generating affinities with contextual information : https://code.amazon.com/reviews/CR
```

Sources :

https://wiki.labcollab.net/confluence/display/Doppler/Project+Mikasa?preview=/1499943958/1528574918/image2021-10-8_17-32-8.png#Project-MikasaMusic

https://www.fool.com/the-ascent/small-business/project-management/articles/weighted-scoring-model/

https://www.abhishekmamidi.com/2020/02/pixie-scalable-graph-based-real-time-recommendation-engine-by-pinterest.html

https://medium.com/pinterest-engineering/an-update-on-pixie-pinterests-recommendation-system-6f273f737e1b

https://math.stackexchange.com/questions/2870341/biased-random-walk

Dataset :

https://wiki.labcollab.net/confluence/display/Doppler/Unified+FTV+Data+-+Component+Datasets

https://wiki.labcollab.net/confluence/display/Doppler/FLARe+Rewrite+Data+Aggregation+from+Greenwich

https://code.amazon.com/packages/AesluTools/blobs/mainline/--/src/aeslu_tools/device_map.py

Appendix :

PROCESS TO GET LOCAL TIME OF THE DAY

Process to get “User local-time of the day” :

1. Get latitude and longitude from the Greenwich dataset. Use the timezonefinder library to get the timezone using latitude and longitude.
2. Use timezone and UTC timestamp combination to get local time for a request.

GROUND TRUTH FOR ODD EVEN INTERACTIONS

For even interactions(the number of interactions are even like 2,4,6), when we don’t consider context, we can imagine there are equal number of morning and evening interactions between a user and entity. Hence, the probability an edge is Morning or Evening is both 0.5. Hence, the difference between these probabilities is then (0.5-0.5 = 0).

For odd interactions(the number of interactions are even like 1,3,5 and so on), when we don’t consider context, we can imagine the difference between morning and evening interactions is always 1. When we don’t consider context, to get the difference in the probabilities between Morning and Evening interactions can be calculated as :

$$ceil(x/2)/x - (ceil(x/2) - 1)/x$$

where x is the number of interactions between a user and entity.

ANALYSIS FOR OTHER CONTEXTS :

In the Music Domain, we have three things : Songs - Podcasts - Radio

For Podcasts

Date	Weekday/Weekend	Total of values	No of podcast requests	Percent of total requests
12/6/2022	**Weekend**	37309973	143296	0.38%
13/06/2022	**Weekday**	34585748	157818	0.46%
14/06/2022	**Weekday**	34666633	161116	0.46%
15/06/2022	**Weekday**	35019035	159827	0.40%
16/06/2022	**Weekday**	35345333	157288	0.44%
17/06/2022	**Weekday**	36508969	151322	0.44%
18/06/2022	**Weekend**	38261688	141250	0.36%

Average on Weekdays : 0.44
Average on Weekends : 0.37

A 17.284 percent difference between weekdays and weekends.

Podcast-Context Probability Distribution :

Context Type	Probability
Weekday	0.543
Weekend	0.456

We use this probability while biasing random walks.

For Radio Stations

Date	Weekday/Weekend	Total of values	No of radio requests	Percent of total requests
12/6/2022	**Weekend**	37309973	1713580	0.04593
13/06/2022	**Weekday**	34585748	1868065	0.05401
14/06/2022	**Weekday**	34666633	1877781	0.05417
15/06/2022	**Weekday**	35019035	1895379	0.05412
16/06/2022	**Weekday**	35345333	1877675	0.05312
17/06/2022	**Weekday**	36508969	1917972	0.05253
18/06/2022	**Weekend**	38261688	1793601	0.04688

Average on Weekdays : 5.35
Average on Weekends : 4.6385

A 14.246 percent difference between weekdays and weekends.

Radio-Context Probability Distribution :

Context Type	Probability
Weekday	0.535
Weekend	0.4635

For FireTV/Other

For FTV:

Date	Total of values	No of FTV requests	Percent of total requests
12/6/2022	37309973	320502	0.00859
13/06/2022	34585748	264688	0.00765
14/06/2022	34666633	256184	0.00739
15/06/2022	35019035	259115	0.0074
16/06/2022	35345333	270374	0.00765
17/06/2022	36508969	282970	0.00775
18/06/2022	38261688	332867	0.0087

Average on Weekdays : 0.75685
Average on Weekends : 0.8645

A 13.2791 percent difference between weekdays and weekends.

FTV-Context Probability Distribution :

Context Type	Probability
Weekday	0.466
Weekend	0.534

For Doppler:

Date	Total of values	No of Doppler requests	Percent of total requests
12/6/2022	37309973	24038461	0.64429
13/06/2022	34585748	22582241	0.65293
14/06/2022	34666633	22599441	0.65191
15/06/2022	35019035	22806929	0.65127
16/06/2022	35345333	22976239	0.65005
17/06/2022	36508969	23715258	0.64957
18/06/2022	38261688	24541611	0.64141

Average on Weekdays : 65.11477
Average on Weekends : 64.28526

There is a 1.28209% difference.

For Knight:

Date	Total of values	No of Doppler requests	Percent of total requests
12/6/2022	37309973	5365490	0.14381
13/06/2022	34585748	4759744	0.13762
14/06/2022	34666633	4801938	0.13852
15/06/2022	35019035	4855746	0.13866
16/06/2022	35345333	4912506	0.13899
17/06/2022	36508969	5070122	0.13887
18/06/2022	38261688	5549909	0.14505

Average on Weekdays : 13.85317
Average on Weekends : 14.44299

There is a 4.15 percent increase.

STATISTICAL ANALYSIS FOR MORNING/EVENING CONTEXT

For an interaction between a user and an entity, let's say the number of morning edges is 3 and the number of evening edges is 7. Then the probability of the morning edge is 0.3 and the evening edge is 0.7. The difference is then 0.4.

When we don't consider context, we can imagine that there are equal number of morning and evening interactions between a user and entity. Hence, the probability that an edge is Morning or Evening is both 0.5. Hence, the difference between these probabilities is then (0.5-0.5 = 0).

Take all interactions where number of interactions between a user and entity is more than 5.

For even number of interactions:
Mean difference : 0.64
Standard deviation : 0.312

So, we can see that 0 is more than 2 standard deviations below the mean. Hence, we can see that a significant amount of interactions have difference in which they are requested in the morning and in the evening.

For odd number of interactions:
Mean difference : 0.67
Standard deviation : 0.33

When we don't consider context, the we can imagine that the difference between the morning and evening interactions isa at-most 1. Hence, the difference between these probabilities is then at most 0.33 (1/3-2/3). but in most cases it is less than 0.2.

So, we can see that 0.2 is around 2 standard deviations below the mean. Hence, we can see that a significant amount of interactions have difference in which they are requested in the morning and in the evening.

DETAILED ALGORITHM:

Algorithm :

```
FUNCTION : contextual_mikasa
Inputs:  Q - the set of HHA entity nodes of a customer
Hyperparameters: total_budget, max_walk_length, high_visit_count_threshold, n_high_vis
Initialization: morning_count = {}, evening_count = {}, Edge_types = ["M", "E"]
Approach:


for q in Q:
    node_budget = total_budget / |Q|
    visit_count_for_morning, visit_count_for_evening = contextual_mikasa_random_walk_w
        high_visit_count_threshold, n_high_visit_threshold)
    ##implementation in the next block##

    # this maintains global score for a node
    for node in morning_visit_counts:
       morning_count[node] = morning_count.get(node, 0) + morning_visit_counts[node]

    for node in evening_visit_counts:
       evening_count[node] = evening_count.get(node, 0) + evening_visit_counts[node]

# boost scores higher valued scores
for node in evening_visit_counts:
     evening_count[node] = evening_count[node] ** boost_visit_count_for_highly_visited

for node in morning_visit_counts:
```

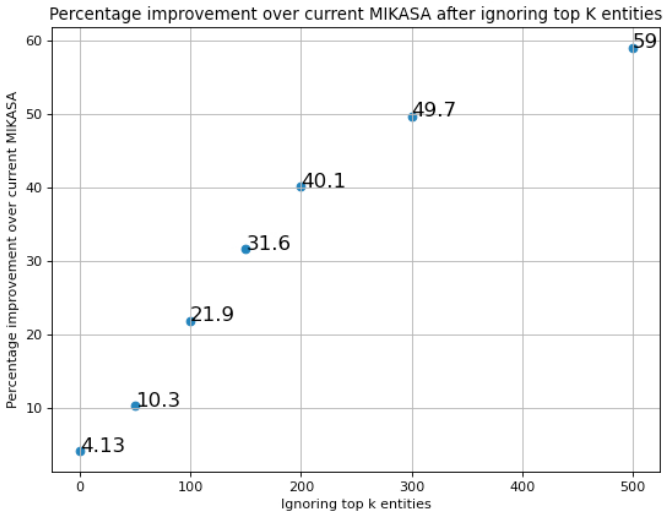
```
     evening_count[node] = evening_count[node] ** boost_visit_count_for_highly_visited

return evening_count, evening_count
```

```
FUNCTION : contextual_mikasa_random_walk_with_early_stopping
Inputs : q, max_walk_length,node_budget,
    high_visit_count_threshold, n_high_visit_threshold

Hyperparameters: total_budget, max_walk_length,
    high_visit_count_threshold, n_high_visit_threshold

Initialization: total_steps = 0, context_count_morning = {}, context_count_evening = {

Approach:

Repeat:

    current_node = q
    current_steps = uniform_sample(0, max_walk_length)
    walk_information = {M: 0, E:0}

    for i in range(0, current_steps):
        next_node, contextual_information = get_random_neighbour_and_associated_edge(c

        walk_information["M"] += contextual_information["M"]
        walk_information["E"] += contextual_information["E"]

        context_count_morning[next_node] += log(walk_information["M"], 2)
        context_count_evening[next_node] += log(walk_information["E"], 2)

        if context_count_morning[next_node] == high_visit_count_threshold OR context_c
            n_high_visited++

        current_node = next_node

Until total_steps >= node_budget or n_high_visited > n_high_visited_threshold Return v

return context_count_morning, context_count_evening
```

HOW TO ADD MORE CONTEXTUAL VARIABLES TO THE ALGORITHM

1. We would need to just modify the graph to add any additional context. It would just look like adding more in more information to the edge that connects a user and entity. For example, change “{”M“:0, ”E“: 1}” to “{”M“:0, ”E“: 1, ”X“:2}”
2. We would need to random walk for that context and generate an index for that index. For example, generate an index for context “X”.
3. We have two indexes at the same time, for example, a request coming in at Morning on a FireTv Device.
4. In that case, for“Morning Index” and “FireTv index”, we would combine entities from Morning and XFireTv index. We would fill up the remaining space in the index with top scoring entities from both indexes.

ADDITIONAL RESULTS



Top entities ignored	Number of entities correctly predicted : Contextual MIKASA	Number of entities correctly predicted : MIKASA	Percentage Difference
0	63871	61283	4.13
50	34444	31039	10.3
100	23589	18926	21.9
150	17985	13069	31.6
200	14474	9639	40.1
300	10639	3741	49.7
500	6914	3741	59