How Artificial Intelligence is being used to match researchers with bug bounty programs

BY ANNA HAMMOND · OCTOBER 15, 2021 · LAST UPDATED ON MARCH 6, 2025

Every security researcher has their specialty, their range of bugs that they focus on or that they have spent the past years researching and perfecting. The greatest power of bug bounty comes into play when considering that all of these bright individuals with diverse skillsets can become a single superbrain when put together.

At Intigriti, we posed the question: “How can we better match researchers with programs”? Finding a superior way to do this would result in a mutually beneficial arrangement whereby the bug bounty programs receive more valid findings, thus further reinforcing their security and protecting them from malicious actions. At the same time, researchers receive more (and more valuable) bounties.

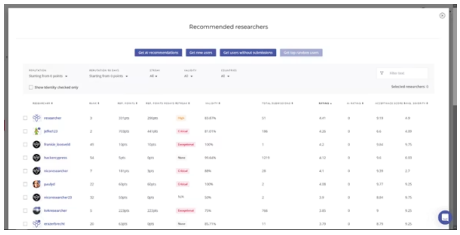
AI: Recommender system

Time to get technical. Can we recommend researchers to bug bounty programs? The type of AI system most fit for this purpose is a recommender system. These are commonly used by online marketplaces, content delivery systems, search mechanisms and such.

For traditional programming approaches, this problem is incredibly difficult to solve. How can we recommend something to a user, that that user has never interacted with? Unless you have some hardcoded relations (or are able to extract some) between items, you won’t be able to properly recommend items to users.

For humans, it is still very hard to make recommendations, definitely when trying to limit the amount of bias introduced. Imagine attempting to recommend a movie to someone without introducing any bias based on your own likings. That is hard, isn’t it?

This is where collaborative filtering comes into play. We created a model which can outperform humans by over 35% and which can do that fully automatically.



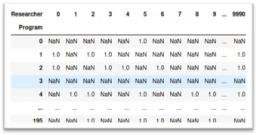
Platform view of recommended researchers

Algorithm: Alternating Least Squares (ALS)

We need a sense of interaction between the programs and the researchers. This is achievable by creating a matrix that maps all interactions between the researchers and the programs. An interaction is defined as a researcher submitting a valid vulnerability to a program.

It is worth noting that the context for user and item might be slightly counterintuitive. When talking about a recommendation system, an item is recommended to a user. In this case, a researcher is recommended to a program. Therefore, the programs are the users, and the researchers are the items.

Such an interaction or rating matrix would look something like the figure below. This matrix can be used to calculate the hamming distance between two users. This distance defines the number of positions at which the corresponding users are different. Thus, the higher this distance, the more the two users match. In theory, this could be used to find users and items with the most similar interactions and recommend them that way.



Researcher	0	1	2	3	4	5	6	7	8	9	...	9990
Program												
0	NaN	NaN	NaN	NaN	NaN	1.0	NaN	NaN	NaN	NaN	...	NaN
1	1.0	NaN	1.0	1.0	NaN	NaN	NaN	NaN	NaN	NaN	...	1.0
2	1.0	NaN	NaN	1.0	1.0	NaN	1.0	NaN	NaN	NaN	...	1.0
3	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...	NaN
4	NaN	1.0	1.0	NaN	NaN	1.0	NaN	NaN	1.0	1.0	...	1.0
...	...	...	...	...	...	...	...	...	...	...	...	...
195	NaN	NaN	1.0	NaN	NaN	1.0	1.0	NaN	NaN	1.0	...	NaN

Sample rating matrix

Matrix factorisation

We further optimised recommendations by using matrix factorisation. The idea here is that by seeing the rating or interaction matrix as the multiplication of a user matrix times an item matrix, it can be stored using less memory and is called a latent feature matrix.

The model we designed uses a compressed sparse row or CSR format which uses three arrays to represent the matrix. One of the great attributes of these arrays is that they do not contain zero values. Without matrix factorisation, it would not be possible to efficiently recommend such sparse rating matrices. From the figure below, it becomes clear how matrix multiplication solves the sparsity problem.

The goal is to find the optimal user and item matrix to minimise a loss function, the least squares error. This is a non-convex problem, which means that if the loss function is graphed, there are local minima that are not the global minimum. An example of a simple loss landscape is depicted in the figure below.

Alternating least squares (ALS) follows the principle of matrix factorisation but in a parallel fashion. This allows it to be executed very smoothly on a GPU which in turn makes it great for large-scale collaborative filtering solutions, such as our own.

In more technical terms, one of the matrices is fixed and the other optimised. Proceeding, the process is alternated. This is repeated until they converge. For example, the user matrix will be frozen, and the convex space of the item matrix will be optimised. Following, the item matrix is fixed, and the user matrix optimised. This process will be repeated until convergence.

Conclusion

Using collaborative filtering, it is possible to effectively optimise the private invitation process of Intigriti. We have already seen great results from this intricate system and will be continuing optimising it to ensure the best results at all times. Lets help make the world more secure, day by day!

REQUEST A DEMO

intigriti.com/demo

VISIT THE WEBSITE

intigriti.com

GET IN TOUCH

hello@intigriti.com