

Qualifier Recommendation for Wikidata^{*}

Andrei Mihai Ducu¹, Michael Cochez¹

¹Vrije Universiteit Amsterdam, The Netherlands

Abstract

Wikidata, a collaborative knowledge base for structured data, empowers both human and machine users to contribute and access information. Its main role is in supporting Wikimedia projects by acting as the central storage database for the Wikimedia movement. To optimize the manual process of adding new facts, Wikidata utilizes the association rule-based *PropertySuggester* tool. However, a recent paper introduced the *SchemaTree*, a novel approach that surpasses the state-of-the-art *PropertySuggester* in all performance metrics. The new recommender employs a trie-based method and frequentist inference to efficiently learn and represent property set probabilities within RDF graphs. In this paper, we adapt that recommendation approach, to recommend qualifiers. Specifically, we want to find out whether the recommendation can be done using co-occurrence information of the qualifiers, or whether type information of the item and the value of statements improves performance. We found that the qualifier recommender that uses co-occurring qualifiers and type information leads to the best performance.

Keywords

Wikidata, Qualifiers, Recommender

1. Introduction

In today's world, the era of big data and rapid ever-changing information, effective systems are needed to organize and structure the abundance of information available online. A great number of databases require constant editing and frequent updates to provide reliable, accurate, and readily available information. An example of such an openly available resource is Wikidata[1].

The Wikidata project is part of the Wikimedia movement. It is widely accessible and used, not only by other Wikimedia projects, but also by external applications and organizations. Just in a one-year period (Apr. 2022 - Apr. 2023), Wikidata reached around 5 billion page views. Most of these are automated actions that propagate information to and from other sources. However, the platform saw a monthly average user base of around 3 million unique devices, proving its direct usefulness to the public, as well. More importantly to the theme of the paper, it had an average of around 43 thousand editors, both human and automated. Another metric of interest is the number of edited pages, which averages at 10 million per month¹. This intensive usage suggests that a better assistive editing system could make the work of many editors easier, more efficient, and less error prone.

^{*}This work is based on the BSc. thesis of Andrei Mihai Ducu, under the supervision of Michael Cochez

Wikidata'23: Wikidata workshop at ISWC 2023

✉ a.ducu@student.vu.nl (A. M. Ducu); m.cochez@vu.nl (M. Cochez)

🌐 <https://www.cochez.nl> (M. Cochez)

🆔 0000-0001-5726-4638 (M. Cochez)



© 2023 Copyright for this paper by its authors. Use permitted under Creative Commons License Attribution 4.0 International (CC BY 4.0).

📄 CEUR Workshop Proceedings (CEUR-WS.org)

¹Wikimedia Statistics: <https://stats.wikimedia.org/#/wikidata.org>

Contributors with diverse backgrounds and varying levels of expertise may encounter difficulties when editing records in a complex knowledge base such as Wikidata. Erroneous updates could potentially result in data inconsistencies and incompleteness. Therefore, assisting the users in the editing process is of utmost importance to preserve data quality and accuracy, while greatly reducing the workload. At the user interface side, there are two systems active on Wikidata to improve the quality. First, there are constraints on properties, which check whether they are applied on items of the right type, and whether the values are within the expected range. Second, there are recommender systems, which can act as a guide for the end users when adding properties to items and qualifiers to properties.

This paper focuses on the latter aspect, improving the recommendations for adding qualifiers on properties, such that manual editors can update qualifier information for statements about items on Wikidata. The recommender system used for this purpose was adapted from the previous work on the SchemaTree property recommender [2]. SchemaTree introduces a novel approach, rooted in trie structures, to compute probability distributions of property sets within RDF graphs. For this project, the recommender suggests qualifiers instead of properties, based on a) co-occurring qualifiers and b) type information of the item (subject), as well as the corresponding property value (object).

In order to determine the best configuration of the new qualifier recommender, four configurations were tested, each corresponding to a different level of information provided to the system. The first configuration recommends qualifiers using only co-occurring qualifier information. The second and third, use either item (subject) or value (object) type information. Lastly, full contextual information was used for the recommendation, namely co-occurring qualifiers, and both item and value type information.

Two questions were formulated around the four configurations investigated. The main question, and a secondary one that stems from the main question. The two research questions are:

- Does including type information improve the performance of the qualifier recommender system?
- What kind of type information is more informative, the type of the item or the type of the value?

These questions are investigated by evaluating each configuration, with two different evaluation methods, against a held-out test set consisting of 20% of all data extracted from Wikidata. The performance of the configurations is compared to a baseline model that make recommendations solely on absolute qualifier occurrence frequency, without using any other contextual information. What we find is that adding type information is nearly always beneficial. The code is available on GitHub².

²Qualifier Recommender: https://github.com/Duculet/QualifierRecommender/tree/eval_handlers

2. Background and Related Work

2.1. SchemaTree Recommender

The *SchemaTree Recommender*³ proposes a novel approach to new property recommendations within the Wikidata project. This system comes as an alternative to the currently used *PropertySuggester*⁴. The newly introduced recommender makes use of the maximum-likelihood of properties to suggest additional ones. The recommender leverages a compact trie-based data structure called the SchemaTree, which integrates the representation of property and type co-occurrences. It specializes on the efficient lookup of such patterns, being constructed as an adapted trie construction of a frequent pattern tree. This data structure type enables efficient probability calculations and efficient property pair retrieval. Next, the SchemaTree structure is introduced and described. This information was adapted from [2].

The SchemaTree is created as a data structure to facilitate property recommendations based on maximum-likelihood estimation. These recommendations are generated for a given item, denoted as E , and its set of properties, denoted as $S = s_1, \dots, s_n$, where $S \subseteq A$ (subset of the available properties A in Wikidata). The goal of recommending maximum-likelihood properties is to identify the most likely property $\hat{a} \in A \setminus S$, meaning a property the item does not have already. The property \hat{a} has to be found such that the following holds:

$$\hat{a} = \operatorname{argmax}_{a \in (A \setminus S)} P(a \mid \{s_1, \dots, s_n\}) = \operatorname{argmax}_{a \in (A \setminus S)} \frac{P(\{a, s_1, \dots, s_n\})}{P(\{s_1, \dots, s_n\})} \quad (1)$$

where $P(\{t_1, \dots, t_m\})$ denotes the probability that a selected entity has at least the properties t_1, \dots, t_m . In line with this, the recommended properties are the ones that exhibit the highest frequency of co-occurrence with the properties already possessed by the given entity [2].

By adopting a frequentist probability interpretation the joint probabilities are estimated based on the relative frequency of occurrence. The absolute frequency of a set of properties, i.e. the number of items that have (at least) this set of properties, is represented as $\operatorname{supp}(A)$. By reformulating Equation 1, the estimation of the most probable property recommendation can be expressed as follows:

$$\hat{a} \simeq \operatorname{argmax}_{a \in (A \setminus S)} \frac{\operatorname{supp}(a, s_1, \dots, s_n)}{\operatorname{supp}(s_1, \dots, s_n)} \quad (2)$$

The SchemaTree structure aims to optimize the computation time for estimating this probability in the context of all data already contained within Wikidata.

Besides finding the properties with the highest probability, the SchemaTree also uses back-off strategies in case the recommendations are not good enough. They found that the best back-off strategy was to rerun the system with the least popular property removed from the property set, in case there are no recommended properties, which happens when all have a zero probability. This gets repeated up to four times, until a recommendation is found. In this work, we use the same setup. In future work, it should be investigated whether there is a better backoff strategy specifically for qualifiers.

³SchemaTree Recommender: <https://github.com/lgleim/SchemaTreeRecommender>

⁴PropertySuggester: <http://gerrit.wikimedia.org/r/admin/projects/mediawiki/extensions/PropertySuggester>

2.2. Other Works

Other recommender systems were proposed throughout the years. Another such system that was recently put forth is the *WikidataRec* [3]. The system employs a hybrid approach that combines content-based and collaborative filtering techniques to rank items for editors. This hybrid approach considers both the features of the items themselves and the previous interactions between items and editors. To achieve this, a neural network called "neural mixture of representations" is developed. This neural network is specifically designed to learn optimal weights for combining item-based representations and editor-based representations, taking into account the interactions between items and editors. By leveraging these interactions, the system aims to optimize the ranking of items and improve the overall recommendation quality for editors. Based on their experimental data, the system proved to perform well in situations where the data fed into the model was dense. However, collaborative filtering was found to be less useful in the case of sparse editing data, which makes up most of the available data [3]. Another approach taken to handle Wikidata qualifiers was by using reasoning. This entails defining inference rules, specifically on ontological properties. The paper proposes handling of qualifiers using inference rules, although the system presented does not implement a recommender system. However, it is interesting to see how they overcame the massive number of qualifiers and practically implemented a prototype that can express all of Wikidata's ontological properties [4].

3. Qualifier Recommender

This section provides a comprehensive description and analysis of the work conducted, from data extraction to data structure, including adaptations made to the SchemaTree code to work with qualifiers instead of properties.

3.1. SchemaTree Adaptation

Several parts of the original SchemaTree recommender code were adapted such that it could be used to enable the recommendation of qualifiers instead of properties. Also, some new additions were made in order to extract qualifier information from the Wikidata dump file, and evaluate the new recommender system. Changes and additions were made to the following components:

Extractor Added the functionality to extract qualifier information from the Wikidata JSON dump and save it in one TSV file for each property type.

Splitter Used for the evaluation to split the TSV files in a train and a test set.

Datatypes Updated so it can detect and count the types that occur alongside the qualifiers (item / value types)

RecommenderServer Added a handler for qualifier recommendation requests. This handler receives the property of the statement for which recommendations are sought, as well as potentially the types of the item and value of the statement.

These updates and additions will be detailed in the following sub-sections.

3.2. Configurations

Four main configurations were explored, as previously mentioned. They are the following:

- FF - No type information included
- TF - Value (object) type information included
- FT - Item (subject) type information included
- TT - Both value (object) and item (subject) type information included

Each of the experiments that follow were conducted four times, once for each configuration. The next subsections will only detail the TT configuration. The same pipeline was applied to the other configurations, leaving out the respective types of information.

Besides these configurations, we also have a baseline configuration, which makes no use of property, nor item and value type information.

3.3. Data Extraction

Data used to generate the models for the recommender system was obtained in the form of a BZIP2 compressed Wikidata JSON dump⁵ consisting of all items and their representative features in the knowledge base. For each unique property in the dataset, a TSV file was generated incorporating information about the occurrences of that specific property throughout Wikidata. For example, in the file for P50, each entry (row) contains information about one occurrence of the property being used on wikidata. We store the used qualifiers, as well as the item (subject) and value (object) types for this occurrence. We call the collected information for one occurrence, a transaction, in accordance with the frequent item set literature. The extraction workflow is further detailed in fig. 1.

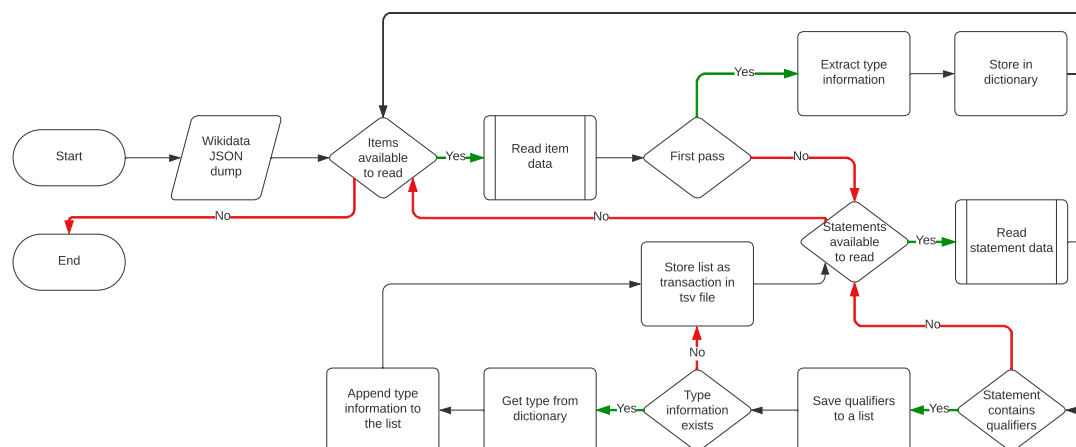




Figure 1: Flowchart of the algorithm constructed for data extraction from the dump.

Referring to the example structure of the item "Douglas Adams" (Q42), an example of a TSV file construction and its format for the property `educated at` (P69) can be seen in fig. 2.

⁵For our experiments we used the one of 27th of March 2023: <https://dumps.wikimedia.org/wikidatawiki/entities/>

educated at (P69)	 St John's College (types Q1055028 and Q19844914)
	<u>end time (P582)</u> 1974
	<u>academic major (P812)</u> English literature
	<u>academic degree (P512)</u> Bachelor of Arts
	<u>start time (P580)</u> 1971
	 Brentwood School (types Q2418495 and Q269770)
	<u>end time (P582)</u> 1970
	<u>start time (P580)</u> 1959

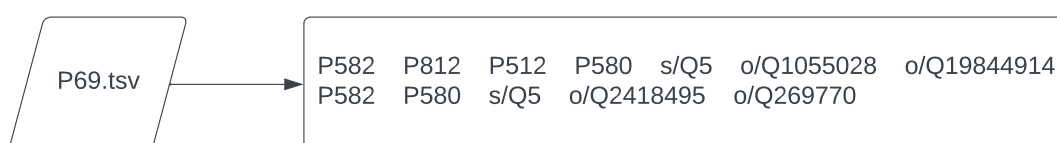


Figure 2: Example TSV file construction based on the Douglas Adams item (Q42), for the property `educated at` (P69). The item has two statements for this property, hence two transactions are generated, it includes the qualifiers on this statement ($P[0-9]^+$), the types of the item – Q5 for Human ($s/Q[0-9]^+$), and the types of the value, e.g., the types of St. John’s College ($o/Q[0-9]^+$).

To adapt the SchemaTree to this data, we treat all parts of the transaction uniformly, and in the same way the SchemaTree dealt with properties. To do this, we rewrite the types by pre-pending information on in which role (as item/subject or as value/object) they occurred. an example can be found in the lower part of fig. 2, the item types are prefixed with “s/”, while the value types are prefixed by “o/”. The qualifiers are saved simply by their property identifier.

3.4. Data Preparation

Once all data available was extracted from the dump, it required further processing to allow for a valid evaluation. Therefore, we separate the extracted data into a train and test set. A random (80% - 20%) split was applied to the transactions of each TSV file. For a real deployment, one would of course use all available data to create the SchemaTree model.

3.5. Model Generation

The next step in the context of the recommender is generating SchemaTrees that work as input models for the final recommender system. We create one SchemaTree for each TSV file constructed before, i.e., one for each property. These models are just SchemaTree structures.

Another setup was explored, where a single large model was created, based on a concatenation of all TSV files into one. This had, however, two negative effects. First, the recommender became slower, because a larger tree needs to be considered. Second, the quality of the recommendations went down; we suspect this is caused by information about other properties

causing mistakes, especially in combination with the backoff strategies. What we notice is that having property-specific models gives the recommender the opportunity to learn the context of qualifier occurrences better and more efficiently. Thus, the choice for many-models was made.

3.6. Recommender Server

Finally, a new method to serve the results was created, with small adaptations to the data structures of the request and response. Examples can be found in fig. 3 below. When the recommender server is started, one model per property type is pre-loaded into memory.

Despite the backoff strategies, the recommendations will sometimes still be empty. To solve this, and to make the evaluation more sound, we make sure that the recommender always ranks all possible qualifiers. That is, first the results of the SchemaTree recommender, then the recommendation where all qualifier information is stripped from the request, and finally the order as provided by a purpose-built SchemaTree that does not even use the property type.

```

{
  "property": "P69",
  "Qualifiers": ["P582", "P580"],
  "subjTypes": ["Q5"],
  "objTypes": ["Q2418495", "Q269770"]
}
{
  "recommendations": [
    {"qualifier": "P512", "probability": 0.0252},
    {"qualifier": "P812", "probability": 0.0101},
    {"qualifier": "P1326", "probability": 0.0050},
    {"qualifier": "P1534", "probability": 0.0050},
    ...
  ]
}

```

Figure 3: Examples of a request on the left and a (truncated) response on the right. The request asks for more qualifiers for the educated at (P69) property with value Brentwood School from the example in fig. 2. Besides the qualifiers, also the types of the item and the value are provided. The response are qualifiers with their corresponding probabilities. Here the answers are academic degree (P512), academic major (P812), latest date (P1326), and end cause (P1534).

4. Evaluation

Two experimental methods were employed to evaluate the recommender system. Each of them provides insight into how informative specific types of data are to the recommender when making suggestions. An additional method was added to act as the baseline when evaluating the system. Key metrics about the individual models were computed and additional and aggregated ones are also presented. This section describes the evaluation protocol.

4.1. Evaluation Methods

To generate recommendation tasks to solve, we first use the **leave-one-out** evaluation method, as was used in [2]. This means that one qualifier of the transaction is left out, to be recommended back by the system. The system thus receives the property type, co-occurring qualifiers, and, depending on the configuration, type information.

A second way to generate recommendations tasks for evaluation is what we call **leave-all-out**. Here, all qualifiers are stripped from the test transactions and the recommender is expected to recommend these back, solely relying on the property type and potentially type information.

4.2. Obtaining Results

For each of the configurations, the two evaluation methods (leave-one-out and leave-all-out) were performed to generate evaluation results. Additionally, results were generated for the baseline recommender, which does not use any contextual information to make predictions, and is hence independent of the method.

For this process, only qualifier models with more than 100 transactions in the test set were used. We noticed that those with fewer transactions lead to very spurious results, most likely since not enough information was available for the model. This led to approximately 1060 models used for the evaluation. The recommendations results are evaluated using ranking metrics, such as rank, hits@1, hits@5, and hits@10. We also record the left out qualifier and the number of co-occurring qualifiers and types for further analysis.

This further analysis is done by either grouping the results by a specific set size, or by computing more general statistics for entire experiments. An important aspect that was considered was the way averages are computed. In this scenario, using the leave-one-out method, each transaction generally equates to more than one evaluation. Therefore, the metrics were first micro-averaged by transaction, then further processed. For instance, when computing the model average rank, the first step was micro-averaging all evaluation ranks by transaction, then macro-averaging those values to obtain the final average. Also, when exploring the evaluation results, some of the transactions appeared to have no qualifier prediction rank, being denoted by the value 5843 (preset for this purpose). The percentage of missing recommendations was saved, but the missing transactions were eliminated from the final evaluation set, as they make up a very small amount, and would otherwise skew the results gravely. The reason for encountering such results stems from the train-test split, as some of the qualifiers in the test set never appear in the train set that was used for modeling. This, however, would never occur in a real production setting where the models would be trained on the full Wikidata dump.

After obtaining the results and closer inspection, two outlier models were identified, namely the ones for P1855 and P5192. The results for these two models were unexpectedly poor. We decided to **not** include them into the final evaluation because they are very generic properties, namely properties for a Wikidata property example, and a Wikidata property example for lexemes.

4.3. Results

For each of the evaluation methods, a table and two plots were generated. The first plot regards the general metrics per configuration, whereas the second groups those metrics by SetSize and aggregates the results. The table contains the evaluation results, calculated in terms of the configuration type.

leave-one-out As can be seen in table 1, the percentage of missing recommendations is very low. The best performing configuration is TT, which includes all type information (both item and value types). The second-best performer is the FT configuration.

leave-all-out As can be seen in table 2, we obtain the same rankings with the TT as best performing configuration and FT as second best. More visualizations of the results can be found in the appendix in fig. 4 and fig. 5.

Table 1

leave-one-out aggregated results.

Recommender	Mean	Median	StdDev	Top1	Top5	Top10	Missing
Baseline	1.8320	1.4962	1.1371	70.6825	96.2079	98.6246	0.0031
No Type Info (FF)	5.1816	3.1143	7.0066	52.6749	77.4949	87.8853	0.0030
Value Type Info (TF)	1.4301	1.0972	0.7393	87.6284	97.8587	99.1271	0.0032
Item Type Info (FT)	1.3535	1.0812	0.6670	89.7055	98.2256	99.3074	0.0031
Full Type Info (TT)	1.3132	1.0666	0.5918	91.1623	98.4774	99.3390	0.0031

Table 2

leave-all-out aggregated results.

Recommender	Mean	Median	StdDev	Top1	Top5	Top10	Missing
Baseline	1.8320	1.4962	1.1371	70.6825	96.2079	98.6246	0.0031
No Type Info (FF)	1.8298	1.4887	1.1380	70.7192	96.2196	98.6489	0.0030
Value Type Info (TF)	1.6807	1.4261	0.8752	73.6173	97.1556	98.9881	0.0032
Item Type Info (FT)	1.5738	1.3534	0.7794	75.8690	97.8243	99.2349	0.0031
Full Type Info (TT)	1.5293	1.3369	0.6823	77.0860	98.1101	99.2896	0.0031

5. Conclusions and Future Work

In this paper, a qualifier recommender system was built around the previously created trie-based SchemaTree property recommender. Adaptations to the data extraction technique were made to allow it to extract qualifier information from the Wikidata dump. Further modifications revolved around the input-output request-response structure of the recommendation server. The results of the four configurations evaluated in the paper were close to expectations. We found that both the item and value type information, as well as the co-occurring qualifiers, are important information when making recommendations. We further found that, models based on item (subject) types outperformed the ones based on value (object) type on average.

The current implementation of the qualifier recommender is limited by several factors. A first improvement would be to restrict the type of qualifiers suggested using Wikidata constraints. Moreover, when extracting the data for building the SchemaTree, more type information could be obtained by traversing the subclass of (P279) type-hierarchy and collecting additional types, rather than only the leaf-type as is currently done.

Another aspect is that we currently use the back-off strategy which was shown to be best for properties. A large scale evaluation could find that a different back-off is better for qualifiers.

Currently, the recommender only gets information about the type of the item, the property, the type of the value of the claim, and other qualifiers on the claim. However, also other claims on the same item might have a useful predictive property. For example, if a Human (Q5) has an employer (P108), then the educated at (P69) property is very likely has the qualifier end time (P582). Incorporating this information is left as future work.

One further idea is that we could make a single model for all properties by including the property as part of the input set of the recommender. This might give some benefits when two similar properties would have similar qualifier information, especially when one of the properties is rarely used. This might also further reduce the overall memory usage at the cost

of a small performance hit.

In terms of evaluation, one more method that would more accurately determine whether the recommender performs well in a real setting could be integrated. Such an evaluation technique would consist in generating a whole list of qualifiers from scratch, only making use of the type information in the statements. Leave-all-out evaluation comes closest to this method.

Finally, the best way to evaluate this is to have an actual A/B testing of the recommender, where the current system used for Wikidata is compared with the proposed system in a practical evaluation.

Acknowledgments

The SchemaTree is rather efficient, but requires a large in memory index to achieve this speed. Besides, we experiment with many different configurations and do perform a lot of queries. Therefore, Snellius (the Dutch national supercomputer) was used to run these experiments.

Michael Cochez was partially funded by the Graph-Massivizer project, funded by the Horizon Europe programme of the European Union (grant 101093202).

References

- [1] D. Vrandečić, Wikidata: A new platform for collaborative data collection, in: Proceedings of the 21st international conference on world wide web, 2012, pp. 1063–1064.
- [2] L. C. Gleim, R. Schimassek, D. Hüser, M. Peters, C. Krämer, M. Cochez, S. Decker, SchemaTree: Maximum-likelihood property recommendation for Wikidata, in: European Semantic Web Conference, Springer, 2020, pp. 179–195.
- [3] K. AlGhamdi, M. Shi, E. Simperl, Learning to recommend items to Wikidata editors, in: The Semantic Web–ISWC 2021: 20th International Semantic Web Conference, ISWC 2021, Virtual Event, October 24–28, 2021, Proceedings 20, Springer, 2021, pp. 163–181.
- [4] S. Aljalbout, G. Falquet, D. Buchs, Handling Wikidata qualifiers in reasoning, arXiv preprint arXiv:2304.03375 (2023).

A. Result visualizations

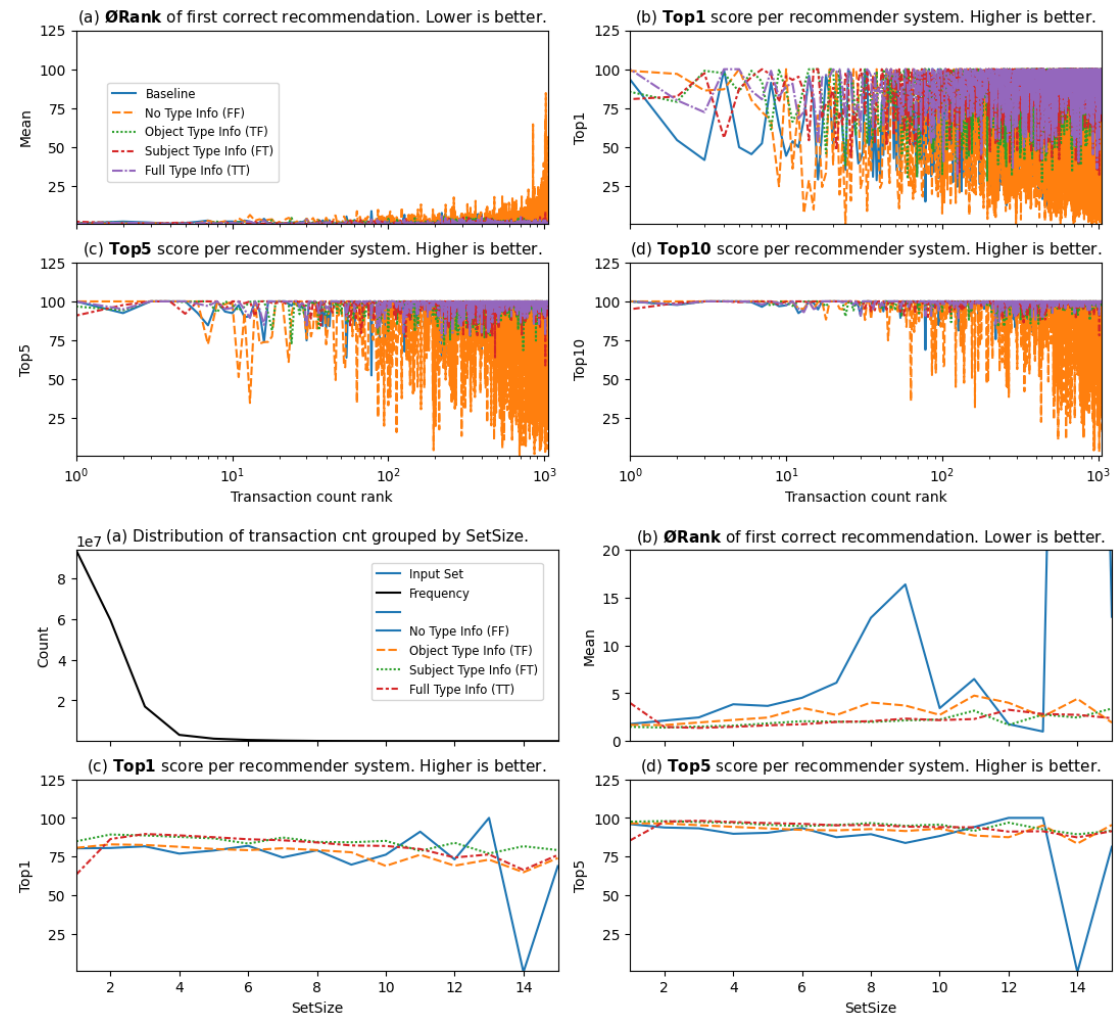


Figure 4: Visualization of leave-one-out results.

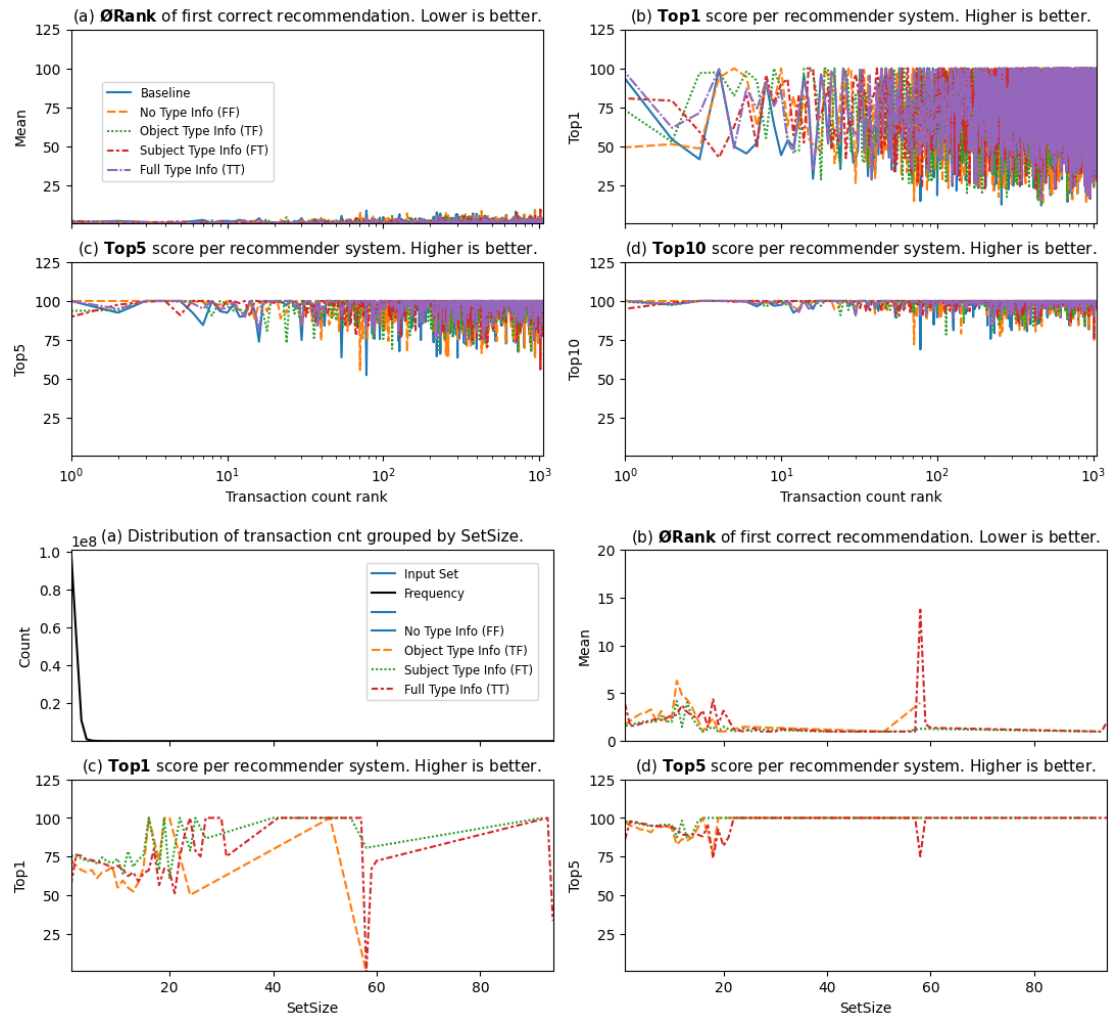


Figure 5: Visualization of leave-all-out results.