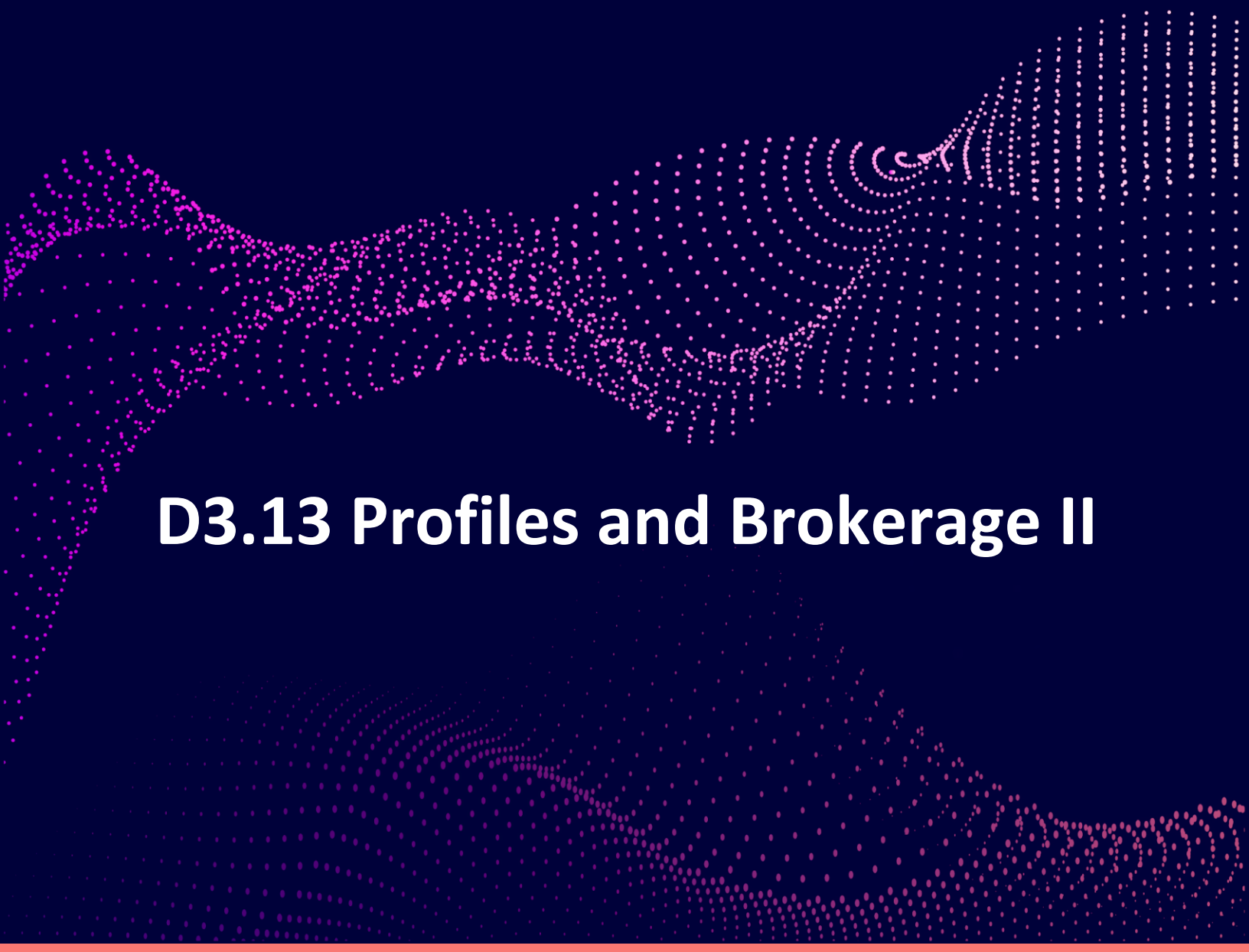


D3.13 Profiles and Brokerage II

Authors: **Dominik Kowald, Dieter Theiler, Peter Müllner, Emanuel Lacic (KNOW)**

Additional Information: -

December 2022

This project has received funding from the European Union's Horizon 2020
research and innovation programme under grant agreement No 871481

TRUSTS Trusted Secure Data Sharing Space

D3.13 Profiles and Brokerage II

Document Summary Information

Grant Agreement No	871481	Acronym	TRUSTS
Full Title	TRUSTS Trusted Secure Data Sharing Space		
Start Date	01/01/2020	Duration	36 months
Project URL	https://trusts-data.eu/		
Deliverable	D3.13 Profiles and Brokerage II		
Work Package	WP3		
Contractual due date	31/12/2022	Actual submission date	05/12/2022
Nature	Demonstrator	Dissemination Level	Public
Lead Beneficiary	KNOW		
Responsible Author	Dominik Kowald		
Contributions from	Dieter Theiler, Peter Müllner, Emanuel Lacic (KNOW)		

Revision history (including peer reviewing & quality control)

Version	Issue Date	% Complete[1]	Changes	Contributor(s)
v1.0	05/08/2022	5%	Deliverable Structure	Dominik Kowald (KNOW)
V1.1	08/09/2022	10%	Definition of T3.6 Research Outputs	Dominik Kowald (KNOW)
V1.2	15/09/2022	15%	Updated executive summary and introduction	Dominik Kowald (KNOW)
V1.3	29/09/2022	40%	Service infrastructure and integration into TRUSTS platform	Dieter Theiler (KNOW)
V1.4	24/10/2022	50%	Related work about recommender systems in data markets	Peter Müllner (KNOW)
V1.5	03/11/2022	75%	Evaluation of the TRUSTS recommender system	Peter Müllner (KNOW)
V1.6	10/11/2022	80%	Research results, System Architecture, Data Scheme and Service Interface	Dieter Theiler, Peter Müllner, Emanuel Lacic, Dominik Kowald (KNOW)
V1.7	18/11/2022	85%	Conclusion and polishing	Dieter Theiler, Peter Müllner, Emanuel Lacic, Dominik Kowald (KNOW)
V2.0	05/12/2022	100%	Integrating feedback from internal review	Dieter Theiler, Peter Müllner, Emanuel Lacic, Dominik Kowald (KNOW)

[1] According to TRUSTS Quality Assurance Process:

1. to be declared

Disclaimer

Copyright message

Table of Contents

List of Figures

List of Tables

Glossary of terms and abbreviations used

Abbreviation / Term	Description
DMA	Data Market Austria
CF	Collaborative Filtering
MP	Most Popular
CB	Content-based Filtering
SP	Service Provider
DML	Data Modification Layer
IDS	International Data Spaces
RE	Recommender Engine
RC	Recommender Customizer
REV	Recommender Evaluator
ML	Machine Learning
KNN	K-Nearest Neighbors

1 Executive Summary

The creation and enrichment of user and corporate profiles is the basis for developing brokerage services that aim to provide recommendations for interlinking users with various offers available in the TRUSTS platform, i.e., services and datasets. Thus, the aim of this deliverable is to describe the second and final version of the TRUSTS recommender system that is developed for these purposes in course of T3.6. This deliverable directly builds on the deliverable D3.12, which described the first and initial version of the TRUSTS recommender system and which was successfully submitted in M18.

In D3.12., we described functional requirements identified in D2.2 and defined architectural requirements as well as six recommendation use cases for the TRUSTS recommender system. This included (i) the recommendation of datasets to users, (ii) the recommendation of services to users, (iii) the recommendation of datasets to services, (iv) the recommendation of services to datasets, (iv) the recommendation of datasets to datasets, and (vi) the recommendation of services to services. Furthermore, we described a scalable recommendation architecture that is capable of supporting these use cases, and that can consume data generated in the TRUSTS platform in line with the IDS information model. In order to test and fine-tune our recommendation algorithms, we proposed an offline evaluation plan using data gathered from the OpenML platform, which is a dataset and service sharing platform. Finally, we presented initial results of a privacy-aware recommendation experiment, in which we aimed to provide quality recommendations with a limited amount of private user data.

In D3.13., we provide an updated and final description of our functional requirements and the service interfaces of the TRUSTS recommender system. We also describe how the recommender system is integrated in the TRUSTS platform. Additionally, we report the offline evaluation results based on the data we gathered from OpenML. Our results show that recommendations based on collaborative filtering are the most accurate ones. However, we also find that content-based recommendations are the least popularity-biased ones that also provide the highest dataset and service coverage. This is in line with the fundamental accuracy-popularity bias trade-off present in recommender systems.

Finally, we summarize all research outputs achieved in course of T3.6. This includes a paper presented at ECIR'2021 (European Conference on Information Retrieval) on privacy aspects of recommender systems as well as two papers presented at ECIR'2022 on popularity bias in recommender systems. In a fourth paper at ECIR'2023, we presented the ScaR[2] recommender framework, which was also used to implement the TRUSTS recommender system. Additionally, we will present a paper at the DataEconomy workshop in 2022 on our offline recommender evaluation using OpenML data mentioned beforehand.

Taken together, the purpose of this deliverable is to demonstrate the final and integrated version of the TRUSTS recommender system. Additionally, this deliverable documents the underlying technical specifications of its service interfaces, and how it is integrated into the TRUSTS platform. This is complemented by the presentation of an offline evaluation study of the recommender system as well as a summarization of all research outputs produced in T3.6.

[2] http://scar.know-center.tugraz.at/, accessed Dec 2, 2022.

2 Introduction

Data is a substantial factor in the economy of the 21st century. It is a driver for growth and innovation and penetrates ever more and more aspects of private and corporate life. In fact, it has become an important input in many commodities (e.g., the internet of things) and services. The data economy is generating value from gathered information which was not possible even a few years ago and the current prospect is that this will even intensify further.

Besides all its benefits, data also has two major drawbacks which prevents it from fulfilling its transformational potential. Firstly, high-quality data is hard to come by and secondly it is even harder to extract valuable information from it. These two points shall be briefly discussed.

The inaccessibility of high-quality data partly stems from the fact that most of it is stored in data silos owned by big tech companies (e.g., Amazon, Facebook, Google), which gain an advantage over their competitors by keeping the data to themselves, or by so called data aggregators (e.g., Bloomberg, Reuters) who act as de facto monopolists and provide their services at high prices. Another factor is the reluctance of organizations to share their own data because there are no valuable business models or security/privacy concerns. As a result, data cannot be seen as a commodity, which can be sold or bought in a frictionless manner, which is detrimental to its distribution. Another difficulty hindering broader access is the lack of transparency regarding property rights, which makes it difficult to determine who owns what kind of data. Potential legal repercussions therefore act as a hindrance to data transfer. Finally, as a result of the preceding arguments, there is often no price, or one that does not reflect actual demand and supply, on data. Because of this fact there might not be enough incentives to distribute or even create data in the first place.

The second drawback is the complexity and difficulty of extracting valuable insights from data. To apply the right algorithm for a given dataset and use case, it requires well-founded knowledge of data science, statistics, and mathematics as well as domain knowledge. Personnel with these skills is rare and often too expensive for small and medium sized firms. This circumstance makes it difficult for all but the largest corporations to reap the full benefits of the data economy.

The two main drawbacks discussed above can both be addressed by establishing a trusted marketplace for data, which is the vision of TRUSTS. Not only would a data market bring together the producers and consumers of data but also the experts developing and applying algorithms. The monetary incentives would ensure that the demand roughly matches the supply and that the quality of goods (i.e., datasets and services) achieve a constantly high level. In addition, transactions would be contract based, clearly reflecting property rights.

Markets are well suited for matching tasks between different actors but also require a high level of knowledge about the respective matter to live up to their full potential. The knowledge intensity is particularly demanding when trading data and algorithms whose value for non-specialists often does not manifest itself at first glance. So, actors in those markets are prone to imperfect information. Therefore, to drive down transaction costs and to reduce the potential of market failure, a brokering instance in the form of a recommender system is needed.

The development and evaluation of this recommender system is the main objective of T3.6. To achieve this, we transferred the functional requirements of the TRUSTS platform related to brokerage to recommendation use cases, which we further translated to architectural requirements for the TRUSTS platform. Based on these architectural requirements, we designed a recommendation system

architecture, a data scheme as well as service interfaces used for integration into the platform. Apart from that, we also performed an offline evaluation using data gathered from the machine learning platform OpenML and conducted research in the areas of privacy-aware recommendations as well as popularity bias of recommendations.

It should be noted that in TRUSTS we also have applications (i.e., apps), but from the perspective of the recommender system, *services* and *applications* can be treated interchangeably. Thus, in this deliverable we solely talk about services but also mean applications. The same is true for *users* and *corporates*, and thus we solely use the term *user* in this deliverable but also mean corporates.

Taken together, the results achieved in T3.6 are three-fold:

1. From an evaluation perspective, T3.6 used publicly available data from the OpenML data and service sharing platform to evaluate the TRUSTS recommender system. This also allowed us to fine-tune the recommendation algorithms, and we shared our created OpenML dataset with the research community.

2. From a research perspective, T3.6 contributed 5 scientific publications. This includes a paper presented at ECIR'2021 on privacy aspects of recommender systems, as well as two papers presented at ECIR'2022 on popularity bias in recommender systems. In a fourth paper at ECIR'2023, we presented the ScaR recommender framework, which was also used to implement the TRUSTS recommender system. Additionally, we will present a paper in the ACM SIGCOMM DataEconomy workshop in 2022 on our offline recommender evaluation using OpenML data.

3. From a technical point of view, T3.6 fully integrated the recommender system into the IDS-based infrastructure of the TRUSTS platform.

2.1 Mapping Project Outputs

Purpose of this section is to map the TRUSTS Grant Agreement commitments, both within the formal deliverable and task description, against the project's respective outputs and work performed.

Table 1: Adherence to TRUSTS GA Deliverable & Tasks Descriptions.

TRUSTS Task		Respective Document Chapter(s)	Justification
T3.6.: User and corporate profiles and brokerage	Based on the work in T3.4, dataset and service descriptions and interactions with the platform are processed by information extraction algorithms. The extracted information is the basis for recommendations and matchmaking algorithms with user and corporate profiles. With regard to datasets, services for the analysis of this data are suggested or other data for enrichment and combination might be suggested.	Section 4.1. Section 4.2. Section 4.3. Section 4.4. Section 5 Section 6	The sections describe the whole design process of the TRUSTS recommender system, starting with the requirements

	Similarly, with regard to services, potential input data as well as pre- and post-processing services might be suggested. The extracted information can be used to improve descriptions and profiles of datasets and services. This leads to brokerage activities, where a mapping between offerings and demands of data and services is made. If no valid mappings can be established, suggestions are generated to publish new data or services.		and use cases, over the system architecture and service interface integration, to the evaluation plan and research results.
TRUSTS Deliverable			
D3.13: Profiles and Brokerage II This deliverable constitutes demonstrator systems that show practical application of the developed algorithms to production data. It describes the evolution of the contributions identified and provided in D3.12 (i.e., suitable recommendation use cases and applicable algorithms and datasets to support them, as well as a proof-of concept demonstrator), incorporating new insights and describing how brokerage functionality is leveraged in the integrated TRUSTS platform.			

2.2 Deliverable Overview and Report Structure

In the following, we give an overview of the structure of this demonstrator deliverable. In Chapter 3, we give an overview about existing recommender systems solutions in data markets and platforms, including the Data Market Austria recommender system. Subsequently, in Chapter 4, we outline the functional requirements (see also D2.2.) as well as the architectural requirements (see also D2.6.) for the TRUSTS recommender system. Additionally, Chapter 4 describes the system architecture, data scheme, and service integration into the TRUSTS platform. Finally, Chapter 5 includes an offline evaluation experiment using data gathered from the OpenML platform, and Chapter 6 summarizes the research outputs of T3.6. We close this deliverable in Chapter 7 with a summary of our findings and contributions as well as an outlook into potential future work.

3 Recommender Systems in Data Markets and Platforms

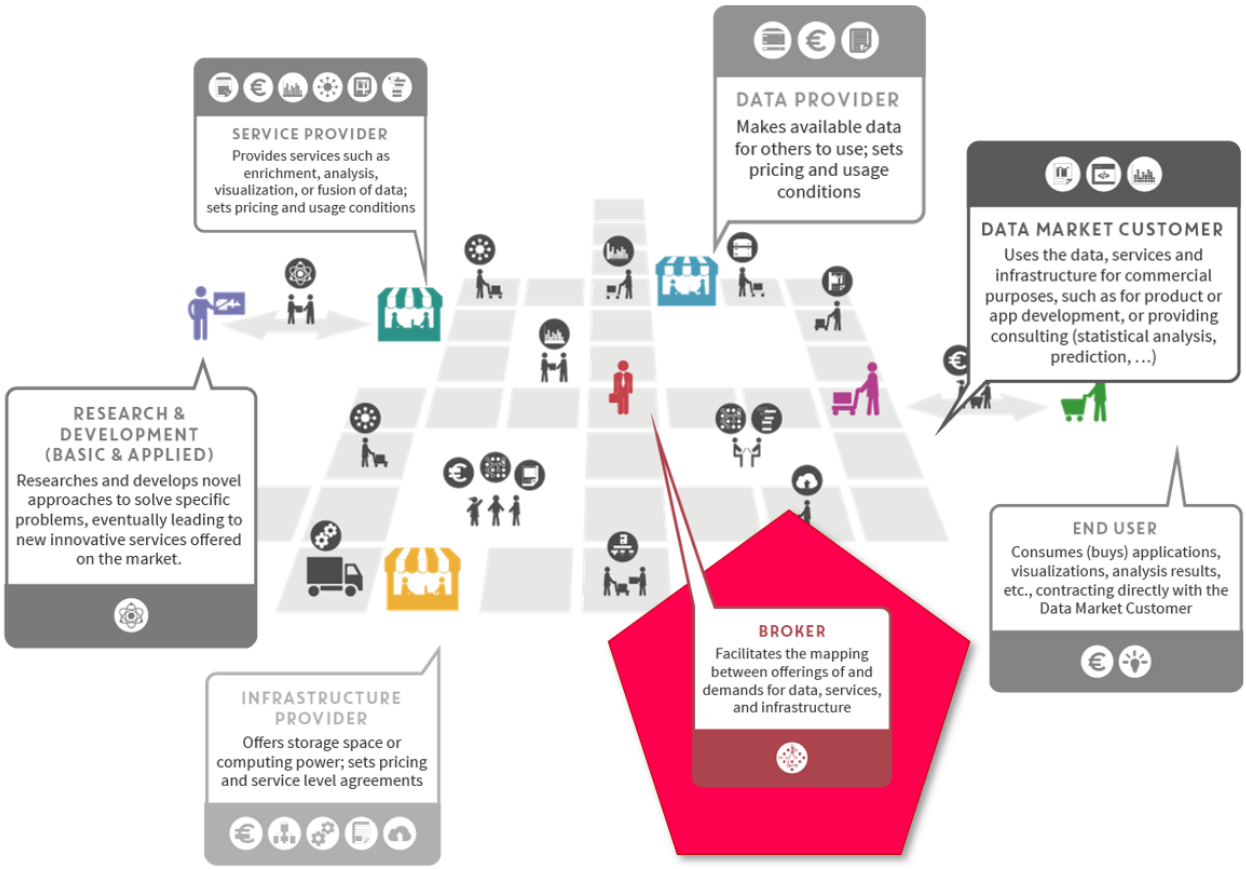
Recommender systems for data and services are of growing interest to both academia and industry in the field of data and AI-driven economies. Traditionally, recommender systems are deployed in domains tightly connected to private customers, for example, the movie, book, music, or e-commerce domain. However, the growing interest in recommender systems for AI-driven economies stems from the immense amount of data that is collected and stored by firms (Stahl et al., 2016; Liang et al., 2018; Fernandez et al., 2020). To filter this overload of information, employing recommender systems is a natural choice.

In this vein, Jess et al. (2015) design a recommender system for the industrial domain and use artificial data to help human decision-making for supply-chain and financial problems. Also, Patra et al. (2020) utilize content-based Filtering for dataset recommendations in the genetics domain.

Besides, research on service recommendations overlaps with research on Automated Machine Learning, which automatically compiles a machine learning pipeline (including algorithms) for a given dataset and problem (He et al., 2021). For example, Zschech et al. (2019) recommend a service in the form of a data mining pipeline, whereas Vainshtein et al. (2018) and Song et al. (2012) recommend services in the form of classification algorithms via exploiting metadata and structural properties of datasets.

Many research works separate dataset and service recommendations. However, research is scarce on how these two strands can be a single recommendation framework for an AI-driven economy. For example, Data Market Austria (DMA)[3], which employs a recommender system to connect users, datasets, and services (Kowald et al., 2019), see also Figure 1. However, the recommender system's interaction-based recommendation algorithms fail to provide accurate recommendations in the case recommendations need to be generated between datasets and services. Moreover, the authors raise concerns about the realism of their dataset used for their study.

In TRUSTS, we will extend this work into three directions: First, besides interaction-based recommendation algorithms, we will employ content-based recommendation algorithms (see Section 4.2). Second, we will develop recommendation use cases between datasets and dataset providers, and services and service providers. Third, we will evaluate our work using a novel dataset based on the



OpenML platform, which enables studying recommender systems for AI-economies.

[3] Data Market Austria: https://datamarket.at/, accessed Dec 2, 2022

Figure 1: Overview of DMA ecosystem with broker/recommender system for matchmaking users, datasets, and services.

4 TRUSTS Recommender System

In this section, we describe the recommender system implemented in TRUSTS. This includes (i) its functional as well as architectural requirements and use cases, (ii) its system architecture, data scheme and service interfaces, and (iii) the integration into the TRUSTS platform.

4.1 Functional Requirements, Recommendation Use Cases, and Architectural Requirements

The functional requirements of the TRUSTS recommender system have been developed in WP2 and are documented in detail in D2.2[4]. In Table 2, we summarize these functional requirements related to our recommender system.

Table 2: Functional requirements of the TRUSTS recommender system.

Recommender system: Related functional requirements	
FR6	The system should be able to provide datasets and services recommendations to its users pertaining to their profile and needs.
FR7	The system should employ matchmaking mechanisms through which hosted datasets are matched with hosted services (e.g., suitable for their analysis) and vice versa.
FR8	The system should identify and match related datasets so as to provide combined and enriched data.
FR9	The system should be able to improve datasets and services profiles based on extracted information originating from the available data.
FR17	The system should provide brokerage mechanisms for addressing the offerings and demands of the hosted datasets and services.
FR25	The system should be able to keep continuously updated profiles of the hosted datasets and services based on related interactions performed with the system.
FR26	Dataset discovery should be based on the FAIR principle.

[4] D2.2: https://www.trusts-data.eu/wp-content/uploads/2020/10/D2.2-Industry-specific-requirements-analysis-definition-of-the-vertical-E2E-data-marketplace-functionality-and-use-cases-definition-I.pdf, accessed Dec 2, 2022.

FR34	The system should allow consumers to announce their need for specific datasets / services if there are not any available, already.

Based on these functional requirements, we derived six recommendation use cases that fulfill these requirements. They are visualized in Figure 2. Here, RUC1 and RUC2 (i.e., recommending datasets/services to users) address FR6, RUC3 and RUC4 (i.e., recommending datasets to services and vice versa) address FR7, and RUC5 and RUC6 (i.e., recommending similar datasets/services for a given dataset/service) address FR8. FR6-FR8 are also summarized in the form of FR17, and FR9 and FR25 deal with creating and updating user/dataset/service profiles needed to generate recommendations. FR26 is realized since recommender systems, by design, support the findability of datasets and services. Finally, FR34 is addressed in T3.6 in the form of providing dataset and service recommendation in case a dataset/service search query leads to an empty result list.

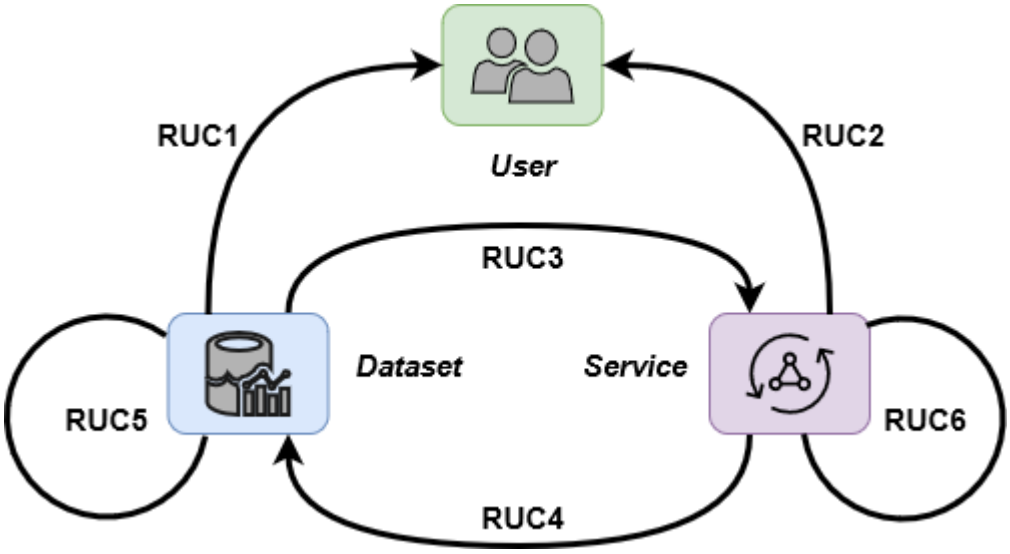


Figure 2: Recommendation use cases in TRUSTS.

In order to implement these use cases, we define architectural requirements to the TRUSTS infrastructure that are also described in D2.6. Table 3 gives an overview of these architectural requirements.

Table 3: Architectural requirements of the TRUSTS recommender system.

Recommender system: Architectural requirements	
AR3.6.1	**Notification mechanism to provide data for the recommender system.** In order to provide the recommender system with data for training its algorithms, the TRUSTS platform should provide a mechanism to transfer data to the recommender system. Therefore, the recommender system will provide REST-based services to add (i) metadata of datasets, (ii) metadata of services, (iii) metadata of users, and (iv) interactions between those entities (e.g., if a user downloads a dataset). The TRUSTS broker and the TRUSTS platform

	should use these services in order to notify the recommender system when new entities or interactions come into the platform or when existing entities are changed.
AR3.6.2	**User interface component to show recommendations.** For visualizing recommendation results to users, the TRUSTS platform should provide a user interface component that is capable of showing an ordered list of recommendations. For this purpose, the recommender system will provide REST-based services for six recommendation use cases: (i) recommend datasets to users, (ii) recommend services to users, (iii) recommend datasets to services, (iv) recommend services to datasets, (v) recommend datasets to datasets, and (vi) recommend services to services. The TRUSTS platform needs to use these services to query recommendations by providing parameters such as the current user, the currently viewed dataset or service, one of the six mentioned use cases, the algorithm (most popular, collaborative filtering, or content-based filtering) and the number of recommendations to generate (the default value is 10).
AR3.6.3	**User interface component to interact with recommendations.** When recommendations are shown to users, the TRUSTS platform should also allow them to interact with the recommendations, i.e., click on the recommendations to get additional information. Thus, for every recommendation request, the recommender system will generate a unique recommendation ID that is provided with the list of recommendations. The TRUSTS platform needs to store this recommendation ID and whenever a user interacts with a recommended entity informs the recommender system about this interaction, which is interpreted as feedback to the recommendation. With this, the recommender system is able to evaluate the quality of the recommendations and adapt the algorithms if necessary. Furthermore, this allows us to conduct A/B tests and compare the quality of three types of algorithms (i.e., most popular, collaborative filtering, and content-based filtering).

In the current architectural vision of the TRUSTS platform, as described in deliverable D2.7[5], the sources of the information mentioned above are threefold. First, the Broker which will register metadata on assets will make available messages (or relevant parts thereof) regarding creation/modification of metadata to the recommender system. Second, the contracting system, which is implemented as a distributed ledger of transactions, will be the source of data regarding user-asset interactions. Finally, the different user interfaces of the platform will provide information regarding the interactions.

4.2 System Architecture, Data Scheme, and Service Interfaces

This section gives a technical overview of all components and services of our recommender system.

4.2.1 ScaR and its Modules

As mentioned earlier, the system architecture used in the TRUSTS recommender system for data markets is based on the scalable recommendation framework ScaR[6]. Its general infrastructure and

[5] D2.7: https://www.trusts-data.eu/wp-content/uploads/2022/01/D2.7-Architecture-design-and-technical-specifications-document-II_Dec2021.pdf, accessed Dec 2, 2022.
[6] ScaR: http://scar.know-center.tugraz.at/, accessed Dec 2, 2022.

modules are depicted in Figure 3. The following briefly elaborates on the function of each of the submodules and their interconnections to each other.

Service Provider (SP): The SP functions as the main entry- and communication-point between the TRUSTS platform and the recommender system. Its RESTful services allow for requesting recommendations for one of the six recommendation use cases described in Table 4 as well as uploading dataset/service-, user- and interaction-metadata to the backend database.

Data Modification Layer (DML): The DML serves as data transfer intermediary between the individual ScaR modules and performs CRUD (create, retrieve, update and delete) operations in interaction with Apache Solr[7]. This particular search platform is being utilized for its multi core system – incorporating item, user, interaction, and feedback data – and provides scalability as well as support for (near) real-time data retrieval.

Recommender Engine (RE): The RE is the centerpiece module of ScaR since its purpose is to calculate recommendations. Apache Solr's built-in data structures allow for efficient similarity calculation. The RE supports standard approaches like collaborative and content-based filtering as well as hybrids between them. In addition, other algorithms can be added as needed depending on the particular use case.

Recommender Customizer (RC): The RC module holds customization profiles for each of the recommendation algorithms. It allows for an easy adjustment of the individual input parameters (e.g., the number of recommended items) by the admins of TRUSTS. The RE automatically takes into account those respective profiles which therefore have a direct effect on the calculation of the recommendations.

Recommender Evaluator (REV): The REV is used for evaluating the algorithms applied by the RE. When triggered it runs an offline evaluation based on training/test set splits or supports the execution of A/B tests. In the TRUSTS project, we will focus on offline evaluation studies.

[7] https://solr.apache.org/

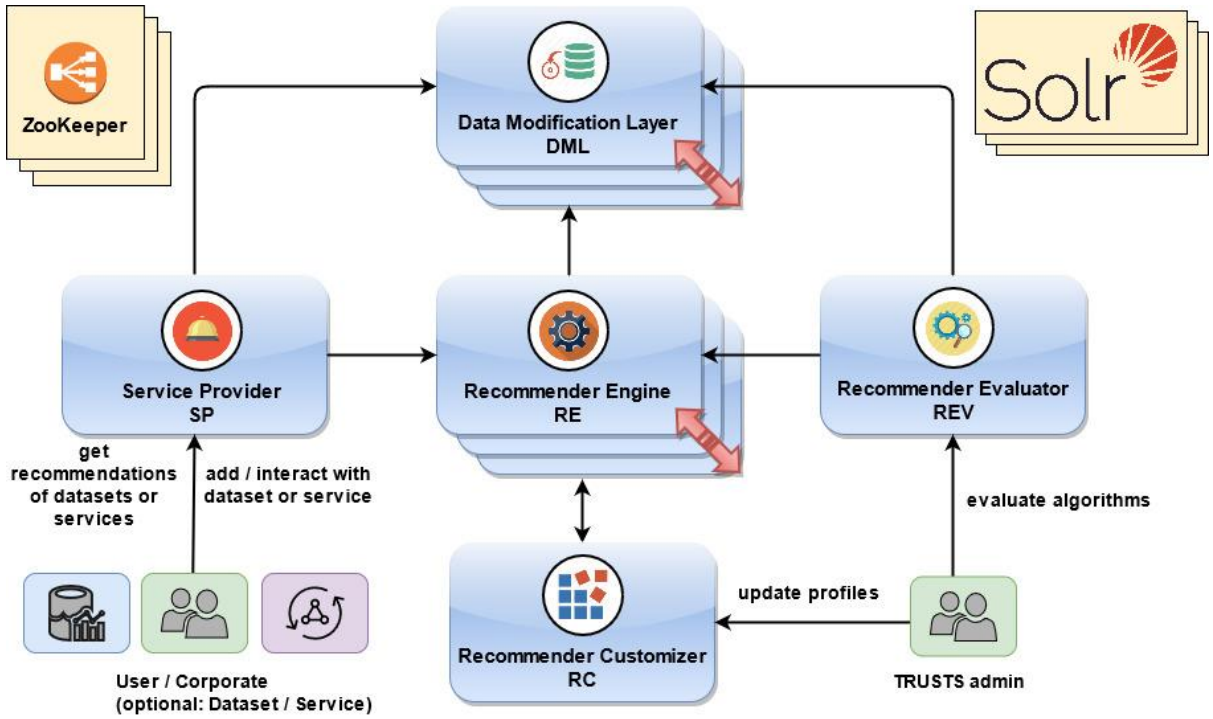


Figure 3: System architecture of TRUSTS recommender system.

4.2.2 REST-Services

This section provides an overview over the REST-API which can be subcategorized into (i) **Data Ingestion** and (ii) **Recommendation Services**. The former encompasses calls to the following REST-Resources:

- **Data Resource**: Handles the storage of metadata related to datasets, services, and users.
- **Interaction Resource**: This service handles the storage of buy-, view- (i.e., click) and download-interactions related to datasets and services. Please note that these three types of interactions are examples of interaction types that we expect to have in the TRUSTS platform.

The latter contains one REST-Resource, which is the **Recommendation Resource** that handles requests of a pre-specified number of recommendations for one of the six recommendation use cases. Figure 4 and Figure 5 depict the Swagger user interface for the aforementioned REST-Resources including their respective endpoints which will be described in the following subsections.

4.3 REST-Services

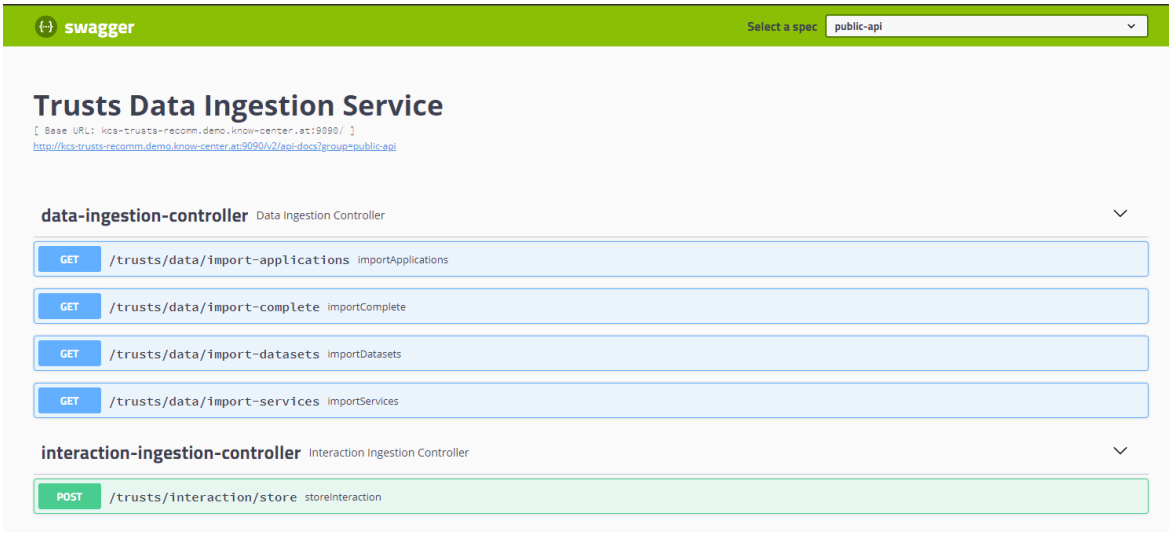


Figure 4: Overview of the Data Ingestion REST-Service.

Data Ingestion Service

This service is used for uploading metadata of datasets, services, and applications to the database. It also serves as a filter and extracts recommendation-relevant information of the received payload.

As the IDS-IM (International Data Spaces – Information Model) defined in the IDS Ontology[8] draft is designed to foster a central agreement between different services which share and apply data assets, the Data Ingestion Service queries the central Broker for assets in respective format and processes the retrieved information so that the data model of the Recommendation Service is populated. Hence, data held in the central Broker and represented with the help of IDS-IM classes like `Resource` or `App Resource` are queried in regular intervals to update data instances held in the recommender backend periodically.

As shown in Figure 4: Overview of the Data Ingestion REST-Service, to explicitly trigger certain updates, the Data Ingestion Service provides four endpoints to import applications, services, datasets or all assets at once, from the central Broker. The result of these endpoints is defined by the `DataResponse` class, which contains a HTTP response status code (*http_status*) and a *message* indicating the success or failure of the call. To insert interaction data, which is generated when users work with the TRUSTS Platform and is incorporated as a crucial source of information (besides assets metadata) for generating recommendations, the `/interaction/store` endpoint allows to store different types of interactions, namely *view*, *download*, *accept_contract*, *publish*, and *view_recomm*. These endpoints are designed to be triggered by the TRUSTS user interface according to user actions. *View* events are triggered when the user opens a data asset, *download* is sent when a user downloads a data asset, *accept_contract* is used when the user accepts a contract to allow her working with a certain asset, *publish* represents the event of making an asset available to be held in the central Broker and so to be used by other users and *view_recomm* is stored when a user actively opens a recommended asset.

The input object for storing interactions consists of the *recommenderId*, which is the id of the recommendation in case it was recommended (empty otherwise), the *type*, which stands for one of the

[8] https://international-data-spaces-association.github.io/InformationModel/docs/index.html

values used to describe the kind of interactions, the *userId*, which is filled with the id of the user responsible for the interaction event, the *timestamp*, which indicates at which point in time (in milliseconds) a certain interaction is made, the *entityId*, which defines the id of the asset which is targeted by respective interaction and the *entityType*, which depicts one the possible assets types, i.e., *application*, *service,* or *dataset*. The response is also defined by the `DataResponse` class.

Recommendation Service

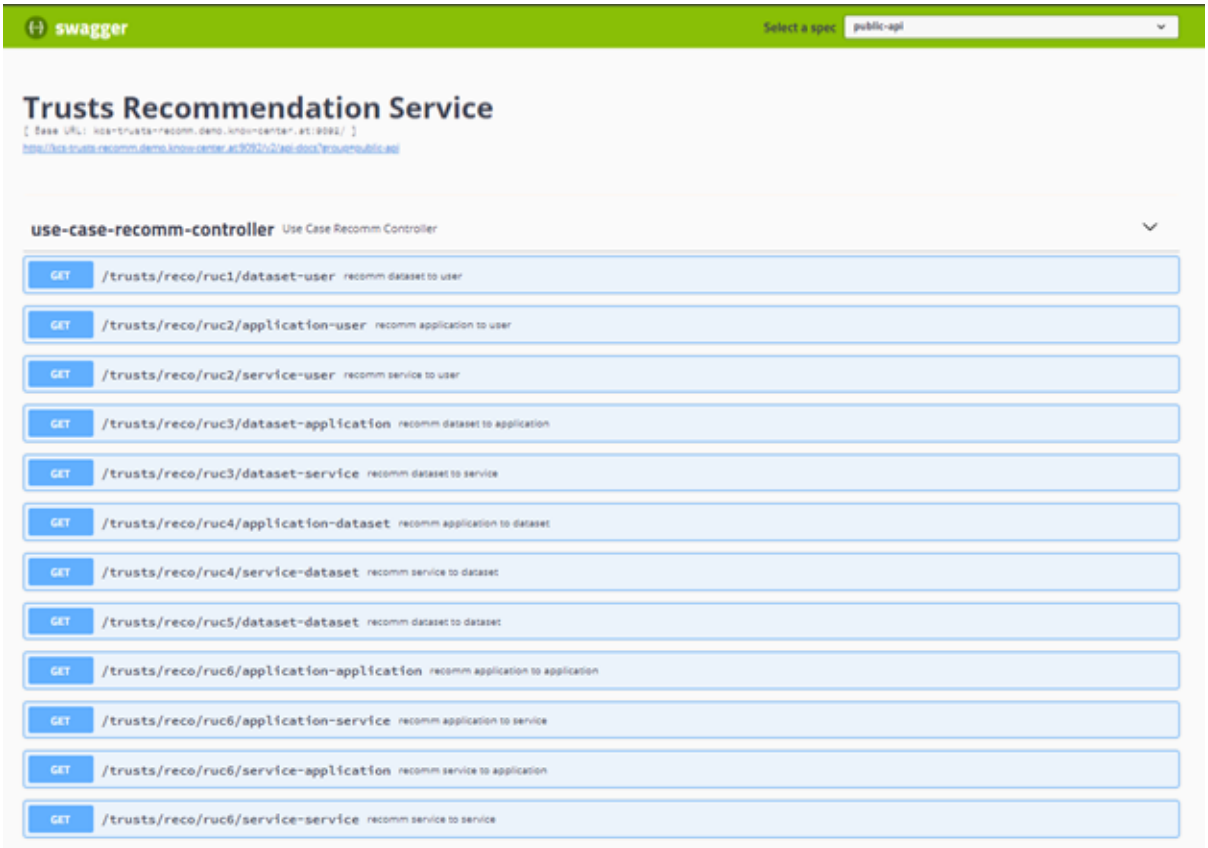


Figure 5: Overview of the Recommendation REST-Service.

This service is used for requesting recommendations for one of the six recommendation use cases. The Recommender Service takes query string parameters as input, which are *count*, *userId* and, depending on the actual type of entity which can be accepted by each endpoint as a recommendation target, either *applicationId*, *serviceId,* or *datasetId* can be given. *Count* is used to state the number of results expected, *userId* is used to define the id of the user requesting the recommendation, and *applicationId*, *serviced*, or *datasetId* represent the id of the asset for which recommendations should be generated. The response of the recommendation call is defined by the *RecoResult* class and contains the *http_status*, *message*, *reco_id* and *results*, where *reco_id* contains the id of the recommendation request and *results*, which is populated with actual recommendations.

4.3.1 Backend Database – Apache Solr

The current database utilized by the ScaR framework is Apache Solr[9]. In principle, ScaR is able to work with different document-based database engines, as the DML module is in charge of encapsulating the underlying database. So far, Solr was chosen as the main search instance as it provides two main advantages for the present use case. The first one is its query speed, which is most relevant for (near) real time applications like recommender systems. The second one is its ability to seamlessly work with several types of entities. If a project-wide decision enforces a different infrastructure at a level of data storage, also other search engines could be applied, e.g., ElasticSearch[10]. The current database of ScaR stores information about items, users, and interactions originating from the TRUSTS portal along with metadata about the generated recommendations in four different cores. These are *items*, which contains items of type application, service, and dataset, *interactions*, which contains interactions of users with items and *feedbacks*, which contains the calculated recommendations as well as information regarding the evaluation of the system.

Items Core

The Items Core contains application, service, and dataset data objects. Its schema-fields can be subdivided into two property-categories:

- **General fields:** These properties contain meta-information about the items themselves.
- **Interaction fields:** These properties contain information about the interaction type and the list of users who interacted with a specific item.

```
<!-- dataset, service, application -->
<field name="id"                          type="string"       indexed="true" stored="true" required="true"    />
<field name="type"                        type="string"       indexed="true" stored="true" required="true"    />
<field name="title"                       type="text_general" indexed="true" stored="true"                     />
<field name="description"                 type="text_general" indexed="true" stored="true"                     />
<field name="themes"                      type="text_general" indexed="true" stored="true" multiValued="true" />

<field name="language"                    type="string"       indexed="true" stored="true"                     />
<field name="version"                     type="string"       indexed="true" stored="true"                     />

<field name="created"                     type="date"         indexed="true" stored="true"                     />
<field name="modified"                    type="date"         indexed="true" stored="true"                     />

<!-- interactions -->
<field name="users_view"                  type="string"       indexed="true" stored="true" multiValued="true" />
<field name="users_view_count"            type="long"         indexed="true" stored="true"                     />

<field name="users_view_recomm"           type="string"       indexed="true" stored="true" multiValued="true" />
<field name="users_view_recomm_count"     type="long"         indexed="true" stored="true"                     />

<field name="users_download"              type="string"       indexed="true" stored="true" multiValued="true" />
<field name="users_download_count"        type="long"         indexed="true" stored="true"                     />

<field name="users_accept_contract"       type="string"       indexed="true" stored="true" multiValued="true" />
<field name="users_accept_contract_count" type="long"         indexed="true" stored="true"                     />

<field name="users_publish"               type="string"       indexed="true" stored="true" multiValued="true" />
<field name="users_publish_count"         type="long"         indexed="true" stored="true"                     />

<field name="interaction_last_modified"   type="date"         indexed="true" stored="true"                     />

<field name="invalid"                     type="boolean"      indexed="true" stored="true"                     />
```

Figure 6: Items Core.

[9] https://solr.apache.org
[10] https://www.elastic.co/elasticsearch

The fields belonging to the objects in the Items Core (see Figure 6) are *id*, the id of the item, *type*, the type of the stored item, *title*, the name given, *description*, a natural language-based description, *themes*, standardized item categories, *language*, a language code, *version*, the version of the item, *created*, the date and time indicating its creation time, *modified*, the date and time of the item being last modified, users_*type*, arrays of unique ids pointing to the users who targeted this item with interactions of type *type*, *users_type_count*, total count of interactions of a given type, *interaction_last_modified*, the date and time indicating the last modification of one of the item's interaction fields and *invalid*, a flag depicting whether the item is still valid for being shown in recommendations.

Interactions Core

The Interactions Core contains interaction data objects of users with applications, services and datasets. The fields with the corresponding data types and meta-information are listed in Figure 7.

```
<field name="id"                type="string"   indexed="true" stored="true" required="true"  />
<field name="type"              type="string"   indexed="true" stored="true" required="true"  />
<field name="timestamp"         type="date"     indexed="true" stored="true" required="true"  />
<field name="user_id"           type="string"   indexed="true" stored="true" required="true"  />
<field name="entity_id"         type="string"   indexed="true" stored="true"                  />
<field name="reco_id"           type="string"   indexed="true" stored="true"                  />
```

Figure 7: Interactions Core.

The fields belonging to the objects in the Interactions Core (see Figure 7) are *id*, the id of the interaction, *type*, the type of the stored interaction, *timestamp*, the date and time indicating when the interaction happened, *user_id*, the id of the user responsible for the interaction, *entity_id*, the id of the item the user interacted with and *reco_id*, the id of the recommendation resulting in the interaction.

Feedbacks Core

The Feedbacks Core contains information regarding recommendations and their respective evaluation metrics. The fields with the corresponding data types and meta-information are listed in Figure 8.

```
<field name="id"                          type="string"   indexed="true" stored="true" required="true" />
<field name="recomm_profile_name"         type="string"   indexed="true" stored="true" />

<field name="recomm_ids"                  type="string"   indexed="true" stored="true" multiValued="true" />
<field name="item_ids"                    type="string"   indexed="true" stored="true" multiValued="true" />
<field name="hybrid_recomm_profile_names" type="string"   indexed="true" stored="true" multiValued="true" />
<field name="hybrid_recomm_ids"           type="string"   indexed="true" stored="true" multiValued="true" />
<field name="hybrid_recomm_parent_id"     type="string"   indexed="true" stored="true" />
<field name="user_id"                     type="string"   indexed="true" stored="true" />
<field name="custom_filters"              type="string"   indexed="true" stored="true" />
<field name="recomm_algo"                 type="string"   indexed="true" stored="true" />

<field name="max_recomm_results"          type="int"      indexed="true" stored="true" />
<field name="recomm_type"                 type="string"   indexed="true" stored="true" />
<field name="recomm_time"                 type="date"     indexed="true" stored="true" />
<field name="duration"                    type="long"     indexed="true" stored="true" />

<field name="eval_id"                     type="string"   indexed="true" stored="true" />
<field name="expected_ids"                type="string"   indexed="true" stored="true" multiValued="true" />

<!-- number of interactions which happend with this recommendation -->
<field name="interaction_count"       type="long"         indexed="true" stored="true" />
```

Figure 8: Feedbacks Core.

The fields belonging to the objects in the Feedbacks Core (see Figure 8) are

- *Id*: the id of the recommendation

- *recomm_profile_name*: the name of the profile in the Recommender Customizer module used for generating the recommendation
- *recomm_ids*: an array of ids indicating the items which were recommended
- *item_ids*: an array of ids indicating the items on which the recommendation is based on
- *hybrid_recomm_**: these fields contain additional properties of the recommendation algorithm
- *user_id*: the id of the user who received the recommendation
- *custom_filters*: the recommendation filter specified on the client side used for filtering the results
- *recomm_algo*: the algorithm which was applied for calculating the recommendation
- *max_recomm_results*: the number of recommendations requested by the client
- *recomm_type*: a parameter indicating whether users or items were recommended
- *recomm_time*: the date and time indicating when the recommendation happened
- *duration*: the time it took the recommendation algorithm to finish
- *eval_id*: the id of the evaluation
- *expected_ids*: an array of items which should have been recommended - used for calculating evaluation metrics
- *interaction_count*: the number of interactions resulting from the recommendation

4.3.2 Recommendation Algorithms

We developed four types of recommendation algorithms to realize our six recommendation use cases:

- **Most Popular (MP)**: This is an unpersonalized algorithm that always recommends the most popular items (e.g., the datasets with the highest number of clicks).
- **Collaborative Filtering (CF)**: This is a personalized algorithm that analyzes the interaction data of items to find similar users, and then recommends items which these similar users interacted with.
- **Content-based Filtering (CB)**: This is another personalized algorithm calculating item-similarities based on content features (e.g., title, description text) and then recommends these similar items.
- **Hybrid (HYB)**: We are also able to combine the aforementioned three recommendation algorithms in a hybrid way, and thus, combine the advantages of the three methods.

4.4 Integration into the TRUSTS Platform

The recommender system is used in different places on the TRUSTS platform, according to the six recommendation use cases. For RUC1, the recommender suggests datasets to the user in the “Datasets” tab (see Figure 9: Datasets Recommendation), for RUC2, services and applications recommendations are generated and shown in respective tabs for “Services” and “Applications” (see Figure 10: Applications Recommendation and Figure 11: Services Recommendation). For RUC3, datasets are recommended for services and applications on the respective page of the selected service or application in the section “Datasets and services recommendations” (see Figure 12: Datasets to Service Recommendation). For RUC4, application and services are suggested for datasets and shown in section “Applications recommendations” and “Datasets and services recommendations” on the page of selected dataset (see

Figure 13: Services and Applications to Dataset Recommendation). For RUC5, datasets are shown for datasets on the respective page of a dataset in section “Datasets and services recommendations” (see Figure 14: Datasets to Dataset Recommendation). Finally, for RUC6, applications and services are recommended for each other in sections “Datasets and services recommendations” and “Applications recommendations” (see Figure 15: Applications and Services to Application Recommendation).

Additionally, interesting datasets, services, and applications are recommended in the top section of the search results page depending on the searched of the users. In each location where recommendations are generated, up to 9 recommendation results are shown (see Figure 16: Search Results Recommendation).

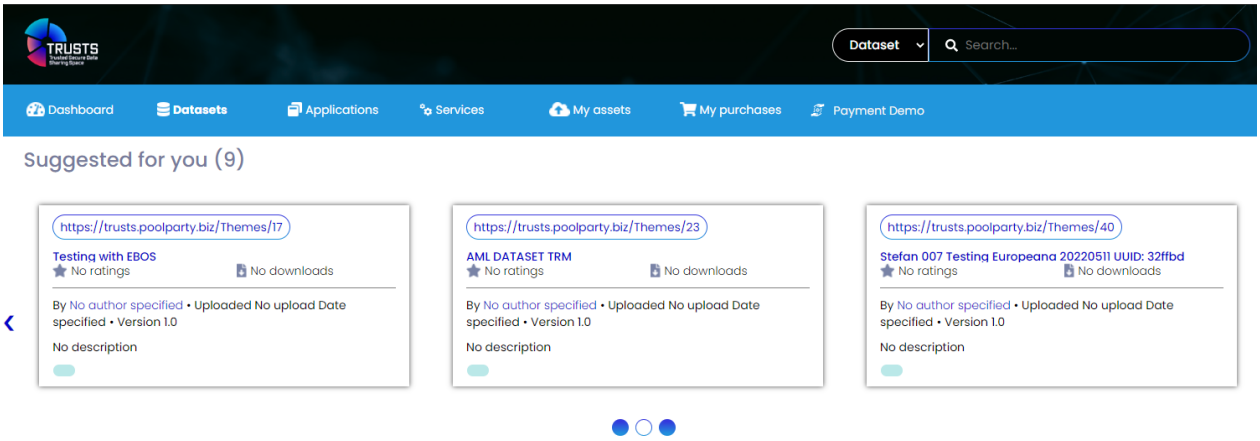


Figure 9: Datasets Recommendation.

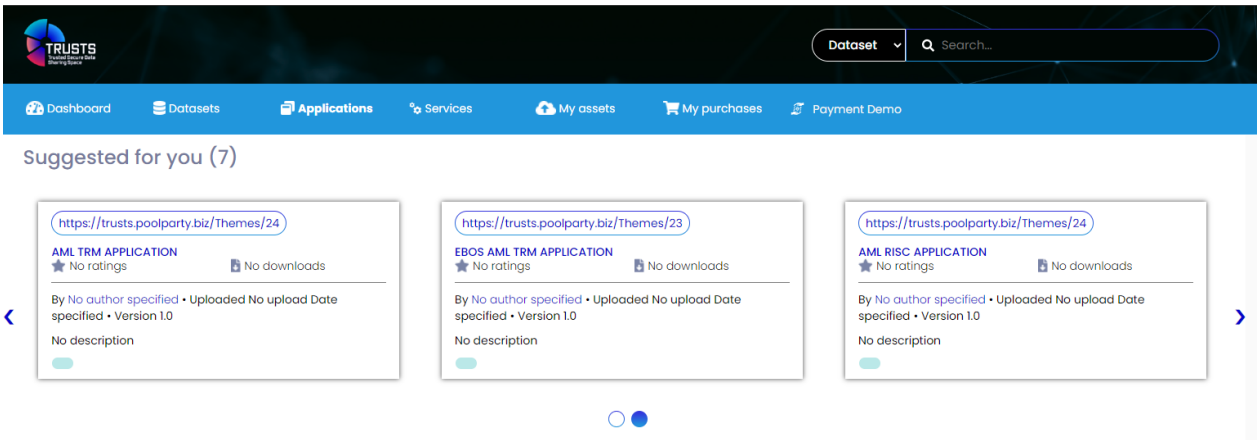


Figure 10: Applications Recommendation.

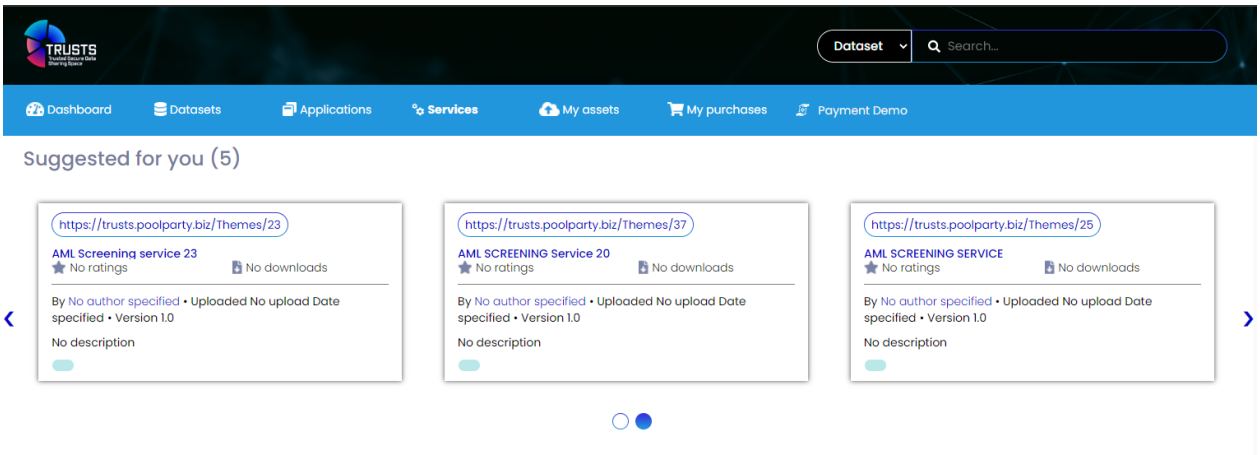


Figure 11: Services Recommendation.

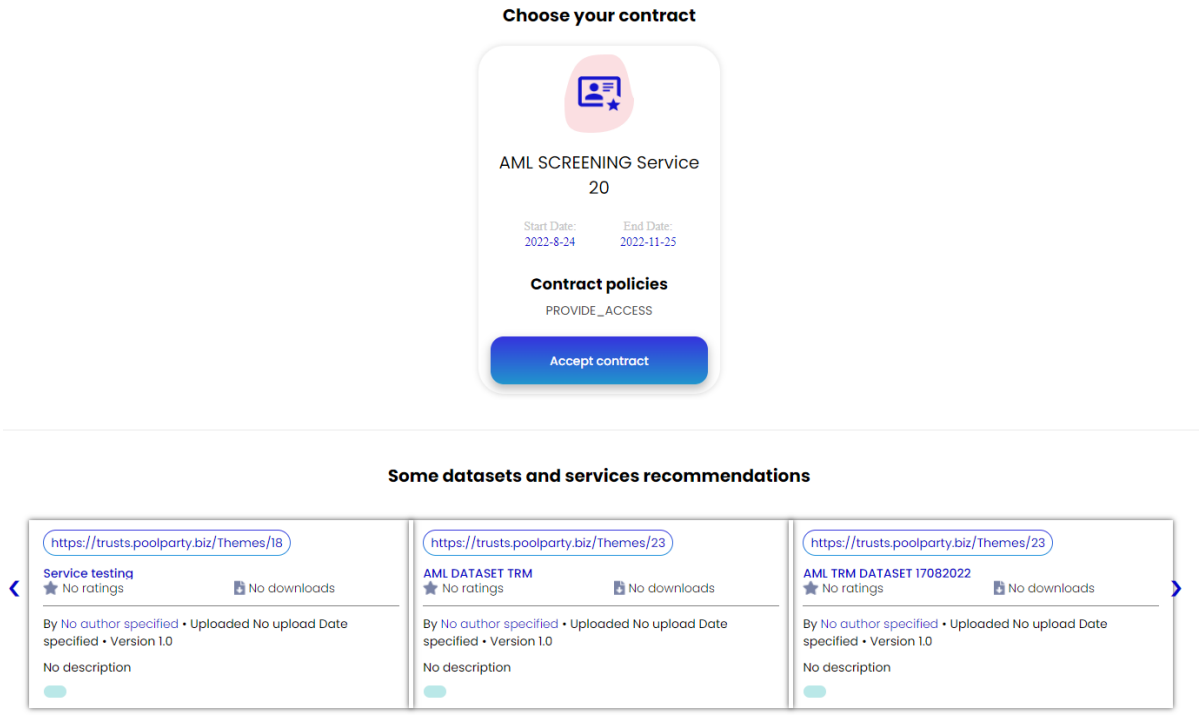


Figure 12: Datasets to Service Recommendation.

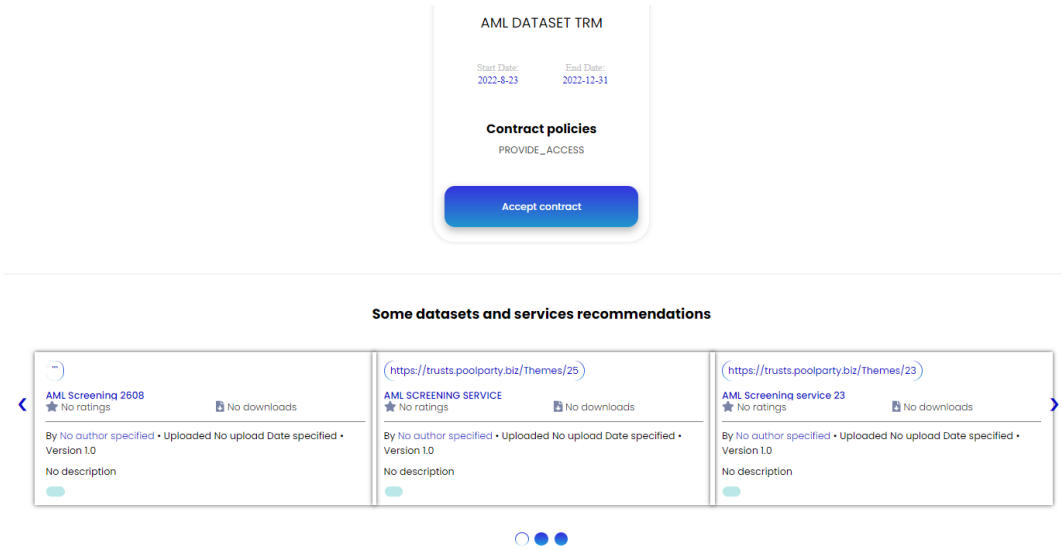


Figure 13: Services and Applications to Dataset Recommendation.

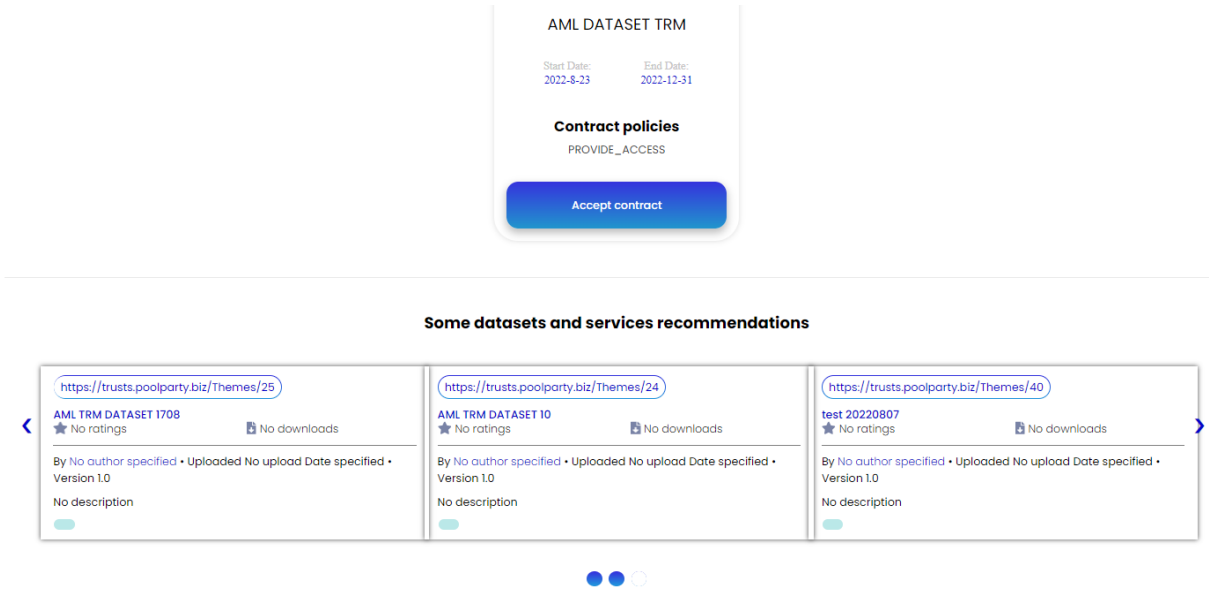


Figure 14: Datasets to Dataset Recommendation.

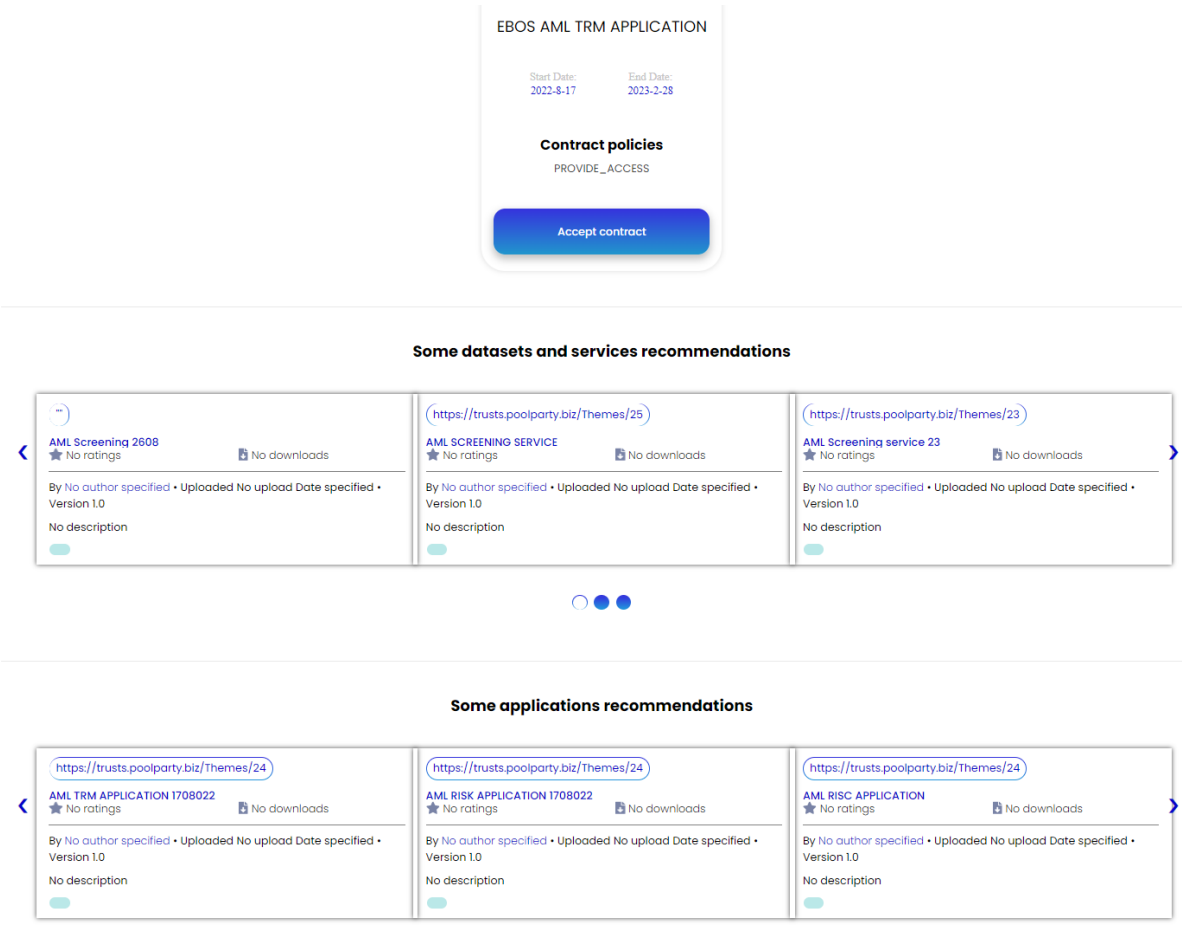


Figure 15: Applications and Services to Application Recommendation.

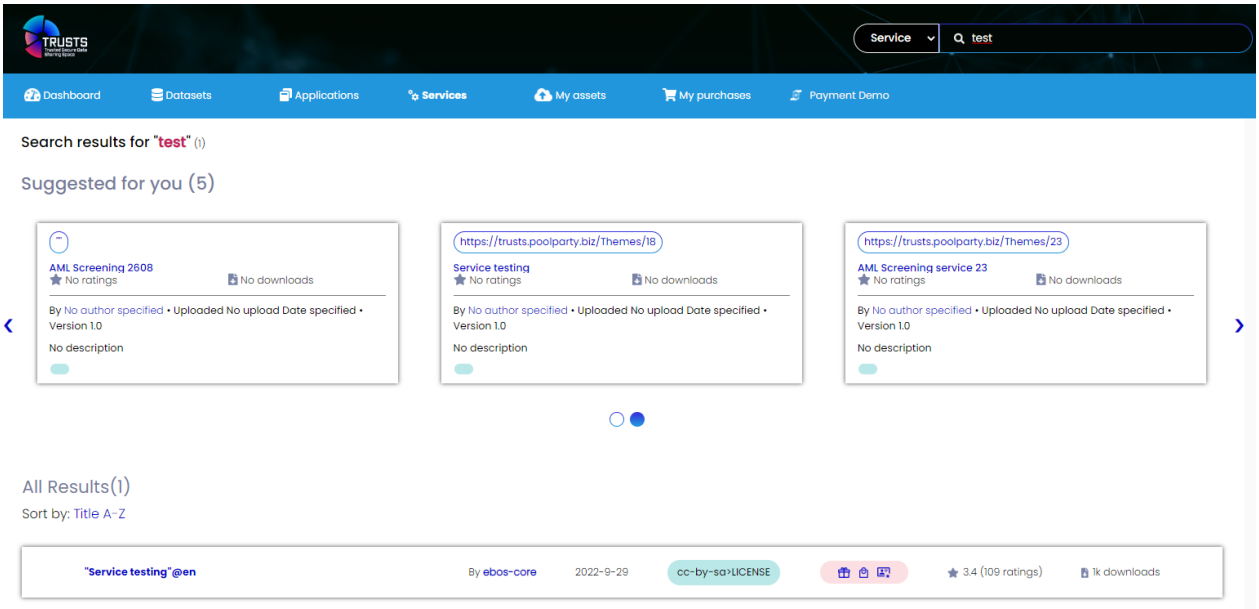


Figure 16: Search Results Recommendation.

To allow the REST-based requests to the recommender API to be instantiated in order to generate the results shown in the GUI of the TRUSTS platform at the described locations, we extended the “ckanext-ids” plugin, which was implemented during the TRUSTS project by partners involved in WP3. Therein, we integrated the graphical (i.e., HTML and JavaScript) skeletons to show and retrieve the recommendation results and integrated the invocation of the Recommender Service in the Python-based backend of the CKAN extension. In addition to the integration of recommender requests to retrieve suggestions at different places in the platform, we equipped the extension with invocations to the recommender system’s Data Ingestion Service, which stores the interactions of users working with the TRUSTS platform in the recommender system’s backend. These interactions in turn are used in real time to extend the recommender’s knowledge base.

The recommender system itself is hosted together with other services on TRUSTS nodes with the help of Docker and in particular Docker Compose. This way, the recommender system is deployed on premise in the customer’s environment and is only accessible via the TRUSTS platform. As the TRUSTS platform only instantiates the recommender system’s API from within the CKAN-based Python-backend, and no ports are open for communicating with the recommender from outside the Docker network, the recommender itself does not need to setup any further means to prohibit unauthorized access. Hence, the services, i.e., the Docker Containers holding the different modules of the recommender system, are instantiable from the CKAN service (i.e., “local-ckan”) and communicate with the Broker through the Dataspace Connector service (i.e., “dsc”) which is hosted in the same Docker environment.

To be able to calculate recommendations, the recommender system imports metadata for datasets, services and applications on a regular basis from the Broker, formats it according to the recommender’s data model and stores it in the described Solr backend. To receive respective metadata held in the Broker in form of IDS-structured entities, the Data Ingestion Service requests the Broker with the help of SPARQL queries sent to the Dataspace Connector’s REST API. The Dataspace Connector encapsulates the request in an IDS-formatted message, forwards it to the Broker and returns the resulting entities in a table-like string-based format, which is subsequently transformed to the needed data model.

5 Evaluation of the TRUSTS Recommender System

In the following, we present the evaluation procedure of our work (Müllner et al., 2022) on recommender systems for data and service sharing platforms. This work was accepted for presentation at the ACM SIGCOMM DataEconomy workshop. In TRUSTS, the recommender system requires profile data that is used for training and ground truth data that is used for testing. Profile data serves as a representation of a target entity and describes its item preferences. Based on this data, the recommender system is trained and eventually, recommendations are compared to the ground truth data to evaluate the predictive capabilities of the recommender system. Typically, recommendations need to be generated between users and items (e.g., recommending datasets to users). In TRUSTS however, recommendations also need to be generated between services and datasets (see Figure 2). This is a different recommendation use case than in case of traditional recommender systems. In total, we identify six recommendation use cases, for which direct or indirect interaction data has to be exploited to evaluate the recommender system (see Table 4). User-to-item interactions (e.g., a user utilizing a dataset) are direct interactions. However, items, i.e., datasets and services, cannot directly interact with each other; a user has to run a service on a dataset. Thus, we use direct user-to-dataset and

user-to-service interactions to connect datasets and services indirectly. Finally, for items of the same type (e.g., datasets and datasets) the ground truth data is unavailable, since in this case, relevant items cannot be identified with our previous approach.

Table 4: Recommendation use cases.

	Profile Data	**Ground Truth Data**
RUC1: Datasets to Users	direct	direct
RUC2: Services to Users	direct	direct
RUC3: Datasets to Services	indirect	indirect
RUC4: Services to Datasets	indirect	indirect
RUC5: Datasets to Datasets	indirect	unavailable
RUC 6: Services to Services	indirect	unavailable

5.1 Evaluated Recommendation Use Cases

In **RUC1 (Datasets to Users)**, recommendations help users to identify relevant datasets. There exists a direct interaction between users and datasets (e.g., a user uses a dataset for a service) and, thus, the recommender system can leverage these interactions to generate recommendations.

In **RUC2 (Services to Users)**, recommendations help users to identify relevant services. As in RUC1, also in RUC2, the recommender system can leverage the direct interactions between users and services to generate recommendations.

In **RUC3 (Datasets to Services)**, recommendations help to identify suitable datasets for a given service. This use case can occur when service providers aim to improve their service via testing it on more datasets. In contrast to RUC1 and RUC2, indirect item-to-item interaction data has to be used to

generate recommendations. Since users interact with services and datasets, we can use user interactions to connect datasets and services. Specifically, the profile and ground truth data of a service consists of the datasets that users used for the service.

In **RUC4 (Services to Datasets)**, recommendations help to identify suitable services for a given dataset. This use case can occur when dataset providers search for other services that can be used with their dataset. Similar to RUC3, the profile and ground truth data of a dataset consists of the services that users used for the dataset.

In **RUC5 (Datasets to Datasets)**, dataset providers can find other datasets that are utilized by the same user-community to identify datasets of competitors. We build the profile data in the same way as in case of RUC4, i.e., the profile of a dataset consists of the services that users apply to the dataset. However, for building the ground truth data, we cannot use this idea, since we need a set of relevant datasets for a given dataset. Therefore, we create a collaboration network (Kowald et al., 2019). This means that we create a link between two datasets if they have been used by the same user. Thus, the ground truth data of a given dataset consists of the datasets that have the largest user overlap with this dataset.

In **RUC6 (Services to Services)**, service providers can find other services that are utilized by the same user-community. Similar to RUC5, this can identify services of competitors. We use the same idea as in case of RUC5 to create profile and ground truth data. Hence, the profile of a service consists of the datasets that users applied this service to and the ground truth data consists of the services with the largest user overlap.

5.2 OpenML Dataset

To evaluate the recommender system developed in TRUSTS with respect to recommendation accuracy and popularity bias, we conduct an offline evaluation study and use data gathered from the open-source machine learning platform OpenML. With our offline evaluation, we can test the TRUSTS recommender without the need to deploy and test it in the real world. This way, we do not risk to negatively impact the experience of entities (e.g., users) while evaluating the usefulness of different recommendation algorithms and settings. We rely on OpenML, since it provides an easy-to-use Python-based API to query the data necessary to generate a dataset for our offline evaluation. In OpenML, users can upload so-called runs. A run indicates that a user *u* applied a flow (i.e., an algorithm *a* with a certain parameter setting) on a certain task. Task describes an objective that is optimized through algorithm *a* on dataset *d*. For example, user *u* applies algorithm *a* on a regression task on dataset *d*.

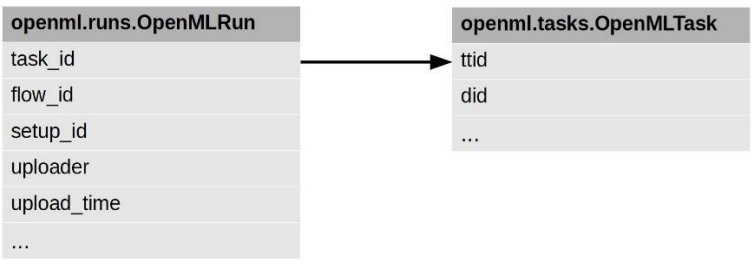


Figure 17: The connection between runs and datasets. task_id and ttid are the unique identifier of a certain task, and did is the identifier of the corresponding dataset.

We generate our evaluation dataset via the following three retrieval steps (i) with *openml.datasets.list_datasets* and *openml.flows.list_flows*, we retrieve all datasets and flows alongside their textual description. Furthermore, we apply stemming on these textual descriptions via using standard python functionality; (ii) with *openml.runs.list_runs*, we retrieve all runs, i.e., a triple containing the user, algorithm, and task. We use *openml.tasks.OpenMLTask* to obtain the dataset for a given task (see **Fehler! Verweisquelle konnte nicht gefunden werden.Fehler! Verweisquelle konnte nicht gefunden werden.**), and (iii) since users can use the same algorithm for the same dataset for many times we merge all repeated interactions. We highlight that the algorithms retrieved from OpenML are referred to as "services" in the remainder of this work.

Our dataset includes users with an extraordinary large number of interactions. We observe that these users are test users and bots, i.e., artificial users to test the platform. Therefore, we delete all data from users, whose number of interactions exceeds the point of maximal curvature of the logarithmic-transformed interaction-distribution using the Python-based Kneed[11] library (Satopaa et al., 2011). With this, we reduce the number of interactions in our dataset from 7,593,184 to 10,945. However, since the removed interactions were generated by artificial users, we do not expect any impact on real users. After these steps, our dataset consists of 10,945 interactions between 512 users, 573 datasets, and 1,307 services. Additionally, it includes 2,104 datasets and 11,037 algorithms without user interactions. Further descriptive statistics of our generated OpenML dataset can be found in Table 5.

Table 5: Descriptive statistics of our dataset.

Users	544
Services	1,307
Datasets	573
Interactions	10,945
Avg. Interactions / User	21.38
Avg. Interactions / Services	8.37
Avg. Interactions / Dataset	19.10
Services without Interactions	11,037
Datasets without Interactions	2,104

In order to evaluate the recommender system in our six recommendation use cases, we split the interaction data from each target entity (user, service, or dataset) into 80% training data and 20% test data. For RUC3 and RUC4, we create training and test data by using indirect item-to-item interactions. For RUC5 and RUC6, we resort to Kowald et al. (2019) and construct the training data via a collaboration network which connects two datasets or two services if they were used by the same users. Plus, for the test data, we use the 10 datasets and services with the highest user overlap.

[11] Kneed: https://github.com/arvkevi/kneed/

5.3 Recommendation Algorithms

We test the recommender system employed in TRUSTS via employing three widely-used recommendation algorithms (Ricci et al., 2011): Most Popular MP, Collaborative Filtering CF, and Content-based Filtering CB. With respect to these recommendation algorithms, we use the built-in variants in the Java-based recommendation framework ScaR (Scalable Recommendation-as-a-service) (Lacic et al., 2013 and Lacic et al., 2015) with the default parameters. Moreover, all recommendation algorithms generate recommendation lists with 10 items. Also, we ensure that the recommendation lists do not contain items that are already known to the target entity.

MP always recommends the most popular datasets and services, i.e., those with the largest number of interactions, ignoring the true preferences of each target entity. Thus, recommendations are unpersonalized and only a small subset of items can be recommended.

In contrast, CF generates personalized recommendations for a target entity via exploiting similar target entities to identify items that are yet unknown, but considered to be relevant. For example, CF recommends a dataset d to a user u, if similar users (u's neighbors) have interacted with d. We use a CF variant with 40 neighbors.

However, MP and CF cannot recommend items without interactions. Thus, we utilize CB to exploit content instead of interactions to generate recommendations. Specifically, in our work, CB exploits TF-IDF representations (Achananuparp, 2008 and Jones, 1972) of textual descriptions of datasets and services. We rely on ScaR's default settings and set the minimum term frequency to 1 and the minimum document frequency to 2.

5.4 Evaluation Criteria and Results

We evaluate the three recommendation algorithms in our six recommendation use cases along two evaluation criteria: recommendation accuracy and popularity bias.

Specifically, for **accuracy**, we use five well-established metrics (Parra and Sahebi, 2013): Precision P@k, Recall R@k, Mean Reciprocal Rank MRR@k (Radev et al., 2002), Mean absolute Precision MAP@k, and Normalized Discounted Cumulative Gain nDCG@k (Järvelin and Kekäläinen, 2002). P@k is the fraction of items in the recommendation list that are relevant, R@k is the fraction of relevant items that are recommended, MRR@k is the average reciprocal position of the first relevant item in the recommendation list, MAP@k penalizes relevant items that occur at higher positions in the recommendation list, and nDCG@k also - via the principle of cumulative gain - penalizes relevant items based on their position.

In case of **popularity bias**, we resort to two widely-used metrics: Item Space Coverage Cov@k (Silveira et al., 2019) and Average Recommendation Popularity RecPop@k. Cov@k is the fraction of all items that is recommended at least once and RecPop@k is the average popularity of items in the recommendation list, where the item popularity is the number of interactions for this item.

In the following, we present the evaluation results of the TRUSTS recommender system. Specifically, we evaluate three recommendation algorithms for six different recommendation use cases with respect to recommendation accuracy and popularity bias (see Figure 18).

For all recommendation use cases, CF (Collaborative Filtering) generates more accurate recommendations than MP (Most Popular) and CB (Content-based Filtering). Across all use cases, CF can

generate the most accurate recommendations for RUC4 (Services to Datasets), and the recommendations for RUC6 (Services to Services) are the least accurate. Our OpenML dataset comprises many more services than datasets. In case of RUC4, this small dataset-to-service ratio shows that there exists a large item catalog (i.e., 1,307 services), that CF can recommend to few target entities (i.e., 573 datasets). As related work shows, this dataset-to-service ratio can positively affect recommendation accuracy (Adomavicius and Zhang, 2012).

Recommendation Use Case	Method	P@1	R@10	MRR@10	MAP@10	nDCG@10	Cov@10	RecPop@10
RUC1 (Datasets to Users)	MP	0.00	0.22	0.04	0.04	0.08	0.01	593.79
	CF	**0.26**	**0.34**	**0.26**	**0.27**	**0.30**	0.06	181.50
	CB	0.05	0.05	0.03	0.02	0.04	**0.12**	**10.25**
RUC2 (Services to Users)	MP	0.03	0.11	0.05	0.05	0.07	0.00	265.75
	CF	**0.12**	**0.26**	**0.14**	**0.14**	**0.18**	0.02	90.51
	CB	0.02	0.06	0.02	0.03	0.03	**0.03**	**9.25**
RUC3 (Datasets to Services)	MP	0.00	0.12	0.02	0.02	0.04	0.01	555.20
	CF	**0.33**	**0.39**	**0.28**	**0.32**	**0.35**	0.06	143.36
	CB	0.00	0.13	0.06	0.06	0.09	**0.14**	**7.07**
RUC4 (Services to Datasets)	MP	0.01	0.29	0.12	0.13	0.18	0.00	270.62
	CF	**0.52**	**0.56**	**0.42**	**0.45**	**0.51**	0.01	97.56
	CB	0.01	0.03	0.01	0.01	0.02	**0.03**	**12.75**
RUC5 (Datasets to Datasets)	MP	0.00	0.02	0.01	0.01	0.01	0.00	650.23
	CF	**0.17**	**0.44**	**0.17**	**0.20**	**0.28**	0.09	55.74
	CB	0.05	0.12	0.05	0.06	0.08	**0.28**	**14.88**
RUC6 (Services to Services)	MP	0.01	0.02	0.01	0.01	0.01	0.00	278.32
	CF	**0.07**	**0.24**	**0.08**	**0.09**	**0.14**	0.02	55.01
	CB	0.04	0.12	0.04	0.05	0.07	**0.04**	**7.87**

Figure 18: Quantitative results of our experiments with respect to accuracy and popularity bias.

For RUC2 (Services to Users), we have the same item catalog consisting of 1,307 services, however, the recommendation accuracy is higher than for RUC4 (Services to Datasets). We investigate this in more detail and find that 50% of users have more than six interactions, while only 28% of datasets have more than six interactions. Due to large profile size for users (in RUC2), generating recommendations for users seems to be more difficult than generating recommendations for datasets. For RUC6 (Services to Services), we again have an item catalog with 1,307 services that needs to be recommended to the very same large set of services. Due to this sparse interaction space, all our three recommendation methods struggle to generate meaningful recommendations. However, the high accuracy for RUC5 (Datasets to Datasets) shows that our approach of constructing the ground truth data is well suited for use cases where an item type is recommended to the same item type.

Besides accuracy, we also evaluate for popularity bias and find that MP can only recommend a small fraction of the item catalog as Cov@10 indicates. This makes sense, since MP always recommends the 10 most popular items which the target entity has not rated yet. This way, target entities cannot properly explore the data and service sharing platform. In contrast, CB recommends the largest fraction of the item catalog and thus, allows exploring large parts of the data and service sharing platform. Plus, CB generates the least popularity-biased recommendations as RecPop@10 is smaller than for MP and CF. This is due to the fact that CB can also recommend services or datasets without interactions, which is the case for many datasets and services. We investigate popularity bias in more detail in Figure 19 and

visualize the fraction of recommendations for popular items (i.e., the 10 most popular items a target entity has not rated yet). Across all use cases, MP provides the recommendations with the most popular items, whereas CB provides the least popularity-biased recommendations. Interestingly, CF can recommend both, popular and non-popular items and this way, provides more popularity-balanced recommendations.

Overall, CB tends to recommend non-popular items and MP recommends only the most popular items to target entities, whereas CF can recommend popular and non-popular. Our findings that CF provides accurate and popularity-balanced recommendations is in line with related research that shows that accurate recommendations should include nonpopular items in addition to popular items (Abdollahpouri et al., 2019 and Kowald et al., 2020).

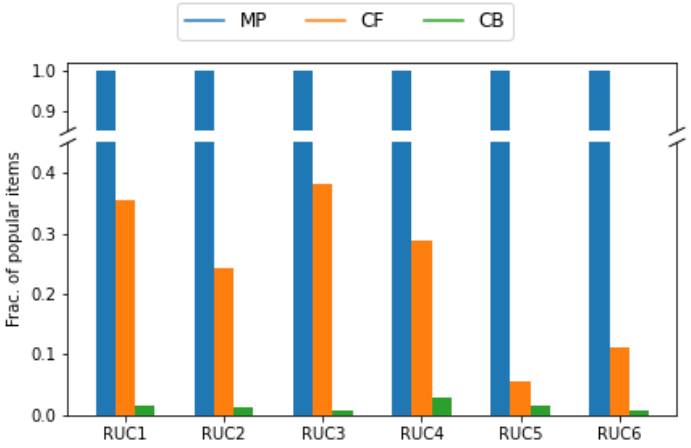


Figure 19: Fraction of recommendations for popular items.

6 Research Outputs of T3.6.

Modern recommender systems collect and process vast amounts of users' personal data. In most cases, this data includes a user's preferences for, e.g., movie genres (Hu et al., 2014; Cantador et al., 2011; Harper and Konstan, 2015; Guo et al., 2014) or jokes (Goldberg et al., 2001). With that, recommender systems pose several severe threats to users' privacy, as Friedmann et al. outline (2015). In particular, users have to share their personal data with the recommender system, which then serves as the basis for the generation of recommendations. This by itself could be already regarded as a breach of a user's privacy, since another party (i.e., the recommender system) has access to the user's personal data. Furthermore, this data could be also used to infer sensitive attributes about the user, e.g., gender or ethnicity. Therefore, it remains an important challenge to serve users with accurate recommendations while protecting their privacy.

Since one goal of the TRUSTS project is to secure the privacy of personal data, i.e., TRUSTS challenge C6 *"Advance the state-of-the-art with respect to scalability, computational efficiency of methods to secure desired levels of privacy of personal data and/or confidentiality of commercial data"*, we conduct research in the area of privacy-preserving recommender systems as part of T3.6. We identify three strands of research, aiming to develop privacy-preserving recommender systems: (i) Homomorphic Encryption (Gentry, 2009), (ii) Differential Privacy (Dwork and Roth, 2014), and (iii) Federated Learning

(McMahan et al., 2017). In Federated Learning, no data ever leaves the users' devices. As such, users do not send their data to the recommender system. Instead, the users share their data with a local copy of the recommender system model on their own device and then send model parameters to the recommender system.

Lin et al. (2020) introduce the MetaMF recommender system. Here, Federated Learning protects users' privacy, while Meta Learning (Ha et al., 2016) increases the degree of personalization and thus, improves accuracy of recommendations. However, according to Nasr et al. (2019), sharing only model parameters in Federated Learning still leaks private data. Intuitively, there can be no data disclosure if there is no data. In this vein, we acknowledge that users may have different inclinations of revealing their data to the recommender system and identify in Muellner et al. (2021) the minimal amount of rating data, users have to share with MetaMF in order to receive accurate recommendations. This work was published and presented at ECIR'2021:

- Muellner, P., Kowald, D., & Lex, E. (2021). Robustness of Meta Matrix Factorization Against Strict Privacy Constraints. In Proceedings of the 43rd European Conference on Information Retrieval (ECIR'2021). Springer.

Additionally, we also researched the popularity bias apparent in recommender systems in two papers published and presented at ECIR'2022:

- Kowald, D., & Lacic, E. (2022). Popularity Bias in Collaborative Filtering-Based Multimedia Recommender Systems. In Proceedings of the BIAS Workshop co-located with the 44th European Conference on Information Retrieval (ECIR'2022). Springer
- Lacic, E., Fadljevic, L., Weissenboeck, F., Lindstaedt, S., & Kowald, D. (2022). What Drives Readership? An Online Study on User Interface Types and Popularity Bias Mitigation in News Article Recommendations. In Proceedings of the 44th European Conference on Information Retrieval (ECIR'2022). Springer.

The recommender systems framework ScaR, which was used as the underlying technology for the recommender system developed in TRUSTS, was also published and presented in the industry track of ECIR'2022:

- Lacic, E., & Kowald, D. (2022). Recommendations in a Multi-Domain Setting: Adapting for Customization, Scalability and Real-Time Performance. In Industry-Day Track of European Conference on Information Retrieval (ECIR'2022)

The offline evaluation of the TRUSTS recommender system was published and presented at the ACM SIGCOMM DataEconomy workshop:

- Muellner, P., Schmerda, S., Theiler, D., Lindstaedt, S., & Kowald, D. (2022). Towards Employing Recommender Systems for Supporting Data and Algorithm Sharing. In Proceedings of the DataEconomy Workshop co-located with the 18th International Conference on emerging Networking EXperiments and Technologies (CoNext'2022). ACM

Finally, our research efforts in TRUSTS have led to three datasets that we published via Zenodo:

- A dataset for studying privacy aspects of recommender systems: https://zenodo.org/record/4031011#.YO_3Q0xCRPZ
- Another dataset for studying popularity bias in recommender systems: https://zenodo.org/record/6123879#.Yx7dQ7TP1PZ

- And a dataset for evaluating recommender systems for data sharing platforms: https://zenodo.org/record/6517031#.Yx7dfLTP1PY

7 Conclusion and Future Outlook

In this demonstrator deliverable, we have described how we provide brokerage in the TRUSTS portal by realizing a recommender system for interlinking users with datasets and services. Thus, the focus of this deliverable has been a technical one, and therefore we provided a detailed description of the TRUSTS recommender system's software architecture, its data scheme as well as its service integration into the platform.

Apart from this technical focus, we have also provided research-related information that we see as important for understanding the functionality of the recommender system. This includes a short overview of recommender systems in data markets as well as our conducted offline evaluation of the recommender system. The scientific work in TRUSTS has led to five papers presented at renowned conferences and workshops in the field. Additionally, we have published three datasets that we made freely available to the scientific community to foster further research on recommender systems for data and service sharing platforms.

The work done in T3.6 on recommender systems for dataset and service sharing platforms will be beneficial for future projects in the field of data markets and data economies. Our research has shown (i) how to implement recommender systems in such a setting, and (ii) how to evaluate recommender systems with respect to different trade-offs (e.g., between accuracy and popularity bias). However, future work also needs to take the monetization factors into account, which we have neglected so far.

Additionally, our research on privacy aspects in recommender systems addresses a very timely topic in light of the GDPR. Here, we have shown that machine learning techniques such as meta learning can help to reduce the personal data needed to generate accurate recommendations. Here, future work should focus on combining privacy-preserving technologies such as differential privacy with recommender systems to add privacy guarantees and to increase the trust in recommender systems.

Finally, our evaluation of the TRUSTS recommender system was conducted solely in an offline-based manner. This enabled us to test the applicability of the recommender algorithms before integrating them into the TRUSTS platform. Also, through the offline evaluation, we did not risk any negative effects the recommendations can have on the users and companies while testing different recommendation algorithms and settings. However, offline evaluation results do not necessarily need to correlate with online evaluation results. Thus, our future work will also focus on evaluating the user acceptance of the recommendations in the running TRUSTS platform after project end. We plan to publish these online evaluation results in additional scientific publications after project end.

8 References

Abdollahpouri, H., Mansoury, M., Burke, R., & Mobasher, B. (2019). The unfairness of popularity bias in recommendation. In *Workshop on Recommendation in Multi-stakeholder Environments (RMSE 2019), in conjunction with RecSys 2019*.

Achananuparp, P., Hu, X., & Shen, X. (2008, September). The evaluation of sentence similarity measures. In *International Conference on data warehousing and knowledge discovery* (pp. 305-316). Springer, Berlin, Heidelberg.

Adomavicius, G., & Zhang, J. (2012). Impact of data characteristics on recommender systems performance. *ACM Transactions on Management Information Systems (TMIS)*, 3(1), 1-17.

Bahls, D., Scherp, G., Tochtermann, K., & Hasselbring, W. (2012). Towards a Recommender System for Statistical Research Data. In *Proceedings of the 2nd International Workshop in Semantic Digital Archives*, (pp. 61-72).

Cantador, I., Brusilovsky, P., & Kuflik, T. (2011, October). Second workshop on information heterogeneity and fusion in recommender systems (HetRec2011). In *Proceedings of the fifth ACM conference on Recommender systems*, (pp. 387-388).

Chatzopoulou, G., Erinaki, M., & Polyzotis, N. Query Recommendations for Interactive Database Exploration. In Winslett M. (ed.). *Scientific and Statistical Database Managment. SSDBM 2009. Lecture Notes in Computer Science*, vol. 5566, (pp. 3-18), Springer, Berlin, Heidelberg.

Chen, X., Gururaj, A.E., Ozyurt, B., Liu, R., Soysal, E., Cohen, T., Tiryaki, F., Li, Y., Zong, N., Jiang, M., Rogith, D., Salimi, M., Kim, H., Rocca-Serra, P., Gonzalez-Beltran, A., Farcas, C., Johnson, T., Margolis, R., Alter, G., Sansone, S., Fore, I.M., Ohno-Machado, L., Grethe, J.S., Xu, H. (2018). DataMed – an open source discovery index for finding biomedical datasets. In *Journal of the American Medical Informatics Association*, vol. 25, no. 3, (pp. 300–308).

Dwork, C., & Roth, A. (2014). The algorithmic foundations of differential privacy. In *Foundations and Trends in Theoretical Computer Science*, 9(3-4), (pp. 211-407).

Erinaki, M., Abraham, S., & Polyzotis, N. (2014). QueRIE: Collaborative Database Exploration. In *IEEE Transactions on Knowledge and Data Engineering*, vol. 26, no. 7, (pp. 1778-1790).

Erinaki, M., Patel, S., (2015). QueRIE reloaded: Using Matrix Factorization to Improve Database Query Recommendations. In *IEEE International Conference on Big Data*, (pp. 1500-1508).

Fernandez, R. C., Subramaniam, P., & Franklin, M. J. (2020). Data market platforms: Trading data assets to solve data problems. *Proc. VLDB Endow.*, (pp. 1933–1947).

Friedman, A., Knijnenburg, B. P., Vanhecke, K., Martens, L., & Berkovsky, S. (2015). Privacy aspects of recommender systems. *In Recommender systems handbook* (pp. 649-688). Springer, Boston, MA.

Gentry, C. (2009, May). Fully homomorphic encryption using ideal lattices. In *Proceedings of the forty-first annual ACM symposium on Theory of computing*, (pp. 169-178).

Goldberg, K., Roeder, T., Gupta, D., & Perkins, C. (2001). Eigentaste: A constant time collaborative filtering algorithm. *information retrieval*, 4(2), 133-151.

Guo, G., Zhang, J., Thalmann, D., & Yorke-Smith, N. (2014, August). Etaf: An extended trust antecedents framework for trust prediction. In *2014 IEEE/ACM International Conference on Advances in Social Networks Analysis and Mining (ASONAM 2014),* (pp. 540-547). IEEE.

Ha, D., Dai, A., & Le, Q. V. (2016). Hypernetworks. arXiv preprint arXiv:1609.09106.

Harper, F. M., & Konstan, J. A. (2015). The movielens datasets: History and context. *Acm transactions on interactive intelligent systems (tiis)*, 5(4), 1-19.

He, X., Zhao, K., & Chu, X. (2021). AutoML: A survey of the state-of-the-art. *Knowledge-Based Systems*, 212, 106622.

Hu, L., Sun, A., & Liu, Y. (2014, July). Your neighbors affect your ratings: on geographical neighborhood influence to rating prediction. In *Proceedings of the 37th international ACM SIGIR conference on Research & development in information retrieval*, (pp. 345-354).

Järvelin, K., & Kekäläinen, J. (2002). Cumulated gain-based evaluation of IR techniques. *ACM Transactions on Information Systems (TOIS)*, 20(4), 422-446.

Jess, T., Woodall, P., Dodwani, V., Harrison, M., McFarlane, D., Nicks, E., & Krechel, W. (2015). An Industrial Data Recommender System to Solve the Problem of Data Overload. In *ECIS 2015 Research-In-Progress Papers*. Paper 52.

Jones, K. S. (1972). A statistical interpretation of term specificity and its application in retrieval. *Journal of documentation*.

Kowald, D., Schedl, M., & Lex, E. (2020, April). The unfairness of popularity bias in music recommendation: A reproducibility study. In *European conference on information retrieval* (pp. 35-42). Springer, Cham.

Kowald, D., Traub, M., Theiler, D., Gursch, H., Lindstaedt, S., Kern, R., & Lex, E. (2019). Using the Open Meta Kaggle Dataset to Evaluate Tripartite Recommendations in Data Markets. In *REVEAL Workshop co-located with ACM Conference on Recommender Systems*.

Lacic, E., Kowald, D., Eberhard, L., Trattner, C., Parra, D., & Marinho, L. B. (2013). Utilizing online social network and location-based data to recommend products and categories in online marketplaces. In *Mining, Modeling, and Recommending'Things' in Social Media* (pp. 96-115). Springer, Cham.

Lacic, E., Traub, M., Kowald, D., & Lex, E. (2015, September). Scar: Towards a real-time recommender framework following the microservices architecture. In *Proceedings of the Workshop on Large Scale Recommender Systems (LSRS2015)* at RecSys (pp. 16-20).

Liang, F., Yu, W., An, D., Yang, Q., Fu, X., & Zhao, W. (2018). A survey on big data market: Pricing, trading and protection. *Ieee Access*, *6*, 15132-15154.

Lin, Y., Ren, P., Chen, Z., Ren, Z., Yu, D., Ma, J., Rijke, M., & Cheng, X. (2020). Meta Matrix Factorization for Federated Rating Predictions. In *Proceedings of the 43rd International ACM SIGIR Conference on Research and Development in Information Retrieval*.

McMahan, B., Moore, E., Ramage, D., Hampson, S., & y Arcas, B. A. (2017, April). Communication-efficient learning of deep networks from decentralized data. *In Artificial Intelligence and Statistics* (pp. 1273-1282). PMLR.

Muellner P., Kowald D., Lex E. (2021) Robustness of Meta Matrix Factorization Against Strict Privacy Constraints. In: Hiemstra D., Moens MF., Mothe J., Perego R., Potthast M., Sebastiani F. (eds) *Advances in Information Retrieval*. ECIR 2021. Lecture Notes in Computer Science, vol 12657. Springer, Cham.

Müllner, P., Schmerda, S., Theiler, D., Lindstaedt, S., & Kowald, D. (2022). Towards Employing Recommender Systems for Supporting Data and Algorithm Sharing*, DataEconomy Workshop at CoNEXT'22*.

Nasr, M., Shokri, R., & Houmansadr, A. (2019, May). Comprehensive privacy analysis of deep learning: Passive and active white-box inference attacks against centralized and federated learning. In *2019 IEEE symposium on security and privacy* (SP), (pp. 739-753). IEEE.

Parra, D., & Sahebi, S. (2013). Recommender systems: Sources of knowledge and evaluation metrics. In *Advanced techniques in web intelligence-2* (pp. 149-175). Springer, Berlin, Heidelberg.

Patra, B.G., Roberts, K., & Wu, H. (2020). A content-based dataset recommendation system for researchers—a case study on Gene Expression Omnibus (GEO) repository. In *Database*, Volume 2020, 2020.

Radev, D. R., Qi, H., Wu, H., & Fan, W. (2002, May). Evaluating Web-based Question Answering Systems. In *LREC*.

Ricci, F., Rokach, L., & Shapira, B. (2011). Introduction to recommender systems handbook. In *Recommender systems handbook* (pp. 1-35). Springer, Boston, MA.

Satopaa, V., Albrecht, J., Irwin, D., & Raghavan, B. (2011, June). Finding a" kneedle" in a haystack: Detecting knee points in system behavior. In *2011 31st international conference on distributed computing systems workshops* (pp. 166-171). IEEE.

Silveira, T., Zhang, M., Lin, X., Liu, Y., & Ma, S. (2019). How good your recommender system is? A survey on evaluations in recommendation. *International Journal of Machine Learning and Cybernetics*, 10(5), 813-831.

Singhal, A., Srivastava, J. (2017). Research Dataset Discovery from Research Publications Using Web Context. In *Web Intelligence*, vol. 15, no. 2, (pp. 81-99).

Song, Q., Wang, G., Wang, C. (2012). Automatic recommendation of classification algorithms based on data set characteristics. In *Pattern Recognition*, vol. 45, no. 7, (2672-2689).

Stahl, F., Schomm, F., Vossen, G., & Vomfell, L. (2016). A classification framework for data marketplaces. *Vietnam Journal of Computer Science*, 3(3), 137-143.

Vainshtein, R., Greenstein-Messica, A., Katz, G., Shapira, B., & Rokach, L. (2018). A Hybrid Approach for Automatic Model Recommendation. In *Proceedings of the 27th ACM International Conference on Information and Knowledge Management*, (pp. 1623-1626).

Zschech, P., Heinich, K., Horn, R., & Höschele, D. (2019). Towards a Text-based Recommender System for Data Mining Method Selection. In *25th American Conference on Information Systems (AMCIS*).