D3.12 Profiles and Brokerage

Authors: **Dominik Kowald, Dieter Theiler, Peter Müllner, Stefan Schmerda (KNOW)**

Additional Information: **-**

June 2021

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

D3.12 Profiles and Brokerage

• 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	D3.12 Profiles and Brokerage		
Work Package	WP3		
Contractual due date	30/06/2021	Actual submission date	30/06/2021
Nature	Demonstrator	Dissemination Level	Public
Lead Beneficiary	KNOW		
Responsible Author	Dominik Kowald		
Contributions from	Dieter Theiler, Peter Müllner, Stefan Schmerda (KNOW)		

○ Revision history (including peer reviewing & quality control)

Version	Issue Date	% Complete[1]	Changes	Contributor(s)
v1.0	21/04/ 2021	5	Deliverable Structure	Dominik Kowald (KNOW)
V1.1	26/04/ 2021	10	Description of functional and architectural requirements	Dominik Kowald (KNOW)
V1.2	07/05/ 2021	30	First draft of System Architecture, Data Scheme and Service Interface	Stefan Schmerda (KNOW)
V1.3	10/05/ 2021	35	Offline Evaluation Plan and Research	Peter Müllner (KNOW)
V1.4	17/05/ 2021	55	Introduction to Recommender Systems in Data Markets	Stefan Schmerda (KNOW)
V1.5	21/05/ 2021	80	Recommender Systems in Data Markets and Platforms, Introduction and Conclusion	Stefan Schmerda, Dominik Kowald (KNOW)
V1.6	26/05/ 2021	85%	System Architecture, Data Scheme and Service Interface	Dieter Theiler, Peter Müllner, Dominik Kowald (KNOW)
V2.0	23/06/ 2021	100%	Incorporated reviewer feedback	Dieter Theiler, Peter Müllner, Stefan Schmerda, 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
CBF	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 current state of the TRUSTS recommender system that is developed for these purposes in course of T3.6.

Based on the functional requirements identified in D2.2, we have defined architectural requirements as well as six recommendation use cases for the TRUSTS recommender system. This includes (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 have designed a scalable recommendation architecture that is capable of supporting these use cases. Apart from that, our proposed recommendation architecture 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 also propose an offline evaluation plan using data gathered from the OpenML platform. OpenML is a machine learning platform that allows users to share datasets and services and thus, provides an ideal data source for offline recommendation experiments. Finally, we present initial results of a privacy-aware recommendation experiment, in which we aim to provide quality recommendations with a limited amount of private user data.

Taken together, the purpose of this deliverable is to demonstrate the capabilities of the TRUSTS recommender system and also to document the underlying technical specifications of its service interfaces. This is complemented by the presentation of an offline evaluation plan as well as initial research results in the area of privacy-aware recommender systems.

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 good 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. 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 particular 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 form of a recommender system is needed.

The development and evaluation of this recommender system is the main objective of T3.6. Specifically, TRUSTS requires that (i) *the system should be able to provide datasets and services recommendations to its users pertaining to their profile and needs*, (ii) *the system should employ matchmaking mechanisms through which hosted datasets are matched with hosted services (e.g., suitable for their analysis) and vice versa*, and (iii) *the system should identify and match related datasets so as to provide combined and enriched data.* We transferred these functional requirements to recommendation use cases, which we further transferred 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. Apart from that, we also worked on an offline evaluation plan using data gathered from the machine learning platform OpenML and conducted research in the area of privacy-aware recommendations.

Finally, 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.

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. 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.	Section 4.1. Section 4.2. Section 4.3. Section 4.4.	The sections describe the whole design process of the TRUSTS recommender system, starting with the requirements and use cases, over the system architecture and service interfaces, to the evaluation plan and research results.
TRUSTS Deliverable			
D3.12: Profiles and Brokerage I This deliverable constitutes demonstrator systems that show practical application of the developed algorithms to production data. It identifies suitable recommendation use cases and applicable algorithms and datasets to support them, as well as a proof-of concept demonstrator.			

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 data platforms, including the Data Market Austria recommender systems. 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 also describes the system architecture, data scheme and service interface of our TRUSTS recommender system. Finally, Chapter 4 also includes an offline evaluation plan using data gathered from the OpenML platform as well as initial research results from the area of privacy-aware recommender systems. We close this deliverable in Chapter 5 with a summary of our findings and contributions as well as an outlook into our future work.

3 Recommender Systems in Data Markets and Platforms

The contribution of T3.6. to TRUSTS is a recommender system with the purpose to function as the required brokering instance. Users will be presented with datasets and services based on their preferences derived from past interactions on the platform.

The approach of utilizing recommender systems for data markets is rather novel and does not have a broad corpus of related work. Notwithstanding this fact, the following paragraphs try to classify the topic at hand and discuss related fields.

The literature explicitly concerned with dataset recommendation is scarce, when compared to the ones about movie, music or e-commerce. Lately, however, the interest in it is growing. This stems from the fact that the number of available datasets, publicly available over the internet or stored in private databases, has increased rapidly over the last decade. The overload of information therefore makes it mandatory for firms and researchers to apply filtering methods, hence the turn to recommender systems for this particular use case.

Jess et al. (2015) proposed a recommender system for the industrial domain (i.e., supply chain-, financial- or human resources- data) and subsequently evaluated it using artificial data. In this context industrial decision makers should be provided with similar data tables based on the one they are currently working on in order to generate additional value. Three different engines (user-based collaborative filtering, item-based collaborative filtering, and content-based filtering) are applied and their results aggregated to form the final set of recommendations.

Patra et al. (2020) utilized a content-based approach to suggest datasets from the genetics domain. Here the similarity between aggregated metadata of different interest clusters, derived from a researcher's publication profile, and the metadata of relevant datasets is calculated to provide recommendations.

Traditional algorithms like collaborative and contend-based filtering often fail in situations where there is a lack of initial usage data from which recommendations can be derived. This is called *the cold-start problem* in the literature and is especially prevalent for dataset recommenders with a relatively small user base. This circumstance is tackled in (Bahls et al., 2012) where the authors outline and propose a knowledge-based approach called "case-based reasoning". Here certain similarity measures reflecting the user's understanding of utility instead of explicit historical usage data are applied to generate recommendations.

A more implicit approach to the dataset recommendation literature is the one about database query recommendations. Here instead of explicitly suggesting datasets, users of databases are provided with queries which can be utilized to extract relevant information. Both (Erinaki et al., 2014) and (Chatzopoulou et al., 2009) apply collaborative filtering to achieve this while (Erinaki and Patel, 2015) uses matrix factorization.

A distantly related field to recommender systems is the one about search engines. Chen et al. (2018) developed a dataset discovery and retrieval system comprising repositories of the biomedical domain. The system incorporates functionalities for automatic dataset metadata extraction and indexing as well as a natural language processing enhanced search engine based on query expansion. In (Singhal and Srivastava, 2017) a non domain specific search engine for research datasets considering their application context was proposed. Here user profiles containing a set of research papers and keywords linked to information from academic search engines are being utilized.

While the literature about dataset recommenders is steadily growing there is less work regarding recommendation systems for machine learning algorithms and data analytics services. This is mainly due to the fact that algorithms per se, when user/item interactions are unavailable, are not suitable for comparison via similarity-based metrics. Notwithstanding this, there is the related literature of automated machine learning, which aims to automatically find the best data mining pipelines, ranging from pre-processing to modelling, for a given dataset or domain. The approach of (Zschech et al., 2019), for example outlines a text based assistant system for data analysis tasks. It takes a domain specific problem description and a general description of the required methods (e.g., classification or clustering), both in natural language form, as inputs and returns a recommended data mining algorithm and the most suitable data scheme for it. In (Vainshtein et al., 2018) the metadata contained in dataset descriptions, statistics derived from the dataset itself and word embeddings, representing a corpus of academic literature, are used to select the most suitable classification algorithm for a given dataset. In (Song et al., 2012) a recommender was proposed on the assumption that similar datasets are also similar in their respective classification algorithm performances. Therefore, feature vectors containing structural and statistical information as well as the best performing classification algorithms (based on accuracy, etc.) of a set of historical datasets were extracted. In a next step KNN is applied to a new dataset to find the most similar ones from the original pool. Algorithm recommendation is then conducted based on the best performing algorithms of the candidate sets.

An example of a particular data market, which incorporates a recommender system for both datasets and services is the Data Market Austria[2] (DMA). In DMA, a recommender system was developed, which acts as a broker to connect different stakeholders in a data market setting such as service providers, data providers and users as depicted in Figure 1. This recommender system was based on the scalable recommendation framework ScaR[3]. Specifically, it was the aim to connect users, datasets and services using two interaction-based recommendation algorithms, i.e., Most Popular (MP) and Collaborative Filtering (CF). The recommendation use cases were evaluated on the Meta Kaggle dataset[4]. The authors found that the recommendation quality strongly depends on the complexity of the recommendation task. Thus, in cases where non-user entities are recommended to other non-user entities (e.g., datasets to services) the interaction-based approaches MP and CF did not provide accurate recommendations.

[2] https://datamarket.at/en/
[3] http://scar.know-center.tugraz.at/
[4] https://www.kaggle.com/kaggle/meta-kaggle

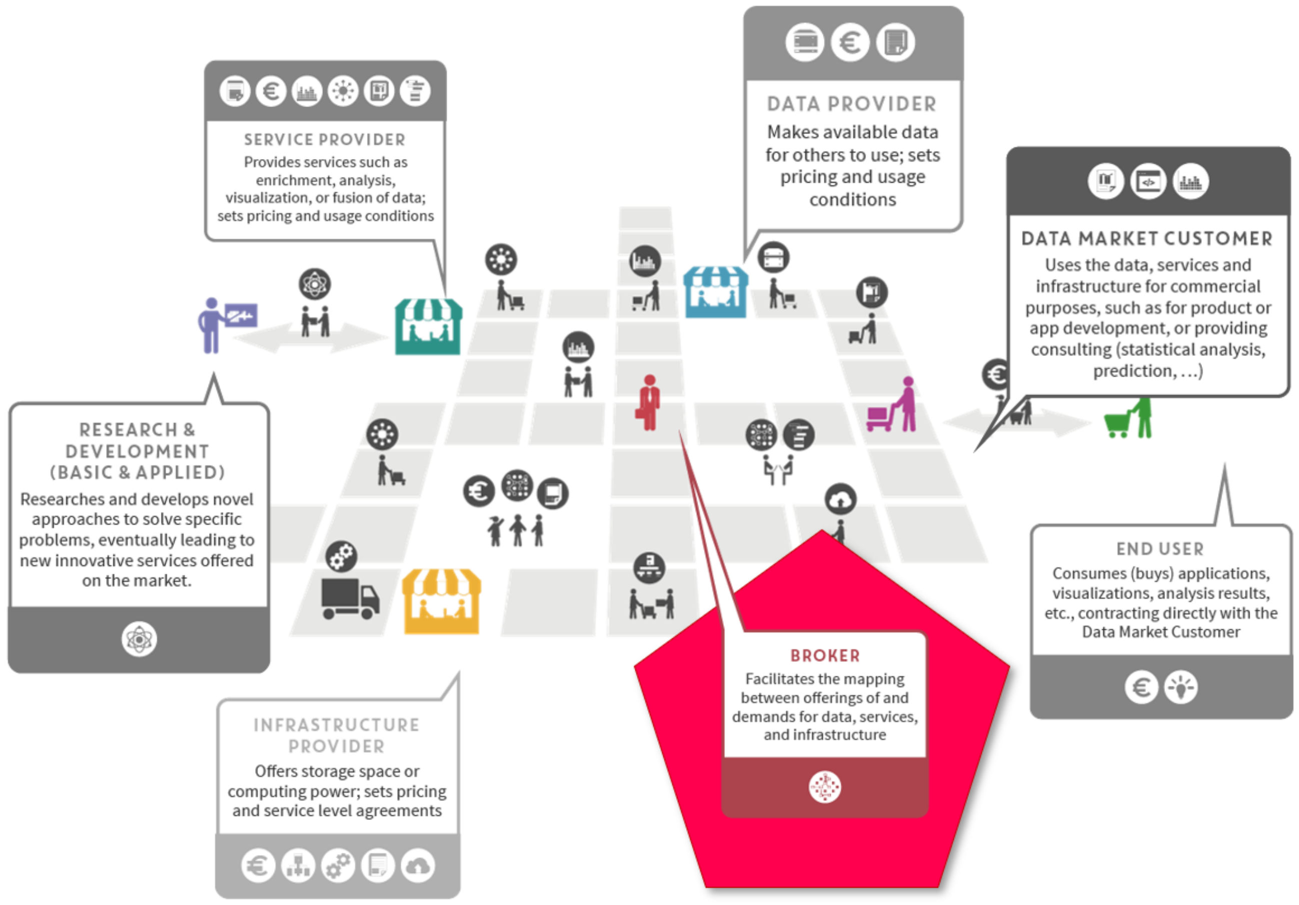

SERVICE PROVIDER
Provides services such as enrichment, analysis, visualization, or fusion of data; sets pricing and usage conditions
DATA PROVIDER
Makes available data for others to use; sets pricing and usage conditions
DATA MARKET CUSTOMER
Uses the data, services and infrastructure for commercial purposes, such as for product or app development, or providing consulting (statistical analysis, prediction, …)
RESEARCH & DEVELOPMENT (BASIC & APPLIED)
Researches and develops novel approaches to solve specific problems, eventually leading to new innovative services offered on the market.
END USER
Consumes (buys) applications, visualizations, analysis results, etc., contracting directly with the Data Market Customer
BROKER
Facilitates the mapping between offerings of and demands for data, services, and infrastructure
INFRASTRUCTURE PROVIDER
Offers storage space or computing power; sets pricing and service level agreements

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

In TRUSTS, we will build upon this work and will also use the ScaR recommendation framework, which proved its useability in the DMA project. We also plan to validate the results obtained in DMA using a dataset gathered from OpenML (see Section **Fehler! Verweisquelle konnte nicht gefunden werden.**). Additionally, we will develop recommendation use cases that connect two non-user entities of the same type with each other (e.g., datasets to datasets) and we will also investigate content-based filtering recommendation algorithms (see Section 4.1).

4 TRUSTS Recommender System

In this section, we describe the recommender system envisioned 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 evaluation plan.

4.1 Functional Requirements, Recommendation Use Cases and Architectural Requirements

The functional requirements of the TRUSTS recommender system have been developed in course of WP2 and are documented in detail in D2.2. In Table 2, we summarize these functional requirements for our recommender system.

Table 2: Functional requirements of the TRUSTS recommender system.

Recommender system: 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

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.

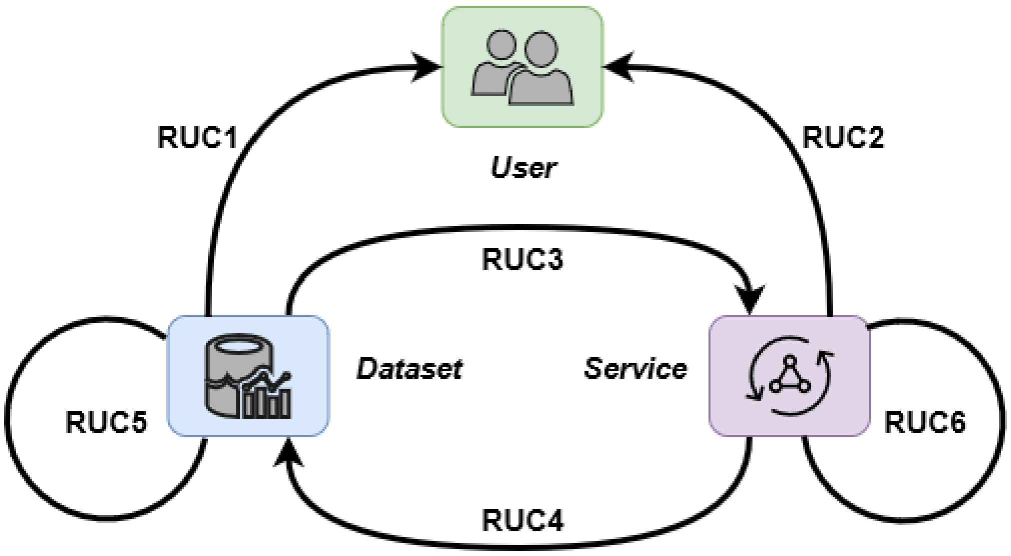


Figure 2: Recommendation use cases in TRUSTS.

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 settings: (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 (e.g., 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 two types of algorithms (e.g., collaborative filtering and content-based filtering).

In the current architectural vision of the TRUSTS platform, as described in deliverable D2.6, 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 will serve 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. Its general infrastructure and 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 use cases described in Table 3 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 [5]. This particular database 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. Herein Apache Solr's built-in data structures are being leveraged in order to allow for efficient similarity calculation. The RE supports standard approaches like collaborative and contend-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.

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

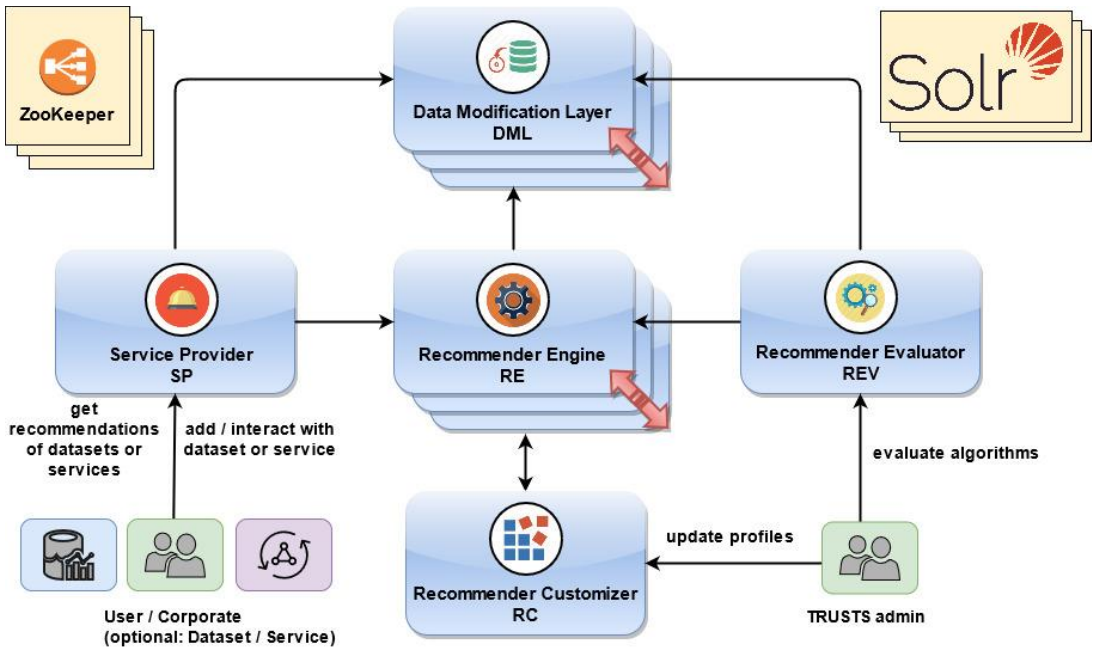


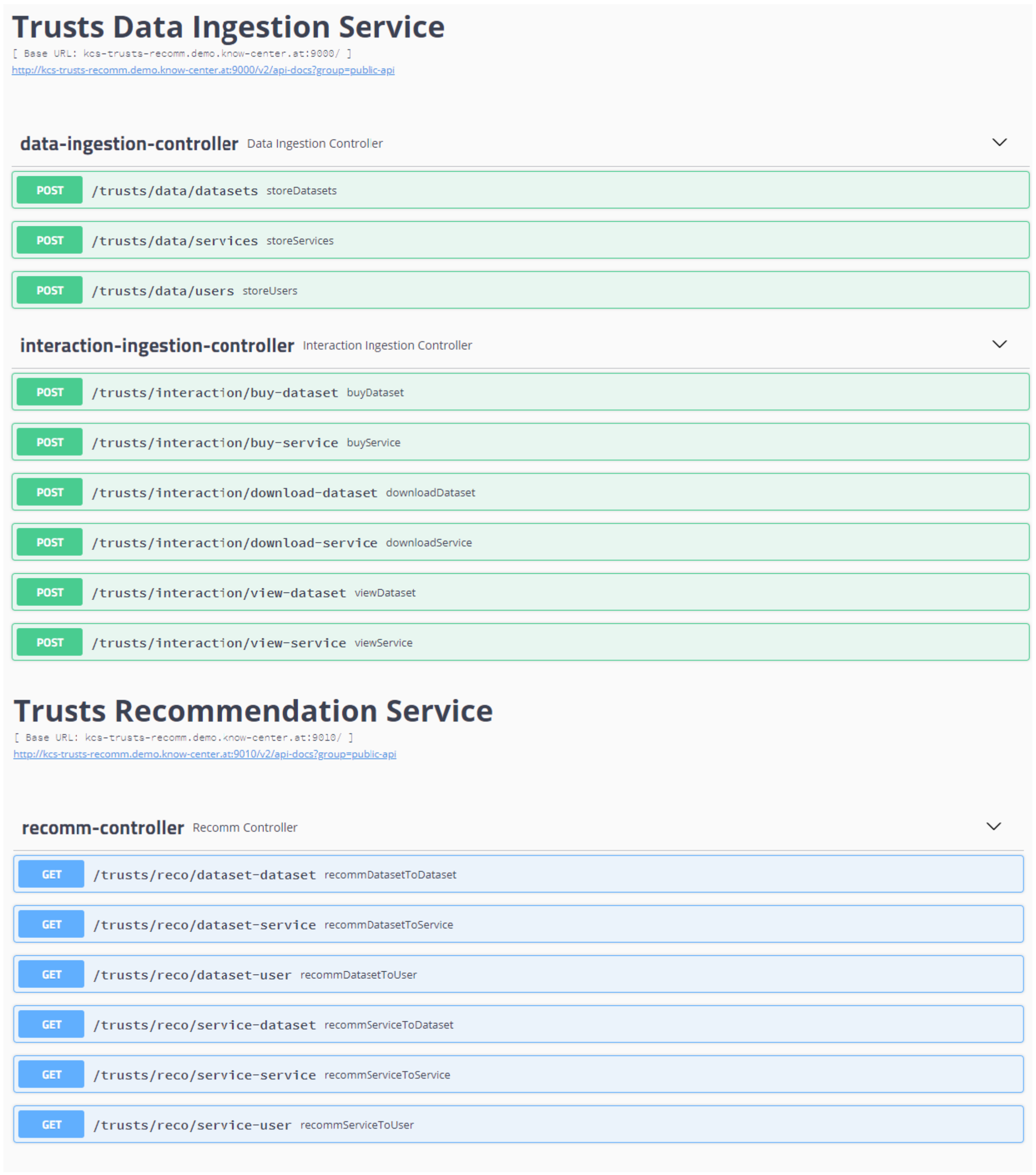
Figure 3: System architecture of TRUSTS recommender system.

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 depicts the Swagger user interface for the aforementioned REST-Resources including their respective endpoints which will be described in more detail in the following subsections.



Trusts Data Ingestion Service

[Base URL: kcs-trusts-recomm.demo.know-center.at:9000/]

http://kcs-trusts-recomm.demo.know-center.at:9000/v2/api-docs?group=public-api

data-ingestion-controller Data Ingestion Controller

POST /trusts/data/datasets storeDatasets

POST /trusts/data/services storeServices

POST /trusts/data/users storeUsers

interaction-ingestion-controller Interaction Ingestion Controller

POST /trusts/interaction/buy-dataset buyDataset

POST /trusts/interaction/buy-service buyService

POST /trusts/interaction/download-dataset downloadDataset

POST /trusts/interaction/download-service downloadService

POST /trusts/interaction/view-dataset viewDataset

POST /trusts/interaction/view-service viewService

Trusts Recommendation Service

[Base URL: kcs-trusts-recomm.demo.know-center.at:9010/]

http://kcs-trusts-recomm.demo.know-center.at:9010/v2/api-docs?group=public-api

recomm-controller Recomm Controller

GET /trusts/reco/dataset-dataset recommDatasetToDataset

GET /trusts/reco/dataset-service recommDatasetToService

GET /trusts/reco/dataset-user recommDatasetToUser

GET /trusts/reco/service-dataset recommServiceToDataset

GET /trusts/reco/service-service recommServiceToService

GET /trusts/reco/service-user recommServiceToUser

Figure 4: Overview of the available REST-Services

4.2.2.1 Data Resource

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

Input Objects: As the IDS-IM (International Data Spaces – Information Model) is designed to foster a central agreement between different services which share and apply data assets, the input to the Data

Resource must conform to the specifications of the IDS-IM as defined in the IDS Ontology[6] draft. To adapt to the needs of the TRUSTS project and to ensure compatibility and interoperability, the IDS-IM will be further refined in the upcoming deliverable D3.7. For data ingestion, objects are based on certain IDS-IM classes:

- `Resource`
- `App Resource`
- `Participant`

Response Object: The `DataResponse` object for the data resource is generic and has the properties depicted in Table 4:

Table 4: `DataResponse` Object

Response Property	Data Type	Description
http_status	Integer	The HTTP response status code.
message	String	A textual message indicating the success or failure of the initial call.

In the following the individual requests to the Data Resource are described:

/data/datasets

A call to this endpoint stores an array of dataset metadata objects in the database of the recommendation service. Please note that this call will update the metadata of the dataset objects if they already exist in the database.

Type: *POST* Request

Input: An array of IDS-IM `Resource` objects

Response: A `DataResponse` object

/data/services

A call to this endpoint stores an array of service metadata objects in the database of the recommendation service. Please note that this call will update the metadata of the services objects if they already exist in the database.

Type: *POST* Request

Input: An array of IDS-IM `App Resource` objects

Response: A `DataResponse` object

/data/users

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

A call to this endpoint stores an array of user metadata objects in the database of the recommendation service. Please note that this call will update the metadata of the user's objects if they already exist in the database.

Type: *POST* Request

Input: An array of IDS-IM `Participant` objects

Response: A `DataResponse` object

4.2.2.2 Interaction Resource

This service is used for uploading buy-, view- and download-interactions of respective items, which can be datasets or services.

Input Object: The input object consists of properties which apply to all endpoints of the interaction resource as well as an identifier property which varies with the actual interaction item type. Therefore, the `Basic Entity` object as described in Table 5, constitutes the basis for different interaction types and is extended with either a specific `Dataset Identifier` or a `Service Identifier` to form the final payload object:

- `Dataset Interaction` ← `Basic Entity` ∪ `Dataset Identifier`
- `Service Interaction` ← `Basic Entity` ∪ `Service Identifier`

Table 5, Table 6, Table 7 show the object properties in detail:

Table 5: Interaction Resource - `Basic Entity`

Body-Parameter	Data Type	Required / Optional	Description
id	String	Required	The id of the interaction.
recommender Id	String	Optional	The id of the recommendation which resulted in the current interaction.
timestamp	Long	Required	The timestamp indicates at which point in time a certain interaction was made. It is based on the Unix format (milliseconds since 1970-01-01T00:00:00).
userId	String	Required	The id of the user who is responsible for the interaction.

Table 6: `Dataset Identifier`

Body-Parameter	Data Type	Required / Optional	Description
datasetId	String	Required	The id of dataset the interaction is based on.

Table 7: `Service Identifier`

Body-Parameter	Data Type	Required / Optional	Description
serviceId	String	Required	The id of the service the interaction is based on.

Response Object: The `InteractionResponse` object for the interaction resource is generic and has the properties depicted in Table 8:

Table 8: `InteractionResponse` Object

Response Property	Data Type	Description
http_status	Integer	The HTTP response status code.
message	String	A textual message indicating the success or failure of the initial call.

In the following the individual requests to the Interaction Resource are described:

/interaction/buy-dataset

A call to this endpoint stores the metadata of a dataset buy-interaction in the database of the recommendation service.

Type: *POST* Request

Input: A `Dataset Interaction` object

Response: An `InteractionResponse` object

/interaction/buy-service

A call to this endpoint stores the metadata of a service buy-interaction in the database of the recommendation service.

Type: *POST* Request

Input: A `Service Interaction` object

Response: An `InteractionResponse` object

/interaction/download-dataset

A call to this endpoint stores the metadata of a dataset download-interaction in the database of the recommendation service.

Type: *POST* Request

Input: A `Dataset Interaction` object

Response: An `InteractionResponse` object

/interaction/download-service

A call to this endpoint stores the metadata of a service download-interaction in the database of the recommendation service.

Type: *POST* Request

Input: A `Service Interaction` object

Response: An `InteractionResponse` object

/interaction/view-dataset

A call to this endpoint stores the metadata of a dataset view-interaction in the database of the recommendation service.

Type: *POST* Request

Input: A `Dataset Interaction` object

Response: An `InteractionResponse` object

/interaction/view-service

A call to this endpoint stores the metadata of a service view-interaction in the database of the recommendation service.

Type: *POST* Request

Input: A `Service Interaction` object

Response: An `InteractionResponse` object

4.2.2.3 Recommender Resource

This service is used for requesting recommendations for one of the six recommendation use cases.

Input Parameters: The Recommender Resource takes query string parameters as input. A subset of those is independent of the endpoint while others vary with it. The variable parameters are listed individually in the respective endpoint references and need to be combined with the general applicable ones (`GenRecoParameter`) depicted in Table 9:

Table 9: Recommender Resource – `GenRecoParameter`

Query String Parameter	Data Type	Required / Optional	Description
count	Integer	Optional	The number of recommendations which should be generated. The default value is 10.
userId	String	Required	The id of the user who requests recommendations.

Response Object: Every request to the Recommendation Resource returns a `RecommendationResponse` object containing the properties depicted in Table 10:

Table 10: `RecommendationResponse` Object

Response Property	Data Type	Description
http_status	Integer	The HTTP response status code.
message	String	A textual message indicating the success or failure of the call.
reco_id	String	The id of the returned recommendation.
[results]	Array of Strings	An array of ids identifying the recommended items.

In the following the individual requests to the Recommender Resource are being described:

/reco/dataset-dataset

A call to this endpoint returns dataset recommendations based on a certain dataset and a user.

Type: `Get` Request

Input: The general `GenRecoParameter` set combined with the endpoint specific `DatasetDatasetRecoParameter` object depicted in Table 11:

Table 11: `DatasetDatasetRecoParameter` Object

Query String Parameter	Data Type	Required / Optional	Description
datasetId	String	Required	The id of the dataset on which the recommendation is based on.

Response: A `RecommendationResponse` object

/reco/dataset-service

A call to this endpoint returns dataset recommendations based on a certain service and a user.

Type: *`Get`* Request

Input: The general `GenRecoParameter` set combined with the endpoint specific `DatasetServiceRecoParameter` object depicted in Table 12:

Table 12: `DatasetServiceRecoParameter` Object

Query String Parameter	Data Type	Required / Optional	Description
serviceId	String	Required	The id of the service on which the recommendation is based on.

Response: A `RecommendationResponse` object

/reco/dataset-user

A call to this endpoint returns dataset recommendations based on a certain user and optionally a dataset.

Type: *`Get`* Request

Input: The general `GenRecoParameter` set combined with the endpoint specific `DatasetUserRecoParameter` object depicted in Table 13:

Table 13: `DatasetUserRecoParameter` Object

Query String	Data Type	Required / Optional	Description

Parameter			
datasetId	String	Optional	The id of the dataset on which, in addition to the user, the recommendation can be based on.

Response: A `RecommendationResponse` object

/reco/service-dataset

A call to this endpoint returns service recommendations based on a certain dataset and a user.

Type: *`Get`* Request

Input: The general `GenRecoParameter` set combined with the endpoint specific `ServiceDatasetRecoParameter` object depicted in Table 14:

Table 14: `ServiceDatasetRecoParameter` Object

Query String Parameter	Data Type	Required / Optional	Description
datasetId	String	Required	The id of the dataset on which the recommendation is based on.

Response: A `RecommendationResponse` object

/reco/service-service

A call to this endpoint returns service recommendations based on a certain service and a user.

Type: *`Get`* Request

Input: The general `GenRecoParameter` set combined with the endpoint specific `ServiceServiceRecoParameter` object depicted in Table 15:

Table 15: `ServiceServiceRecoParameter` Object

Query String Parameter	Data Type	Required / Optional	Description
serviceId	String	Required	The id of the service on which the

			recommendation is based on.

Response: A `RecommendationResponse` object

/reco/service-user

A call to this endpoint returns service recommendations based on a certain user and optionally a service.

Type: *Get* Request

Input: The general `GenRecoParameter` set combined with the endpoint specific `ServiceUserRecoParameter` object depicted in Table 16:

Table 16: `ServiceUserRecoParameter` Object

Query String Parameter	Data Type	Required / Optional	Description
serviced	String	Optional	The id of the service on which, in addition to the user, the recommendation can be based on.

Response: A `RecommendationResponse` object

4.2.3 Backend Database – Apache Solr

The current database utilized by the ScaR framework is Apache Solr[7]. In principial, 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 database 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 different infrastructure at a level of data storage, also other database engines could be applied, e.g., ElasticSearch[8]. 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 depicted in Table 17:

Table 17: Solr Cores

Core Name	Core Description

[7] https://solr.apache.org
[8] https://www.elastic.co/elasticsearch

items	This core contains the following items: • Datasets • Services
users	This core contains the entities using TRUSTS services: • Users
interactions	This core contains interactions of users with datasets and services.
feedbacks	This core contains the calculated recommendations as well as information regarding the evaluation of the system.

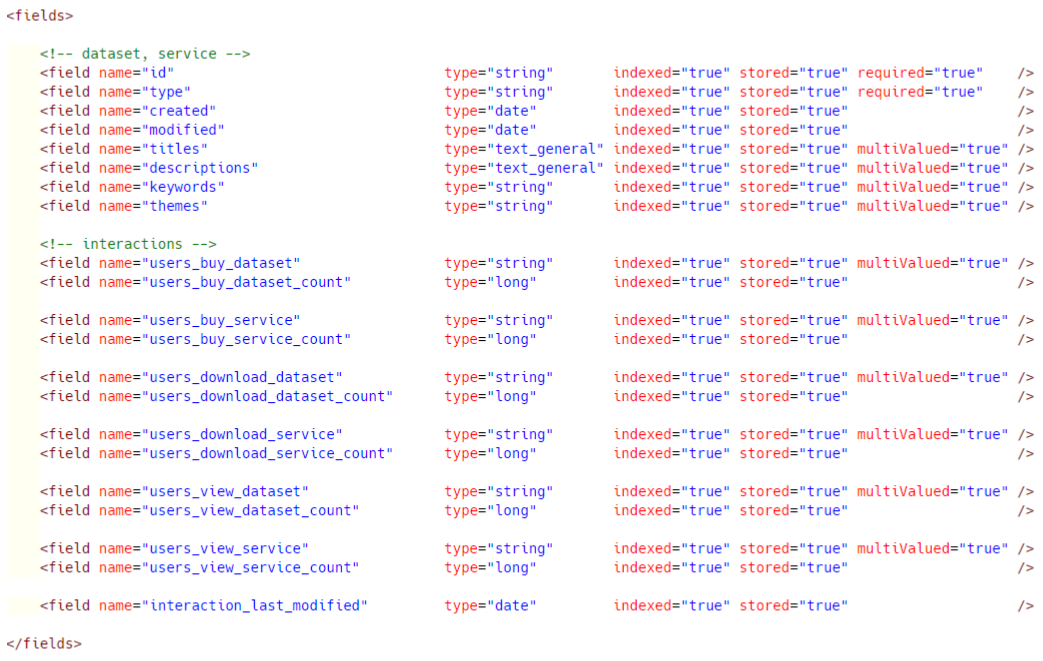
The following subsections elaborate on the structure of the particular data objects stored in those cores:

Items Core

The Items Core contains dataset- and service-data objects, which 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.

The fields of the items core objects with corresponding data types are listed in Figure 5:



```
<fields>

    <!-- dataset, service -->
    <field name="id"                              type="string"       indexed="true" stored="true" required="true"    />
    <field name="type"                            type="string"       indexed="true" stored="true" required="true"    />
    <field name="created"                         type="date"         indexed="true" stored="true"                    />
    <field name="modified"                        type="date"         indexed="true" stored="true"                    />
    <field name="titles"                          type="text_general" indexed="true" stored="true" multiValued="true" />
    <field name="descriptions"                    type="text_general" indexed="true" stored="true" multiValued="true" />
    <field name="keywords"                        type="string"       indexed="true" stored="true" multiValued="true" />
    <field name="themes"                          type="string"       indexed="true" stored="true" multiValued="true" />

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

    <field name="users_buy_service"               type="string"       indexed="true" stored="true" multiValued="true" />
    <field name="users_buy_service_count"         type="long"         indexed="true" stored="true"                    />

    <field name="users_download_dataset"          type="string"       indexed="true" stored="true" multiValued="true" />
    <field name="users_download_dataset_count"    type="long"         indexed="true" stored="true"                    />

    <field name="users_download_service"          type="string"       indexed="true" stored="true" multiValued="true" />
    <field name="users_download_service_count"    type="long"         indexed="true" stored="true"                    />

    <field name="users_view_dataset"              type="string"       indexed="true" stored="true" multiValued="true" />
    <field name="users_view_dataset_count"        type="long"         indexed="true" stored="true"                    />

    <field name="users_view_service"              type="string"       indexed="true" stored="true" multiValued="true" />
    <field name="users_view_service_count"        type="long"         indexed="true" stored="true"                    />

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

</fields>
```

Figure 5: Items Core

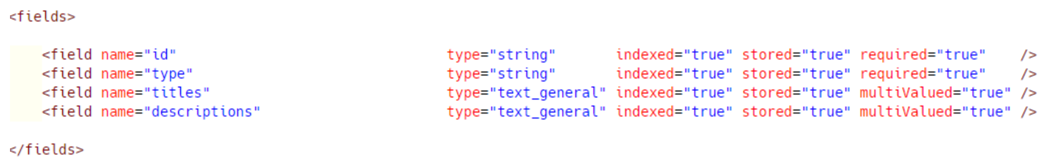
Table 18 contains a description of the individual fields belonging to the objects in the Items Core:

Table 18: Items Core – Field Description

Fieldname	Description
id	The id of the item.
type	The type of the stored item. Can either be dataset or service.
created	Datetime indicating when the item was created.
modified	Datetime indicating the last modification of the item.
titles	An array of names given to the item.
descriptions	An array of natural language-based descriptions of the item.
keywords	An array of tags given to the item.
themes	Standardized item description.
users_buy_dataset	An array of unique ids pointing to the users who bought this dataset.
users_buy_dataset_count	Total buy count of the dataset.
users_buy_service	An array of unique ids pointing to the users who bought this service.
users_buy_service_count	Total buy count of the service.
users_download_dataset	An array of unique ids pointing to the users who downloaded this dataset.
users_download_dataset_count	Total download count of the dataset.
users_download_service	An array of unique ids pointing to the users who downloaded this service.
users_download_service_count	Total download count of the service.
users_view_dataset	An array of unique ids pointing to the users who viewed this dataset.
users_view_dataset_count	Total view count of the dataset.
users_view_service	An array of unique ids pointing to the users who viewed this service.
users_view_service_count	Total view count of the service.
interaction_last_modified	Datetime indicating the last modification of one of the item's interaction fields.

Users Core

The Users Core contains user data objects. The fields with corresponding data types and meta-information are listed in Figure 6.



```
<fields>

    <field name="id"                         type="string"       indexed="true" stored="true" required="true"    />
    <field name="type"                       type="string"       indexed="true" stored="true" required="true"    />
    <field name="titles"                     type="text_general" indexed="true" stored="true" multiValued="true" />
    <field name="descriptions"               type="text_general" indexed="true" stored="true" multiValued="true" />

</fields>
```

Figure 6: Users Core

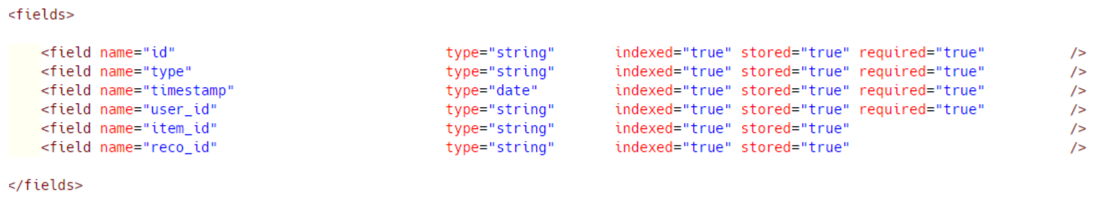
Table 19 contains a description of the individual fields belonging to the objects in the Users Core:

Table 19: Users Core – Field Description

Fieldname	Description
id	The id of the user.
type	The type of the stored user. Can either be user or corporation.
titles	An array of names given to the user.
descriptions	An array of natural language-based descriptions of the user.

Interactions Core

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



```
<fields>

    <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="item_id"                    type="string"       indexed="true" stored="true"                          />
    <field name="reco_id"                    type="string"       indexed="true" stored="true"                          />

</fields>
```

Figure 7: Interactions Core

Table 20 contains a description of the individual fields belonging to the objects in the Interactions Core:

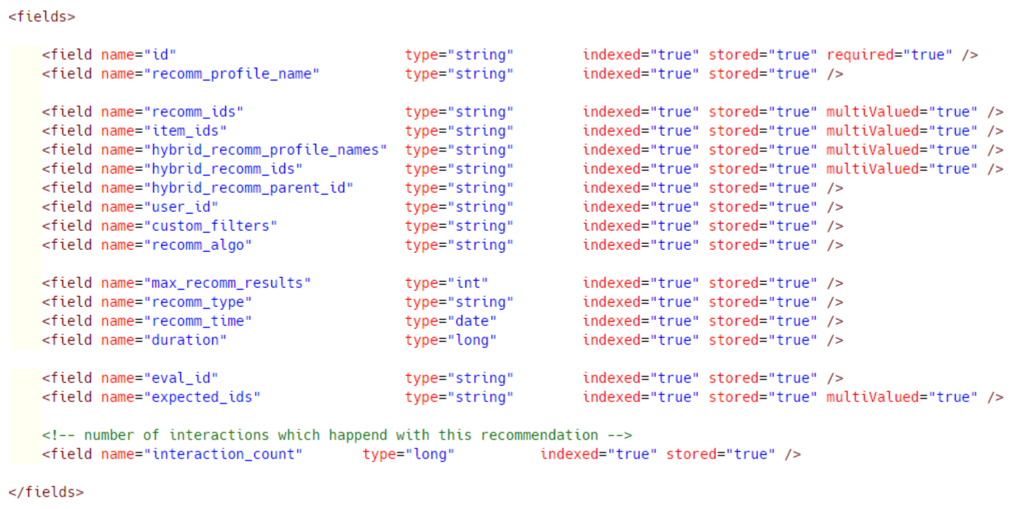
Table 20: Interactions Core – Field Description

Fieldname	Description
id	The id of the interaction.
type	The type of the stored interaction. Can either be buy, view or download.
timestamp	Datetime indicating when the interaction

	happened.
user_id	The id of the user responsible for the interaction.
item_id	The id of the item the user interacted with.
reco_id	The id of the recommendation resulting in the interaction.

Feedbacks Core

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



```
<fields>

    <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" />

</fields>
```

Figure 8: Feedbacks Core

Table 21 contains a description of the individual fields belonging to the objects in the Feedbacks Core:

Table 21: Feedbacks Core – Field Description

Fieldname	Description
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_* [9]	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	Datetime 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.

4.2.4 Recommendation Algorithms

We will develop three types of recommendation algorithms to realize our six recommendation use cases:

- **Most Popular (MP)**: This is an un-personalized 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 on items to find similar users, and then recommends items of these similar users.
- **Content-based Filtering (CBF)**: This also is a personalized algorithm that calculates item-similarities based on content features (e.g., title, description text) and then recommends these similar items.

In the next section, we describe the offline evaluation plan for these recommendation algorithms using OpenML data.

[9] The Feedbacks Core contains a set of parameters starting with the suffix '*hybrid_recomm_*'.

4.3 Offline Evaluation using OpenML Data

In order to evaluate the accuracy of the recommender system developed in TRUSTS in an offline evaluation setting, we will use data gathered from the open-source machine learning platform OpenML[10]. We have chosen OpenML since it provides an easy-to-use Python-based API to query data. Additionally, it contains all four types of data entities that we need to train our recommender system: (i) users, (ii) datasets (i.e., tasks in OpenML terms), (iii) services/algorithms (i.e., flows in OpenML terms), and (iv) interactions between these entities.

In OpenML users can upload so called runs. In general, a run indicates that a user *u* applied a flow (i.e., an algorithm *a* with a certain parameter setting) on a certain task. Moreover, the 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*. Via these runs, we can derive several interaction datasets. For example, users interacted with algorithms, but also with tasks. Plus, there is also an interaction between (i) datasets and algorithms and (ii) datasets and users, since users run algorithms on tasks and as such, datasets. With this, OpenML is a natural match to our TRUSTS recommender (cf. Figure 2), in which we generate recommendations between users, services, and datasets. Here, we underline that we require an algorithm, as included in a flow, as a service, which serves users the results of a task.

Retrieval

In order to demonstrate our TRUSTS recommender on data from OpenML, we built a dataset comprising of interactions between users, datasets, and algorithms. Therefore, we utilize the Python API[11] of OpenML. First, we retrieve all *openml.runs.OpenMLRun* objects, i.e., runs, from the OpenML platform by using the *openml.runs.list_runs* method. With these steps, we retrieve our raw dataset D_{raw}, in which each entry represents an interaction between a user, a flow, and a task. In total, D_{raw} comprises 10,013,752 interactions between 579 users, 5,794 flows, and 20,539 tasks.

Preprocessing

Each entry in our raw dataset D_{raw} *only* represents an interaction between a user, a flow, and a task. The reason is that an *openml.runs.OpenMLRun* object does not directly include the dataset a user interacted with. To also obtain interactions between users and datasets, we leverage the *openml.tasks.get_task* method to query the *openml.tasks.OpenMLTask* object for the specific *task_id* as illustrated in Figure 9. With that, we can link a dataset with id *did* to a user, i.e., *uploader,* and as such, extract interactions between users and datasets. Furthermore, it is important to note that a user could run the same algorithm on the same dataset many times. Also, a user could use different parameter settings of the algorithm in different runs. Both actions result in different flows. With that, multiple flows could represent the same algorithm. This poses a problem, since our raw dataset D_{raw} could include multiple flows for the same algorithm. Since our TRUSTS recommender only considers algorithms, i.e., services, and not flows, we merge flows with the same algorithm, which leads to our base dataset *D. D* only comprises unique (user, algorithm, dataset)-triples. Furthermore, *D* includes 8,637,795 interactions between 544 users, 2,186 datasets and 5,660 algorithms.

[10] https://www.openml.org/
[11] https://docs.openml.org/Python-API/

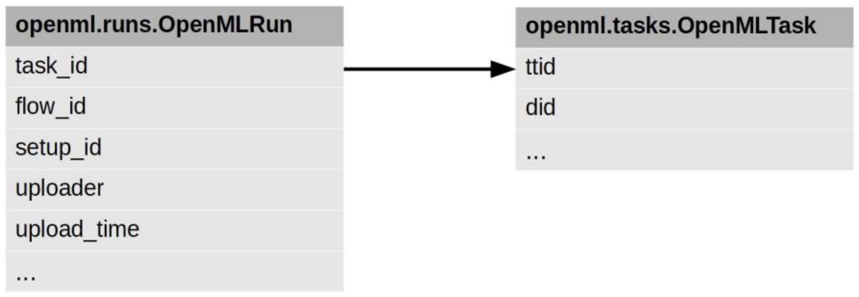


Figure 9: The connection between runs and datasets.

Based on *D*, we derive three datasets, i.e., D_{UA}, D_{UD}, and D_{DA}, which are applicable to different use-cases. D_{UA} contains interactions between users and algorithms, D_{UD} represents interactions between users and datasets and D_{DA} comprises interaction data for datasets and algorithms. It is important to note that since *D* comprises unique (user, algorithm, dataset)-triples, e.g., D_{UA} would include multiple entries per, e.g., user-algorithm pair. Thus, in our derived datasets, we remove all duplicates such that there are only unique pairwise interactions between users, algorithms, and datasets. Statistics of all three datasets can be found in Table 22, Table 23, and Table 24. We will use these datasets to generate train and test splits in order to evaluate the accuracy of our recommender system in the various use cases. We will report on the specific evaluation setting in D3.13.

Table 22: Statistics of our D_{UD} dataset, which comprises interactions between users and datasets.

Nr. of users	544
Nr. of datasets	2,186
Nr. of interactions	9,199
Avg. nr. of interactions per user	16.91

Table 23: Statistics of our D_{UA} dataset, which comprises interactions between users and algorithms.

Nr. of users	544
Nr. of algorithms	5,660
Nr. of interactions	7,986
Avg. nr. of interactions per user	14.13

Table 24: Statistics of our D_{DA} dataset, which comprises interactions between datasets and algorithms.

Nr. of datasets	2,186
Nr. of algorithms	5,660

Nr. of interactions	172,888
Avg. nr. of interactions per dataset	79.09
Avg. nr. of interactions per algorithm	30.55

4.4 Research: Privacy-Preserving Recommendations

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. 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 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. In this work, we refer to the fraction of data a user shares with MetaMF as the user's privacy budget $\beta \in [0; 1]$. Users with large privacy budgets (i.e., $\beta \approx 1$) share lots of their data with the recommender system. Thus, they are willing to give up their privacy in exchange for accurate recommendations. In contrast, users with small privacy budgets (i.e., $\beta \approx 0$) only share a small fraction of their data with the recommender system. Thus, they protect their privacy at the cost of rather poor recommendations. To quantify how the accuracy of user recommendations changes, if users employ a privacy budget β, we measure the recommendation accuracy under β relative to the recommendation accuracy of $\beta=1$. In detail, we introduce our $\Delta MAE@\beta$ measurement which is the Mean absolute error of MetaMF's recommendations if users share only a fraction of β of their data divided by the Mean absolute error of MetaMF's recommendations if users share all of their data.

We ran experiments on five datasets, i.e., Douban (Hu et al., 2014), Hetrec-MovieLens (Cantador et al., 2011), MovieLens 1M (Harper and Konstan, 2015), Ciao (Guo et al., 2014), and Jester (Goldberg et al., 2001) and illustrate ΔMAE@β for decreasing privacy budgets in Figure 10. Interestingly, for a privacy budget of β ≥ 0.5, no substantial changes of the recommendation accuracy can be observed. Only for small privacy budgets of β < 0.5, ΔMAE@β increases. That means that the recommendation accuracy stays high as long as users share more than 50% of their data with MetaMF (i.e., β ≥ 0.5). Recommendation accuracy only drops if users share less than 50% of their data (i.e., β < 0.5). In this sense, by limiting the amount of data users share with MetaMF, privacy could be increased with no substantial loss of accuracy.

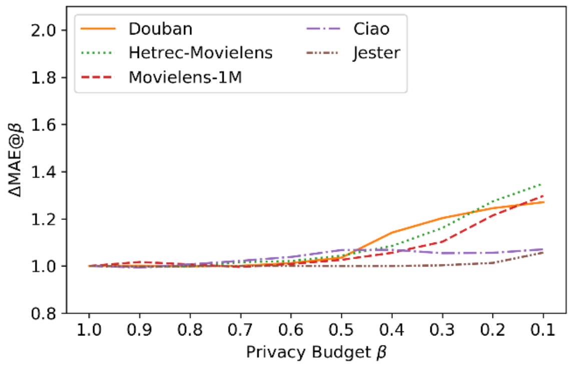


Figure 10: MetaMF

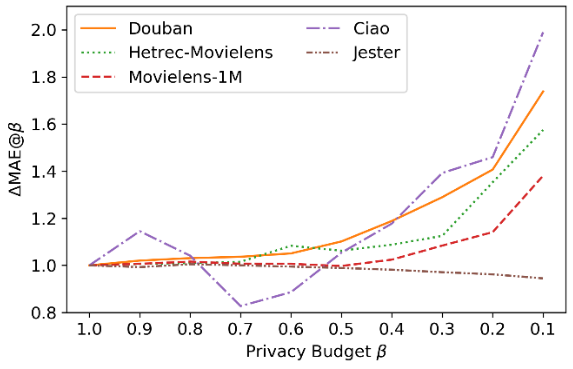


Figure 11: NoMetaMF

Since Meta Learning has been shown to improve model performance in other applications, e.g., Few-Shot Learning (Sung et al., 2018) (Snell et al. 2017) in which a model has to be learnt on only a small amount of data, we pinpoint the impact Meta Learning has on MetaMF's performance under small privacy budgets. Therefore, we develop a variant of MetaMF, i.e., NoMetaMF, in which meta learning is disabled. Experiments for NoMetaMF can be observed in Figure 11. Similar to MetaMF, NoMetaMF performs well for privacy budgets β ≥ 0.5, but shows a substantial decrease in recommendation accuracy for β < 0.5. In this vein, we point out the side-by-side comparison of MetaMF and NoMetaMF in Figures 10 and 11. Here, it is apparent that for small privacy budgets, NoMetaMF's recommendations are far worse than MetaMF's. Thus, if users share less than 50% of their rating data with MetaMF (i.e., β < 0.5), Meta Learning helps to keep high recommendation accuracy.

In conclusion, we observed in our study that for a state-of-the-art recommender system, only ≈ 50% of a user's data is required to generate accurate recommendations. Furthermore, we provided evidence that Meta Learning is beneficial in keeping recommendation accuracy high if users share less than 50% of their data with the recommender system. These findings translate well into TRUSTS, as they show that (i) users have the possibility to reveal only small amounts of personal data to a service and (ii) services may require not all personal data of a user. By limiting the personal data users reveal to services, we hope to make users less identifiable for the service or other parties with malicious intent.

5 Conclusions and Next Actions

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 interface. This should provide other technical partners in the consortium with information on how to use and integrate our recommender system.

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 an evaluation plan and initial research results for integrating privacy aspects into recommendations. In our research, we have shown that Meta Learning is beneficial in terms of keeping the accuracy of recommendations high, even when users only share a small fraction of their data.

Our plans for future work are three-fold: first, and more on the technical side, we aim to fully integrate our recommender system into the IDS-based infrastructure of TRUSTS. This will also include the development of a service that enables the deletion of datasets, services, users and interactions from the database of the recommender system. Second, and according to our presented offline evaluation plan, we will use the gathered OpenML data to evaluate our recommender system, which will enable us to fine-tune the algorithms. Third, we will further research on privacy aspects of recommender system. With this, we also hope to enhance the algorithms implemented in our recommender system with respect to the fundamental privacy-accuracy trade-off. We will report on all three aspects as well as on the final version of our recommender systems in D3.13.

6 References

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).

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.

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).

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.

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*.

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.

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.

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.

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).

Sung, F., Yang, Y., Zhang, L., Xiang, T., Torr, P. H., & Hospedales, T. M. (2018). Learning to compare: Relation network for few-shot learning. *In Proceedings of the IEEE conference on computer vision and pattern recognition* (pp. 1199-1208).

Snell, J., Swersky, K., & Zemel, R. S. (2017). Prototypical networks for few-shot learning. arXiv preprint arXiv:1703.05175.

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*).