Chapter 1
Graph Based Recommendations: From Data Representation to Feature Extraction and Application

Amit Tiroshi, Tsvi Kuflik, Shlomo Berkovsky, Mohamed Ali (Dali) Kaafar

Modeling users for the purpose of identifying their preferences and then personalizing services on the basis of these models is a complex task, primarily due to the need to take into consideration various explicit and implicit signals, missing or uncertain information, contextual aspects, and more. In this study, a novel generic approach for uncovering latent preference patterns from user data is proposed and evaluated. The approach relies on representing the data using graphs, and then systematically extracting graph-based features and using them to enrich the original user models. The extracted features encapsulate complex relationships between users, items, and metadata. The enhanced user models can then serve as an input to any recommendation algorithm. The proposed approach is domain-independent (demonstrated on data from movies, music, and business systems), and is evaluated using several state-of-the-art machine learning methods, on different recommendation tasks, and using different evaluation metrics. Overall, the results show a unanimous improvement in the recommendation accuracy across tasks and domains.

1.1 Introduction

Recommender systems aim at helping users find relevant items among a large variety of possibilities, based on their preferences [1]. In many cases, these personal preferences are inferred from patterns that emerge from data about the users' past interactions with the system and with other users, as well as additional personal characteristics available from different sources. These patterns are typically user-specific and are based on the metadata of both the users and items, as well as on the interpretation of the observed user interactions [39, 87]. Eliciting user preferences is a challenging task because of issues such as changes in user preferences, contextual dependencies, privacy constraints, and practical data collection difficulties [66, 11]. Moreover, the collected data may be incomplete, outdated, imprecise, or even completely inapplicable to the recommendation task at hand. In order to address these issues, modern recommender systems attempt to capture as much data as possible and elicit from this data the desired preferences.

Regardless of the technique exploited by a recommender system, it is inherently bound by the available user data and the features extracted/elicited from it. One major question that arises in this context is *how to engineer*[1] *meaningful features from often noisy user data?* Features may be manually engineered by domain experts. This approach is considered expensive and non-scalable because of the deep domain knowledge that is necessary, the creativity required to conceive new features, and the time needed to populate and evaluate the contribution of the features. A notable example of this challenge is provided by the Netflix Prize winning team, in their recap: "while major breakthroughs in the competition were achieved by uncovering new features underlying the data, those became rare and very hard to get" [42].

An alternative to manual feature engineering is automatic feature engineering, which is a major area of research in machine learning [31, 57, 25]. So far, automatic feature engineering has mainly focused on either algebraic combinations of existing features, e.g., summation or averaging of existing features [54] or elicitation of latent features [38, 83]. The algebraic approaches for automatic feature engineering manage to produce large quantities of features; however, the relationships between the engineered features and the underlying patterns in the data are often not interpretable [44]. Similarly, the latent feature techniques do not provide sufficient insight regarding the meaning of the features [43].

In this work, a novel framework is proposed that uses graph-based representation properties to generate additional features from recommender systems data. The proposed framework is underpinned by the idea of examining the data from the graph theory-based perspective, which represents entities and their relationships as a graph and allows the extraction of a suite of new features computed using established graph-based techniques. The extracted features encapsulate information about the relationships between entities in the graph and lead to new patterns uncovered in the data. In most cases, they are also interpretable; for example, a node's degree represents the importance of the node in the graph, while the path length between two nodes communicates their relatedness.

The proposed framework offers several benefits for automatic feature extraction. Given a new dataset, it is usually impossible to determine a-priori which graph representations will yield the most informative set of features for the recommendation generation. Thus, the proposed framework provides a systematic method for generating and assessing various graph representations, their contribution to the newly extracted features, and, in turn, to the accuracy of the generated recommendations. Additionally, since the number of nodes and relationship types in each graph representation is different, an exhaustive method of distilling the possible graph metrics from each representation is proposed.

Two large-scale case studies are conducted to gather extensive empirical evidence and demonstrate how graph features supplement existing feature sets, improve the accuracy of the recommendations, and perform adequately as stand-alone out-of-the-box features. The case studies answer the following questions:

[1]Feature engineering is also referred to in the literature as feature extraction, generation, and discovery.

- How does the use of graph features affect the performance of rating predictions and recommendation generation in different domains and tasks?
- How are the recommendations affected by the sub-graphs and their representations used to generate the graph features?

Multiple datasets, machine learning mechanisms, and evaluation metrics are used across the case studies, in order to demonstrate the effectiveness of the approach. Overall, the results show that graph-based representation and feature extraction allow for the generation of more accurate recommendations. A comparison across various graph schemes is conducted and the justification for systematic feature extraction is established. Hence, this work concludes the line of research presented in [75, 76, 77] and provides a complete picture that validates the applicability of the graph-based feature generation approach to recommender systems.

The rest of the chapter is structured as follows. Next, the necessary background is provided, and related work is described. Then, the graph representation and graph-based feature extraction process is formalized, and its advantages and disadvantages are discussed. Two case studies demonstrating the contribution of the graph-based features to the recommendation process are then presented. Through these, the overall performance of the framework, as well as the performance of certain graph representations and feature subsets, is evaluated. Finally, the implications of the findings are discussed, together with the suggested future work.

1.2 Background and Related Work

Graphs have been exploited in recommender system for many tasks, mainly due to their ability to represent many entities of different types and their relationships in a simple data structure that offers a broad variety of metrics and reasoning techniques. In this section we provide a general background on the use of graphs in recommender systems, followed by specific aspects of graph representation in recommender systems and feature engineering.

1.2.1 Graph-Based Recommender Systems

Since social networks were identified as a major source for freely available personal information, graphs and networks data structures have been used as tools for user modeling, especially since they combine different entities and links into one simple structure capturing the links between the entities. This section aims at giving the readers an idea about how graph techniques are used in graph-based user modeling and recommender systems. Given the vast amount of prior work, this is only a brief presentation of recent studies and not an in-depth survey.

What was clearly noticeable was that most of the graph-based representations were defined for a specific problem, in specific domains, and in many cases they applied variants of random walk as the only graph feature used for recommendations. [64] suggested to use a simple graph representation for recommending groups to users, tags to groups, and events to users, using a general graph-based model called HeteRS, while considering the recommendation problem as a query dependent node proximity problem. [84] suggested the use of a heterogeneous graph for representing contextual aspects in addition to items and users, and used random walk for context-aware recommendation. [48] used an enhanced version of personalised PageRank algorithm to recommend items to target users and proposed to reduce the size of the graph by clustering nodes and edges. [71] also applied personalised PageRank over the user/item graph augmented with pairwise ranking for items recommendation.

In addition to the wide use of random walk based algorithms, there is a variety of task-specific representations and metrics. It is interesting to note that even for a specific task, a variety of approaches was suggested. For instance, [59] suggested to use tags and sound description represented as a knowledge graph, from which similarity of nodes was extracted using a specific metric they defined. In contrast, [53] suggested using graph representation for music tracks recommendations, where they represented by graphs the relative preferences of users, e.g., pair-wise preference of tracks. They used the graph as a representation for user preferences for tracks and calculated the probability of a user liking a track based on the probability that s/he likes the in-linked tracks.

Some works suggested to use graph representations as an alternative to the classical collaborative and hybrid recommenders. [56] used clustering of graph representation of users and items for generating a model for item- and user-based collaborative filtering. [3] used graphs for representing co-occurrence of mobile apps, as logged from users mobile devices, and the similarity of user graphs was used for finding a neighborhood and generation recommendations. [20] also addressed the app recommendation problem and explored the potential of graph representation for several variants of recommendation strategies for recommending apps. [61] proposed a graph representation for linking items based on their similarity, where users were linked to items they rated, such that items most similar to the items rated by the users could be recommended. [47] suggested an approach for graph-based representation of the user-item matrix, where links among items represent positive ratings, and use entropy to find the items to recommend to users, thus, introducing serendipity into the recommendation process.

A highly relevant line of work focuses on enriching recommender systems dataset with information extracted from graph representation of the data [79]. The authors suggested to enrich a dataset of research papers with what the so-called metapath data links extracted from citations network. They added this information to the existing set of features, then applied classical matrix factorization, and showed an improvement to the results using only the original data. Our framework can be considered as a generalized variant of [79], where a specific set of metrics was extracted from the graph representation of the data and matrix factorization was applied for recommendation generation. The studies presented in this work used a variety of metrics, datasets, and recommendation methods.

Additional applications of graphs for recommendations include cultural heritage, tourism, social networks, and more. [18] used graphs for representing context evolution in cultural heritage: nodes modeled states and transitions between the nodes were based on observation of user behavior. [73] used graphs for representing tourist attractions and their similarity, where different graphs could represent content-based, collaborative, and social relationships. [29] reviewed the use of folksonomies, which can be naturally seen as user-item-tag graphs, in recommender systems. Graph-based approaches in user modeling and recommender systems have become popular and there exists a number of tools for analysis of large graphs. We refer interested readers to [6] and [86] for encompassing reviews of the area.

1.2.1.1 Similarity Measurement Using Graphs and their Application

Previous research on graph-based recommender systems focused on measuring the similarity of two entities in the data (user-item, user-user, or item-item), and associated this with a score or rating [2]. Graph-based similarity measurement is based on metrics extracted from a graph-based representation [23]. Two key approaches for measuring similarity using graphs are based on paths and random walks.

In the path-based similarity, the distance between graph nodes can be measured using the *shortest path* and/or the total *number of paths*. The definition of the shortest path may include a combination of the number of edges connecting the two nodes and the weights of these edges. Shortest path can be computed for a user node and an item node, in order to quantify the extent to which the user prefers the item. The "number of paths" approach works similarly, by calculating the number of paths between the two nodes as a proxy for their relatedness (the more paths, the more related they are). However, this approach is more computationally intensive.

Random walks can be used to compute similarity by estimating the probability of one node being reached from another node, given the available graph paths. The more probable it is that the target node can be reached from the source node, the higher is the relatedness of the two nodes. Random walks can be either unweighted (equal probability of edges) or weighted (edges having different probabilities based on their label, e.g., rating) [23].

Examples of recommendation studies in which the approaches detailed above were applied can be found in [50, 51, 41]. [50] reduced the recommendation problem was to a link prediction problem. That is, the problem of finding whether a user would like an item was cast as a problem of finding whether a link exists between the user and item in the graph. A similarity measure between user and item nodes was computed using random walks. Items were then ranked based on their similarity scores, such that top scoring items were recommended to users. Using classification accuracy metrics, this approach was shown to be superior to other non-graph based similarity ranking methods.

A similar walking distance metric was used in [51], complemented by graph structure metrics such as the number of sub trees. These metrics were used for the purpose of link prediction and property value prediction in RDF semantic graphs, using a learning technique based on an SVM. Experimental results showed that the graph features varied in their performance based on the graph structure on which they operated, for example, full versus partial subtrees. It was also noted that the newly defined features were not dataset-specific, but could be applied to any RDF graph. The graph structures in the context of RDF are less applicable to those used in the approach proposed in this work, because the recommendation dataset graphs do not follow a hierarchical model of RDFs.

Finally, [41] developed a graph-based approach for generating recommendations in social datasets like Last.fm. The work focused on optimizing a random walk with restarts algorithm. The reported results show an improvement in recommendations using the random walk approach, compared to the baseline collaborative filtering. In the presented work, random walks on a graph, although with static parameters, are represented by the PageRank score feature. The above studies are also extended here by generalizing the adoption of graph metrics beyond random walks and their use for similarity measurements, not bound to any specific graph structure.

1.2.1.2 Representing social data and trust using graphs

Other studies involving graph approaches in recommender systems primarily addressed the context of representing social, semantic, and trust data. In some studies, only the graph representation was used as the means to query the data, e.g., neighboring nodes and the weights of edges connecting to them [52, 67], while others utilize both the graph representation and graph-based reasoning methods [55, 65].

A survey of connection-centric approaches in recommender systems [63] exemplifies how the data of an email network [69] and of a co-occurrence in Web documents [37] can be represented in graphs. The graph representation of the email interactions between users defines each user as a node and edges connect users, who corresponded via email. In the case of Web documents, people are again represented as nodes and edges connect people, who are mentioned in the same document. When these graphs are established, they can be used to answer recommendation-related queries. In the email graph, a query regarding the closeness of users can be answered using a similarity or distance metric, such as those mentioned in the previous section. In the Web co-occurrence graph, a query regarding people sharing interests can be answered by counting their common neighbors.

Other graph representation variants are hypergraphs [9]. They differ from graphs by allowing an edge, denoted by a hyperedge, to connect with multiple nodes. Hypergraphs have been proposed in the context of recommendation generation, for the purpose of representing complex associations, such as social tagging [36, 15, 74], where a tag is attached to an item by a user. If the tag, user, and item are represented by nodes, at least two edges are required to represent the association between the three entities. This association can be represented by a hyperedge connecting the three nodes. In these studies, similarity metrics are composed based on this structure and used for the recommendation generation. Results presented in [36] show that the similarity metrics from hypergraphs lead to better recommendations.

Prior works focusing on the means of incorporating trust between users for the sake of improving the recommendations were surveyed in [58]. For example, [52] proposed a graph representation encapsulating trust between users. The representation modeled users as the graph nodes and the trust relationships between them were reflected by the weights on the edges. Data extracted from the graph, e.g., who trusts whom and to what extent, was used in the recommendation process, and it was shown to improve the generated recommendations. However, the graph was used only to represent the data and propagate the trust scores.

Another usage of graphs for recommendation purposes is in the case of geospatial recommendations. Quercia et al. used graphs to find the shortest path between geographical locations, while also maximizing the enjoyment of the path for the user [65]. Locations were represented as nodes and connected to each other based on geographical proximity. Nodes were also ranked based on how pleasant (beautiful, quiet, happy) the locations were. Finally, a route that optimizes the shortness and pleasantness was computed based on a graph method and recommended to the user. In this work, both graph-based representation and graph theory methods are used for recommendation generation.

1.2.2 Feature Engineering for Recommendations

As mentioned at the beginning of the section, another group of related works that covers automatic feature engineering. According to Guyon et al., *"feature extraction addresses the problem of finding the most compact and informative set of features, to improve the efficiency or data storage and processing"* [31]. Basic features are a result of quantitative and qualitative measurements, while new features can be engineered by combining these or finding new means to generate additional measurements. In the big data era, the possibilities of engineering additional features, as well as their potential importance, have risen dramatically.

Feature engineering can be performed either manually or automatically. In the manual method, domain experts analyze the task for which the features are required, e.g., online movie recommendation versus customer churn prediction, and conceive features that may potentially inform the task. The engineering process involves aggregating and combining features already present in the data, in order to form new, more informative features. This approach, however, does not scale well because of the need for a human expert, the time it takes to compose features, and the sheer number of possibilities for the new features [24]. Conversely, automatic feature extraction, the process of algorithmically extracting new features from a dataset, does scale up well.

A basic approach for engineering new features from the existing ones is to combine them using arithmetic functions. In one study that evaluated this approach, arithmetic functions, such as min, max, average, and others, were used [54]. The study also presented a specific language for defining features, where the features were described by a set of inputs, their types, construction blocks, and the produced output. A framework for generating a feature space using the feature language as input was evaluated. The evaluation showed that the framework outperformed legacy feature generation algorithms in terms of accuracy. The main difference between the framework presented at [54] and its predecessors was that the framework was generic and applicable to multiple tasks and machine learning approaches.

Additional automatic feature engineering methods that are domain-specific were surveyed in [57, 25] for image recognition and in [70] for text classification purposes. An example of a feature engineering method for image recognition is quantifying the amount of skin color in an image in order to classify whether it contains a human face or not [28], whereas for text classification a bag-of-words can be generated for every document and used to describe it.

A different suite of methods for eliciting new features, which is also applicable to recommender systems, is latent features computation. Methods such as SVD [38] and PCA [83] can be used to compute new features and support the generation of recommendations by decomposing the available data into components and matching composing factors, i.e., the latent features. When the data is decomposed and there exists a set of latent features that can recompose it with a certain error rate, missing features and ratings can be estimated [2]. Although it has been shown that this approach successfully improves the accuracy of the recommendations [8], it is limited in the interpretability of the latent features found [43].

Unlike previous works, the current work defines an automatic feature engineering process based on graph-based representation of a recommender system data. The details of this process are provided in the following section.

1.3 Graph Based Data Modeling for Recommendation Systems

In this section, an approach for enhancing recommendations based on representing the data as a graph is presented. This representation allows a set of graph algorithms to be applied and a set of graph-related metrics, which offer a new perspective on the data and allow the extraction of new features, to be deduced. Following a brief overview of the approach, the structure of recommender system datasets is formalized (Section 1.3.1). Then a detailed description of porting data from a classical tabular representation to a graph-based representation is given (Section 1.3.2). An elaboration of methods for generating multiple graph representations follows (Section 1.3.2.2) and finally the process of exhaustively distilling graph features from these representations is outlined (Section 1.3.3).[2]

The input to the process (illustrated in Figure 1.1) is a tabular recommender system dataset and the output is a set of graph-based features capturing the relationships between the dataset entities from the graph perspective. The first step deals with the generation of a complete graph representation of the data: the tabular data is converted into a representation where the dataset entities are nodes, connected based on their co-occurrence in the data. Next, a set of partial representations is derived from the complete graph: first the basic representation containing only user and item nodes, and then additional alternative representations, each with a unique combination of relationships filtered from the complete graph. The partial representations are passed to the next step, where the extraction of the graph features is performed. Finally, the newly generated graph-based features are used to supplement the original features available in the dataset and this extended data is fed into the recommender system for the generation of predictions or recommendations. In the following subsections the above steps of the feature extraction process are elaborated.

1.3.1 The Structure of a Recommender System Dataset

In [16, 66], classical recommendation approaches are categorized into several groups: collaborative filtering, content based filtering, knowledge-based, community-based,

[2]An open source package implementing the approach is released at http://amitti.github.io/GraphRecSys/.

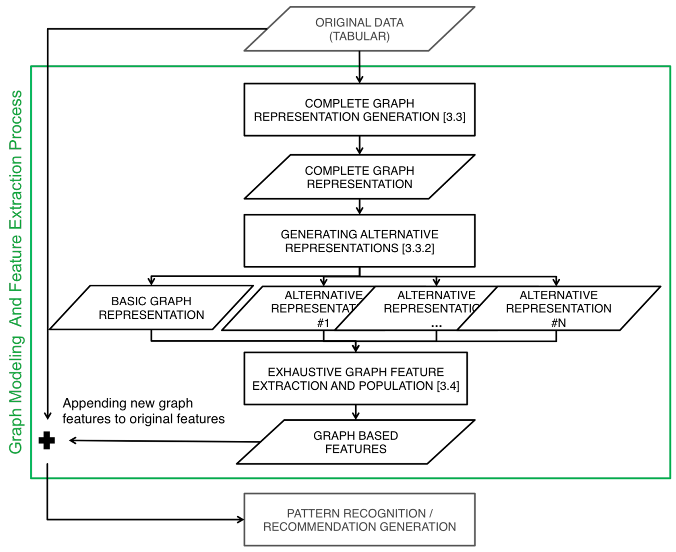

ORIGINAL DATA (TABULAR)
COMPLETE GRAPH REPRESENTATION GENERATION [3.3]
COMPLETE GRAPH REPRESENTATION
GENERATING ALTERNATIVE REPRESENTATIONS [3.3.2]
BASIC GRAPH REPRESENTATION
ALTERNATIVE REPRESENTAT #1
ALTERNATIVE REPRESENTATI ...
ALTERNATIVE REPRESENTATION #N
EXHAUSTIVE GRAPH FEATURE EXTRACTION AND POPULATION [3.4]
GRAPH BASED FEATURES
Appending new graph features to original features
Graph Modeling And Feature Extraction Process
PATTERN RECOGNITION / RECOMMENDATION GENERATION

Figure 1.1: Graph modeling and feature extraction flow chart

and hybrid approaches. We first consider the representation of the data used by these approaches, which can be converted into a tabular form as follows:

- In *collaborative filtering*, the data is represented as a matrix of user feedback on items (matrix dimensions are users$\times$items), where both the users and the items are denoted by their identifiers and the content of the matrix reflects the user feedback for the items, e.g., numeric ratings or binary consumption logs.
- In *content based filtering*, the items are modeled using a set of features, e.g., terms or domain features. Here, the matrix dimensions include the identifiers of the users, as well as the identifiers of the content features, and the values represent the preferences of the users for the features. The model also contains a second matrix with item identifiers and the same content features. The values in this matrix represent the weights of the features in each item.
- Two variants of *knowledge-based recommenders* – case-based and constraint-based – break the items into weighted features, e.g., the price of a product and the importance of the price for the user. This model can be represented by two matrices, one contains the items' weighted features and the second contains the users' ranking of the features importance. In the items matrix, each column represents a feature, each row represents an item, and the values are the strength, or how representative the feature is of the item. Similarly, in the users matrix, each column represents a feature, each row represents a user, and the values represent the importance of the feature for the given user.
- *Community-based recommenders* combine information regarding users' social relations with their ratings. Therefore, ratings of a socially close user are weighted heavier than of others. The items rating information can be represented in a ma-

trix identically to the one described in the collaborative filtering approach. The weights of social relations between users can be represented by a second matrix, where the rows and columns are represent users and the values quantify the degree of the relationship between them.

- Finally, *hybrid approaches* combine some of the above stand-alone recommendation models and, therefore, can be represented using the matrix representation.

The datasets used by the above approaches, which we denote by D, contain two key types of entities. The first refers to the entity for which the recommendations are generated, i.e., the user; it is referred to as the *source* entity and denoted by D_S. The second refers to the entity that is being recommended, e.g., item, content, product, service, or even another user. This entity is referred to as the *target* entity and denoted by D_T. This notation follows the primary goal of a recommender system: to recommend a target item to the source of the recommendation request. Additional data available in the datasets typically represent the features of the source and/or the target entity, or the relationships between the two. The feature set is denoted by D_F.

For example, in a movie recommenderation dataset, D_S refers to the system users and D_T to the recommendable movies. Any available features describing either the users or the movies are denoted by D_F. User features can be the user's age, gender, and location, while movie features can be genre, director, language, and length. A practical assumption is made that in a tabular recommender dataset, all the features associated with an entity are stored in the same table as the entity itself. That is, the gender of a user is stored in the user table rather than in the movie table. A formal representation of the entities and their features in the above example is $D = \{D_S, D_T, D_F\}$, where $D_S = user_{id}$, $D_T = movie_{id}$, and the features D_F are split into $D_F = \{D_{FS}, D_{FT}\}$ as follows: $D_{FS} = \{f_{s1} = age, f_{s2} = gender, f_{s3} = location\}$ and $D_{FT} = \{f_{t1} = genre, f_{t2} = director, f_{t3} = language, f_{t3} = length\}$.

It should be noted that the source and target entities can have common features [10]. For example, in the case of a restaurant recommendation task, the source entity (user) and target entity (business) can both have the "location" feature. The role of the source/target entities and features can also change according to the recommendation task at hand. In the restaurant recommendation example, when the task is to recommend restaurants to users, the users are the source entity, the restaurants are the target entity, and location is a feature of both. However, if the task was to recommend a location, e.g., tourist destinations, for a user to visit based on the restaurants in that location, then the source entity would still be the users, the target entity would be the locations, and the restaurants would be the features of the locations.

An important aspect that needs to be considered is the relationship between the entities, e.g., the fact that a user watched, rated, tagged, or favored a movie. Relationships can be established not only between a source and a target entity, but also between two source/target entities. Examples of relationships between two users are the directional followee-follower relationship or the non-directional friendship. Relationships between two movies can be established because they are directed by the same director, are in the same language, and so forth. Relationships between entities are defined using the tuple $(source \in D_S, \{features\} \in D_F, target \in D_T)$. For example, user rating for a movie is defined by $rel_{rating} = (user, value, movie)$ and friendship between two users is defined by $rel_{friend} = (user, \{\varnothing\}, user)$.[3] The set of all possible relationships in a dataset is denoted by $D_R = \{rel_i\}$, such as in the movies example $D_R = \{rel_1 = rating, rel_2 = friendship\}$.

Given the above formalization of entities, features, and relationships, a recommendation task implies the prediction of a relationship between entities. For example, the task of a movie recommender can be considered as the prediction of the rel_{rating} relationship. This relationship can be numeric (star rating) or binary (interested or not interested), but the recommendations delivered to the users are guided by the predicted values of rel_{rating}. If, on the contrary, the system is a social recommender that recommends online friends, then the relationship in question is rel_{friend} and its task is to recommend a set of candidate friends.

In addition to the original data that is available to the recommender, more features can be generated and distilled, thus, enriching the dataset. For example, two popular features frequently computed in rating-based recommendation datasets are the average rating of a user and the average rating for an item. These features are associated with the users and items, stored in the relevant tables, and they are used to refine, e.g., normalize, the predicted ratings and improve the quality of the recommendations [68]. The question addressed in this work is whether the availability of additional, supposedly more complex, features that encompass more information and stem from graph representation of the data can contribute to the accuracy of the predictions and the quality of the recommendations. In the following sub-sections, the details of extracting and populating features are provided.

1.3.2 Transforming Tabular into Graph-based Representation

1.3.2.1 Basic graph representation for recommender systems data

When moving from the tabular to the graph-based representation of a recommender system data, three key graph design considerations are:

1. Should the graph encompass all the available data? What parts of the dataset are important and need to be represented by the graph?
2. Which entities from the selected data should be represented by graph vertices and which entities by graph edges?
3. How should the edges be defined? Should they be directed or undirected? Should they be labeled? What should the labels be?

Regarding the first question, it is probable that the decision regarding the data to be represented in the graph is data-dependent. For some domains, datasets, and recommendation tasks, certain parts of the data may be more informative than others. Since the space of possible graph-based data representations is too large for determining a-priori the most suitable scheme, a possible alternative is to start with a graph model based on the entire data, and then, to systematically extract all sub-graph representations and their features. This leads to automatic coverage of the entire search space, inherently uncovering the representations that produce the most effective features. Then, the most informative feature set can be selected.

[3]Additional friendship features, such as duration or strength, can also be included.

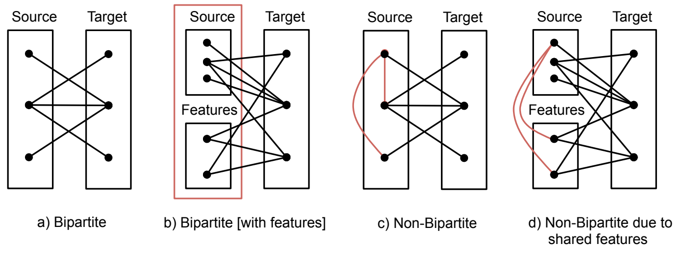


Figure 1.2: Examples of two types of graph schemes representing a recommender system dataset: bipartite (a,b) and non-bipartite (c,d). In (b) the red block confines the multi-part bipartite graph component. In (c) and (d) the red edges break the bipartite structure.

To answer the second and third questions, an intuitive modeling approach is used. Namely, the graph model considers all the source, target, and feature entities as vertices, while their links and relationships between features (including user feedback on items) are the edges. If the information about the relationship is binary, e.g., the item is viewed or not, the edges are not labeled. Otherwise, the edges labels communicate the information about the relationship, e.g., rating or type of association. In most cases, the edges are not directed, as information about a feature connected to an entity or about an entity connected to a feature is equivalent. Although this work does not consider directed edges, the proposed approach can be extended to support this (outlined in Section 1.6.2).

Based on the above abstraction of recommender systems datasets, the following basic graph representation emerges. User and item entities are represented by the graph vertices, and edges connect a user and an item vertex when an association between the two is available. This association can be explicit (ratings or likes) or implicit (content or user view). This graph is called a bipartite graph [81], because it can be split into two partitions consisting of the source and target entity vertices, i.e., the users and items, respectively (Figure 1.2-A).

The basic representation can be extended by adding additional features as new graph vertices and linking them to the existing vertices. For example, if user locations are provided, each location can be represented by a vertex and the users associated with the locations are linked to their vertices. A similar situation may occur in the target partition of the graph, e.g., the target entity of movies and a variety of their content features: genre, actors, keywords, and more (Figure 1.2-B). Adding the feature vertices still preserves the bipartite nature of the graph, but the partition with the added features gets virtually split into two groups of vertices: the entities themselves and their features.

The situation changes, however, when adding information within the source or target partitions, e.g., user-to-user social links or item-to-item domain links. This information introduces new links *within* the partitions, which break the bipartite structure (Figure 1.2-C). Additional information that may break the bipartite structure is the common features shared between the source and target partitions. For example, in the movie domain, the items may be linked to their genres, while the users may also express their preferences towards the genres. Thus, links to the genre vertices are established from both the user and item partitions (Figure 1.2-D) and the graph is no longer bipartite. Note that each of the four schemes shown in Figure 1.2 generates different sets of features and the values of the features also vary.

Following is an outline[4] of a high-level approach for generating the *complete graph*, which includes all the data and relationships of a recommender dataset. The algorithm scans all the tables in the dataset, and for each column that is not a source entity column, target entity column, or feedback column, it generates a graph node for every unique value appearing in the column. Thus, every unique $user_{id}$ and $movie_{id}$ is assigned to a graph vertex, as well as every actor, director, movie genre, keyword, and so forth. Non-categorical features, e.g., movie budget, can be discretized using binning, e.g., under \$10M, \$10M-\$20M, \$20M-\$30M, or based on the observed distribution, e.g., four equal-sized quarters, each containing 25% of the data. Upon discretizing the values in the columns and creating the nodes, all the nodes matching the values that appear in the same row are connected by edges to the source and target nodes of the same row, if available. The result is a graph that contains all the values of the features as the graph nodes, which are connected to the source and target entities based on their co-occurrence in the data.

1.3.2.2 Multiple sub-graph representations

Despite being included in a dataset, not all the features are necessarily informative and contribute to the accuracy of the recommendations. Certain features may be noisy or bear little information, thus, hindering the recommendation process. For example, if a feature is sparsely populated, its values are identical across users, or it is populated only across a certain subset of users, then this feature is unlikely to help the recommender and may not be included in the graph representation. However, it is hard to assess the contribution of the features in advance with a high degree of certainty. This leads to the idea of automatically deriving multiple sub-graph representations from the complete graph and extracting the graph features for each sub-graph first, and selecting the most informative ones in a later stage. Specifically, all the possible sub-graphs are exhaustively generated and their features are extracted. Each sub-graph represents a combination of features influenced by the entities and relationships included in the graph. The process is detailed in Algorithm 1.

The input to the algorithm is the complete graph representation $CompleteGraph$, which was discussed at the end of Section 1.3.2.1, and the edge $PredEdge$ representing the relationship rel_i being predicted. The function `GenerateEdgeCombinations` invoked in line 1 returns all the possible combinations of different types of graph edges. Note that this function receives also the type of the predicted edges $PredEdge$. This is done in order to preserve the $PredEdge$ edges in all the sub-graphs. Namely, this type of edges will not be included in the combinations that are removed from the complete graph and, therefore, will be present in all the sub-graphs.

[4]The pseudo codes omit several technical details that can be found in the accompanying library.

Algorithm 1: Generate sub-graphs and extract features

```
input : CompleteGraph - complete graph representation of the dataset
        PredEdge - edge type of the relationship being predicted
output: ExtractedGraphFeatures - set of features extracted from sub-graph
        representations
1 GraphEdgeTypeCombinations ←
    GenerateEdgeCombinations({EdgeTypes}, PredEdge)
2 ExtractedGraphFeatures ← ∅
3 foreach EdgeCombination ∈ GraphEdgeTypeCombinations do
4     SubGraph ← RemoveEdgesFromGraph(CompleteGraph,
        EdgeCombination)
5     SubGraphFeatures ← ExtractGraphFeatures(SubGraph, PredEdge)
6     ExtractedGraphFeatures ← (ExtractedGraphFeatures ∪ SubGraphFeatures)
7 end
8 return ExtractedGraphFeatures
```

Upon generating all the possible edge type combinations, the set is iterated over and the function `RemoveEdgesFromGraph` is invoked to create a sub-graph *SubGraph* by removing the combination *EdgeCombination* from *CompleteGraph* (line 4). Then, the function `ExtractGraphFeatures` is invoked to extract from *SubGraph* the set of possible graph features referred to as *SubGraphFeatures* (line 5, to be elaborated in Section 1.3.3) and append *SubGraphFeatures* to the set of features *ExtractedGraphFeatures* (line 6). Finally, in line 8 the algorithm returns *ExtractedGraphFeatures* – the set of all the possible graph features from all the possible sub-graphs.

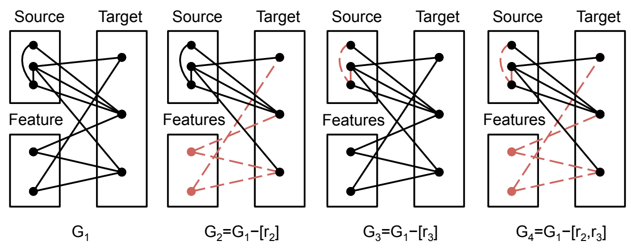


Figure 1.3: Four sub-graph schemes generated from the complete schema based on the relationship permutations. Dashed lines represent links removed from the graph.

The execution of Algorithm 1 is illustrated by an example in Figure 1.3. Consider a graph $G = (V,E)$, where $V = \{V_S \cup V_T \cup V_F\}$ is the set of vertices of the source entities $V_S = \{V_{S1},...,V_{Sm}\}$, target entities $V_T = \{V_{T1},...,V_{Tn}\}$, and domain feature values $V_F = \{V_{F1},...,V_{Sk}\}$. In addition, $E = \{rel_1, rel_2, rel_3\}$ is the set of graph edges, reflecting three relationship types: rel_1 is the source-target relationship being predicted; rel_2 is the relationship between the target entities and domain features; and rel_3 is the relationship between the source vertices. In graph terminology, the recommendation task is to predict the label (or the existence) of an edge $rel_1(i,j)$ between a source vertex V_{Si} and a target vertex V_{Tj}.

For this graph, the set *GraphEdgeCombinations* created by `GenerateEdgeCombinati` includes $GraphEdgeCombinations = \{\{\varnothing\},\{rel_2\},\{rel_3\},\{rel_2,rel_3\}\}$. These are the combinations of edges that are removed from the graph while creating sub-graphs, whereas the predicted relationship rel_1 is preserved in all the sub-graphs. Removing these combinations of edges, function `RemoveEdgesFromGraph` generates four variants of *SubGraph* shown in Figure 1.3: $G_1 \leftarrow CompleteGraph - \{\varnothing\}$, $G_2 \leftarrow CompleteGraph - \{rel_2\}$, $G_3 \leftarrow CompleteGraph - \{rel_3\}$, and $G_4 \leftarrow CompleteGraph - \{rel_2, rel_3\}$. Note that G_1 is the complete graph, whereas other sub-graphs have either rel_2 or rel_3, or both removed. For each *SubGraph*, function `ExtractGraphFeatures` is invoked to extract the respective feature set *SubGraphFeatures* and all the extracted feature sets are appended to *ExtractedGraphFeatures*.

1.3.3 Distilling Graph Features

The function `ExtractGraphFeatures` in line 5 of Algorithm 1 received a sub-graph derived from the complete representation and was invoked to extract a set of graph-based features. Moreover, this function was invoked for all the possible sub-graphs, to ensure that all the possible graph features are extracted. The graph-based features are extracted using a number of functions, each calculating a different graph metric. These functions, referred as *generators*, are divided into several, families according to the number of graph vertices they process.

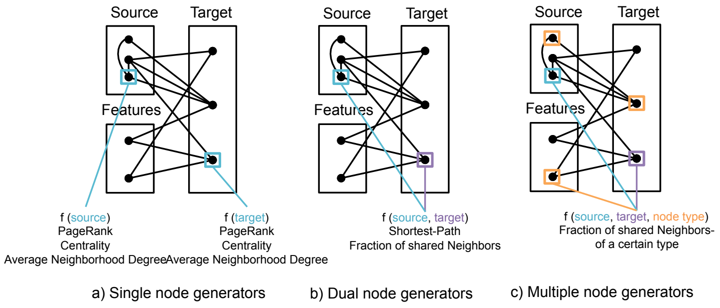

Source
Target
Features
f (source)
PageRank
Centrality
Average Neighborhood Degree
f (target)
PageRank
Centrality
Average Neighborhood Degree
f (source, target)
Shortest-Path
Fraction of shared Neighbors
f (source, target, node type)
Fraction of shared Neighbors-of a certain type
a) Single node generators
b) Dual node generators
c) Multiple node generators

Figure 1.4: Key graph-based feature generator families and their instances

The main steps of `ExtractGraphFeatures` are detailed in Algorithm 2, which uses three types of generators:

- `1-VertexGenerators` are applied to a single vertex, either the source or the target entity, and compute features of this vertex, e.g., PageRank (Figure 1.4-A).
- `2-VertexGenerators` are applied to a pair of vertices, the source and the target entities, and compute graph-based relationships between the two, e.g., the shortest path (Figure 1.4-B).

- `N-VertexGenerators` are applied to $N > 2$ vertices, two of which are the source and target entities and the rest are not, e.g., “number of vertices common neighbors of the source and target vertices” (Figure 1.4-C).

Section 1.3.3.1 lists the functions from each generator family that were used. Note that these are executed iteratively, in order to generate all the possible graph features. By no means this list of functions is exhaustive; it exemplifies a number of popular functions that were used, but more functions can be conceived and added.

Algorithm 2: Extract graph features from a sub-graph

```
input : SubGraph - sub-graph derived from the complete graph representation
        PredEdge - edge type of the relationship being predicted
output: ExtractedSubGraphFeatures - set of features extracted from SubGraph

1  ExtractedSubGraphFeatures ← ∅
2  SubGraphPredictedEdges ← ExtractPredictedEdges(SubGraph,PredEdge)
3  foreach (SourceEntity,TargetEntity) of Edge ∈ SubGraphPredictedEdges do
4      foreach 1-Function in 1-VertexGenerators do
5          SourceFeatures ← 1-Function(SourceEntity)
6          TargetFeatures ← 1-Function(TargetEntity)
7          ExtractedSubGraphFeatures ← (ExtractedSubGraphFeatures ∪
             SourceFeatures ∪ TargetFeatures)
8      end
9      foreach 2-Function in 2-VertexGenerators do
10         SourceTargetFeatures ← 2-Function(SourceEntity,TargetEntity)
11         ExtractedSubGraphFeatures ← (ExtractedSubGraphFeatures ∪
             SourceTargetFeatures)
12     end
13     MultipleEntityCombinations ←
         ExtractEntityCombinations({VertexTypes})
14     foreach EntityCombination ∈ MultipleEntityCombinations do
15         N ← |EntityCombination|
16         foreach N-Function in N-VertexGenerators do
17             MultipleEntityFeatures ← N-Function(SourceEntity, TargetEntity,
                 EntityCombination)
18             ExtractedSubGraphFeatures ← (ExtractedSubGraphFeatures ∪
                 MultipleEntityFeatures)
19         end
20     end
21     return ExtractedSubGraphFeatures
22 end
```

At the initial stage of Algorithm 2, edges belonging to the predicted relationship are copied to the *SubGraphPredictedEdges* set (line 2). For each *Edge* in this set, the generators are invoked as follows. The `1-VertexGenerators` functions are invoked in lines 5 and 6, respectively, on the *SourceEntity* and *TargetEntity* vertices of *Edge*. Applying these functions to other vertices is unlikely to produce features that can contribute to the prediction of the desired relationship, while leading to significant computational overheads. Hence, `1-VertexGenerators` are restricted to these two vertices only. The `2-VertexGenerators` are applied in line 10 to the pairs of vertices *SourceEntity* and *TargetEntity*. Then, the `ExtractEntityCombinations` function is invoked in line 13, in order to create a set of all the possible entity combinations of vertices, *MultipleEntityCombinations*. These combinations necessarily involve *SourceEntity* and *TargetEntity*, and in addition any other type of graph vertices. For each combination *EntityCombination* of size N (line 15), the relevant `N-VertexGenerators` generators are invoked in line 17. Features extracted by `1-VertexGenerators`, `2-VertexGenerators`, and `N-VertexGenerators` are all appended to *ExtractedSubGraphFeatures*.

Note that the value of N determines the `N-VertexGenerators` functions that are invoked and the relationships they uncover. Again, two of the N vertices are necessarily *SourceEntity* and *TargetEntity*, whereas the third vertex can be of any other entity linked to either of them. For instance, for $N = 3$ in the movie recommendation task and entities of user, item, and location, the relationship can be "the number of cinema locations that the user has visited and where the movie is screened". The generator considers the user and movie vertices, and then, scans all the location vertices and identifies those, with edges connected to both. More complex relationships with a higher value of N can be considered. As such, the `N-VertexGenerators` extract a number of features that surpasses by far the set of features that can be engineered manually.

1.3.3.1 Distilled graph features

The set of metrics selected for implementation in this work and used for the evaluation of the approach is now given in detail. The metrics are those that are commonly implemented in widely used graph analysis libraries – NetworkX [32], igraph [22], and Gephi [5]) – and used in social network analysis and measurement works [82, 49]. It is important to stress that this set of metrics is only a portion of those that could be used and serves only as an example. The space of all graph metrics is large, as can be seen in [21, 80, 19], and, thus, could not be exhaustively evaluated within the scope of this work.

The set of `1-VertexGenerators` functions were implemented and used for evaluation are degree centrality [13], average neighbor degree [4], PageRank score [60], clustering coefficient [46], and node redundancy [46]. These metrics are referred to as the *basic* graph features.

- *Degree Centrality* [13] (or, simply, node degree) quantifies the importance of a vertex through the number of other vertices to which it is connected. Hence, in the bipartite graph, the degree centrality of a user vertex S_i is the activity of i, i.e., the number of items with which S_i is associated, and, vice versa, for an item vertex T_j it is the popularity of j, i.e., the number of users who are associated with T_j. In a graph that includes metadata, the number of metadata vertices associated with either the user or the item vertex are added to the degree centrality score. The degree of centrality of a vertex v is denoted by $Deg(v)$.

- *Average Neighbor Degree* [4] measures the average degree of vertices to which a vertex is connected. In the bipartite graph, this metric conveys for S_i – the average popularity of items with which S_i is associated, and for T_j – the average activity of users who are associated with T_j. Formally, if $N(v)$ denotes the set of neighbors of a vertex v, then the average neighbor degree is

$$AvgNghDeg(v) = \frac{1}{|N(v)|} \sum_{u \in N(v)} Deg(v) \tag{1.1}$$

  In a graph with metadata, the average neighbor degree of a user/item vertex also incorporates the popularity of the metadata features with which it is associated.
- *PageRank* [60] is a widely-used recursive metric that quantifies the importance of graph vertices. For a user vertex S_i, the PageRank score is computed through PageRank scores of a set of item vertices $\{T_j\}$ with which S_i is associated and vice versa. Thus, the PageRank score of a user vertex S_i can be expressed as

$$PageRank(S_i) = \sum_{T_j \in N(S_i)} \frac{PageRank(T_j)}{Deg(T_j)}, \tag{1.2}$$

  i.e., the PageRank score of S_i depends on the PageRanks of each item vertex T_j connected to S_i, divided by the degree of T_j. In a graph with metadata, the PageRank scores of user/item vertices are also affected by the PageRank of the metadata vertices to which they are connected.
- *Clustering Coefficient* [46] measures the density of the immediate subgraph of a vertex as the ratio between the observed and possible number of cliques of which the vertex may be a part. Since cliques of a size greater than two are impossible in the bipartite graph, *ClustCoef* measures the density of shared neighbors with respect to the total number of neighbors of the vertex:

$$ClustCoef(v) = \frac{\sum_{u \in N(N(v))} \frac{|N(v) \cap N(u)|}{|N(v) \cup N(u)|}}{|N(N(v))|} \tag{1.3}$$

- *Node Redundancy* [46] is applicable only to bipartite graphs and shows the fraction of pairs of neighbors of a vertex that is linked to the same other vertices. This metric quantifies for user vertex S_a - the portion of pairs of items with which a is associated that are also both associated with another user b. Likewise, for item vertex T_x, it quantifies the portion of pairs of users associated with x and also both associated with another item y. If the vertex is removed from the graph, node redundancy reflects the fraction of its neighbors that will still be connected to each other through other vertices.

Next, multiple-vertex generator functions are detailed. Specifically, the following functions from the `2-VertexGenerators` and `N-VertexGenerators` families were implemented:

- *Shortest Path* [26]. Unlike the above feature generators that operate on a single vertex, shortest path receives a pair of graph vertices: a source entity and a target entity. It evaluates the distance, i.e., the lowest number of edges, between the two vertices. The distance communicates the proximity of the vertices in the graph, as is a proxy for their similarity or relatedness. A short distance indicates high relatedness, e.g., more items shared between users or more features for items, while a longer distance indicates low relatedness.

- *Shared Neighbors of Type X.* This is one of the `N-VertexGenerators` functions, which receives three parameters: source entity vertex, target entity vertex, and entity type X. It returns the fraction of neighbors shared between the source and target vertices that are of the desired type X. The fraction is computed relatively to the union of the source vertex neighbors with the target vertex neighbors. Note that this feature cannot be populated for graphs that do not have a sufficient variety of entities connected to the source and target vertices.
- *Complex relationships across entities.* Apart from the above mentioned generators, system designers may define other `N-VertexGenerators` functions, which could extract valuable features. For example, it may be beneficial for a movie recommender to extract the portion of users, who watched movies from genres g_1, g_2 directed by person p, and released between years t_1 and t_2. It is clear that it is impossible to exhaustively list all the combinations of such features: this is domain- and application-dependent. Hence, the task of defining these complex generators is left open-ended and invites system designers to use the provided library and develop their own feature generators.

To recap, each of the above `1-VertexGenerators` and `2-VertexGenerators` is applied to every source and target vertex and generates features associated with the vertex or a pair of vertices. In addition, `N-VertexGenerators` is applied to the source and target vertices and all the possible combinations of other entity types. Recall that this is done for every sub-graph extracted from the complete graph and the complexity of the feature generation task becomes clear.

1.4 Experimental Setting and Datasets

It is important to highlight that the product of the presented approach is graph-based features that help to generate recommendations using existing recommendation methods. These features can either be used as stand-alone features, i.e., the only source of information for the recommendation generation, or be combined with other features. Hence, the baseline for comparison in the evaluation part is the performance of common recommendation methods when applied **without the newly generated features**.

To present solid empirical evidence, the contribution of the graph feature extraction to the accuracy of the recommendations was evaluated using three machine learning methods: Random Forest [14], Gradient Boosting [27], and Support Vector Machine (SVM) [30]. Both Random Forest and Gradient Boosting are popular ensemble methods that have been shown to be accurate and won recommendation [42] and general prediction [85] competitions. The methods are also implemented in widely used machine-learning libraries [62, 33], and were shown to perform well in prior recommender systems works [35, 7, 78].

In the next section, two case studies showing the contribution of graph-based features are presented. These case studies demonstrate the value of the proposed graph-based approach applied to a range of recommendation tasks and domains. Case study I evaluates the performance of the graph-based approach, evaluating its contribution in different domains and tasks. Case study II focuses on the impact of representing data using different graph schemes on the recommendations.

1.4.1 Dataset I – Last.fm

The first dataset is of users' relevance feedback provided for music performers via the Last.fm online service, which was obtained in [17]. The dataset consists of 1,892 users and 17,632 artists whom the users tagged and/or listened to. More than 95% of users in the dataset have 50 artists listed in their profiles as a result of the method used to collect the data. There are 11,946 unique tags in the dataset, which were assigned by users to artists 186,479 times. Each user assigned on average 98.56 tags, 18.93 of which are distinct. Each artist was assigned 14.89 tags on average, of which 8.76 are distinct. The dataset also contains social information regarding 12,717 bidirectional friendship linkss established between Last.fm users, based on common music interests or real life friendship.

We briefly characterise the dataset. Initially, we consider the distribution of the number of friends per user. The average number of user-to-user edges is low, where the vast majority of users have less than 10 friends and about half of users have less than four friends. Intuitively, a friendship edge between two users can be an indicator of similar tastes, and as such, friendship-based features are expected to affect the recommendations. Then, we consider the distributions of the number of listens per artist, user, and in total. It can be observed that the overall and per artist distribution are highly similar. The user-based distribution resembles the same behaviour, but drops faster. This aligns with the intuition that the number of users who listen to hundreds of artists is smaller than the number of artists who are listened by hundreds of users [34].

There are four relationships in the Last.fm dataset: *[user, listens, artist]*, *[user, uses, tag]*, *[tag, used, artist]*, and *[user, friend, user]*. The task defined for this dataset was to predict the artists to whom a users will listen the most, i.e., the predicted relationship was *[user, listens, artist]*. This task requires first predicting the number of times each user will listen to each artist, then ranking the artists, and choosing the top K artists. Based on the sub-graph generation process detailed in Algorithm 1 and the relationship being predicted, the data can be represented via eight graph schemes in general. Four graph schemas that incorporate the source and target entities were evaluated:

- Bipartite graph that includes users and artists only, denoted as the baseline (BL)
- Non-bipartite graph that includes users, social links, and artists (BL+F)
- Non-bipartite graph that includes users, artists, and tags assigned by users to artists (BL+T)
- Graph that includes all the entities and relationships: users, tags, artists and social links (BL+T+F).

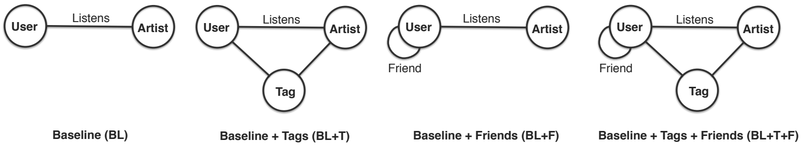


Figure 1.5: Graph representations for dataset I (Last.fm)

The four graphs are illustrated in Figure 1.5. For each of the graphs, two sets of features were generated: basic features, as well as a set of extended features associated with the auxiliary data being included. The generated features are used as the input for a Gradient Boosting Decision Tree regressor [27], trained to predict the number of listens for a given user-artist pair.

A 5-fold cross validation was performed. Users with fewer than five ratings were pruned, to ensure that every user has at least one rating in the test set and four in the training set. For each training fold, a graph was created for each graph model shown in Figure 1.5. For each user, in the test set, a candidate set of artists was created by selecting artists out of the set of the artists listened to by the user and complementing these by randomly selected artists. For example, a candidate set of 100 artists included 10 artists listened to by the user and 90 random artists. Three different candidate sizes were evaluated: 50, 100, and 150.

Then, a regressor was used to predict the number of listens for each artist in the candidate set, rank the set, and compute precision at 10 (P@10) as the performance metric [72]. If candidate set *CS* consists of the artists selected from a user's artist set denoted by *UA* and the randomly selected artists set *RA*, then P@10 is computed by $P@10 = (UA \cap top\_10\_artists(UA \cup RA))/10$, where top_K_artists is the list of top-K artists in *CS* ranked according to the predicted number of listens. Finally, an average of the P@10 scores across all the users in the test set is computed. In order to evaluate the significance in the performance of the various graph schemes feature sets, a two-sided t-test was applied on the results.

1.4.2 Dataset II – Yelp (from RecSys-2013)

The second dataset is of users relevance feedback given for businesses, such as restaurants, shops, and services. The dataset was released by Yelp for the RecSys-2013 Challenge [12]. For the analysis, users with less than five reviews were filtered out, which resulted in 9,464 users providing 171,003 reviews and the corresponding ratings for 11,197 businesses. The average number of reviews per user is 18.07 and the average number of reviews per business is 15.27. A key observation regarding this dataset is the distribution of ratings, which were almost all positive (more than 60% of ratings were at least 4 stars on a 5-star scale), and the low variance of ratings across businesses and users. This phenomenon is common in star rating datasets, where users tend to review fewer items that they did not like.

We discuss the basic statistics of users and businesses in the Yelp dataset. Initially, we consider the distribution of the number of reviews and ratings per user. A long tail distribution of the number of businesses a user reviewed can be observed,

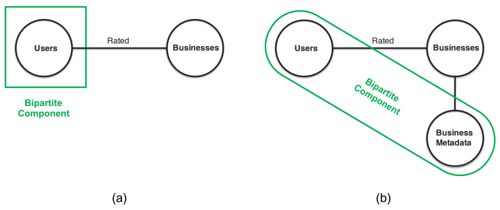


Figure 1.6: Graph representations for dataset II (Yelp - RecSys-2013 Challenge)

with more than 75% of the users providing less than 10 reviews. Then, we consider the distribution of the number of reviews a business received. Only 24% of businesses attract more than 10 reviews, while only a few businesses (less than 2%) have more than 100 reviews. Despite the high number of categories in the data, the average number of categories with which a business is associated is only 2.68. Every business is also associated with a single location.

The task defined for this dataset is the one originally defined for the RecSys-2013 challenge, i.e., to predict the ratings a user will assign to businesses. Two graph models were implemented and evaluated based on this dataset: a *bipartite* model with sets of vertices U and B representing users and businesses and a *tripartite*[5] model with sets of vertices U, B, and M representing users, businesses, and metadata items, respectively. The high-level graph representation models are illustrated in Figure 1.6, while the detailed presentation of the sub-graphs will be given in Section 1.4.3, in which the follow-up dataset is presented.

The features generated for this dataset were aggregated into three groups:

- *Basic* features that include only the unique identifiers of users $\{u_i\} \in U$ and businesses $\{b_j\} \in B$.
- *Manual* features that include the number of reviews by u_i, average rating of u_i, number of reviews for b_j, number of categories $|\{m\}|$ with which b_j is associated, average number of businesses in $\{m\}$, average rating of businesses in $\{m\}$, the main category[6] of b_j, average degree of businesses associated with the main category of b_j, average degree of businesses in $\{m\}$, and the location of b_j.
- *Graph* features that include the degree centrality, average neighbor degree, PageRank score, clustering coefficient, and node redundancy. These features were generated for both user nodes u_i and business nodes b_j, whereas an additional shortest path feature was computed for the pairs of (u_i,b_j).

In this case, a Random Forest regression model [14] was applied for the generation of the predictions of users ratings for businesses. At the classification stage, the test data items were run through all the trees in the trained forest. The value of the predicted rating was computed as a linear combination of the scores of the terminal

[5]The use of 'tripartite' is somewhat misleading, as the "bipartite graph with metadata nodes" notation would be more appropriate. For the sake of brevity, the bipartite and tripartite terminology is used.
[6]Each business in the Yelp dataset is associated with multiple categories, some having an internal hierarchy. The main category is the most frequent root category a business was associated with.

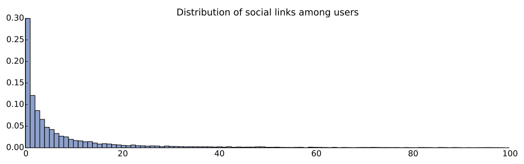


Figure 1.7: Yelp II Dataset characteristics – distribution of social links

nodes reached when traversing the trees. It should be noted that the ensemble of trees in Random Forest and the selection of the best performing feature in each node inherently eliminate the need for feature selection. Since every node uses a single top performing feature for decision making, the most predictive features are naturally selected in many nodes and the ensemble of multiple trees virtually replaces the feature selection process.

A 5-fold cross validation was performed. For each fold, the predictive model was trained using both the original features encapsulated in the dataset and the new graph features. The basic and manual groups of features were populated directly from the reviews, whereas the graph features were populated from the bipartite and tripartite graph representations and augmentrf the former groups of features. Predictive accuracy of various combinations of features was measured using the widely-used RMSE metric [72], computed as $RMSE = \sqrt{\frac{\sum_n (\hat{y}_t - y_t)^2}{n}}$, where n is the number of predictions, $\hat{y}_t$ are the predicted values, and y_t are the actual user ratings. A two-sided t-test was applied to validate the statistical significance of the results.

1.4.3 Dataset III – Yelp II (with social links)

The third dataset is an extension to the previous dataset released by Yelp. The new data contains more users, businesses and reviews, and, more importantly, new information regarding users' social links. The distribution of the social links among users is illustrated in Figure 1.7. It can be seen that the social links follow a long tail distribution, where most users have a small number of links: 29% with no links, 57% with less than 20 links, and only a few users with more than 20 links. The social links break the bipartite structure of the first Yelp dataset, which influences the generated graph features.

The task here is identical to that of the first Yelp dataset, i.e., predicting users ratings for businesses. Eight graph models were generated and evaluated based on this dataset. The models are illustrated in Figure 1.8 and, depending on the availability of the user-to-user friendship edges, categorized as bipartite or non-bipartite. The complete graph is shown in the top-left schema. In the following three schemes one type of edges is missing: either social links, user names, or categories. In the next three, two types of edges are misisng: social and categories, social and names, and names and categories. Finally, in the bottom-right graph all three are missing.

The generated features presented in Section 1.3.3.1 are referred to in the evaluation of this dataset as the *basic* features. These features are aggregated into groups,

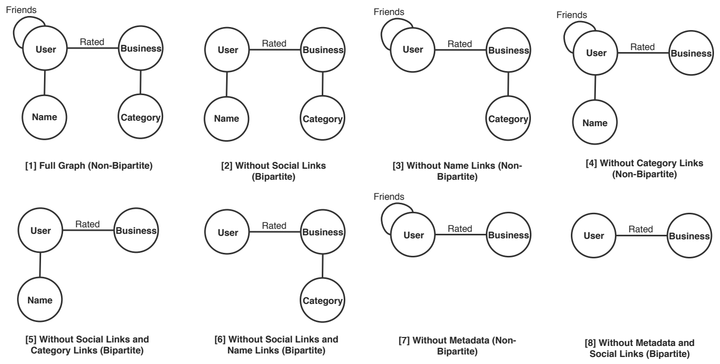

Friends
User
Rated
Business
Name
Category
[1] Full Graph (Non-Bipartite)
[2] Without Social Links (Bipartite)
[3] Without Name Links (Non-Bipartite)
[4] Without Category Links (Non-Bipartite)
[5] Without Social Links and Category Links (Bipartite)
[6] Without Social Links and Name Links (Bipartite)
[7] Without Metadata (Non-Bipartite)
[8] Without Metadata and Social Links (Bipartite)

Figure 1.8: Graph representations for dataset III (Yelp II)

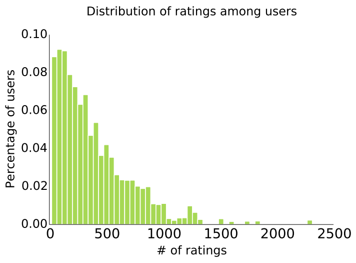
based on the graph scheme from which they were extracted. For example, all the features extracted from the graph named “without category links” in Figure 1.8 were grouped into a combination having the same name. Another evaluated combination includes the union of all the features generated from all the graph schemes, and this is named “all graph features”. Finally, the union of “all graph features” with the “basic features” is referred to as “all features”.

A 5-fold cross validation was performed. For each fold, the predictive models were trained using graph features extracted from each of the above feature sets. The evaluation used different methods (Random Forests, Gradient Boosting, and SVM) to evaluate how the choice of method impacts the results. Predictive accuracy of various feature combinations was measured using the RMSE metric [72], and a two-sided t-test was applied to validate statistical significance.

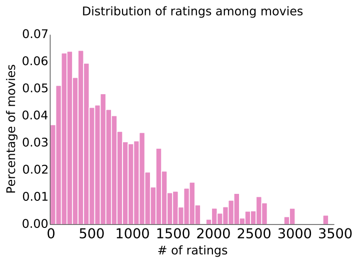
1.4.4 Dataset V – Movielens

Movielens [45] is a classical recommender systems dataset studied in numerous prior works. In this work it is used to show that the graph-based approach is as effective on legacy datasets as on more recent datasets including social data. The 1M Ratings Movielens dataset consists of 1,000,209 ratings assigned by 6,040 users for 3,883 movies, on a discrete scale of 1 to 5 stars. Each user in the dataset rated at least 20 movies. The distribution of ratings across users and movies is illustrated in Figures 1.9a and 1.9b, respectively. The dataset contains metadata of both users and movies. The user metadata includes the gender, occupation, zip code area, and age group, while the movie metadata contains the genre(s) of the movies.

The task defined for this dataset was to predict user ratings for movies. Based on the above description of the dataset, 32 graph schemes were generated and evaluated (see Figure 1.10). The schemes are categorized based on the number of relationships removed from the complete graph. As can be seen, there are four categories: schemes with a single node type removed, containing 5 sub-graphs, schemes with 2 node types removed containing 10 sub-graphs, schemes with 3 node types removed



(a) Distribution of ratings across users



(b) Distribution of ratings across movies

Figure 1.9: Movielens dataset characteristics

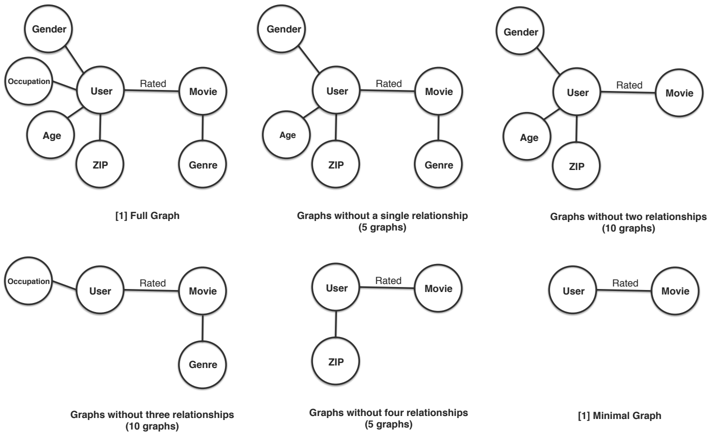


Figure 1.10: Graph representations for dataset IV (Movielens). Each graph is an example of the sub-graphs in the group.

containing 10 more sub-graphs, and finally, schemes with 4 node types removed containing 5 graphs. The minimal graph scheme is the one from which all the entities and relationships were removed, except for the source and target entities and the predicted 'rating' relationships.

5-fold cross validation was performed. For each fold, the predictive models were trained using graph features extracted from each of the above graph schemes. The evaluations used the Random Forest and Gradient Boosting approaches, in order to evaluate how the choice of the learning method impacts the results. The predictive accuracy of various combinations of the above feature sets was measured again using the RMSE and MAE predictive accuracy metrics [72], and a two-sided t-test was applied to validate statistical significance.

Dataset	Source and Target Entities	Graph Schemes	Graphs	Feature Sets	Extracted Features	Learning Method	Metric
Last.fm	1,892 (users) 17,632 (artists)	Bipartite + Non-bipartite (w/ social links, w/ tags, w/ social links+tags)	4	7	Basic graph features, extended graph features	Gradient Boosting	P@K
Yelp	9,464 (users) 11,197 (businesses)	Bipartite + Bipartite with metadata	2	13	Basic graph features, manually engineered	Random Forest	RMSE
Yelp II	13,366 (users) 14,853 (businesses)	Bipartite + Non-bipartite (w/ social links), with and without metadeta	8	13	Basic graph features	Random Forest, Gradient Boosting, Support Vector Machine	RMSE
Movielens	6,040 (users) 3,883 (movies)	Bipartite + Bipartite with metadata	32	36	Basic graph features	Random Forest, Gradient Boosting	RMSE, MAE

Table 1.1 Summary of datasets characteristics

1.4.5 Summary of the datasets, features, and metrics

Table 1.1 summarizes this section and presents the experimental datasets, number of source and target entities, various sub-graph schemes investigated, number of extracted feature sets, groups of features, and evaluation metrics exploited. The datasets contain large numbers of users and items and cover a broad range of data types, application domains, and recommendation tasks. The datasets also contain both legacy and recently collected datasets, such that the evaluation presented in the following section offers solid empirical validity.

1.5 Results and Analysis

1.5.1 Case Study I: Overall Contribution of the Graph-based Approach

This case study answers the broad question: *How does the use of graph features affect the performance of rating predictions and recommendation generation in different domains and tasks?* Each of the above datasets was represented by graphs and graph-based features were extracted from the graphs using the approach detailed in Section 1.3. For each dataset, a matching recommendation task was defined as follows: for the Last.fm dataset the task was to predict the artists to which users will listen; for the two Yelp datasets the task was to predict user ratings for business; and, for the Movielens dataset, to predict user ratings for movies. The tasks were performed and evaluated under three conditions:

- Prediction with versus without the newly extracted graph features
- Prediction with user-related versus item-related graph features
- Prediction using features of a bipartite graph versus extended graph schemes.

The evaluations were conducted using the N-fold cross validation [40], with $N = 5$. For each fold, the complete graph representation was generated based on the entities from both the training and test sets, except for the relationships being predicted in the test set. A two-sided t-test was conducted with the null hypothesis of having identical expected values across the compared prediction sets. The tests assumed that the ratings using feature sets A and B were taken from the same population. The threshold used for a statistically significant difference was p=0.05.

Feature Set	Precision@10
Baseline	0.336
Baseline + Tags	0.548
Baseline + Friends	0.555
Baseline + Tags + Friends	0.571

Table 1.2 Precision of feature combinations using the four graphs - Last.fm dataset.

1.5.1.1 Dataset I – Last.fm Results

Four graph schemes were generated for the Last.fm dataset, as per the structure in Table 1.2. For each graph scheme the set of *basic* graph features listed in Section 1.3.3.1 was extracted and populated. The basic features encapsulate only the user-artist listening data and denoted by $\hat{F}$. In addition, when the social and tagging data is available, namely, in the BL+T, BL+F, BL+T+F schemes, the set of *extended* features can be extracted. These features are denoted by F, e.g., F_{BL+T} denotes the set of extended features extracted from the graph with the tagging data. Note that for the BL schema in Figure 1.5, having neither social nor tagging data, the basic and extended feature sets are identical, i.e., $F_{BL} = \hat{F}_{BL}$.

The average P@10 results obtained for the extended feature sets extracted from the four schemes are summarized in Table 1.2. The baseline for comparison in this case is the performance of the graph features extracted from the bipartite scheme F_{BL}, which scored P@10=0.336. A notable improvement, between 63% and 70%, was observed when the extended feature sets were extracted. For instance, F_{BL+F}, scored P@10=0.555, which is an improvement of more than 65%. A combination of the extended features using the graph that includes both social tags and friendships, F_{BL+T+F} is the best performing feature. This scored the highest P@10=0.571 and improved the baseline P@10 by as much as almost 70%.

In order to evaluate the significance of the results, a paired t-test was performed with each group of features, using the P@10 values obtained for each of the four graphs. The results show that among the extended feature sets, all the differences were significant, $p<0.05$. Thus, the inclusion of auxiliary tagging and friendship data improved the accuracy of the predictions, while their combination led to the most accurate predictions. More importantly, the extraction of graph-based features was shown to consistently and significantly boost the performance of the recommender, in comparison to the variant not using the extracted features.

1.5.1.2 Dataset II – Yelp Results

Improvements due to the use of the graph-based approach were also evident in experiments using the second dataset (Yelp). As per the description in Section 1.4.2, basic (user and business identifiers), manual (number of reviews, average rating, business categoriy and location), and graph-based features were extracted and populated. The latter were broken down into the bipartite and tripartite features. In this dataset, the

	Features combination	Features	RMSE	Improvement
1	All_Features	Basic∪Manual∪Graph	1.0766	8.82%
2	AllExcept_Tripartite	Basic∪Manual∪Bipartite	1.0775	8.75%
3	AllExcept_Basic	Manual∪Graph	1.0822	8.35%
4	Manual_and_Bipartite	Manual∪Bipartite	1.0850	8.11%
5	AllExcept_Bipartite	Basic∪Manual∪Tripartite	1.0896	7.72%
6	Manual_and_tripartite	Manual∪Tripartite	1.1073	6.22%
7	AllExcept_Manual	Basic∪Graph	1.1095	6.04%
8	All_Graph	Bipartite∪Tripartite	1.1148	5.59%
9	AllExcept_Graph	Basic∪Manual	1.1175	5.36%
10	Bipartite		1.1188	5.25%
11	Tripartite		1.1326	4.09%
12	Basic		1.1809	N/A
13	Manual		1.1853	-0.37%

Table 1.3 RMSE of selected feature combinations - Yelp dataset (baseline combination in light gray).

performance of the basic features related to the user-to-business associations was the baseline. Table 1.3 presents the full results for all the feature combinations.

The largest improvement in the RMSE of business ratings prediction was an 8.82% decrease obtained for the combination of graph features with basic and manually engineered ones (row 1). The similarity of the RMSE scores obtained by the various combinations is explained primarily by the low variance of user ratings in the dataset. Since most ratings are similar, they are highly predictable using simple methods and there is only a limited space for improvement. A combination containing only the graph features (row 5) outperformed the baseline performance by 5.59%. On the contrary, the use of manual features (row 13) slightly deteriorated the accuracy of the predictions. This demonstrates the full benefit of the graph-based approach: extracting the graph features took less time than crafting the manual ones, and the graph features also outperformed the manual ones.

An examination of the differences in the accuracy of the results obtained when combining various groups of features revealed a number of findings. An analysis of the performance of each group of features shows that the bipartite and tripartite features performed noticeably better than the basic and manual feature sets (rows 10 and 11 versus rows 12 and 13). A combination of graph features (row 8) still outperforms slightly, although significantly, the combination of the basic and manually engineered features (row 9). To analyze the impact of the feature groups, each group was excluded from the overall set of features and the change with respect to the All_Features combination (row 1) was measured. When the graph features were excluded (row 9), the predictions were less accurate than when the basic (row 3) or manual features (row 7) were excluded. This indicates that the graph features provide the most valuable information, not covered by the basic and manual features.

1.5.1.3 Dataset III – Yelp II (with social links) Results

The results of the evaluation using the extended Yelp II dataset that includes social links between users are in line with the results of the original Yelp dataset. Table 1.4 lists the results of this evaluation for a selected set of feature combinations: basic user

	Features Subset	RMSE	Improvement
1	All Features	1.1416	1.73%
2	All Graph Features	1.1417	1.73%
3	Complete Graph	1.1450	1.45%
4	Business Features	1.1465	1.32%
5	User Features	1.1580	0.33%
6	Basic Features	1.1619	N/A

Table 1.4 RMSE of selected feature combinations - Yelp II dataset (baseline combination in light gray).

and business features, feature of the complete graph, features of all the sub-graphs, and the union of all the available features.

The results show that the combinations including graph features generally outperform the basic feature sets. The best performing combination of graph-based features only, using all the features from all the sub-graph schemes (row 2, RMSE=1.1417), achieves a 1.73% improvement over the baselines. When adding the basic features to all the graph-based features, a slightly lower RMSE=1.1416 (row 1) is obtained. Another noticeable difference is between the business-related features, which achieve RMSE=1.1465 and the user-related features, which achieve RMSE=1.158 (rows 4 and 5, respectively). This intuitively indicates that the predicted ratings assigned to the businesses being predicted are more informative than the ratings of the target user. Again, the achieved improvements are generally modest, primarily due to the low variance of ratings in the Yelp II dataset.

The performance differences between the evaluated combinations are mostly significant, $p<0.01$, except for two pairs of feature sets. The difference between business-related features and complete graph features is borderline, with $p=0.07$. Also the difference between 'All Features' and 'All Graph Features' is expectedly insignificant. This shows that the most important contribution to the predictive accuracy comes from the graph features, while the addition of the basic features improves the prediction only a little.

1.5.1.4 Dataset IV – Movielens Results

Finally, the experimentation with the Movielens dataset re-affirms the contribution of graph-based feature extraction to the recommendation generation. The task in this dataset was to predict movie ratings, whereas the predictions were evaluated using the MAE and RMSE predictive accuracy metrics. Table 1.5 summarizes the perofrmance of a selected group of features. The basic user-item pairs are compared here with the user and item features used individually, all the extracted graph-based features, and the union of all of them, denoted by 'All Features' (row 1).

The already discussed superiority of item features over user features (row 3 versus row 5) can be clearly seen again. In this case, the former improve the accuracy of the predictions by 3-5%, while the latter only deteriorate it. The extraction of the graph-based features (row 2) also leads to an improvement of 3.36% and 5.53% relative to the baseline, using the RMSE and MAE metrics, respectively. When

	Features Subset	RMSE	Improvement	MAE	Improvement
1	All Features	1.0272	4.20%	0.8303	6.06%
2	All Graph Features	1.0362	3.36%	0.8349	5.53%
3	Movie Features	1.0400	3.01%	0.8380	5.18%
4	Basic Features	1.0722	N/A	0.8838	N/A
5	User Features	1.0895	-1.61%	0.8967	-1.46%

Table 1.5 Performance of selected features combinations - Movielens dataset (baseline combination in light gray, rows are sorted by RMSE).

				Results		
	Dataset	Metric	Method	Baseline	Graph Features	Improvement
1	Yelp I	RMSE	Random Forest	1.1809	1.1148	5.59%
2	Yelp II	RMSE	Random Forest	1.1619	1.1417	1.74%
3	Yelp II	RMSE	Gradient Boosting	1.2480	1.1715	6.13%
4	Yelp II	RMSE	SVM	1.1818	1.1783	0.30%
5	Movielens	RMSE	Random Forest	1.1667	1.0268	11.90%
6	Movielens	RMSE	Gradient Boosting	1.0722	1.0362	3.36%
7	Movielens	MAE	Random Forest	0.9144	0.8157	10.79%
8	Movielens	MAE	Gradient Boosting	0.8838	0.8349	5.53%

Table 1.6 Summary of experiments and results for Case Study I.

combined with other features, the graph features achieve the best result, which is RMSE=1.0272, or a 4.20% improvement over the baseline. Those performance differences were statistically evaluated and found significant.

1.5.1.5 Performance across Learning Methods, Datasets, and Metrics

This case investigated the the impact of the graph-based features affect on the accuracy of the recommendations. Although all the evaluations reported so far show that using the graph-based features improves the accuracy of the recommendations, the results cannot be fully corroborated yet, as the conducted experiments use different learning methods, datasets, and evaluation metrics (see Table 1.1). To confidently address the resarch question, the design of the evaluation has overlaps in these factors, so that the contribution of the graph features can be singled out.

The analysis below aims to establish whether the observed improvements should be attributed to the information contributed by the graph features or to the differences in the experimental settings, i.e., learning method, dataset, and metric.The results of all the experiments are summarized in Table 1.6. In all the cases, the performance of the baseline approaches not using the graph features, which were highlighted in light gray in all the tables, is compared to the performance of all the graph-based based features, i.e, row 8 in Table 1.3, row 2 in Table 1.4, and row 2 in Table 1.5.

Included are the results of experiments using the Yelp, Yelp II, and Movielens datasets, which were discussed in sections 1.5.1.2, 1.5.1.3, and 1.5.1.4. That said, results in rows 3, 4, 5, and 7 of Table 1.6 are presented here for the first time. This is due to the fact that previously reported Yelp experiments (both datasets) used Random Forest as their learning method, while the Movielens experiments used Graduate Boosting. Here, new Yelp II results with Gradient Boosting and SVM, and new Movielens results with Random Forest are also presented. Experiments using the Last.fm dataset are excluded from the analysis, since they use classification accuracy metrics and differ both in the dataset and evaluation metric.

In order to demonstrate that the improvement is not due to the selected dataset, the metric and learning method were fixed, while the approaches using different datasets were compared. Two evaluations sets are applicable to this scenario: (1) Random Forest predictions evaluated with the RMSE metric, using the Yelp, Yelp II, and Movielens datasets (rows 1, 2, and 5), and (2) Gradient Boosting predictions also evaluated with RMSE, but using the Yelp II and Movielens datasets (rows 3 and 6). The results of these experiments show an improvement of 1.74% to 11.90%, which allows to eliminate the selected dataset as a possible reason for improvement.

To demonstrate that the improvement is also not due to the selected machine learning method, the dataset and metric were fixed, while the approaches using different learning methods were compared. Three evaluation sets are applicable to this scenario: (1) RMSE of business predictions using the Yelp II dataset, where the learning methods are Random Forest, Gradient Boosting, and SVM (rows 2, 3, and 4), (2) RMSE of movie rating predictions using the Movielens dataset, where the methods are Random Forest and Gradient Boosting (rows 5 and 6), and (3) MAE of movie rating predictions using the the Movielens dataset, where the methods are Random Forest and Gradient Boosting (rows 7 and 8). The results of these experiments show an improvement across all experiments, ranging from 0.30% to 6.13% for the Yelp II dataset, and from from 3.36% to 11.90% for the Movielens dataset. The low variance of ratings in the Yelp datasets, which was discussed earlier, is the main reason for the low improvement observed. This is particularly noticeable with the SVM method, which struggles to linearly separate businesses with moderate ratings. Thus, the learning method cannot be the reason for the accuracy improvement.

Finally, to demonstrate that the improvement is not due to the selected evaluation metric, the dataset and method were fixed, while the performance of approaches using different metrics was compated. Two evaluation sets are applicable to this scenario: (1) Random Forest movie rating predictions using the Movielens dataset, evaluated using RMSE and MAE (rows 5 and 7), and (2) Gradient Boosting movie rating predictions also using the Movielens dataset, and also evaluated using the RMSE and MAE metrics (rows 6 and 8). The results of these experiments show a clear improvement across, ranging from 3.36% to 11.90%, allowing to eliminate the selected evaluation metric as a possible reason for improvement.

Summing up this causal analysis, all three hypotheses that the improved performance is driven by the differences in the experimental settings (dataset, learning method, and evaluation metric) were rejected. Thus, it can be concluded that the reason for the observed improvement lies in the inclusion of graph-based features, contributing new information to the recommendation process.

Feature Set	Basic P@10	Extended P@10
Baseline	0.336	N/A
Baseline + Tags	0.548	0.498
Baseline + Friends	0.555	0.497
Baseline + Tags + Friends	0.571	0.444

Table 1.7 Precision of feature combinations using the four graphs - Last.fm dataset.

1.5.2 Case Study II: Different Graph Schemes and their Impact on Recommendations

As mentioned in the Section 1.3, various sub-graphs and graph schemes can be generated for each dataset. The feature extraction process will, thus, yield a number of graph schemes, corresponding feature sets, and even the values of the same graph features. This leads to to the second research question: *How are the recommendations affected by the sub-graph and its representation used to generate the graph features?* In order to answer this question, another set of experiments was conducted.

In these experiments, the accuracy of recommendations when using various graph schemes was evaluated using four datasets: Last.fm, both Yelp datasets, and Movielens. The recommendation tasks were identical to the previous experiments, i.e., to predict listened artists in the Last.fm dataset, user ratings for businesses in the two Yelp datasets and user ratings for movies in the Movielens dataset. An N-fold cross validation methodology similar to the one reported in Section 1.5.1 was followed. Also, two-sided t-test statistical significance testing was carried out.

1.5.2.1 Dataset I – Last.fm Results

The evaluations using the Last.fm dataset focused on the influence of the social elements, i.e., friendship links and tags, on the obtained recommendation accuracy. In this dataset, the results of recommendations based on the bipartite user-artist graph representation (BL in Figure 1.5) were compared with those of three non-bipartite schemes, BL+T, BL+F, and BL+T+F, including, respectively, the tags assigned by the users to the artists, social friendship links between the users, and tags and friendship links alike. As mentioned in Section 1.5.1.1, two sets of graph features were extracted for each schema: a set of basic features $\hat{F}$ and a set of extended features F. Although the basic feature set $\hat{F}$ is shared across all the schemes, their values may change due to the presence of additional graph nodes. The extended feature set F is composed of the basic features along with new features that were extracted from the social links and tags available in each schema.

Table 1.7 shows the obtained P@10 scores averaged over all the users in the test set, when using both the basic and extended feature sets. First, it can clearly be observed that the inclusion of the social auxiliary data of either the assigned tags or friendships links substantially improves P@10. When both the tags and friendship links are included in the BL+T+F model, the highest average P@10 is observed. Both in the basic and the extended feature sets, the BL+T and BL+F models obtain comparable P@10 scores, showing the effect of the inclusion of auxiliary data in the graph schemes. However, as noted in Section 1.4.1, the tag data includes more than 186K tag assignments, whereas the friendship data consists of only 12K user-to-user links. Since the obtained precision scores are comparable, a single friendship link is more influential than a single artist tag and yields a greater improvement in the recommendation accuracy. Looking at the significance tests conducted within the basic and extended feature sets, significant differences, $p<0.05$, were observed between all the pairs of extended features and all the pairs of basic features except for the $\hat{F}_{BL+T}$ and $\hat{F}_{BL+F}$ pair.

When comparing the performance of the extended graph features to the performance of the corresponding basic features (Basic versus Extended columns in Table 1.7, it can be seen that the extended sets consistently outperformed the basic sets across all the four graph schemes, and the difference within the pairs was statistically significant, $p<0.05$. In the BL+T scheme, the extended graph features from improved on the basic features extracted from it by 10%, P@10=0.548 versus P@10=0.498, while in the BL+F scheme the improvement was by 11.6%, P@10=0.555 versus P@10=0.497. The largest improvement was noted in the BL+T+F scheme, where the extended graph features outperformed the basic features by as much as 28.6%, P@10=0.571 versus P@10=0.444. Surprisingly, when the basic feature set $\hat{F}_{BL+T+F}$ set was found achieve a lower P@10 than $\hat{F}_{BL+T}$ and $\hat{F}_{BL+F}$. A possible explanation for this can be that including both types of social data but not extracting and populating the extended features leads to redundancy in the graph and degrades the performance of the recommender.

	Features combination	Features	RMSE	Improvement
8	All_Graph	Bipartite∪Tripartite	1.1148	5.59%
10	Bipartite		1.1188	5.25%
11	Tripartite		1.1326	4.09%
12	Basic		1.1809	N/A

Table 1.8 Yelp results: RMSE of the bipartite versus the tripartite feature sets.

1.5.2.2 Dataset II – Yelp Results

For the Yelp dataset and the task of business rating prediction, two graph schemes were compared: a pure bipartite graph that contained only the users and businesses, and a tripartite graph that, on top of user and business nodes, also contained metadata nodes describing the businesses. The two graph schemes are illustrated in Figure 1.6. The reason these were the only graph schemes created is that sparse features having a small number of unique features, were filtered from the dataset. These features would have resulted in most nodes of a group, e.g., users, being connected to a single node, which would render it meaningless. For example, adding three "gender" nodes, male, female, and unspecified, would have resulted in all users being connected to either one of the three, essentially creating three large clusters in the graph.

The complete set of graph features was generated for both the bipartite and tripartite representations. The results in Table 1.8 show the RMSE scores obtained for these feature sets, as extracted from Table 1.3 (original row numbers are preserved).

	Features Subset	RMSE	Improvement
2	All Graph Features	1.1417	1.73%
7	Without Name Links	1.1450	1.45%
3	Complete Graph	1.1450	1.45%
8	Without Social Links	1.1463	1.33%
9	Without Social and Name Links	1.1465	1.32%
10	Without Category Links	1.1508	0.95%
11	Without Metadata	1.1508	0.94%
12	Without Social and Category Links	1.1519	0.85%
13	Without Metadata and Social Links	1.1523	0.82%
6	Basic Features	1.1619	N/A

Table 1.9 Yelp II results: RMSE of various sub-graph feature sets.

The experiments showed that the bipartite schema, not including the metadata nodes, performed slightly but significantly better than the tripartite schema with metadata, RMSE=1.1188 versus RMSE=1.1326. The relative improvement with respect to the baseline recommendations was 1.16% higher. This difference in the performance of the schemas led to their unified feature set, which is the All_Graph, to outperform the two feature sets individually.

1.5.2.3 Dataset III – Yelp II (with social links) Results

The richer information provided by the Yelp II datasets allowed for the creation of a larger set of sub-graphs. These are illustrated in Figure 1.8, where various combinations of entities are removed from the complete graph. Thus, in addition to the complete graph, seven sub-graph representations can be created and the performance of the feature sets extracted from these can be compared. The results of this experiment are presented in Table 1.9. The complete graph and the seven sub-graphs are compared to the basic feature set and the union of all the graph features, which were, respectively, the baseline and best performing combination in Table 1.4. The numbering of rows already presented in Table 1.4 is preserved (rows 2, 3, and 6), while the rows of all the sub-graphs from Figure 1.8 are numbered 7 to 13.

1.5.2.4 Dataset IV – Movielens Results

The Movielens dataset offered an even richer information about users and items and allowed for the extraction of 32 sub-graph schemes. Only a small sample of these is illustrated in Figure 1.10. The MAE and RMSE scores obtained for the 32 sub-graphs are listed in Table 1.10. The sub-graphs are compared to the basic feature set and the union of all the graph features, which were presented in Table 1.5 (rows numbered 2 and 4). The rows corresponding to the various sub-graph representations are numbered 6 to 36. For the sake of clarity, the sub-graphs are denoted by the entity types *included* rather than excluded. For example, “graph w/[Age, Genre, Zip]” denotes the sub-graph with the ‘Age’, ‘Genre’, and ‘Zip’ entities, which is identical to the complete graph with the ‘Occupation’ and ‘Gender’ entities excluded.

As can be seen, the ‘genre’ relationship in Movielens sub-graphs plays a similar role to the “category” relationship in Yelp. Sub-graphs containing this relationship (rows 6 to 22) outperformed those, where it was excluded (rows 23 to 36). A com-

	Features Set	RMSE	Improvement	MAE	Improvement
2	All Graph Features	1.0362	3.36%	0.8349	5.53%
6	graph w/[Age, Gender, Genre, Zip]	1.0369	3.29%	0.8353	5.48%
7	graph w/[Age, Gender, Occupation, Zip]	1.0373	3.25%	0.8357	5.44%
8	graph w/[Gender, Genre, Occupation, Zip]	1.0384	3.16%	0.8365	5.35%
9	graph w/[Genre, Occupation]	1.0410	2.91%	0.8391	5.06%
10	graph w/[Age, Genre, Zip]	1.0411	2.90%	0.8386	5.12%
11	graph w/[Genre, Occupation, Zip]	1.0411	2.90%	0.8385	5.12%
12	graph w/[Age, Genre, Occupation]	1.0412	2.90%	0.8390	5.07%
13	graph w/[Age, Gender, Genre]	1.0412	2.89%	0.8392	5.05%
14	graph w/[Age, Gender, Genre, Occupation]	1.0413	2.89%	0.8390	5.07%
15	graph w/[Age, Genre, Occupation, Zip]	1.0413	2.89%	0.8388	5.10%
16	graph w/[Genre]	1.0413	2.89%	0.8393	5.04%
17	graph w/[Gender, Genre, Occupation]	1.0413	2.88%	0.8392	5.04%
18	graph w/[Age, Genre]	1.0414	2.88%	0.8395	5.02%
19	graph w/[Age, Gender, Genre, Occupation, Zip]	1.0414	2.88%	0.8390	5.08%
20	graph w/[Genre, Zip]	1.0415	2.87%	0.8388	5.09%
21	graph w/[Gender, Genre]	1.0416	2.86%	0.8396	5.00%
22	graph w/[Gender, Genre, Zip]	1.0416	2.85%	0.8391	5.06%
23	graph w/[Age]	1.0425	2.77%	0.8413	4.81%
24	graph w/[Zip]	1.0426	2.77%	0.8407	4.88%
25	graph w/[Age, Zip]	1.0426	2.76%	0.8409	4.86%
26	graph w/[Age, Occupation]	1.0426	2.76%	0.8412	4.82%
27	graph w/[Age, Gender]	1.0427	2.76%	0.8413	4.81%
28	graph w/[Age, Gender, Zip]	1.0427	2.75%	0.8408	4.87%
29	graph w/[Age, Occupation, Zip]	1.0427	2.75%	0.8408	4.86%
30	graph w/[Occupation, Zip]	1.0427	2.75%	0.8410	4.84%
31	graph w/[Occupation]	1.0428	2.75%	0.8414	4.80%
32	graph w/[Gender, Occupation, Zip]	1.0428	2.74%	0.8409	4.85%
33	graph w/[Gender, Zip]	1.0431	2.72%	0.8411	4.83%
34	graph w/[Gender]	1.0431	2.71%	0.8418	4.75%
35	graph w/[Gender, Occupation]	1.0432	2.70%	0.8417	4.77%
36	graph w/[Age, Gender, Occupation]	1.0433	2.70%	0.8417	4.77%
4	Basic Features	1.0722	N/A	0.8838	N/A

Table 1.10 Performance of selected features combinations - Movielens dataset (baseline combination in light gray, rows are sorted by RMSE).

mon link between the ‘category’ relationship in Yelp II and the ‘genre’ relationship in MovieLens is that they both divide the item space – be it businesses or movies – into connected groups, which affects values of the item features. In agreement with previous results, the feature set that unifies all the graph features from all the sub-graph schemes (“All Graph Features”, row 2) achieves the highest accuracy and outperforms any other feature set. Again, this is attributed to the broad coverage of the proposed feature extraction mechanism, which produces and aggregates promising feature combinations.

1.5.2.5 Summary

The purpose of this analysis was to analyze the differences driven by the sub-graphs that are used for the feature extraction. To recap the results obtained using the four datasets, the following was established.

- Features extracted from different graph schemes performed differently, not following a certain pattern tied to the entities or relationships included in the sub-graph. This means that it was not possible to conclude which relationships lead to better results if included in the graph. We posit that this is dataset-specific and may be affected by additional factors, such as density of a specific feature, distribution of its values, domain-specific considerations, and so forth. This finding comes through in the 'category' and 'genre' relationships in the Yelp II and Movielens datasets, but not in the Yelp I dataset. Notably, the social links had a major contribution in the Last.fm dataset, but not in the Yelp II dataset, possibly due to the sparsity of the latter.

- Features extracted from the complete graph representations, i.e., those containing all the relationships and entities in the dataset, were not necessarily the best performing feature sets. A negative example can be seen in the basic features of the BL+F+T schema in Table 1.7 that are dominated by the basic feature of BL+F and BL+T alike. Having said that, the feature set that aggregated (that is, unified) all the graph features from all the sub-graph schemes performed the best in the other three scenarios in which it was evaluated: Yelp, Yelp II, and Movielens. We consider this to be a strong argument in favor of using the proposed approach, as its exhaustive nature allows to cover a range of features and determine the most informative ones, as well as their best combination.

The differences across the obtained results do not allow to generalize and determine a priori the best performing sub-graph and feature set. Due to this, the suggested approach of generating sub-graphs, populating features from each of them, and then aggregating the features in the feature sets is more likely to uncover the best performing feature combination. We believe that future research may unveil rating patterns or characteristics of datasets, which may predict the contribution of certain sub-graph, data entities, or even types of features.

1.6 Discussion and Conclusions

1.6.1 Discussion

This work demonstrated the effectiveness of the graph-based approach for improving recommendations. It has been shown that precision and accuracy gains can be achieved by representing tabular data by graphs and extracting new features from them. This contrasts and complements prior approaches that improved recommendations by enhancing the recommendation techniques themselves. Also established are the benefits of the graph-based approach across recommendation domains, tasks, and metrics. These findings show that the graph representation exploits indirect latent links in the data, which lead to an improved recommendation accuracy. Finally, the approach is generic and can be applied to many recommender system datasets.

The process is automatic and can be run end-to-end without human intervention, unlike manual feature extraction methods, which are often time consuming and require domain expertise. Using the proposed approach, rich features, based on intricate relationships between various data entities and sub-graph scheme variations, can be systematically extracted from a dataset. This allows for a better coverage of the features space with a considerable lower effort, as discussed in Section 1.3. Next, we discuss some limitations of our work.

Overfitting

Regarding concerns referring to possible overfitting due to the newly generated features, as long as the volume of available data greatly exceeds the number of extracted features, there is little risk that the features will be the cause of overfitting. The high diversity of unique data characteristics can hardly be captured in full by a smaller subset of features. Recommender system datasets tend to be in the medium to large scale (tens of thousands to millions of data points), while the number of features generated by the current approach is still in the scale of tens to hundreds.

Additionally, machine learning methods such as Random Forests have internal mechanisms for feature selection and can filter out features that overfit. They do so by training on a sample of the dataset and evaluating the performance of the features on the rest of the data. A feature that performs well on the sample but underperforms on the test data is ranked low. In the evaluations, cross validation was used with at least N=5 folds, showing that the models and features on which they are built are generalizable. Moreover, it was shown that in cases of sparse data, which require a higher degree of generalization, the graph features still outperformed other features.

Scalability

A possible disadvantage of the proposed approach is that some graph-based computations, e.g., PageRank, are iterative and may take a long time to converge. In the age of Big Data, recommender system datasets are getting larger and so is their graph representation, which may lead to a computational issue. A general approach for handling this in a deployed system would be to extract the graph-based features offline and use the pre-computed values for real-time predictions. This may resolve the problem under the reasonable assumption that the values do not change substantially. Another means to overcome the computational latency is through using a graph feature computation library, e.g., Okapi, which can use distributed tools in order extract the graph features.

Another factor that adds to the computational complexity of the approach is the exhaustive search for new features. It should be noted that the complexity of the process of generating every possible sub-graph and populating the matching feature combinations is exponential. The number of relationships in current recommendation datasets (as surveyed in Section 1.3.1) is still manageable, and can be accommodated by the proposed approach. However in the future, with additional data sources being integrated for recommendation purposes, this might become unsustainable and will require a long-term solution. Two possible approaches for handling this issue are parallelization, e.g., each sub-graph being processed by a different machine, and heuristics for pruning less relevant sub-graph representations.

Initial Transition to the Graph Model

Another possible disadvantage of graph-based features is the possible need for human intervention when generating the initial complete graph. Non-categorical feature values, e.g., income or price, may generate a large number of vertices, which would lead to a low connectivity of the graph, since not many users or items would share the exact value of the feature. This would lead to a very sparse graph and will need to be addressed by a manual intervention by a domain expert, who can determine how the non-categorical values can be grouped and categorized, e.g., by creating appropriate income or price buckets. A naive solution for this might be to attempt to auto-categorize such features based on the observed distribution of their values, e.g., first quarter, second quarter, and so on. This may, however, mask the differences between fine-grained groups and cause information loss.

Also to be acknowledged in this context is the historic human contribution that was required in order to conceive the graph methods exploited in this work for the generation of the various basic graph features: shortest path, degree, PageRank, etc. Indeed, these methods took a considerable amount of time and effort to evolve; however, they are reusable for generations and the overheads related to their development have been shared across many subsequent applications, while manually engineered features would usually not be highly reusable. Overall, when weighting the ease, quantity, and the possible contribution of the graph-based features to the accuracy of the generated recommendations against the above mentioned disadvantages, it can be concluded that it is worth to generate and populate such features, when designing a recommendation engine.

1.6.2 Conclusions and Future Work

In this work, a new approach for improving recommendations was presented and evaluated. Unlike many previous works, which focused on addressing the recommendation problem by making improvements to the recommendation algorithms, the presented approach does so by suggesting a different way of looking at the dataset used for recommendation. It proposed representing the datasets using graphs and then to extract and populate new features from those graphs, all in a systematic fashion, and feed the new features into existing recommendation algorithms. New features and relationships that were not visible in the original tabular form can be thus uncovered. In this manner, applying this approach may compliment classical recommendation approaches and further enhance them.

The methodology, implementation, and analysis of the approach were described in detail and the approach was evaluated from two main perspectives: the overall contribution to recommendations and the impact of various graph representations. The evaluation encompassed a number of datasets, recommendation tasks, and evaluation metrics. Furthermore, the datasets belonged to four application domains (movies, music, businesses, and personal interests) that in part included metadata and in part included social links. The recommendation tasks varied from binary link predictions to star rating predictions. A number of state-of-the-art classifiers and regressors were used for the generation of the predictions. All in all, the presented evaluations examined the impact of the graph representations and showed that the approach had a profound effect on the accuracy of the recommendations.

The graph-based representation and features were shown to lead to the generation of more accurate recommendations. The variations in performance across various graph schemes and the justification for systematically extracting them, due to that, was established. The approach presented was implemented in a library and is being provided as open source software for the community to use and build on-top. Given such a library, the cost of generating additional features that can improve recommendations becomes substantially lower, in terms of computation time and effort. It can be adopted as a natural first resort, when given a dataset and recommendation task, or as a complementary aid to enhance the standard manual feature engineering.

The conducted evaluations demonstrated the potential of the proposed approach in improving the recommendations by exploiting the benefits of links between entities and characteristics of entities extracted from the graph representations. Therefore, this work lays the foundations for further exploring how graph-based features can enhance recommender systems and automatic feature engineering in the more general context. Several variables were investigated in this work but many more require additional attention. The following paragraphs identify several directions of exploration, which were identified as possible research directions in future works.

- Temporal Aspects. Given a dataset that includs dated actions that are not sparse, the time aspect can be used to build a different type of graphs. Each graph will represent a snapshot in time and will either contain or exclude a link between vertices based on whether it was available in the dataset at that time. A combination of two temporally adjacent graphs will reflect the evolution of the data over that period of time. The main question in this setting is how such temporal graphs will affect the values of features extracted from them and how a recommender systems that use these features will perform in their respective recommendation tasks.
- Weighted and Labeled Graphs. Several features in a dataset can be used to populate the edge labels when constructing the graph based representation. The labels, once set, can be taken into consideration in some graph features being extracted. One example would be to calculate a weighted PageRank score that will have jumps from a vertex to its neighbors based on a skewed probability correlated with the weight on the edge linking to the neighbor. This could lead to further improvement in the recommendations; however, this requires fine-tuning of initial weights on edges that do not naturally have them, e.g., social relationship edges in the Last.fm dataset.
- Directed Graphs. Similarly, in cases where the direction of the edges can be important, the process can be extended to include this aspect by generating additional graph representations, with various combinations of the edge directions. For example, in one variant, edges will be directed from the source vertex to the target vertex, in another, in the opposite direction, and in a third one there will be no direction. This will guarantee coverage in terms of expressing the direction of the edges, and the performance of the features in the various scenarios can be evaluated.

The effects of these modifications on the scalability of the approach can be handled using the previously suggested methods, either by scaling the computations (e.g., computing the features of each subgraph in a separate process), or using distributed graph computation libraries, or identifying heuristics for pruning the feature and subgraph space.

References

[1] Adomavicius, G., Tuzhilin, A.: Toward the next generation of recommender systems: A survey of the state-of-the-art and possible extensions. Knowledge and Data Engineering, IEEE Transactions on **17**(6), 734–749 (2005)

[2] Amatriain, X., Jaimes, A., Oliver, N., Pujol, J.M.: Data mining methods for recommender systems. In: Recommender Systems Handbook, pp. 39–71. Springer (2011)

[3] Bae, D., Han, K., Park, J., Yi, M.Y.: Apptrends: A graph-based mobile app recommendation system using usage history. In: International Conference on Big Data and Smart Computing, BIGCOMP, pp. 210–216 (2015)

[4] Barrat, A., Barthelemy, M., Pastor-Satorras, R., Vespignani, A.: The Architecture of Complex Weighted Networks. PNAS (2004)

[5] Bastian, M., Heymann, S., Jacomy, M., et al.: Gephi: an open source software for exploring and manipulating networks. ICWSM **8**, 361–362 (2009)

[6] Batarfi, O., Shawi, R.E., Fayoumi, A.G., Nouri, R., Beheshti, S., Barnawi, A., Sakr, S.: Large scale graph processing systems: survey and an experimental evaluation. Cluster Computing **18**(3), 1189–1213 (2015)

[7] Bellogín, A., Cantador, I., Díez, F., Castells, P., Chavarriaga, E.: An empirical comparison of social, collaborative filtering, and hybrid recommenders. ACM Transactions on Intelligent Systems and Technology **4**(1), 14 (2013)

[8] Bennett, J., Lanning, S.: The netflix prize. In: Proceedings of KDD cup and workshop, vol. 2007, p. 35 (2007)

[9] Berge, C., Minieka, E.: Graphs and hypergraphs, vol. 7. North-Holland publishing company Amsterdam (1973)

[10] Berkovsky, S., Kuflik, T., Ricci, F.: Mediation of user models for enhanced personalization in recommender systems. User Modeling and User-Adapted Interaction **18**(3), 245–286 (2008)

[11] Berkovsky, S., Kuflik, T., Ricci, F.: The impact of data obfuscation on the accuracy of collaborative filtering. Expert Systems with Applications **39**(5), 5033–5042 (2012)

[12] Blomo, J., Ester, M., Field, M.: Recsys challenge 2013. In: Proceedings of the ACM conference on Recommender systems, pp. 489–490. ACM (2013)

[13] Borgatti, S.P., Halgin, D.S.: Analyzing Affiliation Networks. The Sage handbook of social network analysis (2011)

[14] Breiman, L.: Random forests. Machine learning **45**(1), 5–32 (2001)

[15] Bu, J., Tan, S., Chen, C., Wang, C., Wu, H., Zhang, L., He, X.: Music recommendation by unified hypergraph: combining social media information and music content. In: Proceedings of the international conference on Multime-

dia, pp. 391–400. ACM (2010)

[16] Burke, R.: Hybrid web recommender systems. In: The adaptive web, pp. 377–408. Springer (2007)

[17] Cantador, I., Brusilovsky, P., Kuflik, T.: Second workshop on information heterogeneity and fusion in recommender systems. In: RecSys (2011)

[18] Chianese, A., Piccialli, F.: A smart system to manage the context evolution in the cultural heritage domain. Computers & Electrical Engineering (2016)

[19] Coffman, T., Greenblatt, S., Marcus, S.: Graph-based technologies for intelligence analysis. Communications of the ACM **47**(3), 45–47 (2004)

[20] Cordobés, H., Chiroque, L.F., Anta, A.F., Leiva, R.A.G., Morere, P., Ornella, L., Pérez, F., Santos, A.: Empirical comparison of graph-based recommendation engines for an apps ecosystem. IJIMAI **3**(2), 33–39 (2015)

[21] Costa, L.d.F., Rodrigues, F.A., Travieso, G., Villas Boas, P.R.: Characterization of complex networks: A survey of measurements. Advances in Physics **56**(1), 167–242 (2007)

[22] Csardi, G., Nepusz, T.: The igraph software package for complex network research. InterJournal, Complex Systems **1695**(5) (2006)

[23] Desrosiers, C., Karypis, G.: A comprehensive survey of neighborhood-based recommendation methods. In: Recommender systems handbook, pp. 107–144. Springer (2011)

[24] Domingos, P.: A few useful things to know about machine learning. Communications of the ACM **55**(10), 78–87 (2012)

[25] Due Trier, Ø., Jain, A.K., Taxt, T.: Feature extraction methods for character recognition-a survey. Pattern recognition **29**(4), 641–662 (1996)

[26] Floyd, R.W.: Algorithm 97: shortest path. Communications of the ACM **5**(6), 345 (1962)

[27] Friedman, J.H.: Greedy function approximation: A gradient boosting machine. Annals of Statistics **29**, 1189–1232 (2000)

[28] Garcia, C., Tziritas, G.: Face detection using quantized skin color regions merging and wavelet packet analysis. Multimedia, IEEE Transactions on **1**(3), 264–277 (1999)

[29] Godoy, D., Corbellini, A.: Folksonomy-based recommender systems: A state-of-the-art review. Int. J. Intell. Syst. **31**(4), 314–346 (2016)

[30] Gunn, S.R., et al.: Support vector machines for classification and regression. ISIS technical report **14** (1998)

[31] Guyon, I., Gunn, S., Nikravesh, M., Zadeh, L.: Feature extraction. Foundations and applications (2006)

[32] Hagberg, A., Swart, P., S Chult, D.: Exploring network structure, dynamics, and function using networkx. Tech. rep., LANL (2008)

[33] Hall, M., Frank, E., Holmes, G., Pfahringer, B., Reutemann, P., Witten, I.H.: The weka data mining software: an update. ACM SIGKDD explorations newsletter **11**(1), 10–18 (2009)

[34] Haupt, J.: Last.fm: People-powered online radio. Music Reference Services Quarterly **12**(1-2), 23–24 (2009)

[35] Jahrer, M., Töscher, A., Legenstein, R.: Combining predictions for accurate recommender systems. In: SIGKDD international conference on Knowledge discovery and data mining, pp. 693–702 (2010)

[36] Jäschke, R., Marinho, L., Hotho, A., Schmidt-Thieme, L., Stumme, G.: Tag recommendations in folksonomies. In: Knowledge Discovery in Databases: PKDD 2007, pp. 506–514. Springer (2007)

[37] Kautz, H., Selman, B., Shah, M.: Referral web: combining social networks and collaborative filtering. Communications of the ACM **40**(3), 63–65 (1997)

[38] Klema, V., Laub, A.J.: The singular value decomposition: Its computation and some applications. IEEE Transactions on Automatic Control **25**(2), 164–176 (1980)

[39] Kobsa, A.: Generic user modeling systems. User modeling and user-adapted interaction **11**(1-2), 49–63 (2001)

[40] Kohavi, R., et al.: A study of cross-validation and bootstrap for accuracy estimation and model selection. In: IJCAI, vol. 14(2), pp. 1137–1145 (1995)

[41] Konstas, I., Stathopoulos, V., Jose, J.M.: On social networks and collaborative recommendation. In: SIGIR, pp. 195–202 (2009)

[42] Koren, Y.: The bellkor solution to the netflix grand prize. Netflix prize documentation **81** (2009)

[43] Koren, Y., Bell, R., Volinsky, C.: Matrix factorization techniques for recommender systems. Computer **42**(8), 30–37 (2009)

[44] Kotsiantis, S., Kanellopoulos, D., Pintelas, P.: Data preprocessing for supervised leaning. International Journal of Computer Science **1**(2), 111–117 (2006)

[45] Lam, S., Herlocker, J.: Movielens 1m dataset (2012)

[46] Latapy, M., Magnien, C., Vecchio, N.D.: Basic notions for the analysis of large two-mode networks. Social Networks (2008)

[47] Lee, K., Lee, K.: Escaping your comfort zone: A graph-based recommender system for finding novel recommendations among relevant items. Expert Syst. Appl. **42**(10), 4851–4858 (2015)

[48] Lee, S., Park, S., Kahng, M., Lee, S.: Pathrank: Ranking nodes on a heterogeneous graph for flexible hybrid recommender systems. Expert Syst. Appl. **40**(2), 684–697 (2013)

[49] Lewis, K., Kaufman, J., Gonzalez, M., Wimmer, A., Christakis, N.: Tastes, ties, and time: A new social network dataset using facebook. com. Social networks **30**(4), 330–342 (2008)

[50] Li, X., Chen, H.: Recommendation as link prediction: a graph kernel-based machine learning approach. In: Proceedings of the 9th ACM/IEEE-CS joint conference on Digital libraries, pp. 213–216. ACM (2009)

[51] Lösch, U., Bloehdorn, S., Rettinger, A.: Graph kernels for rdf data. In: The Semantic Web: Research and Applications, pp. 134–148. Springer (2012)

[52] Ma, H., King, I., Lyu, M.R.: Learning to recommend with social trust ensemble. In: Proceedings of the 32nd international ACM SIGIR conference on Research and development in information retrieval, pp. 203–210 (2009)

[53] Mao, K., Chen, G., Hu, Y., Zhang, L.: Music recommendation using graph based quality model. Signal Processing **120**, 806–813 (2016)

[54] Markovitch, S., Rosenstein, D.: Feature generation using general constructor functions. Machine Learning **49**(1), 59–98 (2002)

[55] Massa, P., Avesani, P.: Trust-aware recommender systems. In: Proceedings of the 2007 Conference on Recommender systems, pp. 17–24 (2007)

[56] Moradi, P., Ahmadian, S., Akhlaghian, F.: An effective trust-based recommendation method using a novel graph clustering algorithm. Physica A: Statistical Mechanics and its Applications **436**, 462 – 481 (2015)

[57] Nixon, M.: Feature extraction & image processing. Academic Press (2008)

[58] O'Donovan, J., Smyth, B.: Trust in recommender systems. In: International conference on intelligent user interfaces, pp. 167–174 (2005)

[59] Ostuni, V.C., Noia, T.D., Sciascio, E.D., Oramas, S., Serra, X.: A semantic hybrid approach for sound recommendation. In: International Conference on World Wide Web, pp. 85–86 (2015)

[60] Page, L., Brin, S., Motwani, R., Winograd, T.: The pagerank citation ranking: Bringing order to the web. TR 1999-66, Stanford InfoLab (1999)

[61] Park, Y., Park, S., Jung, W., Lee, S.: Reversed CF: A fast collaborative filtering algorithm using a k-nearest neighbor graph. Expert Syst. Appl. **42**(8), 4022–4028 (2015)

[62] Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., Prettenhofer, P., Weiss, R., Dubourg, V., et al.: Scikit-learn: Machine learning in python. The Journal of Machine Learning Research **12**, 2825–2830 (2011)

[63] Perugini, S., Gonçalves, M.A., Fox, E.A.: Recommender systems research: A connection-centric survey. Journal of Intelligent Information Systems **23**(2), 107–143 (2004)

[64] Pham, T.N., Li, X., Cong, G., Zhang, Z.: A general graph-based model for recommendation in event-based social networks. In: International Conference on Data Engineering, ICDE, pp. 567–578 (2015)

[65] Quercia, D., Schifanella, R., Aiello, L.M.: The shortest path to happiness: Recommending beautiful, quiet, and happy routes in the city. In: Proceedings of the ACM Conference on Hypertext and Social Media, pp. 116–125 (2014)

[66] Ricci, F., Rokach, L., Shapira, B.: Introduction to recommender systems handbook. Springer (2011)

[67] Said, A., Berkovsky, S., De Luca, E.W., Hermanns, J.: Challenge on context-aware movie recommendation: Camra2011. In: Proceedings of the fifth ACM conference on Recommender systems, pp. 385–386. ACM (2011)

[68] Schafer, J.B., Konstan, J., Riedl, J.: Recommender systems in e-commerce. In: ACM Conference on Electronic commerce, pp. 158–166 (1999)

[69] Schwartz, M.F., Wood, D.: Discovering shared interests using graph analysis. Communications of the ACM **36**(8), 78–89 (1993)

[70] Scott, S., Matwin, S.: Feature engineering for text classification. In: ICML, vol. 99, pp. 379–388 (1999)

[71] Shams, B., Haratizadeh, S.: Graph-based collaborative ranking. CoRR **abs/1604.03147** (2016)

[72] Shani, G., Gunawardana, A.: Evaluating recommendation systems. In: Recommender Systems Handbook, pp. 257–297. Springer (2011)

[73] Shen, J., Deng, C., Gao, X.: Attraction recommendation: Towards personalized tourism via collective intelligence. Neurocomputing **173**, 789–798 (2016)

[74] Tan, S., Bu, J., Chen, C., He, X.: Using rich social media information for music recommendation via hypergraph model. In: Social media modeling and computing, pp. 213–237. Springer (2011)

[75] Tiroshi, A., Berkovsky, S., Kaafar, M.A., Chen, T., Kuflik, T.: Cross social networks interests predictions based on graph features. In: ACM Conference on Recommender Systems, RecSys '13, pp. 319–322 (2013)

[76] Tiroshi, A., Berkovsky, S., Kaafar, M.A., Vallet, D., Chen, T., Kuflik, T.: Improving business rating predictions using graph based features. In: International Conference on Intelligent User Interfaces, pp. 17–26. ACM (2014)

[77] Tiroshi, A., Berkovsky, S., Kaafar, M.A., Vallet, D., Kuflik, T.: Graph-based recommendations: Make the most out of social data. In: User Modeling, Adaptation, and Personalization, pp. 447–458. Springer (2014)

[78] Töscher, A., Jahrer, M., Bell, R.M.: The bigchaos solution to the netflix grand prize. Netflix prize documentation (2009)

[79] Vahedian, F., Burke, R.D., Mobasher, B.: Meta-path selection for extended multi-relational matrix factorization. In: Proceedings of the Florida Artificial Intelligence Research Society Conference, FLAIRS, pp. 566–571 (2016)

[80] Wasserman, S.: Social network analysis: Methods and applications, vol. 8. Cambridge university press (1994)

[81] West, D.B., et al.: Introduction to graph theory, vol. 2. Prentice hall Upper Saddle River (2001)

[82] Wilson, C., Boe, B., Sala, A., Puttaswamy, K.P., Zhao, B.Y.: User interactions in social networks and their implications. In: Proceedings of the 4th ACM European conference on Computer systems, pp. 205–218. Acm (2009)

[83] Wold, S., Esbensen, K., Geladi, P.: Principal component analysis. Chemometrics and intelligent laboratory systems **2**(1), 37–52 (1987)

[84] Wu, H., Yue, K., Liu, X., Pei, Y., Li, B.: Context-aware recommendation via graph-based contextual modeling and postfiltering. Int. J. Distrib. Sen. Netw. **2015**, 16:16–16:16 (2015)

[85] Yu, H.F., Lo, H.Y., Hsieh, H.P., Lou, J.K., McKenzie, T.G., Chou, J.W., Chung, P.H., Ho, C.H., Chang, C.F., Wei, Y.H., et al.: Feature engineering and classifier ensemble for kdd cup 2010. In: KDD Cup (2010)

[86] Zoidi, O., Fotiadou, E., Nikolaidis, N., Pitas, I.: Graph-based label propagation in digital media: A review. ACM Comput. Surv. **47**(3), 48 (2015)

[87] Zukerman, I., Albrecht, D.W.: Predictive statistical models for user modeling. User Modeling and User-Adapted Interaction **11**(1-2), 5–18 (2001)