USING STRUCTURED INFORMATION FROM TAGS FOR BOOK RECOMMENDATIONS

By

MELANIA BERBATOVA

SOFIA UNIVERSITY "ST. KLIMENT OHRIDSKI"
Faculty of Mathematiccs and Informatics

MARCH 2020

To the Faculty of Mathematics and Informatics, Sofia University:

The members of the Committee appointed to examine the thesis of MELANIA BERBATOVA find it satisfactory and recommend that it be accepted.

Alexander Popov, Ph.D., Academic advisor

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

Chapter One

Introduction

1.1 Problems and Motivation

Recommendation systems are systems, most commonly using machine learning algorithms, that are built to recommend items (online content, products, people and more) to users of a given web platform. The recommendations are based on consumer preferences (most often in the form of ratings) and the discovery of links between users and items. Today, recommendation systems are increasingly entering our lives and are an important driving force for many businesses.

In this thesis we will present a recommendation system for books based on one of the largest open datasets - goodbooks10k, consisting of various data from the GoodReads.com book site. This dataset if of interest because of its volume and variety, and also because the recommendation system currently used by GoodReads.com has obvious disadvantages. While most published papers on recommendation systems for this dataset explore different algorithms for collaborative filtering and matrix factorization, we will focus on content-based recommendations. To do this, we will use the customer tags provided for the given books.

Customer tags are unstructured textual information that reflects the personal views of users on the web. In order to improve the recommendations, we will try to structure the tag information by organizing it into concepts and linking it to external resources.

A great number of research deals with the problem of learning structured information from systems of customer tags, or so called textitfolksonomies. The motivation for this master's thesis is that the studies dealing with learning structured information from folksonomies don't benefit from methods for word and graph embeddings developed in recent years and are often not evaluated on real-life tasks such as item recommendation.

1.2 Example of Book Recommendations

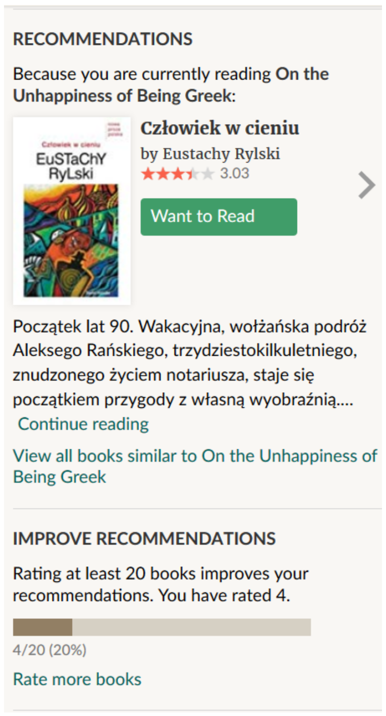
In this section, we will do a review of the current recommendations that the website Goodreads.com uses. The website has two places when it offers recommendations - on the homepage and at a book's page.

1.2.1 On the homepage

The first place where a recommendation is shown is at the homepage, as seen in figure 1.1. From the message we can guess that item-based filtering is used. There are two main issues we can notice: firstly, it is based only on one of the user's listed books and not taking into account the others, and secondly, the language of the suggested book does not match the language of the used book.

1.2.2 On a book's page

The second place that a recommendation occurs is on a book's page. Again, this could be seen as item-based filtering. The recommender suggested "Harry Potter", which is a book with very high rating (4.62) and a best-seller, similar to "Hunger games". But on the customer side, it would be much more useful to see books a little less known.


RECOMMENDATIONS

Because you are currently reading On the Unhappiness of Being Greek:

Człowiek w cieniu

by Eustachy Rylski

3.03

Want to Read

Początek lat 90. Wakacyjna, wołżańska podróż Aleksego Rańskiego, trzydziestokilkuletniego, znudzonego życiem notariusza, staje się początkiem przygody z własną wyobraźnią.... Continue reading

View all books similar to On the Unhappiness of Being Greek

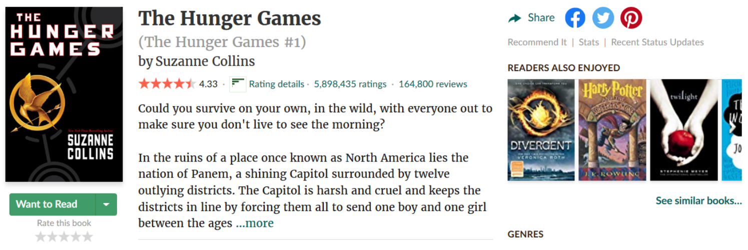
IMPROVE RECOMMENDATIONS

Rating at least 20 books improves your recommendations. You have rated 4.

4/20 (20%)

Rate more books

Figure 1.1 Recommendation on the Goodreads homepage


The Hunger Games

(The Hunger Games #1)

by Suzanne Collins

4.33 · Rating details · 5,898,435 ratings · 164,800 reviews

Could you survive on your own, in the wild, with everyone out to make sure you don't live to see the morning?

In the ruins of a place once known as North America lies the nation of Panem, a shining Capitol surrounded by twelve outlying districts. The Capitol is harsh and cruel and keeps the districts in line by forcing them all to send one boy and one girl between the ages ...more

Want to Read

Rate this book

Share

Recommend It | Stats | Recent Status Updates

READERS ALSO ENJOYED

See similar books...

GENRES

Figure 1.2 Recommendation on "The Hunger Games" page on Goodreads.

1.3 Goals and Tasks

The main goal of this thesis is to develop a system that can effectively use a book's user tags for giving relevant recommendations. In this section we will define a list of tasks that need to be executed in order to achieve this goal:

1. *Overview of related work.* This task includes searching for papers, systems and any other work done on the topic of content-based recommendations for books. Then, we need to analyze the results that the found research achieves and how they can be beneficial to the development of our system.

2. *Data exploration.* As the data set is heterogeneous and voluminous, we need to look at the characteristics and the patterns of the data and create smaller data set for experiments. This task also includes searching for external data resources that are suitable for our task, such as knowledge bases and ontologies, for example.

3. *Choosing a set of approaches.* As the task of book recommendation is very broad, we need to choose a subset of the data and set of approaches we will work with. Based on our research, we can define a sublist of tasks and perform incremental experiments.

4. *Building a recommendation system.* This task includes choosing the architecture of the recommender, choosing learning algorithm and choosing the evaluation metrics.

5. *Feature engineering.* In this task we need to organise the information about the books in features suitable for the recommender system's architecture. For this task we will create different feature sets that will be fed to the system.

6. *Planning and executing the experiments.* In this task we will define the experiments that we will execute. We need to choose a model as a baseline and incrementally try to improve it.

7. *Evaluation and interpretation of the results.* In the end, we need to compare the models we developed to the baselines and to other pre-existing systems. We should also try to interpret the results and explain why some approaches give better results than others.

Chapter Two

Background and related work

Overview of the existing methods for book recommendations.

2.1 Recommendation Systems

Recommendation systems are engines that use algorithms leveraging the interaction between users to generate personalized recommendations. They provide users with recommendations for new content these users might be interested in (music, movies, books, etc).

Recommendation systems can be divided into three main types: Collaborative Filtering (CF), Content-based Filtering (CBF) and Hybrid systems. Collaborative filtering systems analyze users' interactions with the items (e.g. through ratings, likes or clicks) to create recommendations. On the other hand, content-based systems use semantic information (frequently called "metadata") about the items in the system. Hybrid systems are a combination of these two approaches. If compared to collaborative or content-based systems, hybrid ones usually exhibit higher recommendation accuracy. This is due to the fact that collaborative filtering lacks information about domain dependencies, while content-based filtering systems do not take into account users' preferences[18].

2.1.1 Collaborative filtering

Collaborative filtering recommenders are systems that suggest recommendations based on users' interactions (most commonly, ratings). A great deal of the most efficient collaborative filtering algorithms are based on matrix factorization (MF). Matrix factorization algorithms work by decomposing the user-item interaction matrix into the product of two lower dimensionality rectangular matrices that refer to user representations and item representation[16]. When multiplied, these matrices return the original matrix. Using matrix factorization, we can project users and items to a common vector space and compute closeness between them. This family of methods gained popularity around the Netflix prize challenge and showed state-of-art results for many RS tasks [4].

2.1.2 Content-based filtering

Content-based recommenders are a type of recommender systems that use item metadata (description, rating, product features, reviews, tags, genres) to find items similar to those the user has enjoyed in the past. To generate recommendations, we use items that are most similar to the ones liked by a given user. In the context of books, main characteristics, and even the whole of the book's content can be present as metadata. As descriptions and reviews are purely natural language data, and categorical data such as tags and genres can also be represented in a way suitable for natural language processing, employing such techniques is crucial for the design of successful content-based recommenders.

In content-based recommendation systems, the features of products, e.g. the genre and the author of a book, are represented most often as a bag of words or a vector space model. Features of a book might refer to its title, summary, outline, whole text, or metadata, including the author, year of publication, publisher, genre, number of pages, etc. A promising way to enrich book metadata is automatic genre identification. Users on community-based book websites can assign tags and organize books under custom-defined "shelves". These tags and shelves can serve as genres and can be used to indicate patterns in users' opinions.

Content-based recommenders use a variety of machine learning algorithms, including Naive Bayes, support vector machines, decision trees, and kNN. As bag-of-words and vector representations can have hundreds or thousands of dimensions, techniques as Latent Dirichlet Allocation (LDA) are often adopted. The content may also require natural language processing (NLP) techniques to make use of semantic and syntactic characteristics.

2.2 Book recommendation systems

2.2.1 Overview

Recommender systems for books have been studies for years, because it is believed that it can help with solving the reading decline, especially among young, by recommending literature that is relevant to personal preferences. As recommender systems of books have been commonly studies for the use cases of libraries and educational institutions, they also have commercial importance, as they can drive the sales of websites for online book trading, such as Amazon.com, which acquisited Goodreads in 2013.

In their paper "A survey of book recommender systems", Alharthi, Inkpen, and Szpakowicz [2] present a detailed survey of different approaches to book recommendation, compiled from over 30 papers up to 2017. These publications report results from content-based filtering, collaborative filtering, and other methods, obtained on the Book-Crossing, LitRec, LibraryThing, INEX, and Amazon reviews datasets. Only the LitRec dataset uses data from GoodReads.

In the current study, we will focus on models for GoodReads built on the goodbooks-10k dataset. Recent publications written on this dataset mostly deal with collaborative filtering. Out of 11 unique papers in English on recommender systems retrieved by Google Scholar when search is performed for "goodbooks-10k": Le [20], Kula [19],Recommendation [31], Greenquist, Kilitcioglu, and Bari [11], Zhang et al. [38], Zhang et al. [37], Paudel, Luck, and Bernstein [27], Khanom et al. [15], Kouris et al. [17], Yang et al. [36], Hiranandani et al. [14], 10 examine algorithms for Collaborative filtering, two [20, 11] implement hybrid systems, and only one [20] implement a simple content-based recommender. We will examine the content-based systems or components of hybrid systems, developed on goodbooks-10k, and will compare them to systems using another dataset for GoodReads - LitRec.

2.2.2 Content-based book recommender systems

In their bachelor thesis, Le [20] implement simple collaborative, content-based and hybrid systems for book recommendations. Their content-based recommender uses only ratings data and leaves aside books metadata. They achieve a best score of 0.842 of root-mean-square error (RMSE) for FunkSVD algorithm.

A CBF/CF hybrid system is implemented by Greenquist, Kilitcioglu, and Bari [11]. To gather more information, they merge goodbooks-10k data with Amazon reviews data. For books representations, they use tf-idf vectors of the books descriptions, tags, and shelves. Authors report using book descriptions in their content-based approach, but it is unclear how they obtained the descriptions, as the latter are not present in the goodbooks-10k dataset. There best score is 0.842 for RMSE, achieved with FunkSVD algorithm.

In addition to published ones, there are many other approaches to the goodbook-10k dataset, implemented and shared by community members on platforms such as Kaggle.com. On Kaggle's goodbook-10k dataset page, there are 31 shared kernels[1]. Some of them contain demonstrations on the development of content-based systems, including ones that use tags information. It can be seen that, as mentioned in [11], tags are turned into tf-idf vectors, and cosine similarity is used for determining the books that are the most similar to a given one.

[1]https://www.kaggle.com/zygmunt/goodbooks-10k/kernels

Authors	Dataset	Features	Evaluation metrics	Algorithms
Le [20]	goodbooks-10k	ratings	MAP, CC, MPS, MNS, MDS	cosine similarity
Greenquist, Kilitcioglu, and Bari [11]	goodbooks-10k	tf-idf vectors of tags	RMSE	cosine similarity
Alharthi and Inkpen [1]	Litrec	linguistic and stylometry features	precision@10, recall@10	kNN

Table 2.1 Overview of recent published papers on content-based recommender systems for books.

Alharthi and Inkpen [1] use the Litrec dataset [35] to develop a book recommendation system based on the linguistic features of the books. Litrec dataset has ratings of 1,927 users of 3,710 literary books and contains the complete text of books tagged with part-of-speech labels.

The content-based systems that Alharthi and Inkpen develop are based on the analysis of lexical, character-based, syntactic, characterization and style features of the books texts. Feature sets are learned from book texts converted into a numerical value using one-hot encoding. For evaluation, they extract the ratings from Goodreads, representing ratings of 1-2 as dislikes and 3-5 as likes. They obtain their best results with kNN, which are 0.498 for recall@10 and 0.223 for precision@10.

2.2.3 Critiques of the published work

After the examination of the previous work on content-based systems for books, and particularly on goodbooks-10k, we can see that none of the methods proposed makes use of structuring the metadata. The metadata which is used is fed directly to the algorithm, or vectorized by one-hot encoding or tf-idf. In order to deal with hierarchy, ambiguity, and synonyms, some data normalization and natural language processing techniques could be adopted. Also, it is arguable whether tf-idf is the most suitable way for vectorizing tags information, as their distribution does not necessarily follow the one of words in natural text. Another observation is that the systems developed do not take advantage of recent advances in machine learning, particularly of embeddings, despite the large volume of the data available.

2.2.4 Proposed improvements

In this thesis we propose a method for structuring metadata information in the form of a graph and extracting graph embeddings out of it, that we later use as features for our recommendation algorithm. We chose to use graph structure and graph embeddings as graphs are commonly used data structures for many domains. Graph embeddings will help us extract implicit information such as structural relations and hierarchy. We can also use graph structure for ontology representation and enrichment of the given data with external resources as ontologies and knowledge bases, which will help us gain additional semantic information about the data, such as genre relations, for example.

In order to use a graph structure, we will start the graph with the books as nodes. We will then add the tags as nodes and link the books with the tags they have been labeled with. We will try to enrich the graph with links between the tags based on their semantic features. Then we will add the other metadata, such as authors, languages, and years of publication, to the developed graph. Finally, we will use the node embeddings of the books as features for the recommendation algorithm.

2.3 Learning structured knowledge for tags

Tags are used for the annotation of resources in many social media platforms. The accumulated social tags, contributed by millions of online users (folks) collaboratively, are referred to as folksonomies. [2] As data in folkonomies is highly unstructured and often ambiguous, there is numerous research done on the task of extracting "useful" tags and organising them into some forms of structured knowledge, such as concept hierarchies or lightweight terminological ontologies. For an overview of methods in this section, we will use the structure suggested in the paper "Learning Structured Knowledge from Social Tagging Data: A Critical Review of Methods and Techniques" [9]. This paper provides a categorized overview of the state-of-the-art of knowledge organization methods for folksonomies. Methods are divided into two main groups: learning term lists from folksonomies and learning relationships from folksonomies, where the first group contains methods for word sense disambiguation (WSD) and word sense induction (WSI).

2.3.1 Learning term lists

Word sense disambiguation (WSD) is defined as the task of selecting the correct sense of a word for a particular context, normally from a fixed inventory of potential senses. WSD is similar to the task of *Named Entity Disambiguation (NED)*, whose purpose is to map named entities, such as names of people or locations, from an input text document to corresponding unique entities in a target knowledge base. Both methods use mappings from text to target ontology or knowledge base. The most widespread knowledge bases are DBPedia and WordNet, whereas the choice of ontology is being dependent on the domain of the folksonomy.

[2] http://vanderwal.net/folksonomy.html

Word Sense Induction is the task of automatic creation of set of senses, e.g. cluster of tags, from the tags in the dataset. Compared to WSD, WSI does not need external resources. Most methods based on WSI employ unsupervised learning techniques, more specifically, clustering, and form the clusters by computing similarity between tags. For most of the current methods, similarity is computed by co-occurrence features of tags in annotating same resources (tag-resource matrix) or created by same users (tag-user matrix), as summarized in [10]. In case of tag-resource matrix, tags used for the common items will score high similarity, whereas in case of tag-user matrix, tags used by common users will be highly similar. Commonly used supervised techniques include clustering algorithms such as Hierarchical Agglomerative Clustering (HAC), Maximal Complete-Link Clustering, K-means and DBSCAN, as well as dimensionality reduction techniques such as non-negative matrix factorization (NMF).

2.3.2 Learning relationships in tags

Research in this category aims to learn relations among tags, which can help generate a hierarchy of concepts or tags. These hierarchies can be further extended to lightweight or even formal ontologies. A variety of techniques from the areas of social network analysis, machine learning, data mining, natural language processing, the Semantic Web, etc., have already been proposed in the literature.

Lee et al. [21] use divisive hierarchical clustering for generating hierarchical structures from social tags and deterministic annealing (DA) algorithm for automatic determination of the number and the sizes of clusters are automatically determined. Unfortunately, the final tag structure cannot discriminate all sub, related and parallel relations from the ancestor and child nodes, as discussed in the study.

Si, Liu, and Sun [32] propose three methods to discover the subsumption relation between tags, as well as a greedy algorithm to eliminate the redundant relations by constructing a

Layered Directed Acyclic Graph (Layered-DAG) of tags. They evaluate their method by empirical study of the constructed Layered-DAG and error analysis.

The problem of learning a *collabulary* - a special ontology that represents the knowledge of both users and experts in an integrated fashion - is introduced by Marinho, Buza, and Schmidt-Thieme [24]. The authors first take a folksonomy and a domain-expert ontology and project them into an enriched folksonomy through semantic mapping. Then they apply an algorithm based on frequent itemset mining techniques to learn an ontology over the enriched folksonomy.

A more recent method for learning relationships between tags is proposed by Dong, Wang, and Coenen [8]. Their work is based on extracting domain-independent features from tag pairs according to similarity, topic distributions and probabilistic associations. They chose probabilistic topic analysis over word embedding approaches as data representation technique, as dimensions in word embeddings are not probabilistically and semantically interpretable and leave embedding approaches for a further study.

2.3.3 Problems with current methods

Some of the problems pointed out by [9] and [8] which are relevant to our current tasks are as follows:

- Learning term lists using the WSD-based methods heavily relies on the external lexical resources or domain ontologies. The main issue of these methods is concerned with the low coverage and/or low quality of external resources.

- An important issue of the methods for learning relationships between tags is that they are not able to differentiate different types of relations.

- The co-occurrence based methods typically need to re-compute the whole model when new data is available and therefore do not scale well.

2.3.4 Evaluation of structures

The methods for evaluation of structured information from folksonomies can be conditionally divided in 3 groups: semantic evaluation and evaluation of navigation, as mentioned by Strohmaier et al. [33], and evaluation of recommendation. Semantic evaluation examines the truthfulness of the learned relations. Evaluation of navigation shows to what degree the obtained structure helps users better navigate through the system. Finally, evaluation of recommendation shows how the structures help in recommending relevant items to the users.

As Marinho, Buza, and Schmidt-Thieme [24] point out, in most of the literature on learning structural knowledge (ontology) from folksonomies the quality of the ontologies is measured based on whether they match people's common sense or a reference ontology, instead of how good it is for the task it was designed for. For this reason, they propose to plug the investigated knowledge structures in recommender systems and evaluate the outcome as an indicator of their usefulness. For evaluation, they use the minimal elements necessary for a task-based evaluation of an ontology, defined by [30]: a task, one or more ontologies, an application and a gold standard, which give a reasonable framework for evaluating such structures.

2.3.5 Book tags

While book tags for the Goodreads website and goodbook-10k dataset are not studyed enough, more research has been done on customer tags for other social networks, such as ones for image and audio content, but also including the social network for books Library Thing[3], which is quite similar to GoodReads. Many researchers, such as Bartley [3], Thomas, Caudle, and Schmitz [34], Lu, Park, and Hu [23], Zubiaga, Körner, and Strohmaier [39], make studies on the content of Library Thing tags and point out that these tags are full of noise

[3]https://www.librarything.com/

and represent personal attitudes towards the books rather than the books' characteristics.

2.4 Graph Embeddings

In this section, we will present an overview of existing graph embedding algorithms, suitable for our problem.

2.4.1 Representation learning and embeddings

In machine learning, representation learning is learning representations of input data typically by transforming it, that makes it easier to perform a task like classification or prediction. On of the most common techniques for representation learning is embedding.

An embedding is a relatively low-dimensional space into which you can translate high-dimensional vectors. Embeddings make it easier to do machine learning on large inputs like words corpora and graphs. Ideally, an embedding captures some of the semantics of the input by placing semantically similar inputs close together in the embedding space. An embedding can be learned and reused across models. While word embeddings are widely used in natural language processing for various tasks, graph embeddings are still gaining popularity.

2.4.2 General problem

The graph embedding problem is widely examined in the paper "A comprehensive survey of graph embedding: Problems, techniques, and applications" [5]. Graph embeddings are related to two traditional research problems: graph analytics and representation learning. Graph analytics is used for mining useful information from graph data. Representation learning obtains data representations that make it easier to extract useful information when building classifiers or other predictors. Graph embedding lies in the overlap of the two problems and focuses on learning the low-dimensional representations of the parts of the graph (nodes and edges), while the graph structures are preserved.

Graph embedding methods are divided into 4 groups: node embedding, edge embedding, hybrid embedding and whole graph embedding. As for our task we would like to have representations only of the books nodes, we focus on the methodologies for node embeddings.

2.4.3 Node embedding

Node Embedding is the most common embedding output setting, where a node embedding represents each node as a vector in a low-dimensional space. Nodes that are "close" in the graph are embedded to have similar vector representations. The differences between various graph embedding methods lie in how they define the "closeness" between two nodes.

2.5 Node2vec

In this master thesis, we will use the node embedding algorithm Node2vec, proposed by Grover and Leskovec [13]. Node2vec is an algorithmic framework for learning continuous representations for nodes in networks. Node2vec learns a mapping of nodes to a low-dimensional space of features that maximizes the likelihood of preserving network neighborhoods of nodes. It works for both directed/undirected and weighted/unweighted networks.

Node2vec generates the embeddings by using the skip-gram model, used in word2vec word embeddings. Word2vec is a group of shallow, two-layer neural networks that are used to produce word embeddings. Word2vec takes as input a large corpus of text and returns a vector space of several hundred dimensions, with each unique word in the corpus being assigned a corresponding vector in the space. The word vectors obtained are positioned in the vector space in a way that words that appear in similar contexts in the corpus are located close in the vector space.[26]

Node2vec generates its corpus by running biased random walks on the graph. Each walk starts at a random node and performs a series of steps, where each step goes to a random

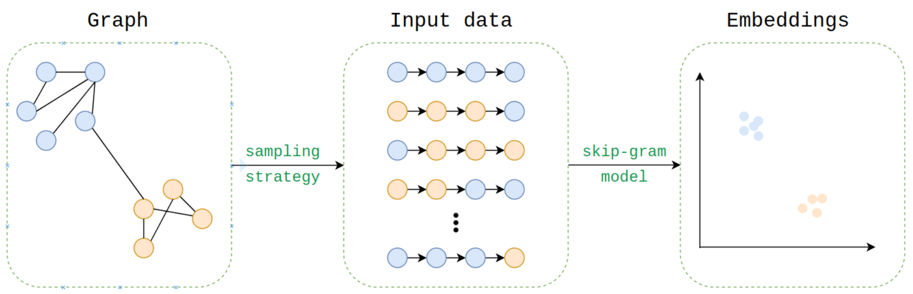


Figure 2.1 Node2vec embedding process

neighbor. Each random walk forms a sentence that can be fed into word2vec. The emdedding process is illustrated in figure 2.1 (source: "node2vec: Embeddings for Graph Data" [4]).

2.5.1 Sampling strategies

As graphs can be directed or not, weighted or not, and cyclic or not, they are much more complex in structure than text. In order to solve that, node2vec uses a sampling strategy, controlled by hyperparameters, which gives the probability of "jumping" to a neighbour node.

Node2vec is based on two main sampling strategies - for generating a neighbourhood of a node - breadth-first sampling (BFS) and depth-first sampling (DFS).

With the BFS strategy, the neighborhood generated is restricted to nodes which are immediate neighbors of the source node. With DFS strategy, the neighborhood consists of nodes sequentially sampled at increasing distances from the source node. Illustration of the different neighbourhoods generated can be seen in 2.2.

The authors observe that BFS is correladed with creating embeddings that lead to structural equivalence, meaning that nodes that have similar structural roles in networks have close enbeddings.On the other hand, DFS leads to embeddings exhibiting homophily, meaning nodes that are highly interconnected are embedded closely together.

Looking through the prism of our project, structural equivalence means that books with a

[4]https://towardsdatascience.com/node2vec-embeddings-for-graph-data-32a866340fef

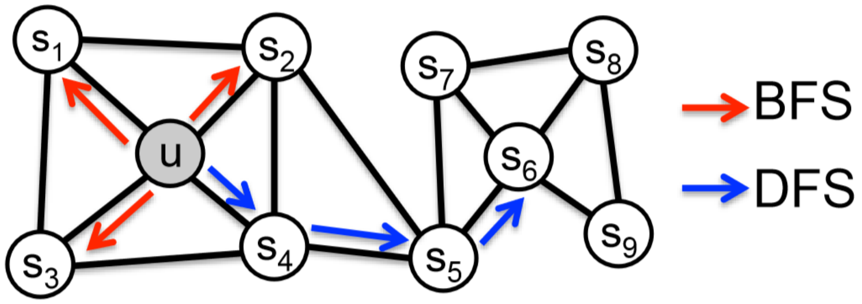


Figure 2.2 Difference between BFS and DFS sampling strategies. [12]

similar number and structure of neighbours should have similar embeddings, while homophily means that books that have same neighbours and are close to each other in the graph should have similar embeddings. Apparently, the second is more important for the task of book recommendations. Therefore, we would like to put more empphasis on DFS than BFS in the way we create random walks and graph embeddings.

2.5.2 Random walks

In practice, the graph often exhibit characteristics that are a mix of homophily and structural equivalence. For that reason, node2vec presents a sampling strategy that can easily interpolate between DFS and BFS. This is achieved by developing a flexible biased random walk procedure.

The simplest way to bias the random walks is to sample based on the edge weights, if any. However, this method does not take into account the graph structure and is not helpful for exploring different types of neighborhoods. In order to deal with this, the authors introduce two parameters to the random walks - return parameter p and in-out parameter q.

The return parameter p controls the likelihood of going back to the previous node in the walk. If p is high ($> \max(q,1)$), we are less likely to sample an already visited node in the following two steps. If p is low ($< \min(q,1)$), it would would keep the walk “local”, close to the starting node.

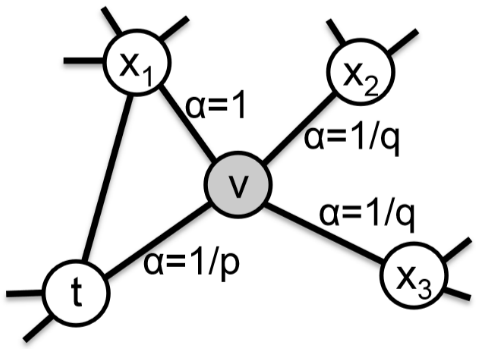


Figure 2.3 Illustration of the random walk procedure in node2vec[12]

The in-out parameter q allows the search to differentiate between "inward" and "outward" nodes. If $q > 1$, the random walk is biased towards nodes close to node the previous node. Such walks approximate BFS behavior and obtain a local view of the graph with respect to the start node. If $q < 1$, the walk is more inclined to visit nodes which are further away from the previous node. Such behavior approximates DFS and encourages outward exploration.

Figure 2.3 illustrates the random walk procedure in node2vec, where t is the previous node, v is the current, and alpha is the search bias. Each edge label represents the probability of its target node being the next in the random walk. [12]

2.5.3 Graph weights

In his PhD thesis, Compagnon [6] explores whether graph embedding algorithms based on random walks, such as node2vec, take into consideration the weights while they produce embeddings of weighted graphs. For node classification, performed on the Wiki 5k dataset, no significant differences are observed between the weighted and unweighted versions of the embedding. Having these results in mind, we will run our experiments with both weighted and unweighted graphs and explore if there is difference in performance for the task of node recommendation.

2.5.4 Node2vec implementation

For our experiments, we use a node2vec implementation, based on the original paper [12], that is publicly available on Github [5]. As input parameters, it takes integer representation of the graph, values of the return parameter p (default is 1), in-out parameter q (default is 1), and other arguments, such as if the graph is directed and weighted, number of walks per source, length of the walks, and number of desired dimensions of the learned representation.

[5]https://github.com/aditya-grover/node2vec

Chapter Three

Data

3.1 Goodbooks-10k dataset

Goodbooks-10k is one of the most popular open datasets for evaluating recommendation algorithms. It is a compilation of 5,976,479 ratings for the 10,000 most popular books in the book website Goodreads, as well as book metadata for each book. The dataset is available online on the FastML website. Data, in the form of ratings, books metadata, to-read tags, and user tags and shelves, is organised in 5 files.

Some previous research on the topic of book recommender systems relies on datasets such as Book-Crossing, LitRec Vaz, Ribeiro, and Matos [35], LibraryThing Lu, Park, and Hu [23]. The main advantage of the goodbooks-10k dataset over the above-mentioned ones is the volume of data. As the number of records is close to 6 million and the data presented is diverse and consistent, it allows experimenting with different algorithms, including ones that are designed for big data.

3.1.1 Ratings

Customer ratings are between 1 ("did not like it") and 5 ("it was amazing"). The distribution of ratings in this dataset is centered around 100 ratings per user, where the average rating per user is 4. The distribution of the number of ratings per user and the average rating per

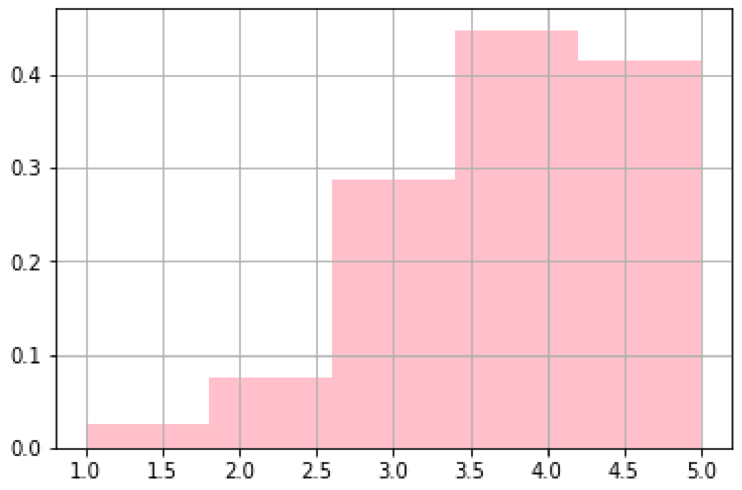


Figure 3.1 Distribution of ratings in the dataset

user seem to follow a multivariate normal distribution.

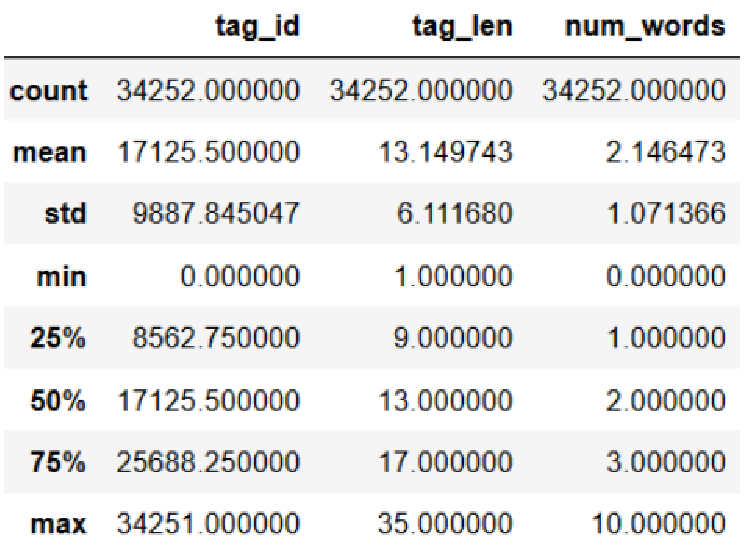
3.1.2 Tags

The total number of defined tags is 34,251. Around 1000 of the tags are from languages different from English, such as Arabic, Persian, Russian, Greek etc. The set of tags per book is similar to textual data, except that there is no sequence between the tags.

The tags of a book show its genres (e.g. "young-adult", "fiction" ,"biography"), readers' intents ("to-read", "to-buy", "to-be-finished"), books features ("printed", "books-in-spanish"), awards ("printz-award"), authors ("oscar-wilde"), etc. and many of the tags have similar or equal meanings. Every book is characterized by its tags and the number of occurrences of every tag. We have almost 1 million records of (book_id, tag_id, count), or an average of 100 tags per book.

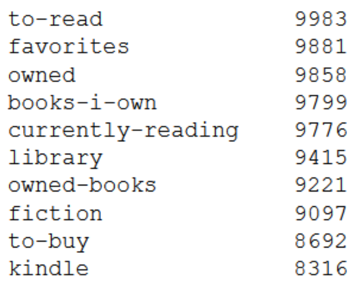
In figure 3.2 we can see some descriptive statistics on the tags. Most of the tags consist of 1 to 3 words are of a length of 9 to 17 characters.

In figure 3.3 we show the most popular tags in the dataset. We can see that most of them show the customers personal attitude to the book, rather than particular characteristics of the books themselves.



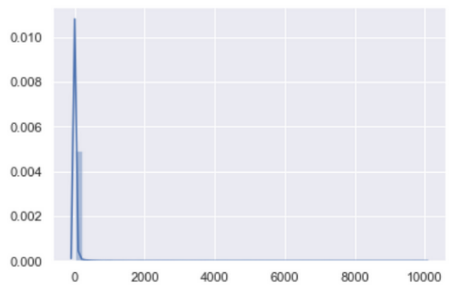
	tag_id	tag_len	num_words
count	34252.000000	34252.000000	34252.000000
mean	17125.500000	13.149743	2.146473
std	9887.845047	6.111680	1.071366
min	0.000000	1.000000	0.000000
25%	8562.750000	9.000000	1.000000
50%	17125.500000	13.000000	2.000000
75%	25688.250000	17.000000	3.000000
max	34251.000000	35.000000	10.000000

Figure 3.2 Descriptive statistics of the tag



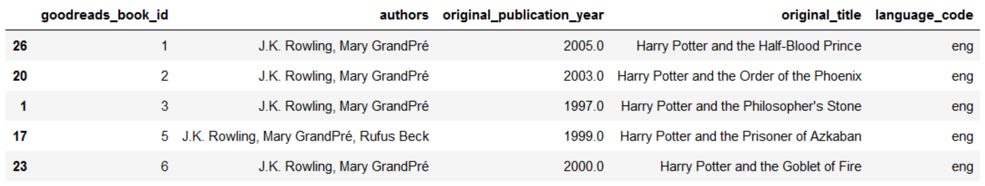
```
to-read              9983
favorites            9881
owned                9858
books-i-own          9799
currently-reading    9776
library              9415
owned-books          9221
fiction              9097
to-buy               8692
kindle               8316
```

Figure 3.3 10 most popular tags



```
count    34252.000000
mean        29.192806
std        277.254715
min          1.000000
25%          1.000000
50%          1.000000
75%          5.000000
max       9983.000000
Name: tag_name, dtype: float64
```

Figure 3.4 Distribution of the unique tags



	goodreads_book_id	authors	original_publication_year	original_title	language_code
26	1	J.K. Rowling, Mary GrandPré	2005.0	Harry Potter and the Half-Blood Prince	eng
20	2	J.K. Rowling, Mary GrandPré	2003.0	Harry Potter and the Order of the Phoenix	eng
1	3	J.K. Rowling, Mary GrandPré	1997.0	Harry Potter and the Philosopher's Stone	eng
17	5	J.K. Rowling, Mary GrandPré, Rufus Beck	1999.0	Harry Potter and the Prisoner of Azkaban	eng
23	6	J.K. Rowling, Mary GrandPré	2000.0	Harry Potter and the Goblet of Fire	eng

Figure 3.5 Books metadata

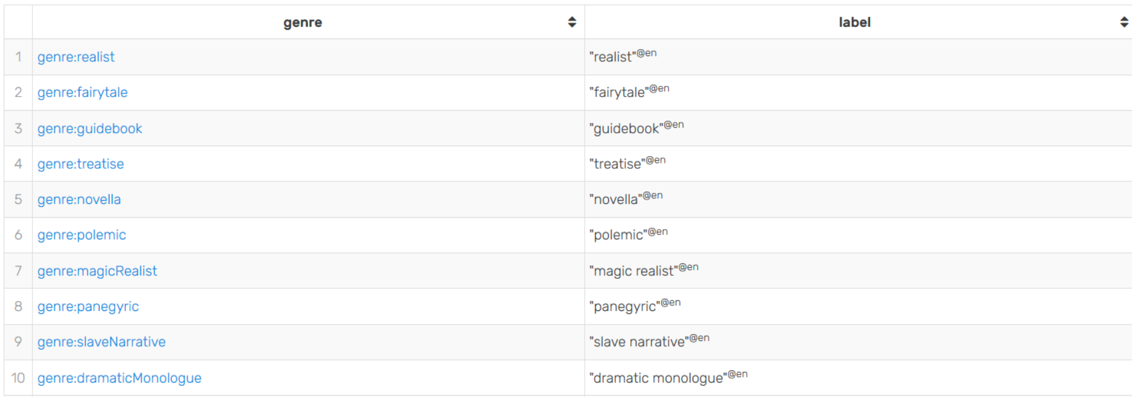
Figure 3.4 show number of books tagged with every tag. We can see that the data is highly unbalanced, as nearly half of the tags occur on only one book. We do not need these tags, as we cannot measure similarity of books based on them.

3.1.3 Books metadata

We are provided with metadata for every book. The dataset for of the following columns:

book_id, goodreads_book_id, best_book_id, work_id, books_count, isbn, isbn13, authors, original_publication_year, original_title, title, language_code, average_rating, ratings_count, work_ratings_count,work_text_reviews_count, ratings_1, ratings_2, ratings_3, ratings_4, ratings_5, image_url, small_image_url.

Since most of this columns are irrelevant, or rely to information that is already included in the ratings, as books metadata we will use only the authors, language and publication year of the book.



	genre	label
1	genre:realist	"realist"@en
2	genre:fairytale	"fairytale"@en
3	genre:guidebook	"guidebook"@en
4	genre:treatise	"treatise"@en
5	genre:novella	"novella"@en
6	genre:polemic	"polemic"@en
7	genre:magicRealist	"magic realist"@en
8	genre:panegyric	"panegyric"@en
9	genre:slaveNarrative	"slave narrative"@en
10	genre:dramaticMonologue	"dramatic monologue"@en

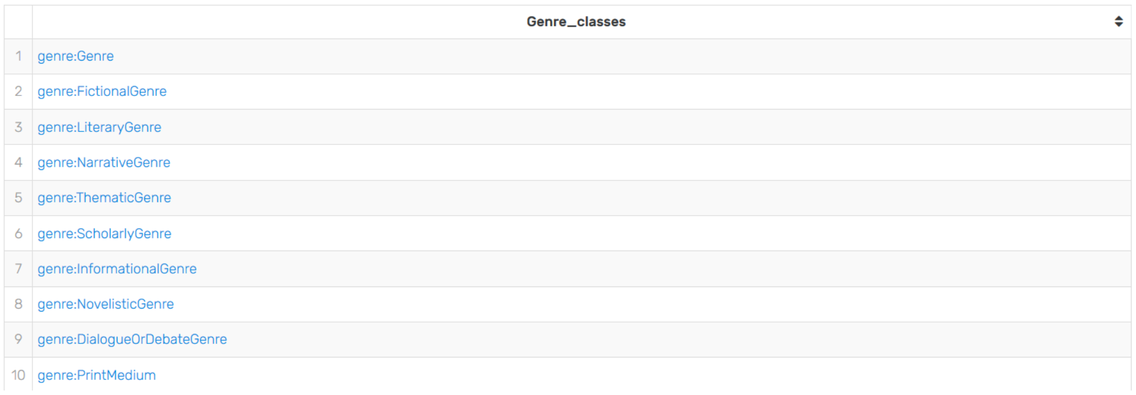
Figure 3.6 Examples of genre instances

3.2 Genre ontology

To organize the information about genres included in the tags, we will link the tags to an ontology of genres. After some research, we decided that the most suitable recourse for this purpose whould be the CWRC Genre Ontology. [1] This ontology is used by the Canadian Writing Research Collaboratory to assign genres to different types of cultural objects, particularly but not exclusively written ones. In addition to literature genres, it consist also of music genres and genres from other arts, as well as document and journalistic genres. A quick look at the tags data shows us that the genres included are not limited to literature ones, therefore we decided to work with the whole ontology as it is. The CWRC Genre Ontology consists of a total of 284 genres, labeled both in English and French, that are organized in 64 genre classes. In figures 3.6 and 3.7 we can see some examples for both genre classes and genre instances.

We chose to use this ontology would like to extract information about the genres of the books, that is not included in the metadata file, but it is one of the main characteristics of the books. In chapter 5 we show how the ontology is mapped to the tags.

[1] http://sparql.cwrc.ca/ontologies/genre-2019-07-09.html



	Genre_classes
1	genre:Genre
2	genre:FictionalGenre
3	genre:LiteraryGenre
4	genre:NarrativeGenre
5	genre:ThematicGenre
6	genre:ScholarlyGenre
7	genre:InformationalGenre
8	genre:NovelisticGenre
9	genre:DialogueOrDebateGenre
10	genre:PrintMedium

Figure 3.7 Examples of genre classes

Chapter Four

Design of the system

4.1 Tensorrec

We used the python library TensorRec [1], which is a recommendation system library based on the deep learning library tensorflow, and employs the power of tensors and computational graphs. Recommender systems designed with TensorRec can be customized by choosing representation graphs, prediction graphs and loss graphs. Representation graphs are used for choosing the algorithms for latent representations (embeddings) of users and items. Prediction graphs are used for computing recommendation scores from the latent representations of users and items. Loss functions are the algorithms that define how loss is calculated from a set of recommendations.

We chose TensorRec over other libraries for recommendation systems, as it allows quick implementation and evaluation of a recommendation system and also because it is more suitable for working with voluminous data.

4.1.1 Content-based recommendations

TensorRec scores recommendations by consuming user and item interactions (ratings) and building two low-dimensional vectors, a user representation and an item representation, that

[1] https://github.com/jfkirk/tensorrec

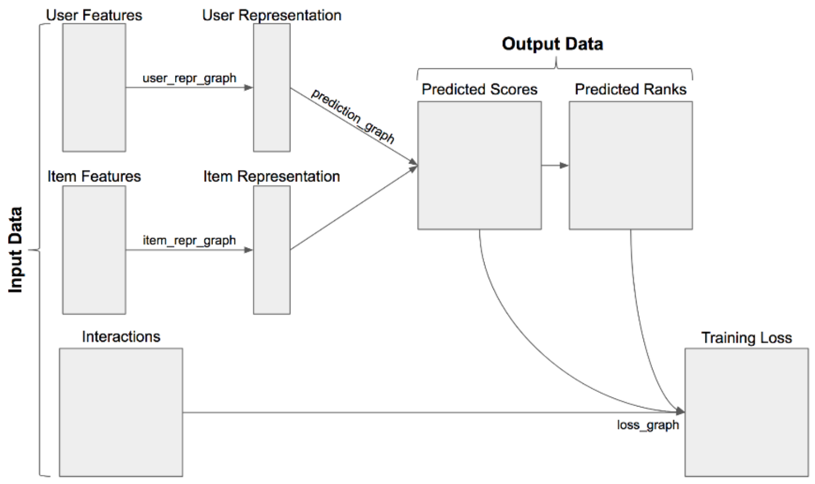


Figure 4.1 Design of TensorRec

are called indicator features. The dot product of these two vectors is the score for the relationship between that user and that item — the highest scores are predicted to be the best recommendations.

```
predictions = model.predict(user_features=user_indicator_features,
                            item_features=item_indicator_features)
```

In case of content-based recommendations, instead of the built item features, we pass the item metadata we would like to work with. In our case, these features are the embeddings we would test.

```
predictions = model.predict(user_features=user_indicator_features,
                            item_features=books_features)
```

4.1.2 Loss graphs

As proposed by James Kirk in his blogpost "Getting Started with Recommender Systems and TensorRec"[2], we will try to improve results by changing the loss graph that the system

[2]https://towardsdatascience.com/getting-started-with-recommender-systems-and-tensorrec-8f50a9943eef

uses. The loss graph takes in the predictions and and the actual ratings and calculates a penalty (loss) that the system will try to decrease as it learns.

The default loss graph that TensorRec uses is RMSE (root mean square error). This means that TensorRec is trying to estimate the values of the ratings exactly as they are in the training set. However, for our recommender system, we do not need to predict ratings exactly, but just to be able to rank books a user will like above these that they won't like.

For this reason, many systems function by "learning to rank." We can make our TensorRec system work this way by using a loss graph called WMRB.

WMRB or "weighted margin-rank batch", proposed by [22], works by taking a random sample of items the user has not rated and comparing their predicted ratings to items the user likes. Over time, this pushes the items that the user likes to the top of the rankings. In our case, we would want to train the model on only the ratings considered as a "like" (4.0 or more), so WMRB pushes those to the top.

4.2 Data selection

4.2.1 Data sampling

For our experiments, we sample 20% of the data. We make this by taking the books with the first 20% Goodreads IDs. As IDs are not correlated to popularity, in this way we produce sample comprised of books with mixed popularity. We sample data so we can perform quicker experiments with it.

4.2.2 Training and test set

We chose the standard split of 80% percent of randomly sampled ratings for the training data and 20% for the test data. Because of the large volume of data, we preferred this method over k-fold cross validation, as the cross validation would have slowed down the work of the predictive algorithms.

4.2.3 Definition of a "like"

As shown in fFigure 3.1, the higher ratings significantly outnumber the lower ones. Therefore, we chose to define a “like” as a rating of 4 or more stars, instead of 3 or more, as defined in previous research. We estimated that this would lead to more balanced data and a better approximation of readers’ perception, which was further proved by our experiments.

4.3 Evaluation metrics

We choose recall@k as an evaluation metric. Recall@k shows the number of a user’s liked test items that were included in the top k of the predicted rankings, divided by k. Similar to Alharthi and Inkpen [1], we chose k to be 10.

$$\textit{recall at } k = \frac{\#\textit{ of recommended items at } k \textit{ that are liked}}{\#\textit{ of liked items}}$$

Recall@K is an often metric used for many recommender systems because it emulates the behavior of a recommendation product: If a book website is only going to show me my top 10 recommendations, then their algorithm will want to be effective at putting books that I like in my top 10.

In the case of book recommendations, recall@k is a more reliable metric than precision@k, because if the user has rated less than 10 books, precision@10 for their prediction cannot be 1.

We prefer recall@k and over RMSE and other metrics that measure the ratings prediction error, as for the design of the current recommender system, the exact score predicted is not as important as ranking the liked items higher than those that are not liked.

Chapter Five

Experiments

In this chapter, we will present our various experiments on learning structured information from the goodbooks-10k tags.

5.1 Creating a subset for testing

For testing different methods for learning relations between tags, we create a subset of 1000 random consecutive tags, with (ids from 28446 to 28628). We will select tags that have different kinds of relationships between them:

- Synonymy
- Specificity (hypernym/hyponym)
- Acronyms
- Words with same roots
- Semantic extension
- Different spellings (misspelling) of same words
- Same named entity

We use this dataset for evaluation and visualization of our experiments. In Table 5.1 we can see examples of the listed relations from the sample.

Type of relation	Number of pairs	Examples
Synonyms	3	“started-but-didn-t-finish”, “started-but-not-finished”
Extension of meaning	40	“spy”, “spy-military”
Hypernym/hyponym	4	“spy-books”, “spy-fiction”
Same root	1	“stalker”, “stalking”
Different forms	5	“startup”, “startups”
Different spellings	7	“steampunk”, “steam-punk”
Same named entity	4	“steinbeck-john”, “steinbeck”

Table 5.1 Examples of relations between tags

5.2 Word embeddings

Word embeddings are a widely used methodology in natural language processing. The idea of word embeddings is to map every word to an n-dimantional vector, in a way that close words (that occur often in the same context) are represented by vectors that are close in the n-dimensional vector space. In recent years there has been a boom in the development of word embeddings based on a huge variety of architectures, including the Skip-gram model [25], LSTMs [29], and Transformers [7].

5.2.1 Naive methods

Before jumping into word embeddings, we first tried to find relationships between tags based on fuzzy string matching. We use the fuzzy token sort ratio method which takes the tokens of the tags, sorts them, and then calculates the ratio of the overlap of the two token lists.

As table 5.2.1 demonstrates, this methods does not give good results, especially on the "extension of meaning" relation, where there are more tokens in one of the tags. Another problem is that it matches words that are close in spelling but have nothing in common in meaning.

tags	fuzzy token sort ratio
"states", "stats"	0.91
"star-wars", "star-wars-legends-novels"	0.55

Table 5.2 Examples of tags similarity with fuzzy token matching

5.2.2 GloVe embeddings

GloVe embeddings [28] are embeddings obtained by the Skip-gram model. In our experiments we used GloVe embeddings with 300 dimensions, trained on Wikipedia 2014 + Gigaword 5. We performed tests on our predefined test set, predicting similarity between tags based on their cosine similarity. The problems we noticed with the GloVe embeddings are that different numbers map to very close embeddings, which leads to very high scores of tags that contain numbers, but have different meanings. Another problem are out-of-vocabulary (OOV) and misspelled words that are frequently seen in user-generated content, such as tags, as they do not have corresponding embeddings. For example, tags such as “-post”, “post-apocolypse” score perfect similarity, because the misspelled token "apocolypse" is an OOV token and does not have a corresponding embedding.

Examples of the issues discussed can be seen in the Table 5.2.2.

tags	cosine similarity
“-post”, “post-apocolypse”	1.0
“-read”, “mangás-read”	1.0
“1920”, “1920s”	0.582
“1-star”, “4-star”	0.946

Table 5.3 Calculating tag similarity in GloVe embeddings

5.2.3 ELMo embeddings

In order to deal with out-of-vocabulary words, we tried ELMo embeddings [29], which are character-level contexual embeddingns, based on an bi-directional LSTM. They are very powerful for tasks such as part-of-speech tagging and named entity recognition. Unfortunately, as tags are sequences of just a few words, we do not have the amount of context we would have in a text. This leads to poor results for the task of similarity of tasks. (Table 5.2.3).

tags	cosine similarity
"spy", "spys"	0.647
"startup", "startups"	0.742
"spy", "spy-novels"	0.746

Table 5.4 Examples of tag similarity of ELMo embeddings

5.2.4 BERT embeddings

Another recent popular word embeddings are the BERT embeddings [7]. BERT stands for Bidirectional Encoder Representations from Transformers, and BERT embeddings are pre-trained embeddings based on the Transormers architecture and developed by Goodle, that can be fine-tuned for various NLP tasks. Like ELMo embeddings, BERT embeddings are also contexualized, meaning that BERT produces word representations that are dynamically informed by the words around them. In table 5.2.4 we can see that BERT embeddings deal better with OOV and numbers than GloVe embeddings, but for our task they have the same weaknesses as ELMo, as tags lack context.

tags	cosine similarity
“-post”, “post-apocolypse”	0.321
“-read”, “mangás-read”	0.705
“1920”, “1920s”	0.756
“1-star”, “4-star”	0.905
“spy”, “spys”	0.587
“startup”, “startups”	0.561
“spy”, “spy-novels”	0.328

Table 5.5 Examples of tag similarity of BERT embeddings

5.2.5 Results on the test set

Table 5.2.5 we show how good each of the listed above methods performed on finding relations between the tags in the subset for testing with different thresholds. Best results are obtained with Glove and threshold of 0.90.

	Method	Threshold	Precision	Recall	F1-score
Fuzzy matching	0.75	0.338	0.359	0.348	
Fuzzy matching	0.80	0.439	0.281	0.342	
Fuzzy matching	0.90	0.736	0.218	0.337	
Glove + cos sim	0.75	0.232	0.718	0.351	
Glove + cos sim	0.80	0.345	0.609	0.440	
Glove + cos sim	0.90	0.523	0.172	0.258	
BERT + cos sim	0.80	0.171	0.531	0.258	
BERT + cos sim	0.85	0.191	0.343	0.245	
BERT + cos sim	0.90	0.272	0.141	0.185	
ELMo + cos sim	0.75	0.188	0.781	0.304	
ELMo + cos sim	0.80	0.214	0.562	0.310	
ELMo + cos sim	0.90	0.409	0.141	0.209	

Table 5.6 Performance of methods on the subset for testing

Table 5.2.5 we show how good each of the listed above methods performed on finding relations between the tags in the subset for testing with different thresholds. On this subset GloVe outperforms BERT because of the small number of numbers and out-of-vocabulary words.

5.2.6 Stacked embeddings

Having in mind the strenght and weaknesses of all the methods listed above, we stacked together GloVe embeddings, non-contetxual character embeddings and BERT embeddings. Stacking embeddings helped in dealing with numbers and gave high scores for tags with implicit semmantic relation, as for example "04-babies-and-toddlers" and "04-caldecott", where *caldecott* refers to Caldecott Medal - annual award for the most distinguished American picture book for children. In our final experiments, we used stacked embeddings with similarity threshold of 0.90 to find relations between tags.

tags	cosine similarity
“04-babies-and-toddlers”, “04-caldecott”	0.903
“04-babies-and-toddlers”, “04-preschooler”	0.922

Table 5.7 Examples of tag similarity in stacked embeddings

5.3 Clustering

5.3.1 DBSCAN

DBSCAN, standing for *density-based spatial clustering of applications with noise*, is a data clustering algorithm based on density. It is commonly used where the number or size of the clusters can't be easily predefined and there the data contains a lot of noise. It works by grouping together points that are close in the high-density regions, marking as outliers points that lie alone in low-density regions.

We decided to experiment with DBSCAN as our data contains a lot of noise. Other suggested clustering methods for learning relations between tags, such as hierarchical aglomerative clustering and variants of k-means, request the number of desired clusters as a parameter, which in our case cannot be easily determined.

The downside of using DBSCAN is that there is no direct way to control the size of the output clusters. In a case where an embedding of a tag is close to two clusters, these clusters merge and can't be easily separated later. On a bigger part of the dataset, BDSCAN outputs cluster sets where some clusters are huge and contain quite heterogeneous information.

We first tested DBSCAN with the small sample and got almost perfect results for the clusters.

```
cluster 0 :  ['sports-star', 'star']
cluster 1 :  ['spy', 'spy-action', 'spy-books', 'spy-espionage', 'spy-fiction', 'spy-kids',
'spy-military', 'spy-mystery', 'spy-novel', 'spy-novels', 'spy-stories', 'spy-stuff', 'spy-thriller',
'spy-thriller-espionage', 'spy-thrillers']
cluster 2 :  ['sri', 'sri-lanka']
cluster 3 :  ['stage', 'stage-play']
cluster 4 :  ['stand-alone', 'stand-alone-books', 'stand-alones', 'stand-up']
cluster 5 :  ['star-wars', 'star-wars-books', 'star-wars-cannon', 'star-wars-canon', 'star-wars-collection',
'star-wars-disney-canon', 'star-wars-eu', 'star-wars-expanded-universe', 'star-wars-legends',
'star-wars-legends-novels', 'star-wars-new-canon', 'star-wars-novels', 'star-wars-old-republic-era',
'star-wars-own', 'star-wars-owned', 'star-wars-read', 'star-wars-to-read', 'star-wars-universe']
cluster 6 :  ['started-but-didn-t-finish', 'started-didn-t-finish']
cluster 7 :  ['steamy', 'steamy-romance']
cluster 8 :  ['steel', 'steel-danielle', 'steel-danielle']

cluster 9 :  ['steinbeck', 'steinbeck-john']
```

When we scaled with more data, however, there appeared clusters with huge amount of unrelated tags.

```
cluster 4 :  ['-dean', '1800s', '1830s', '1840s', '1850s', '1860s', '1870s', '1880s', '1890s', '1900s',
'1910s', '1920s', '1930s', '1940s', '1950s', '1960s', '1970s', '1980s', '1990s', '2-non-fiction', '2010s',
'20s', '60s', '90s', 'a', 'a-christie', 'a-shaw', 'abandon', 'abandonados', 'abandoned', 'abandonment',
'abbey', 'abduction', 'ability', 'abraham', 'abundance', 'abuse', 'abused', 'abused-heroine', 'academia',
'acceptance', 'accident', 'accidents', 'ace', 'aching', 'action-adventure', 'action-adventure-thriller',
'action-and-adventure', 'action-fiction', 'action-mystery', 'action-packed', 'action-romance',
'action-suspense', 'action-thriller', 'action-thrillers', 'actresses', 'adam', 'adams', 'adams-douglas',
'adaptation', 'adaptations', 'addicted', 'addiction', 'adoption', 'adorable', 'adrian', 'adrian-lara',
'adult-books', 'adult-christian-fiction', 'adult-fantasy', 'adult-fiction', 'adult-historical-fiction',
'adult-literature', 'adult-non-fiction', 'adult-nonfiction', 'adult-novels', 'adult-paranormal-romance',
'adult-romance', 'adultery', 'adventure', 'adventure-action', 'adventure-books', 'adventure-fiction',
'adventure-non-fiction', 'adventure-nonfiction', 'adventure-sports', 'adventure-survival',
'adventure-thriller', 'adventure-travel', 'adventures', 'aeronautics', 'affair', 'africa',
'africa-non-fiction', 'african', 'african-american', 'african-american-authors',
'african-american-experience', 'african-american-fiction', 'african-american-history', 'african-american-lit',
'african-american-literature', 'african-american-romance', 'african-american-studies', 'african-americans',
'african-authors', 'african-fiction', 'african-history', 'african-lit', 'african-literature',
'african-writers', 'afro-american', 'agatha', 'agents', 'aging', 'airborne', 'airplane', 'alcohol',
'alcoholism', 'alex', 'alex-craft', 'alex-cross', 'alex-delaware', 'alex-rider', 'alien', 'aliens', 'alpha',
'alphabet', 'alvin', 'amanda', 'amazing', 'amazing-books', 'amber', 'amelia-sachs',
'american-author', 'american-authors', 'american-drama', 'american-fiction', 'american-historical-fiction',
'american-literature', 'american-studies', 'amnesia', 'amy', 'amy-harmon', 'amy-plum', 'amy-tan', 'anarchy',
'ancient', 'ancient-classics', 'ancient-egypt', 'ancient-greece', 'ancient-greek', 'ancient-greek-literature',
'ancient-history', 'ancient-lit', 'ancient-literature', 'ancient-mysteries', 'ancient-mystery',
'ancient-philosophy', 'ancient-rome', 'ancient-texts', 'ancient-world', 'andrews', 'angel-books',
'animal-books', 'animal-fantasy', 'animal-fiction']
```

5.3.2 Exploring co-occurences

As mentioned in Chapter 2, most of the research done so far on learning relations between tags relies on finding similarity based on the co-occurences of tags in annotating same resources (tag-resource matrix) or created by same users (tag-user matrix). As we do not have information about the users in the tags data, we can research only the tag-resource co-occurences.

We take one big cluster formed by DBScan and try to form one or more semantically coherent sub-clusters, based on the co-occurence features in the tags inside. Our idea is to take the most popular tag in the cluster as a representable of the whole cluster and leave inside only the points that are similar to it. The most popular tag in our case is 'adult-fiction'. We then select the tags that occur for a subset of the books that this tag is used for, which are the tags `['1830s', '1870s', '1890s', '90s', 'action-suspense', 'adult-christian-fiction', 'adult-fiction', 'adult-literature', 'adult-novels', 'adult-romance', 'adultery', 'adventure-fiction', 'african-american-fiction', 'african-fiction', 'alex', 'alex-cross', 'alien', 'alvin', 'american-drama', 'american-historical-fiction', 'amnesia', 'amy-tan']`, out of which only half are relevant. This experiment showed us that there is not a direct correlation between the co-occurences and the semantic relations between tags.

5.3.3 Discussion

Our experiments with clustering and exploring co-occurences did not give a promising direction for finding hierarchical structure in the tags. For that reason, we decided to not include structures formed by clusters in the final experiments.

5.4 Mapping to external resources

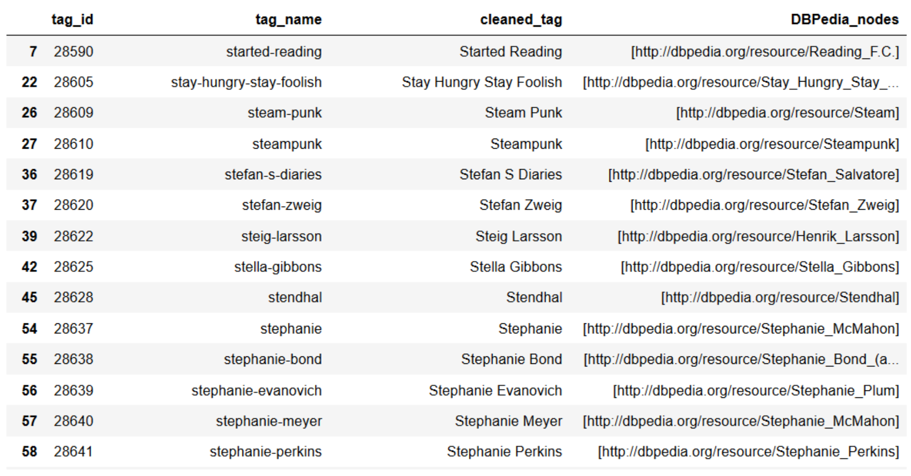
5.4.1 Mapping to the genre ontology

In order to extract structured information about the genres of the books, we map the tags to the genre ontology presented in Chapter 3. We use similarity-based metrics on stacked embeddings to find corresponding concepts in the ontology. After several experiments with different similarity thresholds, we concluded that best results are obtained with cosine similarity above 0.90. After removing the wrong mappings, we ended with 112 new links between tag-genre pairs and 100 new links between genre-class pairs. This results show the little overlap in the semantics of the tags and the ontology, and the little coverage of the tags, as we have total number of 284 genres in the ontology, organized in 64.

5.4.2 Named entity disambiguation

We noticed that named entities, such as authors and book series, are frequently seen in the dataset. Therefore, we wanted to map the tags related to the same entity, to the corresponding node in DBPedia. For this task, we decided to use DBPedia Spotlight.

We obtained good results with DBPedia spotlight on a small subset of the data, as we can see in figure 5.1. Unfortunately, when we tried in on a bigger dataset, the server was responding all the time and we could not scale it for the whole dataset. If mapping tag concepts to DBPedia nodes where successful, we ccan enrich our graph with the links between classes and instances existing in DBPedia and obtain a much richer in information graph.



	tag_id	tag_name	cleaned_tag	DBPedia_nodes
7	28590	started-reading	Started Reading	[http://dbpedia.org/resource/Reading_F.C.]
22	28605	stay-hungry-stay-foolish	Stay Hungry Stay Foolish	[http://dbpedia.org/resource/Stay_Hungry_Stay_...
26	28609	steam-punk	Steam Punk	[http://dbpedia.org/resource/Steam]
27	28610	steampunk	Steampunk	[http://dbpedia.org/resource/Steampunk]
36	28619	stefan-s-diaries	Stefan S Diaries	[http://dbpedia.org/resource/Stefan_Salvatore]
37	28620	stefan-zweig	Stefan Zweig	[http://dbpedia.org/resource/Stefan_Zweig]
39	28622	steig-larsson	Steig Larsson	[http://dbpedia.org/resource/Henrik_Larsson]
42	28625	stella-gibbons	Stella Gibbons	[http://dbpedia.org/resource/Stella_Gibbons]
45	28628	stendhal	Stendhal	[http://dbpedia.org/resource/Stendhal]
54	28637	stephanie	Stephanie	[http://dbpedia.org/resource/Stephanie_McMahon]
55	28638	stephanie-bond	Stephanie Bond	[http://dbpedia.org/resource/Stephanie_Bond_(a...
56	28639	stephanie-evanovich	Stephanie Evanovich	[http://dbpedia.org/resource/Stephanie_Plum]
57	28640	stephanie-meyer	Stephanie Meyer	[http://dbpedia.org/resource/Stephanie_McMahon]
58	28641	stephanie-perkins	Stephanie Perkins	[http://dbpedia.org/resource/Stephanie_Perkins]

Figure 5.1 Examples of mapping tags to DBPedia nodes

5.5 Graph embeddings

5.5.1 Experiments

In order to test the methodologies for creating the graph structure we listed above on a real task, we will embed the obtained graphs and use them for a content-based recommendation system.

The algorithm for this is as follows:

Algorithm 1: Test a graph structure

```
1. Create node embeddings for the whole graph
2. Take the embeddings for the book nodes only
3. Divide book embeddings into train and test set
4. Train a content-based model with embeddings as features
5. Evaluate as explained in Chapter 4.
```

.

We experimented with creating node2vec embeddings for the graphs with different parameterisations, in terms of: number of dimensions, number of random walks, properties of the graph, such as directed-undirected and weighted-unweighted.

For a baseline model we prepare a book-tag matrix with the count of occurrences of every tag, and take the obtained vectors of the books as features.

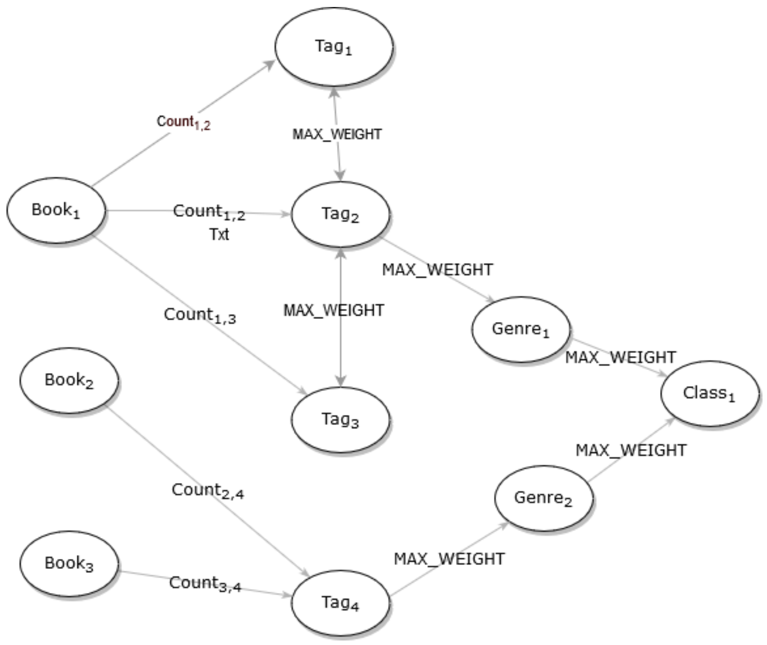


Figure 5.2 Possible nodes and edges in the graph

Then, creation of the graph structure follows these steps:

1. Let b be a book node and t be a tag node. Create a graph from the data, where books and tags are nodes, and the number times that book has been tagged with that tag is the weight of the edge *(b, t)*.

2. For every genre g, that has been mapped to some tag t, create and edge *(t, g)*. Then, add an edge *(g, c)* for every class c that g is instance of and complete the graph with the class hierarchy of the classes mentioned. Set edge weights to MAX_WEIGHT (we set it to 1,000,000) for the new edges.

3. For every two tags t_1, t_2 that are predicted in relation, add edges (t_1, t_2) and set the weight to MAX_WEIGHT.

Figure 5.2 illustrates possible nodes and edges that can exist in the obtained graph.

5.5.2 Results

In table 5.5.2 we demonstrate some of the experiments we have run. The column Features shows what kind of edges of the graph were included in the experiment. The column "# of walks" refers to the number of random walks we use in the graph embedding algorithm, and "Weighted" and "Unweighted" show us the properties of the created graph. "Train"and "Test" refer to the train and test scores of recall@10.

Features	# of walks	Weighted	Directed	Train	Test
Tags count vectors	-	-	-	0.0394	0.0193
Book-tag	5	yes	yes	0.0412	0.0406
Book-tag	100	yes	yes	0.0677	0.0389
Book-tag	100	yes	no	0.0565	0.0236
Book-tag + tag-genre	5	yes	yes	0.0463	0.0462
Book-tag + tag-genre + genre-class	5	yes	yes	0.0506	0.0509
Book-tag + tag-genre + genre-class	5	no	yes	0.0113	0.0120
Book-tag + tag-genre + genre-class	100	yes	no	0.0861	0.0356
Book-tag + tag-tag	5	yes	yes	0.0623	0.0617
Book-tag + tag-tag	100	yes	no	0.0800	0.0424

Table 5.8 Results of content-based recommendation with graph embeddings features

5.5.3 Discussion

When analyzing the results, we can note the following observations:

- Our initial experiments of putting books and tags into a graph, increased performance, compared to using the tag information directly as count vectors.
- Adding additional information (nodes and edges) to the graph led to slight improvement.

- The results shown in the table refer to the best results obtained with the corresponding edges and parameters. There were fluctuations in the results, depending on the random choice of train and test set and also depending on the embeddings that node2vec outputed.
- The bigger number of walks (in our case 100) should mean that the data of the graph is presented in a fuller way. However, in our experiments the bigger number of walks, and therefore, the bigger corpus that the algorithm works with, leads to overfitting (having results that are significantly better on the train set than on the test set). This problem may be solved with regularization or with adding more edges in the graph.
- We can see that for our task adding weights to the graph leads to better performance, where we could not observe correlation between adding directions and change in performance.
- There is still plenty of room for improvement in order to reach a satisfactory state that can be used in a production recommendation system. This can be achieved by using more data, better external resources and a combination with collaborative filtering.

Chapter Six

Conclusion and future work

6.1 Conclusion

As mentioned in previous research, the tasks of learning structured data from tags is quite challenging, namely because the social tagging data is quite sparse and full of noise. There are many approaches that can be chosen for this task, and in this master thesis we conducted experiments with variety of them. The conclusions we came to are as follows:

1. The field of using graph embeddings for the task of item recommendation is still unexplored and gives space for experimenting with huge variety of methods.

2. The success of mapping the folksonomy to an external resource, such as ontology or a knowledge base, is dependent of the quality of the resource and the semantic overlap between the folksonomy and the resource.

3. Word embeddings give us a promising direction for finding relations between tags. However, they do not reflect the variety of relations that can exist between two tags and are not equally useful for the different types of relations.

4. In order to test the usability of graph structure through using the graph embeddings for recommendation system, we need experiments with more data and with various recommendation methods.

6.2 Future work

There is a number of improvements that can be done to the current state of the project.

1. *Design a custom recommeder system.* In this master thesis, we use the library TensorRec to build a prototype of a recommender system. In order to have more control of the architecture and algorithms used in the system, we can design our own recommender system from scratch.

2. *Use a knowledge base.* In the future, the graph created from the tag ca be represented as a knowledge base. This will help us store in the graph additional information, as different roles of the edges and type of nodes (class-instance).

3. *Develop hybrid methods for learning structured information.* As we saw, there are many methods for learning structured information and none of them good enough for the task by itself. In future, we need to develop methods that combine various techniques in order to automatically structure voluminous and noisy information, such as the one examined in the dataset we chose.

4. *Use full data.* We would like to perform all experiments with the full data available, which is quite expensive in terms of time and resources.

5. *Test embeddings with more algorithms.* For more credibility, the node embeddings obtained should be tested with more recommendation algorithms, such as kNN and regression for example.

6. *Parameter tuning.* Parameters of the graph embedding algorithm, such as p, q, number of output dimensions and number of weights can be further tweaked for better results. Also, different and more sophisticated weighting schemes can be developed and tested.

Bibliography

[1] Haifa Alharthi and Diana Inkpen. “Study of linguistic features incorporated in a literary book recommender system”. In: *Proceedings of the 34th ACM/SIGAPP Symposium on Applied Computing*. ACM. 2019, pp. 1027–1034.

[2] Haifa Alharthi, Diana Inkpen, and Stan Szpakowicz. “A survey of book recommender systems”. In: *Journal of Intelligent Information Systems* 51.1 (2018), pp. 139–160.

[3] Peishan Bartley. “Book tagging on LibraryThing: how, why, and what are in the tags?” In: *Proceedings of the American Society for Information Science and Technology* 46.1 (2009), pp. 1–22.

[4] James Bennett, Stan Lanning, et al. “The netflix prize”. In: *Proceedings of KDD cup and workshop*. Vol. 2007. Citeseer. 2007, p. 35.

[5] Hongyun Cai, Vincent W Zheng, and Kevin Chen-Chuan Chang. “A comprehensive survey of graph embedding: Problems, techniques, and applications”. In: *IEEE Transactions on Knowledge and Data Engineering* 30.9 (2018), pp. 1616–1637.

[6] Paul Compagnon. “Weighted graph embedding with random walks”. PhD thesis. Oct. 2017. DOI: 10.13140/RG.2.2.34118.32329.

[7] Jacob Devlin et al. “BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding”. In: *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*. Minneapolis, Minnesota: Association for

Computational Linguistics, June 2019, pp. 4171–4186. DOI: 10.18653/v1/N19-1423. URL: https://www.aclweb.org/anthology/N19-1423.

[8] Hang Dong, Wei Wang, and Frans Coenen. “Learning Relations from Social Tagging Data: 15th Pacific Rim International Conference on Artificial Intelligence, Nanjing, China, August 28–31, 2018, Proceedings, Part I”. In: July 2018, pp. 29–41. ISBN: 978-3-319-97303-6. DOI: 10.1007/978-3-319-97304-3_3.

[9] Hang Dong, Wei Wang, and Hai-Ning Liang. “Learning Structured Knowledge from Social Tagging Data: A Critical Review of Methods and Techniques”. In: Dec. 2015. DOI: 10.1109/SmartCity.2015.89.

[10] Andres Garcia-Silva et al. “Review of the state of the art: Discovering and associating semantics to tags in folksonomies”. In: *The Knowledge Engineering Review* 27.1 (2012), pp. 57–85.

[11] Nicholas Greenquist, Doruk Kilitcioglu, and Anasse Bari. “GKB: A Predictive Analytics Framework to Generate Online Product Recommendations”. In: *2019 IEEE 4th International Conference on Big Data Analytics (ICBDA)*. IEEE. 2019, pp. 414–419.

[12] Aditya Grover and Jure Leskovec. “node2vec: Scalable Feature Learning for Networks”. In: *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. 2016.

[13] Aditya Grover and Jure Leskovec. “node2vec: Scalable feature learning for networks”. In: *Proceedings of the 22nd ACM SIGKDD international conference on Knowledge discovery and data mining*. ACM. 2016, pp. 855–864.

[14] Gaurush Hiranandani et al. “Clustered Monotone Transforms for Rating Factorization”. In: *Proceedings of the Twelfth ACM International Conference on Web Search and Data Mining*. ACM. 2019, pp. 132–140.

[15] Aniqa Zaida Khanom et al. “BookCeption: A Proposed Framework for an Artificially Intelligent Recommendation Platform”. In: *Proceedings of the 2019 8th International Conference on Software and Computer Applications*. ACM. 2019, pp. 253–257.

[16] Yehuda Koren, Robert Bell, and Chris Volinsky. “Matrix factorization techniques for recommender systems”. In: *Computer* 8 (2009), pp. 30–37.

[17] Panagiotis Kouris et al. “A versatile package recommendation framework aiming at preference score maximization”. In: *Evolving Systems* (2018), pp. 1–19.

[18] Oleksandr Krasnoshchok and Yngve Lamo. “Extended content-boosted matrix factorization algorithm for recommender systems”. In: *Procedia Computer Science* 35 (2014), pp. 417–426.

[19] Maciej Kula. “Mixture-of-tastes models for representing users with diverse interests”. In: *arXiv preprint arXiv:1711.08379* (2017).

[20] Hieu Le. “Building and evaluating recommender systems”. In: (2019).

[21] Kangpyo Lee et al. “Tag sense disambiguation for clarifying the vocabulary of social tags”. In: *2009 International Conference on Computational Science and Engineering*. Vol. 4. IEEE. 2009, pp. 729–734.

[22] Kuan Liu and Prem Natarajan. “WMRB: Learning to Rank in a Scalable Batch Training Approach”. In: *arXiv preprint arXiv:1711.04015* (2017).

[23] Caimei Lu, Jung-ran Park, and Xiaohua Hu. “User tags versus expert-assigned subject terms: A comparison of LibraryThing tags and Library of Congress Subject Headings”. In: *Journal of information science* 36.6 (2010), pp. 763–779.

[24] Leandro Balby Marinho, Krisztian Buza, and Lars Schmidt-Thieme. “Folksonomy-based collabulary learning”. In: *International Semantic Web Conference*. Springer. 2008, pp. 261–276.

[25] Tomas Mikolov et al. "Distributed representations of words and phrases and their compositionality". In: *Advances in neural information processing systems*. 2013, pp. 3111–3119.

[26] Tomas Mikolov et al. "Efficient estimation of word representations in vector space". In: *arXiv preprint arXiv:1301.3781* (2013).

[27] Bibek Paudel, Sandro Luck, and Abraham Bernstein. "Loss Aversion in Recommender Systems: Utilizing Negative User Preference to Improve Recommendation Quality". In: *arXiv preprint arXiv:1812.11422* (2018).

[28] Jeffrey Pennington, Richard Socher, and Christopher D Manning. "Glove: Global vectors for word representation". In: *Proceedings of the 2014 conference on empirical methods in natural language processing (EMNLP)*. 2014, pp. 1532–1543.

[29] Matthew Peters et al. "Deep Contextualized Word Representations". In: *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long Papers)*. New Orleans, Louisiana: Association for Computational Linguistics, June 2018, pp. 2227–2237. DOI: 10.18653/v1/N18-1202. URL: https://www.aclweb.org/anthology/N18-1202.

[30] Robert Porzel and Rainer Malaka. "A task-based approach for ontology evaluation". In: *ECAI Workshop on Ontology Learning and Population, Valencia, Spain*. Citeseer. 2004, pp. 1–6.

[31] Collaborative-Filtering Recommendation. "Building Information Systems Using Collaborative-Filtering Recommendation Techniques". In: *Advanced Information Systems Engineering Workshops: CAiSE 2019 International Workshops, Rome, Italy, June 3–7, 2019, Proceedings*. Springer, p. 214.

[32] Xiance Si, Zhiyuan Liu, and Maosong Sun. “Explore the structure of social tags by subsumption relations”. In: *Proceedings of the 23rd International Conference on Computational Linguistics*. Association for Computational Linguistics. 2010, pp. 1011–1019.

[33] Markus Strohmaier et al. “Evaluation of folksonomy induction algorithms”. In: *ACM Transactions on Intelligent Systems and Technology (TIST)* 3.4 (2012), pp. 1–22.

[34] Marliese Thomas, Dana M Caudle, and Cecilia Schmitz. “Trashy tags: problematic tags in LibraryThing”. In: *New Library World* (2010).

[35] Paula Cristina Vaz, Ricardo Ribeiro, and David Martins de Matos. “LitRec vs. Movielens”. In: ().

[36] Fan Yang et al. “Towards Interpretation of Recommender Systems with Sorted Explanation Paths”. In: *2018 IEEE International Conference on Data Mining (ICDM)*. IEEE. 2018, pp. 667–676.

[37] Shuai Zhang et al. “Metric Factorization: Recommendation beyond Matrix Factorization”. In: *arXiv preprint arXiv:1802.04606* (2018).

[38] Xuejian Zhang et al. “An Interpretable and Scalable Recommendation Method Based on Network Embedding”. In: *IEEE Access* 7 (2019), pp. 9384–9394.

[39] Arkaitz Zubiaga, Christian Körner, and Markus Strohmaier. “Tags vs shelves: from social tagging to social classification”. In: *Proceedings of the 22nd ACM conference on Hypertext and hypermedia*. 2011, pp. 93–102.