Semantics-Aware Autoencoder

Department of Electrical and Information Engineering
Polytechnic University of Bari
Ph.D. Program
SSD: ING-INF/05 - INFORMATION PROCESSING SYSTEMS

Final Dissertation

Semantics-Aware Autoencoder

---

by
Vito Bellini

Supervisor:
Prof. Tommaso Di Noia

Co-ordinator of Ph.D. Program:
Prof. Alfredo Grieco

Course n°32, 01/11/2016 - 31/10/2019

To my family

Contents

Abstract

In the digital era, in which users are overwhelmed by information, it is not easy for them to find what they are looking for. Recommender Systems became a fundamental tool nowadays since they mitigate the information overload problem by filtering relevant content in a personalized fashion to the users. Many algorithms have been developed over the years to tackle the recommendation problem by using different machine learning techniques. Recently, in the past few years, we assisted at the rising of Deep Learning. Many state-of-the-art machine learning algorithms have been outperformed by deep learning techniques and every now and then, new deep learning architectures outperform previous state-of-the-art deep learning techniques. Deep Learning has proven its strength in several fields: Computer Vision, Text-To-Speech, Automatic Machine-Translation, and many others. Very recently, it has been adopted in Recommender Systems field. Even though Deep Learning is effective in tackling the recommendation problem, it doesn't provide any explanation about the recommended items; it works as a black-box. Users, on the other hand, would like to know the reason an item has been recommended to them.

This thesis investigates a new technique that combines the representational power of Neural Networks with Knowledge-Graphs, to provide both effective and explainable recommendations to the users. Specifically, we propose a new method to label hidden units by using a Knowledge-Graph and train a not fully-connected Neural Network to extract users' preferences that are used to compute a recommendation within an explanation.

Experimental results showed, analyzed, and discussed in this thesis, support the validity of the proposed method.

Chapter 1

Introduction

1.1 Motivation

The first Web site can be dated on the 6th of August 1991 when Tim Berners-Lee published for the very first time a web page, at the CERN of Ginevra, where he worked as a researcher. The web was born as a solution to share knowledge and scientific findings with other researchers across the world. Since then, the web has been evolving: technologies and market changed and user behavior as well. The evolution of the web can be summarized in three phases: Web 1.0, Web 2.0, and Web 3.0.

Web 1.0 has been characterized by static generated content, produced and published on web portals; the interaction is one-way between the visiting user and the web site. The term Web 2.0 has been born during the first O'Reilly Media Web 2.0 Conference on October 2004. It was just a "slogan" to denote a change in the web paradigm. It indicates a new way in which users and web interact; the technologies which the web relies on was almost the same. What is changed is that now, users became not only contents' consumer, but they produce new content. Prominent examples of this new paradigm are blogs and social networks, in which the user interacts with other users and moreover, the user begins to rate products on e-commerce web sites. Web 3.0, also known as Semantic Web, denotes a transformation of the World Wide Web in which the published documents (HTML pages, files, images, etc...) are associated to metadata that specifies the semantics context in a

format well-suited to be queried and interpreted by computer agents (web search bots). Because of the massive proliferation of data on the Web, a new issue has been raised, the Information Overload. The information overload refers to the vast quantity of data available to the users that make them hard to find what they are looking for. In order to mitigate this problem, a new class of algorithms has been proposed to filter the information so that users can find relevant content to them. These new techniques are called Recommender Systems; they suggest items that users might be interested in. Many companies have based their business around these systems, for example: Amazon, Netflix, Spotify, and many others.

Recently, Deep Learning approaches are emerging, and several companies leverage the representational power of neural networks to provide effective recommendations to the users. Over the years, Deep Learning techniques outperformed several machine learning state-of-the-art methods. Even though Deep Learning leads to better accuracy, neural networks are black-boxes. Interpreting how the neural networks provide results is still an open question in the research field. Nowadays, interpretability and explainability of neural networks are gaining momentum. Many recommender systems are relying on neural networks, and users want to know why a certain item has been recommended.

The main contribution of this thesis is to provide a new architecture of a particular neural network which turns out to be interpretable and explainable to the users.

1.2 Overview

This thesis is organized as follows:

- **Chapter 1** presents motivation, contributions and publications related to the thesis.
- **Chapter 2** presents Linked Open Data and Knowledge Graphs.
- **Chapter 3** introduces recommender systems and describes the main classes of recommendation approaches: content-based, collaborative filtering, and hybrid method.
- **Chapter 4** introduces Neural Networks and provides an overview on Autoencoders Neural Networks.
- **Chapter 5** presents a Semantics-Aware Autoencoder Neural Network which combines the representational power of Autoencoders with Knowledge Graphs in order to design a not fully-connected architecture which

turns out to be both interpretable and explainable for recommendation scenarios.

- **Chapter 6** describes different explanation techniques in a recommendation scenario and provides a method which relies on Semantics-Aware Autoencoder to provide explanation to the users.
- **Chapter 7** provides a qualitative analysis of Knowlege Graphs in Recommendation scenarios through Semantics-Aware Autoencoders.
- **Chapter 8** describes the overall conclusions of the work presented in this thesis and the potential future work.

1.3 List of publications

This thesis is based on the following publications produced during the Ph.D. program:

- Semantics-Aware Autoencoder
  Bellini, V., Di Noia, T., Di Sciascio, E. and Schiavone, A.
  IEEE Access
- Computing recommendations via a Knowledge Graph-aware Autoencoder
  Bellini, V., Schiavone, A., Di Noia, T., Ragone, A., and Di Sciascio, E.
  Knowledge-aware and Conversational Recommender Systems Workshop 2018 (RecSys 2018)
  CEUR Workshop Proceedings 2018
- Knowledge-aware Autoencoders for Explainable Recommender Systems
  Bellini, V., Schiavone, A., Di Noia, T., Ragone, A., and Di Sciascio, E.
  Proceedings of the 3rd Workshop on Deep Learning for Recommender Systems
- Exploiting Knowledge Graphs for Auto-Encoding User Ratings in Recommender Systems
  Bellini, V., Di Noia, T., Di Sciascio, E. and Schiavone, A.
  IIR 2018 - 9th Italian Information Retrieval Workshop
- Auto-encoding user ratings via knowledge graphs in recommendation scenarios
  Bellini, V., Anelli, V.W., Di Noia, T., and Di Sciascio, E.

Proceedings of the 2nd Workshop on Deep Learning for Recommender Systems

Chapter 2

Linked Open Data

2.1 Introduction

The World Wide Web was born as a media for human beings; this means that the data exchanged across it are unstructured, which in turn are hard to understand by computer agents. The semantic web is an attempt to address this issue. The idea behind the semantic web is to use structured data, modeled by ontologies that computer agents or other software can handle, interpret, and inference over them. Semantic web architecture is composed of differentß levels, as depicted in Figure 2.1.

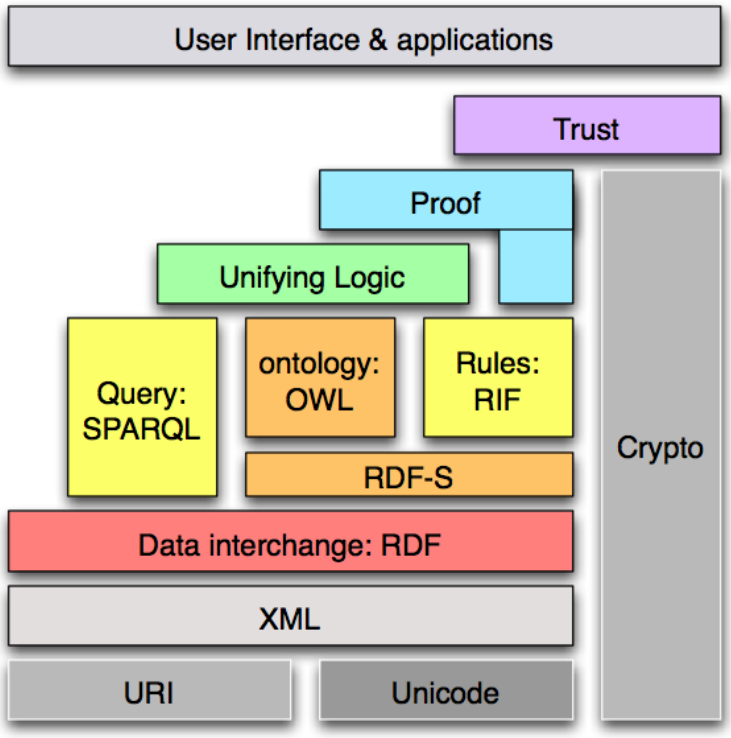


Figure 2.1: Semantic Web architecture.

2.2 Architecture

The first layer contains URI[1] and Unicode, which are technologies of the WWW. Unicode is a standard to encode international characters, independently from the language, platform, and software used. URI is an identifier that allows identifying every resource of the web, such as documents or images. Moreover, a URI can be extended to a URL[2] when it identifies not only a resource of the web, but it also specifies the protocol used to access the resource. The XML layer, the XML namespace, and XML schema provide a standard syntax for the semantic web. XML forces structured data to be uniform and independent from the application or manufactures. XML is a language to represent document and structured data; its syntax is rigid but flexible that allows representing very complex structured data. An XML document contains elements that have attributes and data. Nevertheless, elements, in turn, can be nested to represent complex data. The

[1]Uniform Resource Identifier
[2]Uniform Resource Locator

XML namespace provides a method to avoid conflicts with elements' name by specifying the vocabulary of the markup. XML Schema is a set of formal rules of an XML document which specify the name for elements and the elements that are allowed in the document, how they can be combined and their values. The schema is defined in a DTD file (Document Type Definition). An XML file is defined as valid if it is compliant with the rules defined in the DTD file, otherwise, it is defined as a well-formed XML file. Another way to represent structured data is RDF that allows representing resource through graph data. RDF is based on triples of subject-object-predicate. Nowadays, all the data in the semantic web are represented through RDF. The above layer consists of ontologies, RDF schema, and query languages. An ontology is a formal description of concepts and relations among them. Ontologies are defined in OWL (Ontology Web Language); therefore, OWL is a language for knowledge representation. RDFS (RDF Schema) allows us to define taxonomies of classes and their properties to create ontologies. In order to query data in the semantic web, a new query language has been introduced, SPARQL (Simple Protocol RDF Query Language). It is a SQL-Like W3C standard language to query RDF databases. Since RDFS and OWL are both build atop RDF, SPARQL can be used to query ontologies and knowledge-bases directly. Nevertheless, it represents a protocol to access RDF data. In the Proof layer, the truthfulness of the new knowledge is deducted from semantic relations through inference processes. Cryptography is used in the Trust layer to mark assertions to the aim to guarantee their authenticity and source.

2.3 RDF

Resource Description Framework is used to represent resources of any type; it is a set of elements and formal rules to describe resources. Any resource is identified by a URI, which might be a not accessible object of the web. Properties are attributes to store within a resource (key-value pairs); every property has its meaning and a set of allowed values can be associated with each resource. Assertions, also known as statements, have a subject-predicate-object resource, as shown in Figure 2.2. RDF statements represent the association between a property and a resource. RDF Schema allows validating values for properties. Differently from XML Schema, it doesn't force any constraints on the structure of the document; it provides a way to interpret the meaning of the document itself.

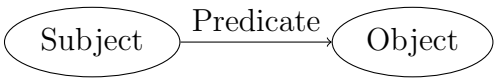


Figure 2.2: RDF tripe example.

An RDF model is represented through an oriented graph in which nodes and edges respectively denote resources and properties, as depicted in Figure 2.3.

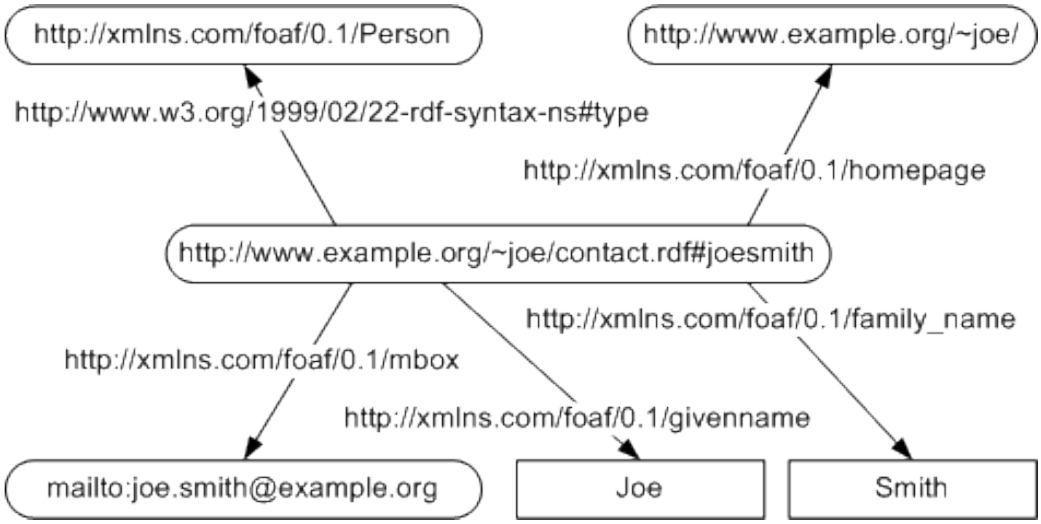


Figure 2.3: RDF graph.

As stated above, an object can be a resource or a literal value. For example, the resource `http://www.example.org/~joe/contact.rdf#joesmith` (subject) through the predicate `http://xmlns.com/foaf/0.1/family_name` specifies that "Smith" is the last name (object). The literal value represents a string and not a resource.

2.4 Linked Open Data

The term Linked Data has been coined by Tim Berners-Lee in 2009, to which it followed the term Linked Open Data (LOD). According to Berners-Lee, information has not only to be easily accessible but also structured and interconnected to be automatically processed by a software agent. Through a semantic structure, raw data gets a higher value and real meaning.

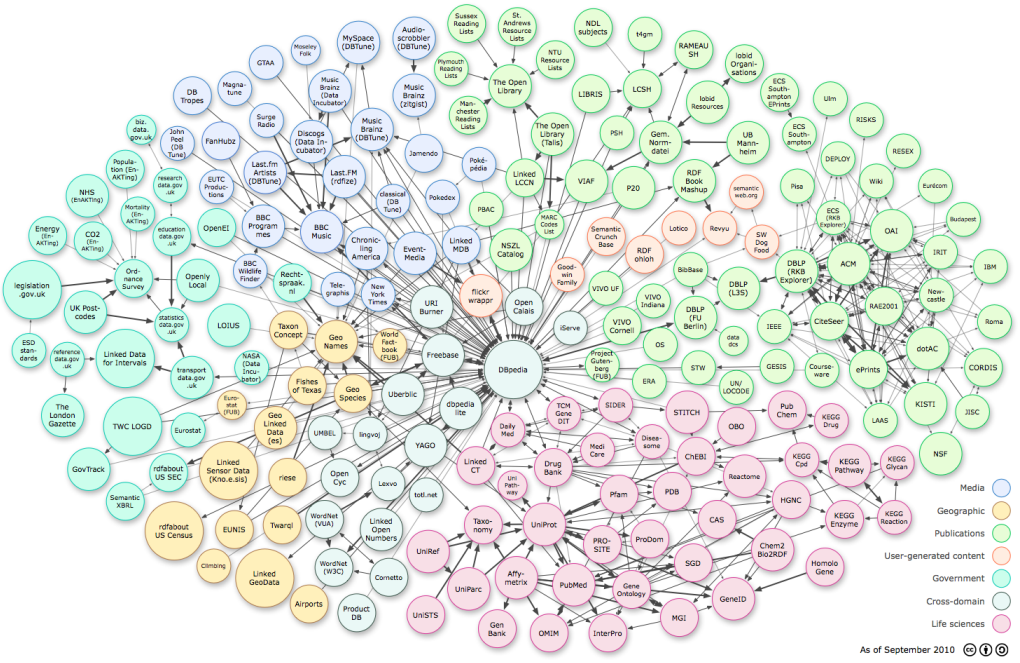


Figure 2.4: Linked Open Data cloud.

The idea of LOD is strictly related to the Semantic Web; even tough the Semantic Web is much more than Linked Open Data, they can be seen as a tool on which the Semantic Web is built upon. In order to use LOD for building the Semantic Web, there are some rules to follow that allow the creation of an accessible layer of content for automatic software agents.

Tim Berners-Lee, drawn up a list of principles in order to publish data as Linked Data[3]. The Linked Data Principles are the followings:

- use URIs as names for things;
- use HTTP URIs so that people can look up those names;
- when someone looks up a URI, provide useful information, using the standards (RDF, SPARQL);
- include links to other URIs. so that they can discover more things.

The RDF language identifies resources through URI; resources can be documents, real-world objects, or abstract concepts. In order to comply with the Linked Data Principles, a way to represent resources is needed. RDF alone doesn't provide a mechanism to access those resources. HTTP,

[3] `https://www.w3.org/DesignIssues/LinkedData.html`

on the other hand, represents a widely used mechanism to access resources on the Web. Therefore, using URIs and HTTP protocol allow us to combine a globally unique identification method with a simple and widely adopted well-known mechanism to fetch resources. According to the second principle, resources have to be identified by using URIs through the HTTP protocol, in order to obtain a description of the object identified by the URI. The fourth principle instead, requires the use of links to create interconnections among resources. In this way, we can connect resources across different paths to explore the graph. Ontologies define the types of resources and how they can be connected to other resources according to the domain they belong to.

In the past few years, LOD had tremendous growth, as shown in Figure 2.4.

2.5 Knowledge Graph

Several definition for KGs have been proposed in the literature as summarized in [1], in which authors propose the following new definition:

> *A knowledge graph acquires and integrates information into an ontology and applies a reasoner to derive new knowledge.*

Concerning the work presented in this thesis, we can state that Knowledge Graphs (KGs) encode the human knowledge within graph structured data in multiple domains by linking different domain-related KGs. In a recommendation scenario, KGs are a massive source of information that can be leveraged to produce accurate recommendations. The publication and spread of freely available Knowledge Graphs in the form of Linked Open Data datasets, such as DBpedia [2], has paved the way to the development of knowledge-aware recommendation engines in many application domains and, still, gives the possibility to easily switch from a domain to another one by just feeding the system with a different subset of the original graph.

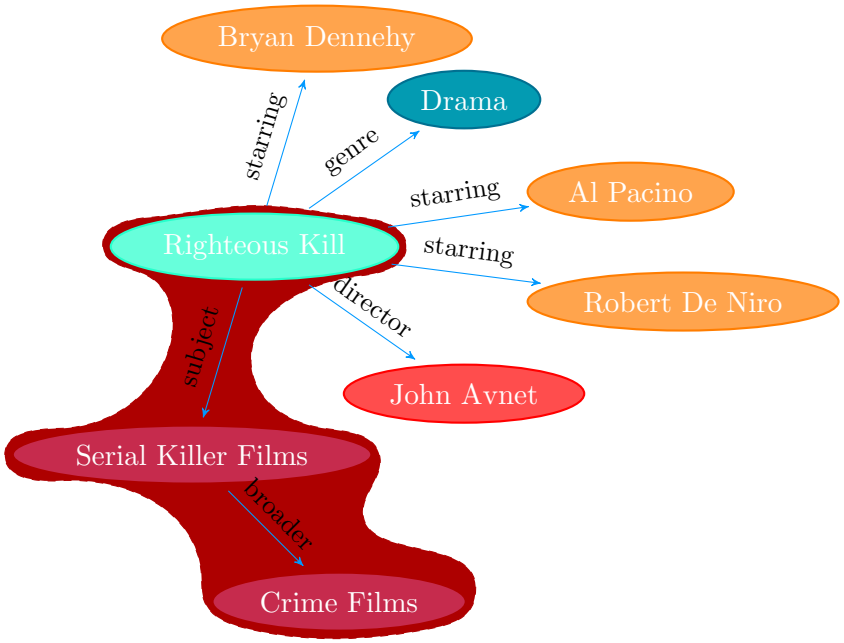


Figure 2.5: Example of Knowledge Graph in movie domain.

In the past few years, several works have been proposed to leverage the side information coming from KGs to enhance the performance of recommender systems. Most of them rely on the usage of DBpedia as KG. In [3], authors leveraging the knowledge encoded in DBpedia, they show that it is possible to build an accurate content-based recommender system. For the very first time, in [4], a LOD-based recommender system is proposed to alleviate some of the major problems that affect collaborative techniques mainly the high sparsity of the user-item matrix. The effectiveness of such an approach seems to be confirmed by a large number of methods that have been proposed afterward. In [5], a detailed review of LOD-based recommender systems is presented. It is worth noticing how KGs are recently being used in lots of applications; they freely offer a large amount of structured data which turned out to be very useful also in recommendation scenarios.

Chapter 3

Recommender Systems

3.1 Introduction

Recommender Systems (RSs) are techniques and algorithms that provide users with personalized suggestions about items they might be interested in [6]. The term item indicates what the system recommends to the users: movies, songs, news articles or products to buy, to cite a few. RSs aim to mitigate the information overload problem: they support users in the choice of the next items to consume among a vast number of alternatives. In order to generate a recommendation list for each user, RSs leverage the past users' interaction with the system: ratings, consumed items, or implicit feedback inferred by users' actions (clicks, query searches, mouse movements, time spent listening to a song). Different techniques exist to tackle the recommendation problem; they are classified into three main categories:

- **Content-Based**: recommends items similar to items the user liked in the past; the similarity is calculated considering the description of the items (content).
- **Collaborative Filtering**: recommends items leveraging the users' behavior and suggests items liked by other users with similar tastes; the similarity is computed on the rating history of the users.
- **Hybrid**: combines both content-based and collaborative filtering tech-

niques in order to get the best from the two and mitigate their respective weaknesses.

Several definitions for the recommendation problem have been proposed in the literature, although the most adopted is [7]. Formally, the recommendation task consists in estimating the rating $r(u, i)$ for an unseen item i to the user u through a recommendation function $r^* : \mathbf{U} \times \mathbf{I} \rightarrow \mathbf{R}$ where $\mathbf{U}$ and $\mathbf{I}$ are respectively the set of users and the set of items. Hence, the problem consists in estimating for each user $u \in \mathbf{U}$ those items $i^{max,u} \in \mathbf{I}$ that maximize the utility function r^*, as formally described in Equation 3.1:

$$\forall u \in \mathbf{U}, i^{max,u} = arg \max_{i \in \mathbf{I}} r^*(u, i) \tag{3.1}$$

As pointed out by [6], the user profile is represented by a set of feedback between users and the system. Feedback are explicit when they are stated by the user (rates, likes), or they are implicit when acquired by analyzing clicks, search queries, or time spent on web pages. Explicit feedback such as ratings are in different forms, many web site and e-commerce use the 5-star scales. However, other forms of explicit feedback are binary feedback (e.g., like/dislike) or unary positive-only are becoming popular thanks to social networks and content sharing websites. Implicit feedback, on the other hand, are inferred by analyzing users' behavior, and they are particularly useful when explicit feedback are not available. Nevertheless, they can be leveraged together with explicit feedback in order to provide even more effective recommendations.

3.2 Content-Based

Content-Based (CB) recommender systems, recommend items similar to those the user liked in the past. Therefore, a CB recommender system has to match attributes in the user profile, which contains the user's preferences, with the item's attribute, to recommend relevant items to the user.

3.2.1 Architecture

CB recommenders need specific techniques to represent items and to generate the user profile, as shown in [5].

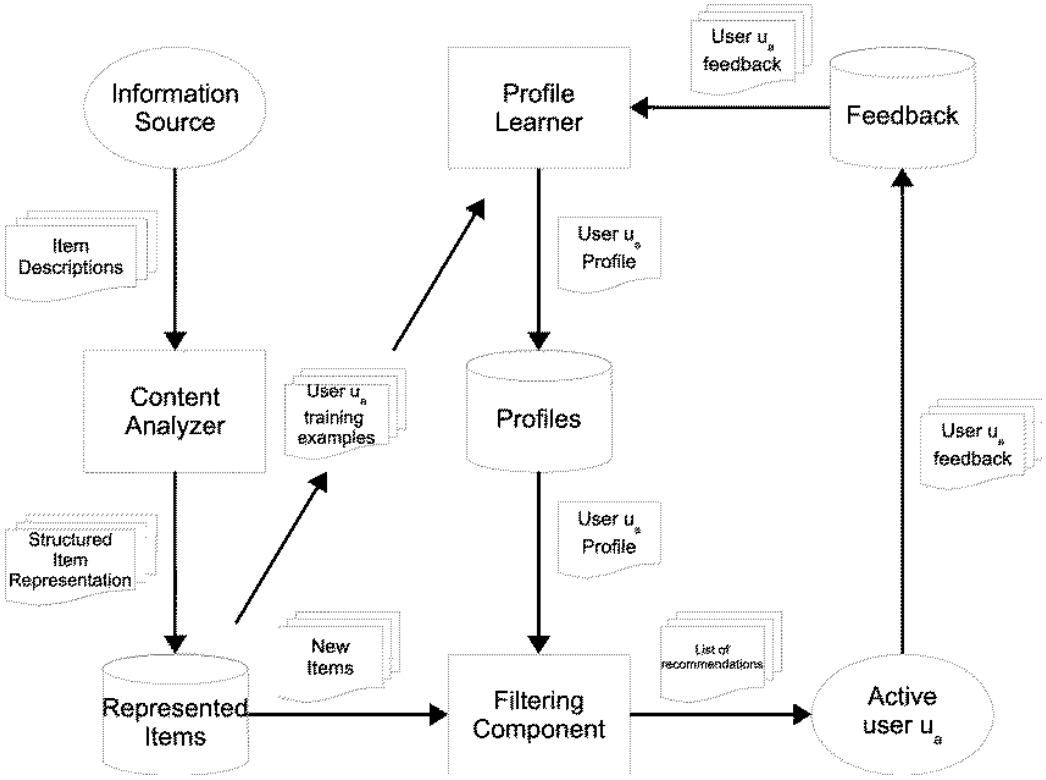


Figure 3.1: High level architecture of a content base recommender.

A CB recommender process can be split in three different phases:

- **Content Analyzer**: pre-processes unstructured data such as the text of a review, to extract relevant structured data; it provides as output a representation that describes an item in terms of features which are later used by the Profile Learner and the Filtering Component.
- **Profile Learner**: gets data that represent the user preferences and generalizes over them in order to build the user profile; it leverages on machine learning algorithms to infer a model by using preferences over items given by the user.
- **Filtering Component**: leverages the user profile to recommend relevant items by matching the representation of the user profile and items; the most widely used similarity technique is the cosine similarity.

The very first step in a CB recommendation task is done by the Content Analyzer which exploits Information Retrieval techniques to extract features from the description of items, in order to generate a structured representation

to store in the Represented Items component. In order to build and keep the user profile updated for the user, her feedback for the recommended items are stored in the Feedback repository. Feedback are exploited in the learning phase of the model, which learns a function to predict the user's relevance for the suggested items. Usually, feedback are of two kinds: positive and negative. Moreover, they can be explicit or implicit. Explicit feedback consist of explicit item ratings given by the user; on the contrary, implicit feedback don't require the active involvement of the user, since they are inferred by analyzing the user's actions. Explicit feedback have the advantage of simplicity, but they can not get the emotions of the user on an item, on the other hand, implicit feedback have the advantage that they don't require active interactions.

In order to generate the user profile for the user u_a, a training set TR_a for the user u_a has to be defined. TR_a is a set of pairs $< I_k, r_k >$ where r_k is the rating given by the user u_a to the item I_k. The Profile Learner, using supervised learning algorithms, builds a predictive model which is stored in the profile repository. Given a representation for an item, the Filter Component predicts whether the item can be of interest for the user u_a. Generally, this component uses different strategies to rank items by their relevance to the user profile. Most relevant items are placed in the recommendation list L_a, which is presented to the user u_a. User's tastes change over time; that is the reason why the user profile has to keep updated in order to provide effective recommendations.

3.2.2 Vector Space Model

The simplest and widely adopted way to represent items through features in a CB recommender is to represent them in a Vector Space Model (VSM). Each item is so represented by an n-dimensional vector, where each dimension corresponds to a feature. Features are then weighed to indicate the relevance of them with the item.

Let $D = \{d_1, d_2, \ldots, d_n\}$ the set of items and $T = \{t_1, t_2, \ldots, t_n\}$ the vocabulary of all the features contained in D. Each item d_j is represented as a vector in a vector space of dimension n, so that $d_j = \{w_{1,j}, w_{2,j}, \ldots, w_{n,j}\}$ where $w_{k,j}$ is the weight for the feature t_k in the item d_j.

In order to represent items in a VSM, it is important to define how to weigh features and how to compute a similarity score between two vectors. The most used technique to weigh feature is the TF-IDF (Term Frequency-Inverse Document Frequency). It was born from two empirical observations from the Information Retrieval field [8]:

- rare terms (features) are not less important than frequent terms (IDF);
- multiple occurrences of a term (feature) in a document (item) are not less important of a single occurrence (TF);
- long documents (items) should not be preferred to short document (normalization).

More informally, features that frequently occur in an item (TF) but rarely in all the other items (IDF) are probably the most important to discriminate the item. Normalization helps to avoid giving too much consideration at longest documents because the same features might be repeated over and over. Formally, the TDF-IDF is described the following equation:

$$TF - IDF(t_k, d_j) = TF(t_k, d_j) \log \frac{N}{N_k}$$
$$TF(t_k, d_j) = \frac{f_{k,j}}{\max_z f_{z,j}}$$

where N and n_k are respectively the number of items and number of items that have the feature t_k and $\max_z f_{z,j}$ is computed on the frequencies $f_{z,j}$ for all the features t_z that occur in the item d_j. Finally, weights are normalized in the range $[0, 1]$ according to the following equation:

$$w_{k,j} = \frac{TF - IDF(t_k, d_j)}{\sqrt{\sum_{s=1}^{|T|} TF - IDF(t_s, d_j)^2}} \tag{3.2}$$

To determine a similarity score between two items, the most adopted measure is the cosine similarity, described by Equation 3.3:

$$sim(d_i, d_j) = \frac{\sum_k w_{k,i} w_{k,j}}{\sqrt{\sum_k w_{k,i}^2 \sum_k w_{k,j}^2}} \tag{3.3}$$

Recommender Systems that are based on a VSM to represent both users and items, leverage a distance measurement between users vector and items vector to predict the relevance of an item for a user: the higher is the affinity, the lower is the distance.

3.2.3 Advantages and Disadvantages

Information Retrieval and Artificial Intelligence have inspired research on CB recommender systems. It gets techniques to extract items' features from

unstructured data from the Information Retrieval field. Conversely, it gets from the Artificial Intelligence field, machine learning methods to build a model starting from the users' profile.

CB filtering has many advantages with respect to collaborative filtering techniques:

- **Independence from users**: it leverages the ratings only to build the user profile, on the contrary, collaborative filtering methods need to keep track of the users' ratings to find out similar users.
- **Transparency**: it easy to explain how the RS is working and why it recommends items, since the recommendation it is based on the extracted features; on the other hand, collaborative filtering methods are like black-boxes, the only explanation they can provide is that the item is being recommended because similar users liked it.
- **Cold Start**: it recommends new items that haven't already rated since it is based on features and not on explicit ratings as the collaborative filtering is; without a decent number of ratings, the collaborative filtering methods are not able to recommend new items while content-based techniques work well.

However, there are also some downsides:

- **Feature Engineering**: it requires knowledge on the recommender's domain of application; items have to be described by enough features to allow the recommender to provide effective recommendations.
- **Serendipity**: CB recommenders are prone to overfitting; they provide the same kind of recommendations over and over again with a limited novelty and no serendipity at all because the recommendations are computed on the features contained in the user profile.
- **Cold users**: it requires a decent number of ratings in order to correctly model the user's preferences; a user with few ratings gets inaccurate recommendations.

CB recommenders are based on the features extracted from unstructured data, but features suffer from polysemy and synonymy. The former indicates a word with different meanings; the latter denotes different word with the same meaning. Because of synonymy, relevant information might be lost if the user profile doesn't contain the exact keyword (feature); on the other hand, because of polysemy, some items might be relevant even if they are not.

In order to solve these problems, a new innovative approach is the Semantic Analysis. The base idea is to exploit knowledge bases to annotate items, and to obtain a semantic representation of the users' profile.

3.3 Collaborative Filtering

The aim of Collaborative Filtering (CF) approaches is to predict the relevance of the item i for the user u leveraging the ratings of similar users. More formally, the relevance $r(u, i)$ of the item i for the user u is computed on the ratings $r(u_j, i)$ given by other users $u_j \in \mathbf{U}$ on the item i. In a movie recommender system, in order to suggest a movie to the user u, a CF approach finds peer users of u which are those users that share the same tastes of u because they rated the same movies in a similar fashion. The predicted rating of the item i for user u is, therefore, a function of the rating that similar users of u gave to the item i.

CF techniques have advantages over the CB methods:

- a description for items is not required;
- items are recommended on feedback received from other users and not on the features of them;
- it provides serendipitous recommendations, a CF recommender system might suggest very different items with respect to the ones the user is used to like, it introduces a novelty in the recommended list.

CF algorithms are classified into two categories: memory-based (heuristic-based) and model-based, as pointed out by [7, 9, 10, 11]. The former techniques compute recommendations on-the-fly, those algorithms don't need to train and to store a model to provide recommendations; instead, the latter methods, require to train a model.

3.3.1 Nearest-Neighbor

This approach is a memory-based method since it doesn't require a model to be trained in order to provide recommendations [10]. It exploits heuristics to predict ratings of unseen items; hence, predictions depend on other user ratings. Nearest-Neighbor approach are grouped in user-based and item-based. User-based methods [12, 13, 14] exploit statistics on ratings to find sets of users, also called neighbors, that have rated items similarly; they infer the rating of the item i for user u by using rates that neighbors of u gave to the item i. Item-based methods, on the other hand, predict the rating

for the item i of a user u based on the ratings of u for items similar to i, as shown in [11, 15].

Nearest-Neighbor methods are called k-Nearest-Neighbor (k-NN) where k is the size of the neighborhood. In a user-based approach, the rate of user u for the item i is predicted as described in Equation 3.4.

$$\hat{r}(u,i) = \frac{1}{|N_i(u)|} \sum_{v \in N_i(u)} r(v,i) \tag{3.4}$$

where $N(u)$ is the set of the most similar users to the user u that have rated the item i. A limit of Equation 3.4 is that it doesn't take into account the similarity among users. A widely adopted solution is to weigh the contribution of each user by its similarity to the user u, as described by Equation 3.5. In this case, since weights might don't sum to 1, it is necessary to normalize these weights.

$$\hat{r}(u,i) = \frac{\sum_{v \in N_i(u)} w_{u,v} r_{v,i}}{\sum_{v \in N_i(u)} |w_{u,v}|} \tag{3.5}$$

Users may use different rating values for the same level of satisfaction for an item. Unfortunately, Equation 3.5 doesn't take it into account. To address this issue, a common approach is to normalize neighbors' rating $r_{v,i}$ to $h(r_{v,i})$ as shown in [10, 16]. The predicted rating is computed as follows:

$$\hat{r}(u,i) = h^{-1}\left(\frac{\sum_{v \in N_i(u)} w_{u,v} h(r_{v,i})}{\sum_{v \in N_i(u)} |w_{u,v}|}\right) \tag{3.6}$$

On the other hand, item-based approaches leverage the ratings given to similar items, as investigated in [11, 15]. Let be $N_u(i)$ the set of most similar items to the item i rated by the user u. The rating of u for i is predicted as weighted average of the ratings given by u on the items of $N_u(i)$ as shown in Equation 3.7.

$$\hat{r}_{u,i} = \frac{\sum_{j \in N_u(i) w_{i,j} r_{u,j}}}{\sum_{j \in N_u(i) |w_{i,j}|}} \tag{3.7}$$

Analogously to Equation 3.6, in order to take into account the differences in the users' individual rating scales, Equation 3.7 can be normalized as follows:

$$\hat{r}_{u,i} = h^{-1}\left(\frac{\sum_{j \in N_u(i) w_{i,j} h(r_{u,j})}}{\sum_{j \in N_u(i) |w_{i,j}|}}\right) \tag{3.8}$$

Choosing between user-based or item-based method depends on different aspects. Regarding the accuracy, item-based approaches produce more accurate recommendations when the number of users is much greater than the number of items, as shown in [17]. On the other hand, when the number of items is greater than the number of users, user-based approaches work better, as studied in [18]. Taking into account the efficiency, when the number of users exceeds the number of items, item-based methods are faster than user-based ones, since they require less memory and time to compute similarity weights. Regarding the stability, if the catalog of items doesn't change in comparison to the users of the system, an item-based approach is preferable since items' similarities can be computed periodically saving both time and computational costs. Justify a item-based recommendation is easier than a user-based one, because the user, in this case, is aware that items are being recommended because of they are similar to other items liked by the user; conversely, a user-based recommendation is less amenable to be explained because the active user doesn't know about other users served as neighborhood in the recommendation process. Finally, user-based approaches turn out to generate more serendipitous recommendation than item-based ones. Item-based approaches rely on items' similarity; hence, they recommend items that are similar to those already liked by the users. For instance, in a movie recommender, they might recommend items of the same genre or by the same director. On the other hand, user-based methods are more likely to generate serendipitous recommendations when a small number of neighbors k is used.

3.3.2 Model-Based

Memory-based approaches, differently from model-based ones, exploit the ratings to learn a model that predicts the rate for the item i of user u. Several approaches have been proposed in the literature, such as Latent Dirichlet Allocation [19], Neural Networks [20], Factorization Machines [21] and many others. Recently, Matrix Factorization (MF) models gain popularity after the Netflix Prize[1] competition. These models, get the inspiration from the SVD applied in the Information Retrieval field, in which SVD is used to identify latent semantic factors [22]. MF models map users and items together in a latent factor space of dimensionality f in which user-item interactions are modeled as inner products in the aforementioned space. In that space, latent factors are learned automatically from user feedback. More formally, each item i is represented with a vector $q_i \in \mathbf{R}^f$ and each user u is represented with a vector $p_u \in \mathbf{R}^f$ such that the dot product of those vectors is the

[1] `https://www.netflixprize.com`

expected rating:

$$\hat{r}(u,i) = q_i^T p_u \tag{3.9}$$

Vectors q_i and p_u are learned in such a way that the squared error difference between their dot product and the actual rate (ground-truth) in the user-item ratings matrix is minimum:

$$\min(p,q) = \sum_{u,i \in f} (r(u,i) - q_i^T p_u)^2 \tag{3.10}$$

To reduce the error between the predicted and actual value, the algorithm makes use of the baseline (biases) predictor which leverage some characteristics of the dataset. In particular for each user-item (u,i) pair, three parameters are being exploited:

- μ the overall average rating;
- b_i the average rating of item i - μ;
- b_u the average rating given by user u - μ.

Adding the baseline predictor to Equation 3.10, the final function to minimize is the following:

$$\min(p,q,b_i,b_u) = \sum_{u,i \in f} (r(u,i) - q_i^T p_u - \mu - b_i - b_u)^2 \tag{3.11}$$

Typically, to minimize Equation 3.11, either stochastic gradient descent [23, 20, 24] or alternating least squares are performed.

3.4 Hybrid

Content-Based and Collaborative Filtering techniques have complementary strengths and weaknesses. Obtaining the best from the two methods is achievable by combining them together. In a movie recommender system, recommendations could be computed by taking into account both movies that similar users liked (CF) and movies with features liked by the users (CB).

Different hybridization techniques have been proposed and a comprehensive survey is given in [25]:

- **Weighted**: it weigh the score given by different recommenders; the simplest strategy consists in equally weighing both the CB and CF scores and then adjust the weights depending on the users' feedback;

- **Switching**: the recommender choses which criteria to adopt in order to compute the recommendation; it switches from CB to CF, for instance, whether the recommendation is not enough accurate;
- **Mixed**: it combines together recommendations from several recommenders into the same list by means of a ranking or combination strategy;
- **Feature combination**: it uses features derived from different knowledge sources (e.g. content and collaborative features); this technique allows using CF features without relying only on them so that the system is less affected by the cold-start problem;
- **Cascade**: it leverages several recommenders to refine the ranking of the recommendation list: the ranked list provided by the first recommender is then re-ranked by the subsequent recommenders in order to produce a more and more accurate ranking;
- **Feature augmentation**: the output of a recommender augments the feature space of the subsequent recommender;
- **Meta-level**: the model generated by one recommender is used as input by a principal recommender.

In this thesis, are of particular interest two types of methods: Feature combination and Feature augmentation.

3.5 Evaluation

Evaluating a recommender system is a difficult task. The same algorithm may perform better or worse, depending on the dataset. Different aspects of a recommender require different settings and metrics for the evaluation. Adopting a rigorous evaluation strategy is necessary to obtain correct results and providing all the details about the experiments is crucial to compare different algorithms and for the reproducibility of experiments. A recommender system can be evaluated either offline or online. In the former case, past data are used to train and test the recommender. In this setting, the dataset is partitioned in train and test set [26]. Usually, a spitting strategy is applied for each user in the dataset, in order to split her ratings in both train and test set. A common practice is to user 80% of the ratings for the train set and the remaining 20% for the test set. The training set is used to train a model, while the test set is performed only to evaluate the recommender

system. Performance metrics are computed comparing recommendation lists with test data. In an online evaluation, a recommender system is through an A/B test. Users are split into two groups: the control group and the variation group. To the former, recommendations from the baseline are provided, conversely, to the other group recommendations from the new algorithm are provided. Several metrics are taken into account for the evaluation. In this work, we focus on the offline evaluation that allows faster experimentation without impacting the user experience on the production system.

3.5.1 Accuracy metrics

To evaluate the accuracy of a recommender, several metrics have been proposed. In this thesis, we focus on *Precision@N*, *Recall@N*, *F1-Score* and *nDCG@N*.

Precision@N represents the fraction of relevant items in a top-N recommendation. Let $rel(u, i)$ be a boolean function that represents the relevance of item i for the user u, with the value of 1 for a relevant item and a value of 0 for a non-relevant one. *Precision@N* is computed as follows:

$$Precision@N = \frac{\sum_{i=1}^{N} rel(u, i)}{N} \tag{3.12}$$

Recall@N denotes the fraction of relevant items from the test set that occur in the top-N recommendation list. Being $test(u)$ the set of relevant items in the test set for the user u, *Recall@N* is defined as:

$$Recall@N = \frac{\sum_{i=1}^{N} rel(u, i)}{|test(u)|} \tag{3.13}$$

Precision and Recall can be combined with each other in the F1 measure computed as the harmonic mean between them, as follows:

$$F1@N = 2\frac{Precision@N Recall@N}{Precision@N + Recall@N} \tag{3.14}$$

Precision@N, *Recall@N*, *F1-Score* are not rank-sensitive. *nDCG@N*, differently from the aforementioned metrics, takes into account the position of a relevant item in the recommendation list. Formally, it is defined as follows:

$$nDCG@N = \frac{1}{iDCG} \sum_{i=1}^{N} \frac{2^{rel(u,i)} - 1}{\log_2(i + 1)} \tag{3.15}$$

where $iDCG$ is a normalization factor that sets $nDCG@N$ value to 1 when an ideal ranking is returned [27].

Chapter 4

Deep Learning

4.1 Introduction

Deep Learning (DL) is a field of Artificial Intelligence, based on a hierarchical representation on multiple layers, where concepts of a layer are defined upon the previous layer. Even tough the term Deep Learning is relatively new, its history can be traced back to 1943, when Walter Pitts and Warren McCulloch created a computer model based on the neural networks of the human brain. They used a combination of algorithms and mathematics they called "threshold logic" to mimic the thought process. Since that time, Deep Learning has evolved steadily, with only two significant breaks in its development. At that time, DL models were called Artificial Neural Networks (ANNs) because they were intended to be computational models of biological learning. The modern term Deep Learning goes beyond that, it is a more general principle of learning multiple levels of composition (deep) which can be applied in machine learning frameworks that are not necessarily inspired by the human brain.

4.2 A single neuron

The basic unit of computation in a neural network is the neuron, also called a node or unit. It receives input from some other neurons, or from an external

source and computes an output. Each input has an associated weight w, which is assigned on the basis of its relative importance to other inputs. The node applies a function f to the weighted sum of its inputs as shown in Figure 4.1.

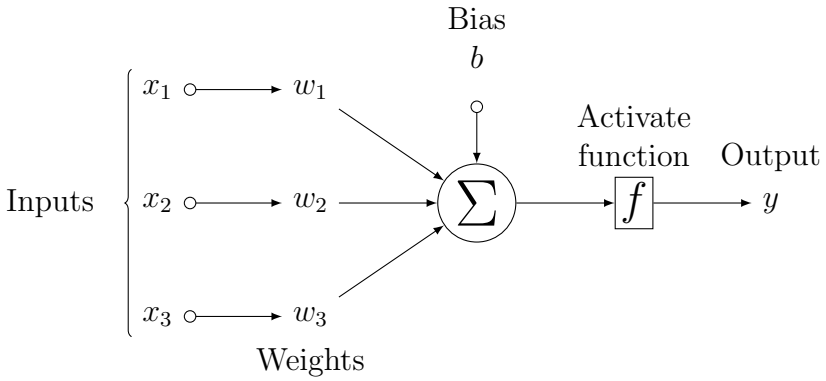


Figure 4.1: Structure of a single neuron.

The output of the neuron depicted in Figure 4.1 is computed as follows:

$$y = f(w_1x_1 + w_2x_2 + w_3x_3 + b) \tag{4.1}$$

The above network takes numerical inputs x_1, x_2 and x_3 and has weights w_1, w_2 and w_3 associated with those inputs. Additionally, there is another input 1 with weight b (bias) associated with it. The role of the bias term is to allow the activation function f to shift to the left or right, which may be critical for successful learning.

The output y from the neuron depicted in Figure 4.1 is computed as shown in the Equation 4.1. The function f is non-linear and it is called the Activation Function. The purpose of the activation function is to introduce non-linearity into the output of a neuron. This is important because most real world data is non linear and we want neurons to learn these non linear representations.

Every activation function (or non-linearity) takes a single number and performs mathematical operations on it. Different non-linearities have been proposed over the years, as shown in Figure 4.2. The most common activation functions are:

- **Sigmoid**:

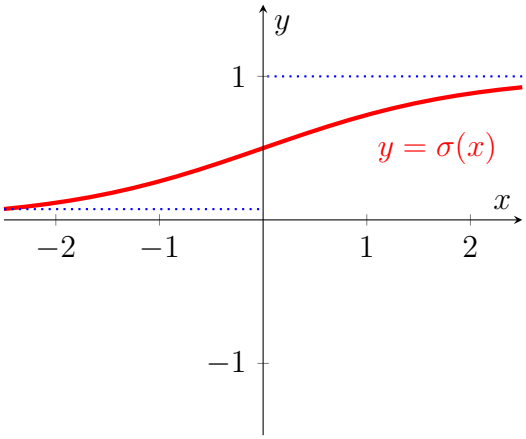
$$\sigma(x) = \frac{1}{1 + \exp^{-x}}$$

- **Hyperbolic Tangent**:

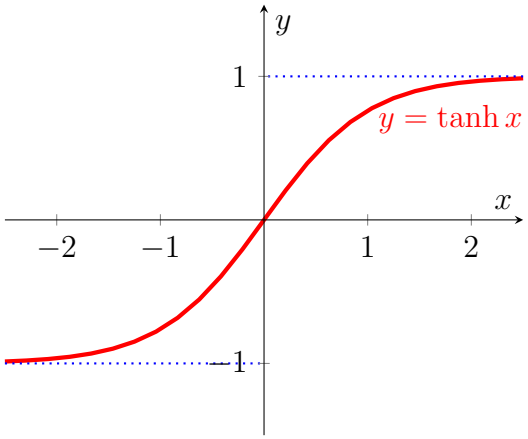
$$tanh(x) = 2\sigma(2x) - 1$$

- **Rectified Linear Unit**:

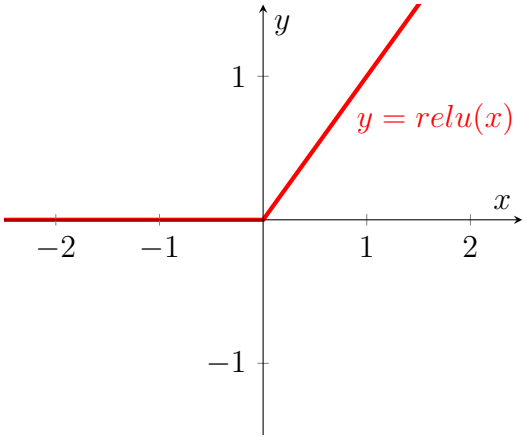
$$relu(x) = \max(0, x)$$



(a) Sigmoid



(b) Hyperbolic Tangent



(c) Rectified Linear Unit

Figure 4.2: Activation functions.

4.3 Neural Networks

An Artificial Neural Network is a computational model that is inspired by the way biological neural networks in the human brain process information. Artificial Neural Networks have generated a lot of excitement in machine learning research and industry, thanks to many breakthrough results in speech recognition, computer vision and text processing.

The feedforward neural network was the first and simplest type of artificial neural network devised. It contains multiple neurons arranged in layers. Neurons from adjacent layers have connections or edges between them. All these connections have weights associated with them, as shown in Figure 4.3.

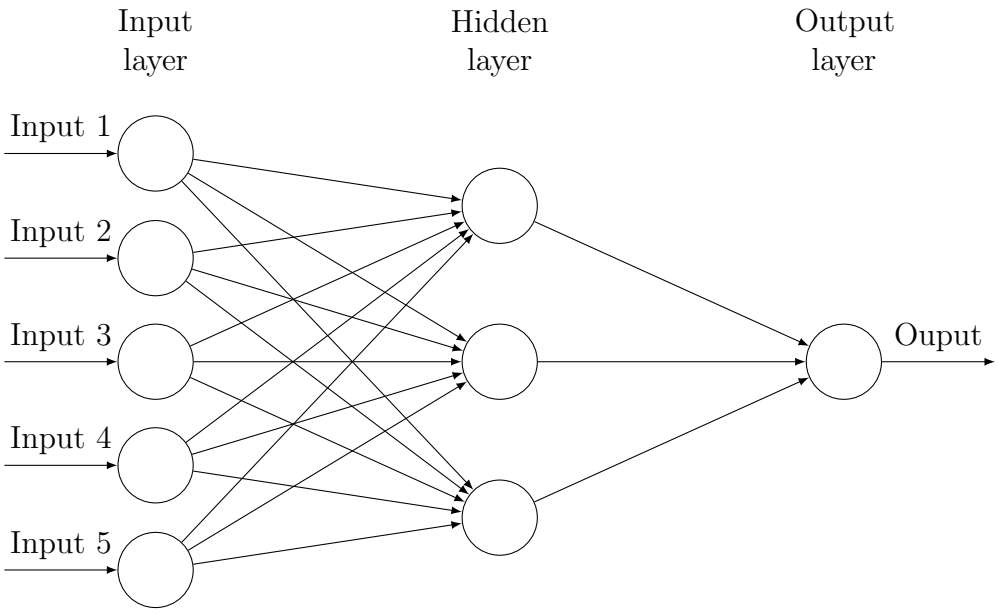


Figure 4.3: Structure of an Artificial Neural Network.

A feedforward neural network has three type of layers:

- **Input layer**: provides information from the outside world to the network; no activation function is applied to the input nodes, they merely pass on the information to the hidden layer;
- **Hidden layer**: it has no direct connection to the outside world (hence the name "hidden"), an activation function is applied on neurons in this layer and the output is passed to the output layer;

- **Output layer**: it is responsible for computations and transferring information from the network to the outside world.

The simplest feedforward neural network is the Single Layer Perceptron which does not contain any hidden layer. On the other hand, a feedforward neural network with one ore more hidden layer is called Multi Layer Perceptron. While a single layer perceptron can only learn linear functions, a multi layer perceptron can also learn non-linear functions.

In this thesis we focus on Autoencoders Neural Networks.

4.4 Autoencoders

An autoencoder neural network is an unsupervised learning algorithm that applies backpropagation, setting the target values to be equal to the inputs. Basically, it tries to reconstruct at the output layer, the input data it is fed with, by using a latent representation of the original input data which is encoded in the hidden layer. In other words, it learns an approximation to the identity function, so as to output $\hat{x}$ that is similar to x.

Since an autoencoder reconstructs the input data at the output layer, it means that the number of neurons in the input has to be the same as the output layer, as depicted in Figure 4.4.

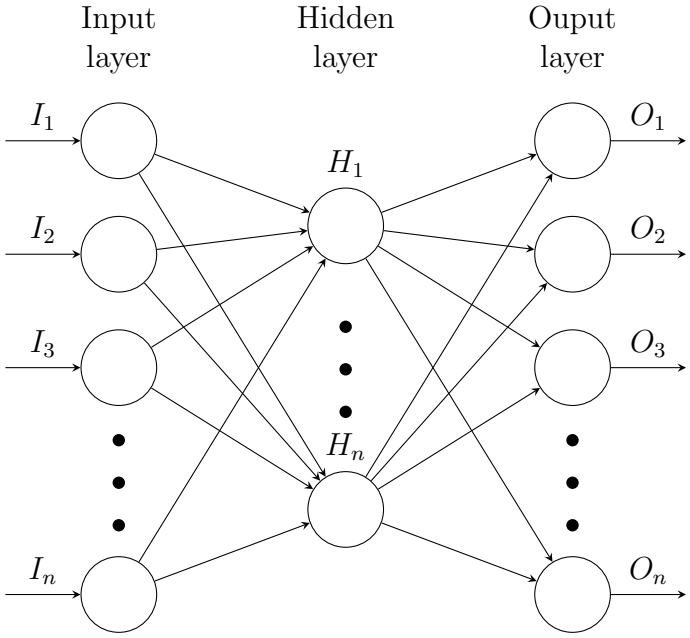


Figure 4.4: Architecture of Autoencoder Neural Network.

Learning an identity function to approximate the network's input data seems trivial; but by placing constraints on the network, such as by limiting the number of hidden units, we can discover interesting structure about the data. Suppose the input x is the pixel values from a 10×10 image (100 pixels) and there are 20 hidden units in the hidden layer. The network has to reconstruct the input data at the output layer by using only 20 hidden units. It turns out that the network is forced to learn a compressed representation of the input. An autoencoder is able to discover correlations in the input features; in fact, a simple autoencoder as the one shown in Figure 4.4 learns a low-dimensional representation very similar to the PCA algorithm.

Usually, autoencoders are trained with only a single hidden layer even though this is not a requirement. Nowadays, in the era of Deep Learning, it is quite common to deal with Deep Autoencoders with several hidden layers. The main advantage of using several hidden layers is that they approximate better any mapping from input to latent representation, given enough hidden units. Deep Autoencoders lead to better compression then corresponding shallow, as investigated in [28].

4.4.1 Applications of Autoencoders

Autoencoder have been successfully applied to dimensionality reduction and information retrieval task. Lower-dimensional representations are usually obtained to improve performances on many tasks or barely to speed up the computational time (e.g. using lower-dimensional input vector to train a NN takes less time). Another task that benefits from dimensionality reduction is Information Retrieval where search become efficient in low-dimensional spaces.

Even tough Autoencoders are successfully used in several task, as all neural network models, they work as black-boxes. Explaining the reason behind their prediction is not an easy task.

4.5 Deep Learning for Recommender Systems

Several approaches have been proposed in the last few years, thanks to the popularity Deep Learning is gaining. In Neural Network Matrix Factorization (NNMF) [29], authors propose the old fashioned matrix factorization in a deep learning re-visitation. In another work, DeepFM [30], the well-known Factorization Machine (FM) has been proposed in an end-to-end model which integrates factorization machines and multilayer perceptron. In DeepFM, FM computes linear and pairwise interactions between features while MLP lever-

age non-linearities and deep neural network structure to capture high-order feature interactions. In a similar way, in Wide&Deep [31], the architecture authors propose consists of two main components: the wide component and the deep component. The wide component is a single layer perceptron which can be seen as a linear model. The deep component is a multilayer perceptron. The idea behind Wide&Deep is to combines these two components in order to capture both memorization(wide component) and generalization (deep component). This model has been successfully implemented for App recommendation in Google play. Collaborative Filtering is successfully performed by Autoencoders, as shown in AutoRec [32, 33, 34, 35]. In Collaborative Deep Learning (CDL) [36], authors propose a hierarchical Bayesian model which integrates stacked denoising autoencoder (SDAE) into probabilistic matrix factorization. A session-based recommender based on Recurrent Neural Networks has been proposed in [37], in which authors propose a short session-based data recommender instead of leveraging on long user histories.

A complete survey about Deep Learning techniques applied to the recommendation problem has been proposed by [38].

4.6 Interpretability

Very recently, few attempts have been made in order to design interpretable neural networks. A generic approach, Linear Proxy Models (LIME) has been proposed by [39] in which a generic black-box system is explained by perturbing the input data and then that data is used to construct a local linear model that is leveraged as a simplified proxy for the full model in the neighborhood of the input. Many works focus on Convolutional Neural Networks for image classification. In [40], authors modify the CNN so that each filter in a high convolutional layer represents a specific part of the object. In [41], authors propose the use of attention mechanism combined with CNN fed with reviews in order to predict user ratings. The attention mechanism enables an interpretable representation of users and items. Attention-based mechanisms learn functions that provide a weighting over inputs or internal features to focus on a certain portion of the data, which turns out to be more relevant. Attention approaches have shown remarkable success in the field of Natural Language Processing, as shown in [42] and in imagine classification tasks as presented in another work [43].

Even though the attention mechanism is effective in learning which part of the data is the most relevant, it is not trained for the purpose of creating human-readable explanations.

Chapter 5

Semantics-Aware Autoencoder

In the last years, deep learning has shown to be a game-changing technology in artificial intelligence thanks to the numerous successes it reached in diverse application fields. Among others, the use of deep learning for the recommendation problem, although new, looks quite promising due to its positive performances in terms of accuracy of recommendation results. In a recommendation setting, in order to predict user ratings on unknown items a possible configuration of a deep neural network is that of autoencoders typically used to produce a lower dimensionality representation of the original data. In this paper we present SEMAUTO, an autoencoder that bases the structure of its neural network on the semantics-aware topology of a knowledge graph thus providing a label for neurons in the hidden layer that are eventually used to build a user profile and then compute recommendations. We show the effectiveness of SEMAUTO in terms of accuracy, diversity and novelty by comparing with state of the art recommendation algorithms.

5.1 Introduction

Recommender systems (RS) have become pervasive tools we experience in our everyday life. While browsing a catalog of items RSs exploit users' past preferences in order to suggest new items they might be interested in.

Knowledge Graphs have been recently adopted to represent items, compute their similarity and relatedness [44] as well as to feed Content-Based

(CB) and hybrid recommendation engines [45]. The publication and spread of freely available Knowledge Graphs in the form of Linked Open Data datasets, such as DBpedia [2], has paved the way to the development of knowledge-aware recommendation engines in many application domains and, still, gives the possibility to easily switch from a domain to another one by just feeding the system with a different subset of the original graph.

Another technology that surely boosted the development of a new generation of smarter and more accurate recommender systems is deep learning [46]. Starting from the basic notion of an artificial neural net (ANN), several configurations of deep ANN have been proposed over the years, such as autoencoders.

In this chapter, we show how autoencoders technology can benefit from the existence of a Knowledge Graph to create a representation of a user profile that can be eventually exploited to predict ratings for unknown items. The main intuition behind the approach is that both ANN and Knowledge Graph expose a graph-based structure. Hence, we may imagine building the topology of the inner layers in the ANN by mimicking that of a Knowledge Graph.

5.2 Related Work

Autoencoders and Deep Learning for RS. The adoption of deep learning techniques is for sure one of the main advances of the last years in the field of recommender systems. In [33], the authors propose the usage of a denoising autoencoder performs a top-N recommendation task by exploiting a corrupted version of the input data.

A pure Collaborative-Filtering (CF) model based on autoencoders is described in [32], in which the authors develop both user-based and item-based autoencoders to tackle the recommendation task. Stacked Denoising Autoencoders are combined with collaborative filtering techniques in [47] where the authors leverage autoencoders to get a smaller and non-linear representation of the users-items interactions. This representation is eventually used to feed a deep neural network which can alleviate the cold-start problem thanks to the integration of side information. A hybrid recommender system is finally built.

Wang et al. [48] suggest to apply deep learning methods on side information to reduce the sparsity of the rating matrix in collaborative approaches. In [49] the authors propose a deep learning approach to build a high-dimensional semantic space based on the substitutability of items; then, a user-specific transformation is learned in order to get a ranking of

items from such a space. Analysis about the impact of deep learning on both recommendation quality and system scalability are presented in [50], where the authors first represent users and items through a rich feature set made on different domains and then map them to a latent space. Finally, a content-based recommender system is built.

Knowledge Graphs and Linked Open Data for RS. Several works have been proposed exploiting side information coming from knowledge graphs and Linked Open Data (LOD) to enhance the performance of recommender systems. Most of them rely on the usage of DBpedia as knowledge graph. In [4], for the very first time, a LOD-based recommender system is proposed to alleviate some of the major problems that affect collaborative techniques mainly the high sparsity of the user-item matrix. The effectiveness of such an approach seems to be confirmed by a large number of methods that have been proposed afterward. A detailed review of LOD-based recommender systems is presented in [5]. By leveraging the knowledge encoded in DBpedia, it is possible to build an accurate content-based recommender system [3].

5.3 Semantics-Aware Autoencoder for rating prediction

The main idea of our approach is to map the connections in a knowledge graph (KG) with those between units from layer `i` to layer `i+1`, as shown in Figure 7.1. There we see that we injected only categorical information in the autoencoder and we left out factual one. As a matter of fact, if we analyze these two kinds of information in DBpedia we may notice that:

- the quantity of categorical information is higher than the factual one. If we consider movies, the overall number of entities they are related with is lower than the overall number of categories;
- categorical information is more distributed over the items than the factual one. Going back to movies we see that they are more connected with each other via categories than via other entities.

Hence, we may argue that for a recommendation task where we are looking for commonalities among items, categorical data may result in more meaningful than the factual one. The main assumption behind this choice is that, for instance, if a user rated positively *Cloud_Atlas* this may be interpreted as a positive rating for the connected category *Post-apocalyptic_films*.

In order to test our assumption, we mapped the autoencoder network topology with the categorical information related to items rated by users.

As we build a different autoencoder for each user depending on the items she rated in the past, the mapping with a KG makes the hidden layer of variable length in the number of units, depending on how much categorical information is available for items rated by the specific user.

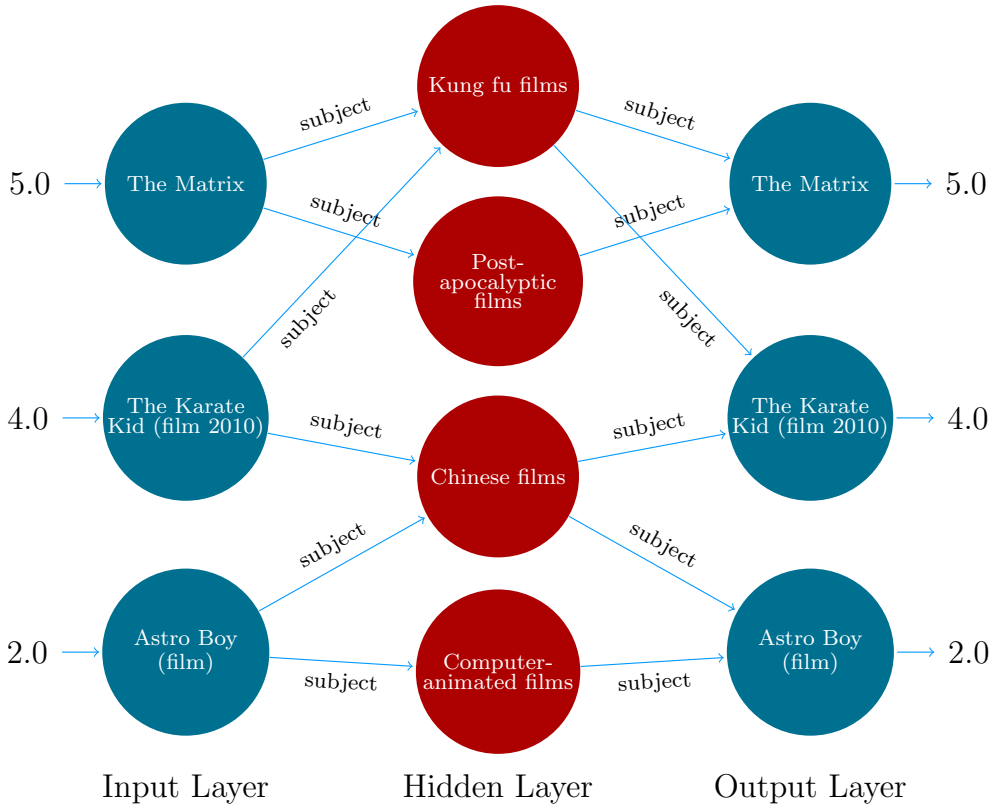


Figure 5.1: Architecture of a semantic autoencoder.

Let n be the number of items rated by u available in the graph and $C_i = \{c_{i1}, c_{i2}, \ldots, c_{im}\}$ be the set of m categorical nodes associated in the KG to the item i. Then, $F^u = \bigcup_{i=1}^{n} C_i$ is the set of features mapped into the hidden layer for the user and the overall number of hidden units is equal to $|F^u|$. Once the neural network setup is done, the training process takes place, feeding the neural network with ratings provided by the user, normalized in the interval `[0,1]`. It is worth noticing that, as the autoencoder, we build to mimic the structure of the connections available in the Knowledge Graph, the neural network we build is not fully connected. Moreover, it does not need bias nodes because these latter are not representative of any semantic data in the graph.

Nodes in the hidden layer correspond to categorical information in the

knowledge graph. At every iteration of the training process, backpropagation will change weights accordingly on edges among units in the layers, such that the sum of entering edges in an output unit will reconstruct the user rating for the item represented by that unit. Regarding the nodes in the hidden layer, we may interpret the sum of the weights associated to entering edges computed at the end of the training process as the importance of that feature in the generation of an output which, in our case, are the ratings provided by the user.

5.3.1 User profiles

Once the network converges we have a latent representation of features associated with a user profile together with their weights. However, very interestingly, this time the features represented by nodes in the hidden layer also have an explicit meaning as they are in a one to one mapping with categories in a knowledge graph. Our autoencoder is, therefore, able to learn the semantics behind the ratings of each user and weight them through backpropagation. In our current implementation we used the well known sigmoid $\sigma(x) = \frac{1}{1+e^{-x}}$ activation function since we normalized the design matrix to be within $[0, 1]$. We trained each autoencoder for 10,000 epochs with a learning rate of $r = 0.03$; weights are initialized to zero close values as Xavier et al. suggest in [51].

Starting from the trained autoencoder, we may build a user profile by considering the categories associated to the items she rated in the past as features and by assigning them a value according to the weights associated to the edges entering the corresponding hidden units. Given a user u, the weight associated to a feature c is then the summation of the weights $w_k^u(c)$ associated to the edges entering the hidden node representing the Knowledge Graph category c after training the autoencoder with the ratings of u.
More formally, we have:

$$\omega^u(c) = \sum_{k=1}^{|In(c)|} w_k^u(c)$$

where $In(c)$ is the set of the edges entering the node representing the feature c. We remember that since the autoencoder is not fully connected, $|In(c)|$ varies depending on the related connections to the category c in the knowledge graph.

By means of the weights associated with each feature, we can now model a user profile composed by a vector of weighted categorical features. Given F^u as the set of categories belonging to all the items rated by u and $F = \bigcup_{u \in U} F^u$

as the set of all features among all the users in the system we have for each user $u \in U$ and for each feature $c \in F$:

$$P(u) = \{\langle c, \omega \rangle \mid \omega = \omega^u(c) \text{ if } c \in F^u\}$$

Considering that users provide a different number of ratings, we have an unbalanced distribution in the dimension of user profiles. Moreover, as a user usually rate only a small subset of the entire catalog, we have a massive number of missing features belonging to items not rated by u. In order to compute values associated with missing features, we leverage an unsupervised deep learning model inspired by the *word2vec* approach [**?**]. It is an efficient technique originally conceived to compute word embeddings (i.e., numerical representations of words) by capturing the semantic distribution of textual words in a latent space starting from their distribution within the sentences composing the original text. Given a corpus, e.g., an excerpt from a book, it projects each word in a multidimensional space such that words similar from a semantic point of view result closer to each other. In this way, we are able to evaluate the semantic similarity between two words even if they never appear in the same sentence. Given a sequence of words $[x_1, \ldots, x_n]$ within a window, *word2vec* compute the probability for a new word x' to be next one in the sequence. More formally, it computes $p(x' \mid [x_1, \ldots, x_n])$.

In our scenario, we may imagine replacing sentences represented by sequences of words with user profiles represented by sequences of categories in $c \in F^u$ and then use the *word2vec* approach to compute for a given user u the weight of missing features $c' \notin F^u$.

We need to prepare the user profiles $P(u)$ to be processed by *word2vec*. Hence, we first generate a corpus made of sequences of ordered features where the order is given by ω. The very preliminary step is that of selecting an order among elements $c \in F^u$ which results coherently for all $u \in U$ thus moving from the set $P(u)$ to a representative sequence of elements $s(u)$.

For each $\langle c, \omega \rangle \in P(u)$ we create a corresponding pair $\langle c, norm(\omega) \rangle$ with $norm$ being the mapping function

$$norm : [0, 1] \mapsto \{0.1, 0.2, 0.3, \ldots, 1\}$$

that linearly maps[1] a value in the interval $[0, 1]$ to a real value in the set $\{0.1, 0.2, 0.3, \ldots, 1\}$. The new pairs form the set

$$P^{norm}(u) = \{\langle c, norm(\omega) \rangle \mid \langle c, \omega \rangle \in P(u)\}$$

[1] In our current implementation we use a standard `minmax` normalization.

For each normalized user profile set $P^{norm}(u)$ we then build the corresponding sequence

$$s(u) = [\ldots, \langle c_i, norm(\omega_i^u)\rangle, \ldots \langle c_j, norm(\omega_j^u)\rangle, \ldots]$$

with $\omega_i^u \geq \omega_j^u$.

Once we have the set $S = \{s(u) \mid u \in U\}$ we can feed the *word2vec* algorithm with this corpus in order to find patterns of features according to their distribution across all users. In the prediction phase, by using each user's sequence of features $s(u)$ as input for the trained *word2vec* model, we estimate the probability of $\langle c', norm(\omega')\rangle \in \bigcup_{v \in U} P^{norm}(v) - P^{norm}(u)$ to belong to the given context, or rather to be relevant for u. In other words, we compute $p(\langle c', norm(\omega')\rangle \mid s(u))$.

It is worth noticing that given $c' \in F^u$ we may have multiple pairs with c' as first element in $\bigcup_{v \in U} P^{norm}(v) - P^{norm}(u)$. For instance, given the category `dbc:Kung_fu_films` we may have both $\langle$`dbc : Kung_fu_films`$, 0.2\rangle$ and $\langle$`dbc : Kung_fu_films`$, 0.5\rangle$, with the corresponding probabilities $p(\langle$`dbc : Kung_fu_films`$, 0.2\rangle \mid s(u))$, $p(\langle$`dbc : Kung_fu_films`$, 0.5\rangle \mid s(u))$. Still, as we want to add the category `dbc:Kung_fu_films` together with its corresponding weight only once in the user profile we select only the pair with the highest probability. The new user profile is then

$$\hat{P}(u) = P(u) \cup \{\langle c, \omega\rangle \mid \underset{\omega \in \{0.1,\ldots,1\}}{\operatorname{argmax}} \; p(\langle c, \omega\rangle \mid s(u)) \text{ and } \langle c, \omega\rangle \notin P^{norm}(u)\}$$

We point out that while the original $P(u)$ is built by exploiting only content-based information, the enhanced user profile $\hat{P}(u)$ also considers collaborative information as it based also on the set S containing a representation for the profiles of all the users in U.

5.3.2 Computing Recommendations

Given the user profiles represented as vectors of weighted features, recommendations are then computed by using a well-known k-nearest neighbors approach. User similarities are found through projecting their user profile in a Vector Space Model, and then similarities between each pair of users u and v is computed using the cosine similarity.

For each user u we find the top-k similar neighbors to infer the rate r for the item i as the weighted average rate that the neighborhood gave to it:

$$r(u, i) = \frac{\sum_{j=1}^{k} sim(u, v_j) \cdot r(v_j, i)}{\sum_{j=1}^{k} sim(u, v_j)} \tag{5.1}$$

where $r(v_j, i)$ is the rating assigned to i by the user v_j. We use then ratings from Equation (5.1) to provide top-N recommendation for each user.

5.4 Experiments

In this section, we present the experimental evaluations performed on three different datasets. We first describe the structure of the datasets used in the experiments and the evaluation protocol and then we move to the metrics adopted for the evaluation and the discussion of obtained results.

Our experiments can be reproduced through the implementation available on our public repository[2].

5.4.1 Dataset

In order to validate our approach we performed experiments on the three datasets summarized in Table 7.1.

	#users	#items	#ratings	sparsity
MovieLens 20M	138,493	26,744	20,000,263	99.46%
Amazon Digital Music	478,235	266,414	836,006	99.99%
LibraryThing	7,279	37,232	626,000	99.77%

Table 5.1: Datasets

In our experiments, we referred to the freely available knowledge graph of DBpedia[3]. The mapping contains 22,959 mapped items for MovieLens 20M[4], 4,077 items mapped for Amazon Digital Music[5] and 9,926 items mapped for LibraryThing[6]. For our experiments, we removed from the datasets all the items without a mapping in DBpedia.

5.4.2 Evaluation protocol

Here, we show how we evaluated the performances of our methods in recommending items. We split the dataset using Hold-Out 80/20, ensuring that every user has 80% of their ratings in the training set and the remaining

[2]https://github.com/sisinflab/SEMAUTO-2.0
[3]`https://dbpedia.org`
[4]`https://grouplens.org/datasets/movielens/20m/`
[5]`http://jmcauley.ucsd.edu/data/amazon/`
[6]`https://www.librarything.com`

20% in the test set. For the evaluation of our approach we adopted the "all unrated items" protocol described in [52]: for each user u, a top-N recommendation list is provided by computing a score for every item i not rated by u, whether i appears in the user test set or not. Then, recommendation lists are compared with the test set by computing both performance and diversity metrics such as Precision, Recall, F-1 score, nDCG [53], aggregate diversity, and Gini index as a measure of sales diversity [54].

5.5 Results Discussion

In our experiments, we compared our approach with three different states of the art techniques widely used in recommendation scenarios: BPRMF, WRMF and a single-layer autoencoder for rating prediction. BPRMF [55] is a Matrix Factorization algorithm which leverages Bayesian Personalized Ranking as the objective function. WRMF [56, 57] is a Weighted Regularized Matrix Factorization method which exploits users' implicit feedback to provide recommendations. In their basic version, both strategies rely exclusively on the User-Item matrix in a pure collaborative filtering approach. They can be hybridized by exploiting side information, i.e. additional data associated with items. In our experiments, we adopted categorical information found on the DBpedia Knowledge Graph as side information. We used the implementations of BPRMF and WRMF available in MyMediaLite[7] and implemented the autoencoder in Keras[8]. We verified the statistical significance of our experiments by using the Wilcoxon Signed Rank test we get a *p-value* very close to zero, which ensures the validity of our results.
In Table 5.3 we report the results gathered on the three datasets by applying the methods discussed above. As for our approach SEMAUTO, we tested it for a different number of neighbors by varying k.
In terms of accuracy, we can see that SEMAUTO outperforms our baselines on both MovieLens 20M and Amazon Digital Music datasets, while on LibraryThing the achieved results are quite the same. In particular, on the LibraryThing dataset, only the fully-connected autoencoder performs better than our approach with regard to accuracy.
Concerning diversity, we get much better results on all the datasets. Furthermore, by analyzing the gathered results, it seems that our approach provides very discriminative descriptions for each user, letting us identify the most effective neighborhood and compute both accurate and diversified recommendations. As a matter of fact, we achieve the same results in terms of

[7] `http://mymedialite.net`
[8] `https://keras.io`

accuracy as the baselines by suggesting much more items.

	avg #features	std	avg #features/avg #items
Movielens 20M	1015.87	823.26	8.82
Amazon Digital Music	7.22	9.77	5.17
LibraryThing	206.88	196.64	1.96

Table 5.2: Summary of hidden units for mapped items only.

As shown in Table 5.2, SEMAUTO performs better on those datasets whose items can be associated with a large amount of categorical information, which implies the usage of many hidden units. This occurs because very complex functions can be modeled by ANNs if enough hidden units are provided, as Universal Approximation Theorem points out. For this reason, our approach turned out to work better on MovieLens 20M dataset (whose related neural networks have a high number of hidden units) rather than the others. In particular, the experiments on LibraryThing dataset show that the performances get worse as the number of the neurons decreases, i.e. available categories are not enough.

5.6 Conclusion and future work

In this work, we have presented a recommendation approach that combines the computational power of deep learning with the representational expressiveness of knowledge graphs. As for classical applications of autoencoders to feature selection, we compute a latent representation of items but, in our case, we attach an explicit semantics to selected features. This allows our system to exploit both the power of deep learning techniques and, at the same time, to have a meaningful and understandable representation of the trained model. We used our approach to autoencode user ratings in a recommendation scenario via the DBpedia knowledge graph and proposed an algorithm to compute user profiles then adopted to provide recommendations based on the semantic features we extract with our autoencoder. Experimental results show that we are able to outperform state of the art recommendation algorithms in terms of accuracy and diversity. Furthermore, we will compare our approach with other competitive baselines as suggested in more recent works [58]. The results presented in this paper pave the way to various further investigations in different directions. From a methodological and algorithmic point of view, we can surely investigate the augmentation of further deep learning techniques via the injection of explicit and structured knowledge

	k	F1	Prec.	Recall	nDCG	Gini	aggrdiv
MOVIELENS 20M							
AUTOENCODER	–	0.21306	0.21764	0.20868	0.24950	0.01443	1587
BPRMF	–	0.14864	0.15315	0.14438	0.17106	**0.00375**	3263
BPRMF + SI	–	0.16838	0.17112	0.16572	0.19500	0.00635	3552
WRMF	–	0.19514	0.19806	0.19231	0.22768	0.00454	766
WRMF + SI	–	0.19494	0.19782	0.19214	0.22773	0.00450	759
SEMAUTO	5	0.18857	0.18551	0.19173	0.21941	0.01835	**5214**
	10	0.21268	0.21009	0.21533	0.24945	0.01305	3350
	20	0.22886	0.22684	0.23092	0.27147	0.01015	2417
	40	0.23675	0.23534	0.23818	0.28363	0.00827	1800
	50	0.23827	0.23686	0.23970	0.28605	0.00780	1653
	100	**0.23961**	**0.23832**	**0.24090**	**0.28924**	0.00662	1310
AMAZON DIGITAL MUSIC							
AUTOENCODER	–	0.00060	0.00035	0.00200	0.00102	0.33867	**3559**
BPRMF	–	0.01010	0.00565	0.04765	0.02073	**0.00346**	539
BPRMF + SI	–	0.00738	0.00413	0.03480	0.01624	0.06414	2374
WRMF	–	0.02189	0.01236	0.09567	0.05511	0.01061	103
WRMF + SI	–	0.02151	0.01216	0.09325	0.05220	0.01168	111
SEMAUTO	5	0.01514	0.00862	0.06233	0.04365	0.03407	3378
	10	0.01920	0.01091	0.07994	0.05421	0.05353	3449
	20	0.02233	0.01267	0.09385	0.06296	0.08562	3523
	40	0.02572	0.01460	0.10805	0.06980	0.14514	3549
	50	0.02618	0.01486	0.10974	0.07032	0.17192	3549
	100	**0.02835**	**0.01608**	**0.11964**	**0.07471**	0.24859	3448
LIBRARYTHING							
AUTOENCODER	–	**0.01562**	**0.01375**	**0.01808**	**0.01758**	0.07628	2328
BPRMF	–	0.01036	0.00954	0.01134	0.01001	0.06764	3140
BPRMF + SI	–	0.01065	0.00994	0.01148	0.01041	0.10753	**4946**
WRMF	–	0.01142	0.01071	0.01223	0.01247	**0.00864**	439
WRMF + SI	–	0.01116	0.01030	0.01217	0.01258	0.00868	442
SEMAUTO	5	0.00840	0.00764	0.00931	0.00930	0.13836	4895
	10	0.01034	0.00930	0.01163	0.01139	0.07888	3558
	20	0.01152	0.01029	0.01310	0.01248	0.04586	2245
	40	0.01195	0.01073	0.01347	0.01339	0.02800	1498
	50	0.01229	0.01110	0.01378	0.01374	0.02403	1312
	100	0.01278	0.01136	0.01461	0.01503	0.01521	873

Table 5.3: Experimental Results

coming from external sources of information. Giving an explicit meaning to neurons in an ANN as well as to their connections can fill the semantic gap in describing models trained via deep learning algorithms. Moreover, having an explicit representation of latent features opens the door to a better and explicit user modeling. We are currently investigating how to exploit the structure of a Knowledge Graph-enabled autoencoder to infer qualitative preferences represented by means of expressive languages such as CP-theories [59]. Providing such a powerful representation may also result in being a key factor in the automatic generation of explanation to recommendation results.

Chapter 6

Explainability

Recommender Systems have been widely used to help users in finding what they are looking for thus tackling the information overload problem. After several years of research and industrial findings looking after better algorithms to improve accuracy and diversity metrics, explanation services for recommendation are gaining momentum as a tool to provide a human-understandable feedback to results computed, in most of the cases, by black-box machine learning techniques. As a matter of fact, explanations may guarantee users satisfaction, trust, and loyalty in a system. In this paper, we evaluate how different information encoded in a Knowledge Graph are perceived by users when they are adopted to show them an explanation. More precisely, we compare how the use of categorical information, factual one or a mixture of them both in building explanations, affect explanatory criteria for a recommender system. Experimental results are validated through an A/B testing platform which uses a recommendation engine based on a Semantics-Aware Autoencoder to build users profiles which are in turn exploited to compute recommendation lists and to provide an explanation.

6.1 Introduction

In recent time we assisted to the rising of Deep Learning models in many fields such as Computer Vision, Speech Recognition, Natural Language Processing and, more recently, few attempts have also been made to solve the Recom-

mendation problem. Deep Learning techniques have proven their strength thus gaining the attention of both researchers and companies and they are widely deployed in nowadays recommender systems. While research has mainly focused on improving accuracy metrics in recommenders, under the hood, their algorithms are becoming more and more complex thus making extremely hard to understand the reasons behind model predictions for a particular input. This recently led both researchers and companies to pay more attention to explainable models. Indeed, it has been proven that showing to users an explanation for the provided recommendation leads to better interaction with the recommender system. Moreover, when users understand how the system works, they can refine their preferences in order to get a better recommendation according to their tastes. However, in many popular recommenders such as Amazon or Netflix, the explanation provided is still very poor, as it is essentially based on a popularity basis: it just tells that users with similar tastes have enjoyed the suggested items. It turns out that this kind of explanation is not perceived as a valid justification of why the system is recommending certain items and it hardly improves users loyalty in the system. On the other hand, a content-based explanation turns out to be more engaging from the user's point of view because it makes users aware about item's attributes that might be relevant for them.

Providing a content-based explanation seems to be much more difficult because item descriptions are not always available and they are not easy to maintain. Thus, some attempts have been made in order to exploit Knowledge Graphs as data source for items' content description.

In this work we propose to exploit a Semantics-Aware Autoencoders (SEMAUTO) [60] to compute explainable recommendations. Originally developed to cope with the cold start problem, in SEMAUTO the structure of the DBpedia KG is injected within an Autoencoder Neural Network, whose structure is built by mimicking the existing connections in the KG. Then, after feeding such a network with user ratings, weights associated to the hidden neurons are extracted and then used to build knowledge-aware user profiles which are eventually used to compute recommendations. In [61] we prove that SEMAUTO can also be effectively used to compute recommendations in non-cold situations reaching very competitive results in terms of accuracy and diversity.

Here we show how the model built by SEMAUTO can also be adopted to compute content-based explanations to recommended items. We evaluated the effectiveness of our approach through an A/B testing platform with 892 volunteers and compared its results to two baselines. We tested both a pointwise and a pairwise explanation style by exploiting different kinds of

available information in DBpedia[1] (categorical and factual), in order to investigate how the effectiveness of the proposed explanation changes according to the selected properties. The main research questions we address in this paper are then:

RQ1 Can we assume that the information encoded in the hidden layer of the SEMAUTO autoencoder is representative of user preferences?

RQ2 Given a content-based explanation built upon the SEMAUTO model, is a pairwise explanation better than a simple pointwise one for the user?

6.2 Related works

Making a Recommender System (RS) transparent to users is getting more and more relevance since it may lead to users retain [62]. Different studies [63, 64] have pointed out that introducing transparency in the recommendation process may have lots of advantages because users appear to be more satisfied with the recommendation if they are aware of the reasons why certain items are suggested. Furthermore, the provided explanation may also convince users to try items they would have normally ignored, thus improving users confidence in the system.

Since the explanation may be decoupled from the recommendation process, a distinction between *transparency* and *justification* has to be made [65]. The explanation brings *transparency* to the system if it makes users aware about how the recommender engine works, explaining somehow the underlying algorithm behind the proposed suggestions. This is usually the case of those explanations computed along with the recommendation. On the other hand, *justification* implies an explanation which is not directly related to the recommendation algorithm, thus it can be generated in a more freely way. Such kind of explanations may be preferred to *transparency* because of algorithms that are difficult to explain or have not to be spread.

The main advantages users may get from the explanation are described in [66] and they include: *transparency*, *scrutability*, *trust*, *effectiveness*, *persuasiveness*, *efficiency* and *satisfaction*. In [67], the authors show how they can be exploited as evaluation metrics for explanatory services. However, providing effective explanations is not always a trivial task; RSs have surely proven to be very accurate in accomplishing their tasks, but they usually work just as black boxes, being not transparent at all. In order to overcome

[1] `http://dbpedia.org`

this issue, new methods have been developed in order to generate an explainable recommendation ([64] provides an overview of the most successful approaches proposed over the years) such as *MoviExplain* [68], which exploits movies metadata to justify its recommendation lists. Other interesting works include: a RS based on Restricted Boltzmann Machines which looks at the rating distribution to identify the most explainable items [69], a Latent Factor Model leveraging users reviews to compute more transparent recommendations [70] and, finally, a novel approach based on movies information encoded in the Linked Open Data cloud which generates natural language explanation for the computed recommendation presented in [71]. Looking at the last mentioned method, it is worth noticing how Knowledge Graphs (KG) are recently being used in lots of applications; they freely offer a large amount of structured data which turned out to be very useful also in recommendation scenarios [3, 5, 45]. In particular, in [60], the authors introduce the idea of a Semantics-Aware Autoencoder which paves the way to compute explanation by leveraging deep learning techniques.

As a matter of fact, all the approaches based on deep learning models that have been proposed over the years, turned out to barely leverage latent factors to which no meaning can be attached to. Among them, Autoencoder Neural Networks have proven their effectiveness in CF settings as shown in [32], in which the authors use an Autoencoder fed with user ratings in order to predict missing value for users' unseen items. In other works such as [48] a stacked architecture made of Autoencoders is proposed to perform a generalization over a higher set of latent features that every stacked autoencoder is able to learn. More recently, in [72] the authors propose a hybrid architecture for Autoencoders in order to incorporate both users' feedback and content description about items. A similar approach has been proposed in [73], in which they exploit side information in a CF setting by using Stacked Autoencoders in order to overcome the cold start problem and data sparsity.

6.3 Background technologies

Our approach relies on two main technologies: Knowledge Graphs and Autoencoder Neural Networks. The proposed method shows how to use the former to map KG's connections to the topology of the latter, in order to give an explicit meaning to the connections in the NN.

6.3.1 Knowledge Graphs

In the past few years, we have assisted to the publication of freely ontological data on the Web, thanks to diverse communities that began to develop Knowledge Graphs as well-structured graph data encoding the human knowledge. KGs are oriented graphs in which nodes identify resource entities and edges provide labeled relationships between them. Some prominent examples of KGs are DBpedia and Wikidata, which are community driven projects that leverage on Wikipedia pages to automatically parse structured data. Mainly, two kind of information exist in DBpedia: semantics-aware and factual one. The former can be divided into categorical and ontological data. Categorical information is encoded through the `dct:subject` predicate and represents items categories parsed from Wikipedia infoboxes, such as `Vigilante films` or `Cyborg films`. Categories in Wikipedia are collaboratively maintained by community's editors thus leading to a rich set of categories that reflects a human classification by encoding knowledge about classes attributes and other semantic relations [74]. Ontological data basically captures entities types (classes) and their hierarchy; it does not only represent their taxonomy but extends it by using restrictions on its relationships to other classes or on the properties a particular class is allowed to posses. Finally, factual knowledge is merely made of *facts*; it identifies items' attributes, as it can be in the movies domain that the actor `Will Smith` `starred` in the movie `I, Robot`, as depicted in Figure 6.1. Differently from categorical information, factual one is identified via different attributes/predicates connecting an item to different entities as in the case of `director`, `starring`, etc..

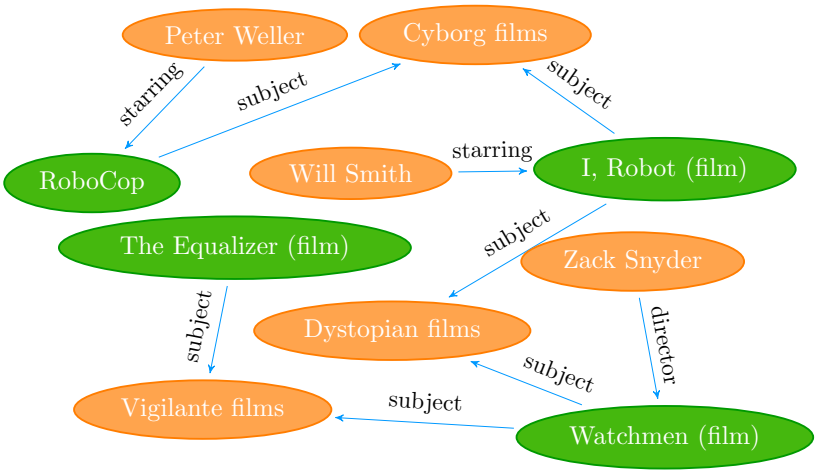


Figure 6.1: Part of a KG related to the movie domain.

6.4 Semantics-Aware Autoencoder

Neural Networks model are generally made by one input layer, one or more hidden layers and an output layer. Every layer contains neurons, and every neuron of layer i is connected to all neurons of layer $i + 1$. In particular, Autoencoders are a special kind of unsupervised learning Neural Networks that learn a function able to reconstruct the original data available at the output layer. In the training phase, autoencoders learn how to reconstruct the input vector x through a latent representation encoded in the hidden layers.

In a semantics-aware autoencoder, the hidden layers and their connections are substituted by the knowledge graph thus having an explicit representation on the meaning associated both to hidden nodes and to their mutual connections [60]. This means that each neuron represents an entity in the adopted KG and the edge between two autoencoder nodes exist if the corresponding KG entities are connected with a predicate (labeled edge).

In our implementation, we adopted three different configurations based on a single hidden layer semantics-aware autoencoder (see Figure 7.1) which exploits one of the following sets of information available in DBpedia: (i) *semantic data*, or rather categorical attributes of items; (ii) *factual data*, specific items properties (such as `actors` and `directors` in the movie domain); (iii) *semantic and factual information*, a mixture of the previous two.

Hence, the resulting autoencoder has three layers: input layer, hidden layer and output layer where the input and output layers represent items in the catalog while the middle hidden layer contains their DBpedia categories and/or properties.

As previously said, in the training phase, an autoencoder learns how to reconstruct the input vector (in our case user ratings) using the latent representation encoded in the hidden layer. As we train an autoencoder per user, once the model converges, in a semantics-aware autoencoder, for each user we have a latent representation of item's features which, actually, result to be no more latent because every neuron corresponds to an entity in the KG. It turns out that features belonging to positively rated items tend to have a higher weight, differently from those of negatively rated items. This behavior is quite understandable considering that a rating feeding an input node (representing an item in the catalog) flows throughout the neural network by crossing only features/nodes connected to it in the KG. We want to stress here that, although each autoencoder is trained over a not huge number of samples, in [61] we prove that recommendation results have very good performance in terms of accuracy and diversity also compared to state-

of-the-art algorithms[2].

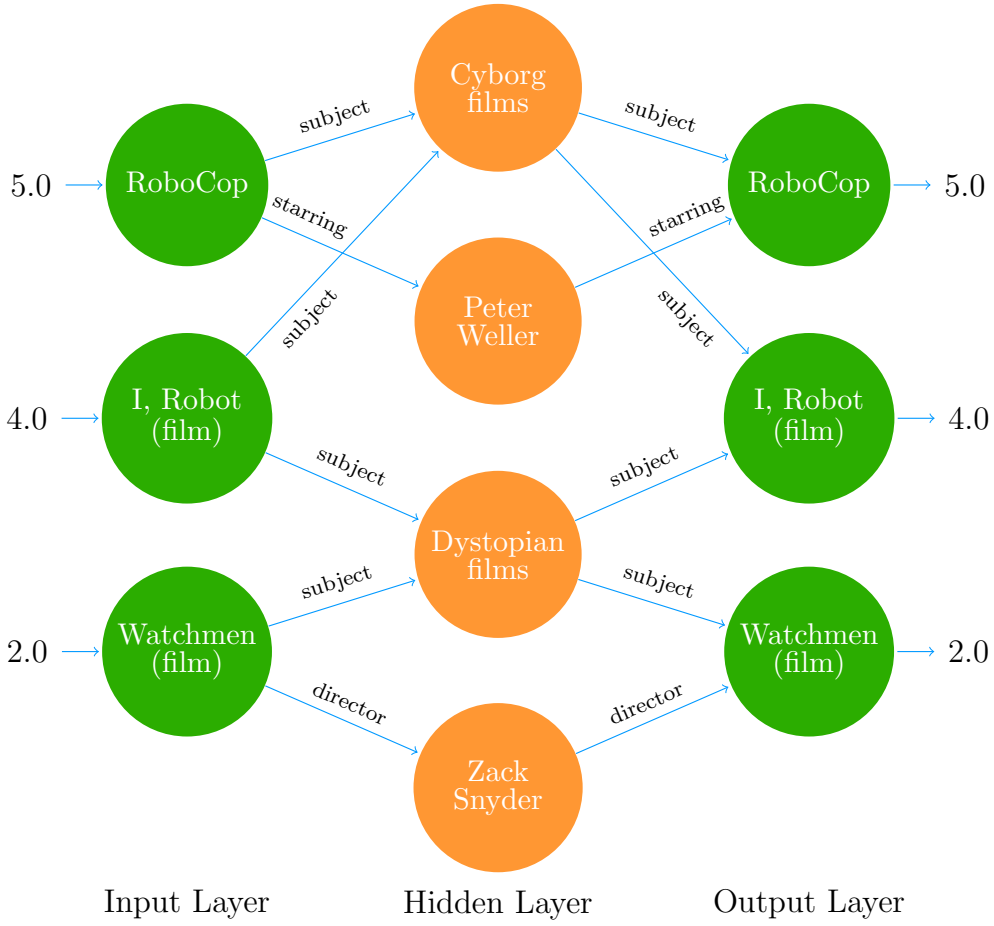


Figure 6.2: Architecture of a semantic autoencoder.

To train such a kind of autoencoder, we inhibit the feedforward and back-propagation step for those neurons which result to be not connected in the KG by using a masking multiplier matrix M where rows and columns represent respectively items and features.

$$M_{m,n} = \begin{pmatrix} a_{1,1} & a_{1,2} & \cdots & a_{1,n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{m,1} & a_{m,2} & \cdots & a_{m,n} \end{pmatrix} \tag{6.1}$$

The matrix in Equation (6.1) represents the adjacency matrix of the KG

[2]The code implementing SEMAUTO has been developed by using TensorFlow and is available at `https://github.com/sisinflab/SEMAUTO-2.0`.

where a generic entry is a binary value indicating whether a connection among entities exists in it. In other words, we have

$$a_{i,j} \in M_{m,n} = \begin{cases} 1, & \text{if item } i \text{ is connected to feature } j \\ 0, & \text{otherwise} \end{cases}$$

Hence, hidden (h) and output (o) layers are computed by the following two equations:

$$\begin{aligned} h &= g(X \times (W_1 \circ M)) \\ o &= g(h \times (W_2 \circ M^T)) \end{aligned}$$

During the backpropagation step, gradients are computed as usually for W_2 and W_1 with respect to a mean squared error loss $E = \frac{1}{2} \sum_i \parallel x_i - y_i \parallel^2$ being x_i and y_i the elements of the input and output vector respectively.

The weights update step in SGD (Stochastic Gradient Descent) backpropagation has been modified according to Equations (6.2) in order to take into account the masking matrix:

$$W_1 = (W_1 \circ M) - r \cdot \frac{\partial E}{\partial W_1} \qquad W_2 = (W_2 \circ M^T) - r \cdot \frac{\partial E}{\partial W_2} \tag{6.2}$$

Where E is the mean squared error loss while W_1 and W_2 represent the weight matrices for the connections between the input and hidden layer (W_1) and between the hidden layer and the output layer (W_2). They are both initialized randomly using Xavier initialization [51]. In our experiments, we trained the model for 1000 epochs with a learning rate $r = 0.03$ and we used the well-known sigmoid $\sigma(z) = \frac{1}{1+e^{-z}}$ as activation function. Since we train one autoencoder per user and we want it to overfit on user ratings, we did not use any form of regularization.

Computing user profiles. After training the autoencoder for each user u, we extract the weights of the hidden neurons and use them to build a user profile $P(u)$:

$$P(u) = \{\langle f_{u1}, w_{u1} \rangle, \ldots, \langle f_{um}, w_{um} \rangle\}$$

being f_u the label associated to the neuron and w_u its corresponding weight for u. Indeed, as each hidden neuron represents an entity in DBpedia, we may assume that its weight after the training is an indicator of the importance of the corresponding entity for u.

6.5 Semantics-Aware Explanations

As previously said, in this work we explore the adoption of a semantic autoencoder to provide an explanation for top-N recommendations. In our experimental setting aimed at evaluating the explainability of the trained model, while building the structure of the SEMAUTO autoencoder we used those KG entities reachable through the predicate `dbo:subject` as item categories, while we used the approach originally proposed in [75] to select the top-3 factual movie properties: `dbo:starring`, `dbo:director`, `dct:writer`[3].

In order to formulate a human-understandable explanation for the provided results, we rely on the weights associated to features in the user profile, which also appear in the description of the recommended items. In particular, given a user u and a recommendation list $rec(u) = [\langle i_1, \tilde{r}_1^u \rangle, \ldots, \langle i_n, \tilde{r}_n^u \rangle]$, with $\tilde{r}_k^u$ being a score/rating computed for the item i_k by a recommendation engine, we may compute a pointwise and a pairwise personalized explanation.

pointwise personalized. Given an item $i = \{f_{1i}, f_{2i}, \ldots, f_{ni}\}$ described by a set of features f_i, the pointwise explanation $e1^k(i)$ is computed by considering the set of top-k highest weighted features in $P(u)$ which also appear in i.

pairwise personalized. Given two items i and j such that $\tilde{r}_i^u > \tilde{r}_j^u$, the pairwise explanation $e2^k(i,j)$ is computed by evaluating both $e1^k(i)$ and $e1^k(j)$. In case m features are in common between $e1^k(i)$ and $e1^k(j)$, we compute $e1^{k+m}(j)$ and leave them only in $e1^k(i)$ thus avoiding any overlap between the explanation for i and that for j.

To verify that the explanation generated through a Semantics-Aware Autoencoder is able to satisfy the main explanatory criteria of *transparency*, *persuasiveness*, *effectiveness*, *trust* and *satisfaction*, we built a web platform[4] that returns the top-5 recommendations and then asks for users' feedback about the provided explanation.

6.5.1 Explanation styles

We provided our platform with four different explanation styles: as in [71], we used a *popularity-based explanation* and a *non-personalized* one as baselines

[3]We selected only the top-3 properties to reduce the dimension of the feature space and then minimize the noise in the provided explanation. Finding the best number of properties to compute explanations is not in the scope of this paper and is part of our future work.

[4]Available at `http://sisinflab.poliba.it/semanticweb/lod/recsys/explanation`

[67] while as third and fourth style we provide our pointwise and pairwise approaches. During the usage of the platform by a user, we randomly select one of the four styles and show the associated explanation, which is generated for the top-2 recommended items in a *pairwise* fashion. Hence, the user may receive one of the following explanations:

popularity-based *We suggest these items since they are very popular among people who like the same movies as you.*

(non-/pointwise) personalized *We guess you would like to watch i and j since they are about* $\tilde{f}_{u1}, \ldots \tilde{f}_{uk}$

pairwise personalized *We guess you would like to watch i more than j because you may prefer* $e1^k$(i) *over* $e1^{k+m}$(j) (Example 6.5.1)

Example 6.5.1 *In order to show the difference between a pointwise and a pairwise personalized explanation, hereafter we report the two explanation styles with reference to a recommendation having Terminator 2: Judgment Day and Transformers: Revenge of the fallen as the first two items in the recommendation list. The pointwise personalized explanation may look like:*

We guess you would like to watch Terminator 2: Judgment Day (1991) and Transformers: Revenge of the Fallen (2009) because you may prefer:

- *(subject) 1990s science fiction films*
- *(subject) Science fiction adventure films*
- *(subject) Drone films*
- *(subject) Cyberpunk films*

and:

- *(subject) Science fiction adventure films*
- *(subject) Films set in Egypt*
- *(subject) Robot films*
- *(subject) Films shot in Arizona*
- *(subject) Ancient astronauts in fiction*

while the pairwise version (see also Figure 6.4) is a bit different:

We guess you would like to watch Terminator 2: Judgment Day (1991) more than Transformers: Revenge of the Fallen (2009) because you may prefer:

- *(subject) 1990s science fiction films*
- *(subject) Science fiction adventure films*
- *(subject) Drone films*
- *(subject) Cyberpunk films*

over:

- *(subject) Films set in Egypt*
- *(subject) Robot films*
- *(subject) Films shot in Arizona*
- *(subject) Ancient astronauts in fiction*
- *(subject) IMAX films*

The *popularity-based explanation* may be considered as the less meaningful, since it justifies recommender choices by just leveraging the popularity of suggested items among the users with similar tastes of the active user u. The *non-personalized explanation*, instead, tries to explain the provided recommendation by using additional information about the suggested items. In our experiments, we randomly select $k = 5$ features from the set $F_{ij} = F_i \cup F_j = \{f_{1i}, f_{2i}, \ldots, f_{ni}\} \cup \{f_{1j}, f_{2j}, \ldots, f_{n'j}\}$. In a similar manner, in a *pointwise personalized explanation* we selected the top-5 features from each set F_i and F_j. The value $k = 5$ has been selected also to compute $e2^k(i, j)$ in the *pairwise personalized explanation*.

Please notice that the considered set of features per item varies according to the different configuration adopted for the SEMAUTO autoencoder; it may include just item categories, factual data or both of them.

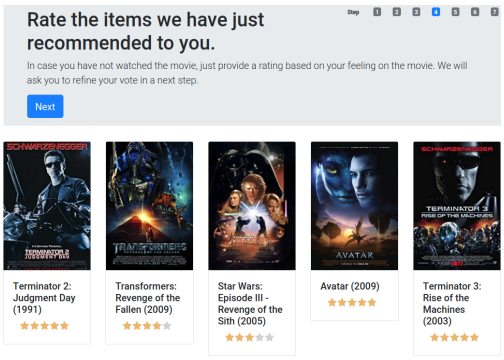


Figure 6.3: Step 4. The user is asked to rate the recommended items, even if she has not watched them.

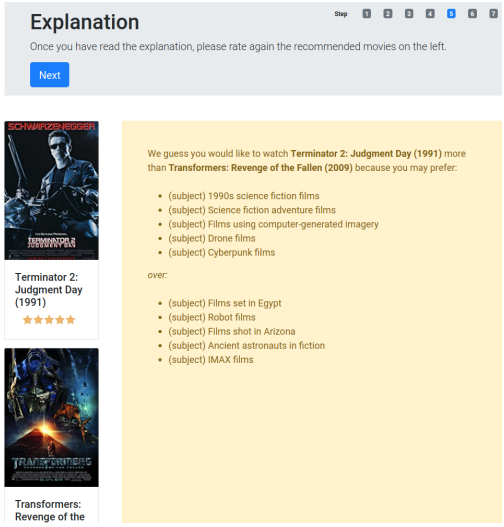


Figure 6.4: Step 5. The user is asked to read the explanation and after that to rate again the top-2 recommended items.

6.5.2 Evaluation Protocol

During the online A/B testing phase, we fixed a sequence of steps in order to measure the aforementioned explanatory criteria.

Steps 1-3. At the beginning of the experiment, the user u selects at least 15 movies she has watched among the ones randomly listed by the platform. The movies belong to the well-known MovieLens 20M dataset [5]. Then, she is invited to rate each selected movie on a five-stars rating scale; data so gathered are exploited to get both the user profile computed with the semantic autoencoder and a top-5 recommendation list.
Step 4. Once the recommendation has been generated, the user is asked to rate the suggested items, even if no explanation has been shown yet (see Figure 6.3): these ratings will be relevant to determine the impact the explanation has on the user (*persuasiveness*).
Step 5. The next step consists of showing to u one of the four randomly selected explanation styles deployed within the application (see Figure 6.4). After enjoying the explanation, the user has to re-rate the top-2 recommended items, letting us measure how different is the items evaluation before and after the explanation has been provided.

[5] `https://grouplens.org/datasets/movielens/20m/`

Step 6. Similarly, in the last part of the experiment, the user is asked to re-rate the recommended movies after watching the related trailers. This phase allows u to emulate the items consumption, and makes her more aware about the topics of the suggested movies. In this way we can evaluate how much effective the selected explanation style was (*effectiveness*).
Step 7. Finally, the user fills a questionnaire, aimed at measuring the explanation *transparency*, *trust* and *satisfaction* (see Table 6.1).

6.6 Metrics

When evaluating an explanation system, the main characteristics to evaluate are [64]:

- *transparency*, which refers to the capability of the explanation to make users aware of how the system works;
- *trust*, or rather the confidence users have in the system;
- *satisfaction*, if users have an enjoyable experience in the usage of the system;
- *persuasiveness*, which evaluates how much convincing is the proposed explanation;
- *effectiveness*: the explanation is said to be effective if it helps users to correctly estimate items relevance before the consumption.

The first three characteristics are evaluated by collecting answers from users after filling the questionnaire at Step 7. As a final score for the first and the second metric we used the percentage of users that answered positively to the questions, while we exploited the average score assigned by users to quantify the overall *satisfaction*.

In order to evaluate the *persuasiveness* of the proposed explanation, we asked users to rate each recommended item before and after showing them the explanation: if the rating provided after looking at the explanation is higher than the original one, then the explanation has been able to persuade the user to try the suggested item. More formally we measure persuasiveness as [64]:

$$persuasiveness = \frac{1}{|U|} \cdot \sum_{u \in U} \frac{1}{N} \cdot \sum_{i \in I_N^u} (r_{ui}^e - r_{ui})$$

where U stands for the collection of users; I_N^u represents the set of top-N recommended items for u; r_{ui} and r_{ui}^e are, respectively, the ratings u assigns to i just before and after the explanation is provided.

Analogously, we evaluated the *effectiveness* as the difference between two ratings (see Equation (6.6) [64]), where r_{ui}^t represents the rating the user gives to the suggested movie after watching the related trailer (r_{ui}^t).

$$effectiveness = \frac{1}{|U|} \cdot \sum_{u \in U} \frac{1}{N} \cdot \sum_{i \in I_N^u} ||r_{ui}^e - r_{ui}^t||$$

The lower this value, the more effective the explanation, since it implies that users have rated each item with very similar values before and after the explanation has been provided.

METRIC	**QUESTION**
transparency	I understood the reason why the two movies have been ranked in the proposed order.
trust	The explanation increased my trust in the system.
satisfaction	The provided explanation: **really** captures my tastes. **partially** captures my tastes. **does not** capture my tastes.

Table 6.1: The final questionnaire.

6.7 Results Discussion

We conducted our experiment with the help of 892 volunteers[6], with at least 73 subjects for each of the implemented settings. As stated in [76], 73 has to be considered as the minimum acceptable sample size for such kind of experiments. This assures the significance of our experimental results. Furthermore, we verified the statistical significance of our experiment by using *Wilcoxon Rank-Sum Test*, getting $p \ll 0.01$.

As shown in Figure 6.5, a content-based explanation is always preferred by users, since the *popularity-based* style gets the worst results in all the considered explanatory criteria. The only exception is represented by *persuasiveness*: quite interestingly, the *non-personalized* explanation leveraging semantic/categorical information gets negative values, as it happens when

[6]They were recruited both among our students and via Amazon Mechanical Turk.

users rate items with lower values after looking at the explanation than before, being not convinced to consume the suggested items at all. Hence, users overestimate their interest in the recommended items or underestimate it because of the provided explanation; this may be interpreted as users dissatisfaction for the shown categories, since they are chosen randomly without taking into account users interests. As a matter of fact, the *personalized* approaches, which compute the explanation by leveraging users preferences, outperform the two baselines. Furthermore, it is worth noticing how convincing the categories are: by looking at the results, if the *personalized* style exploits categorical features, then it performs very well in terms of *persuasiveness* if compared to others. It is worth noticing that when categorical features are combined with factual information, they lead to a better *persuasiveness*, in particular the pairwise approach gets better results than the pointwise one. This may be explained by considering that we simulate item consumptions through their associated trailers: in our experiments, users provide a certain rating to a movie by just considering a few scenes, that are those shown in the trailer. Therefore, users may get information about the movie topics, subjects and how good or interesting an actor's or a director's performances are. This condition may influence the way the explanation is perceived by users, who demonstrated to find more convincing an explanation involving both categorical and factual information rather than an explanation based on item factual properties only.

On the other hand, still considering the pairwise *personalized* approach, factual properties turn out to be more effective as concerning *satisfaction*, *trust* and *effectiveness*; we suppose that users feel more confident in specific information such as actors or directors rather than just a set of movie categories. This trend is already confirmed by the pointwise approach, and the pairwise one gets even higher score with those metrics. As a matter of fact, the system *transparency* has the highest values when both semantic and factual properties are exploited; as these values are very close to those achieved by just using factual properties, we can claim that it is the factual information itself that improves the measured performances. Analogously, richer items descriptions make more *effective* the explanation: when the system leverages both categories and factual attributes, the *effectiveness* achieves its best results. Hence, providing more information about the suggested items surely lets users better evaluate them before their consumption. By considering the *non-personalized* style, it is quite interesting that by using both categorical and factual attributes, the gathered results for all the adopted criteria are usually the best, far from the performances measured by the other settings based on the KG. Once again, as discussed above, this may depend on the random aspect behind it: e.g., users may be more or less *satisfied* with the

provided explanation according to the randomly shown features, which they may like or dislike, know or ignore. Summing up, from the experimental results, we may argue that the pairwise approach with factual information gets better performance in users' *satisfaction*, *effectiveness* and *trust*, while it outperforms the pointwise one in *persuasiveness* and *transparency* when both factual and categorical information are exploited.

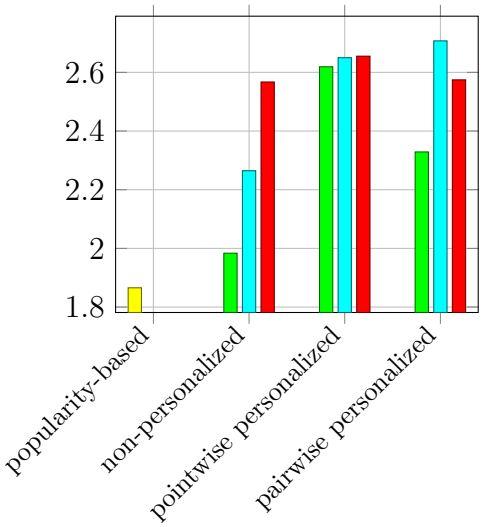
To provide an answer to **RQ1**, examining the results, it turns out that our SEMAUTO provides reliable users' descriptions as evidenced in the *effectiveness* metric which gets the lowest value by using a pairwise explanation. This can be interpreted as a strong signal that the information encoded in the autoencoder hidden layer is representative of the users' preferences because the users is less prone to change her ratings after she read the explanation.

As for **RQ2**, we can assert that the pairwise approach outperforms the pointwise one in all metrics especially in *transparency* because it provides a better justification on how the system ranks items according to the importance of the features in the user profile. This lets the user to better understand how her preferences are involved in the recommendation process. In fact, this has an impact especially for the *persuasiveness* metric where the pairwise approach has a higher score with respect to the pointwise explanation, thus leading users in consuming an item after they have read the provided explanation.

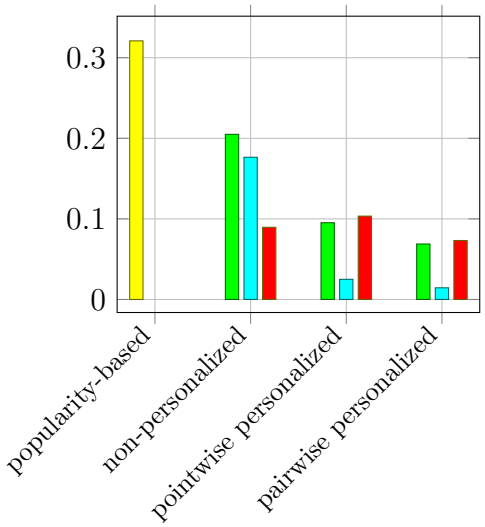
6.8 Conclusion and Future work

In this paper we present results on the capability for a semantics-aware autoencoder [60] to generate explanation to recommendation lists via the exploitation of data coming from the DBpedia knowledge graph. Online experimental results show that a content-based explanation is preferred by users, as it outperforms other baselines in terms of transparency, trust, satisfaction, persuasiveness and effectiveness. As we can see in the *satisfaction*, *effectiveness* and *trust* plots in Figure 6.5 for both pointwise and pairwise approaches, an interesting point is that, in order to build an explanation, factual data works better than the semantic/categorical one, achieving the same results as when both semantic and factual data are exploited. A possible reason for this behavior is that the probability for a user to know factual data and accepts it as explanation is higher if compared to categorical one. Very interestingly, a pairwise approach has the same trends for all the evaluation metrics of the pointwise one but it outperforms the latter.

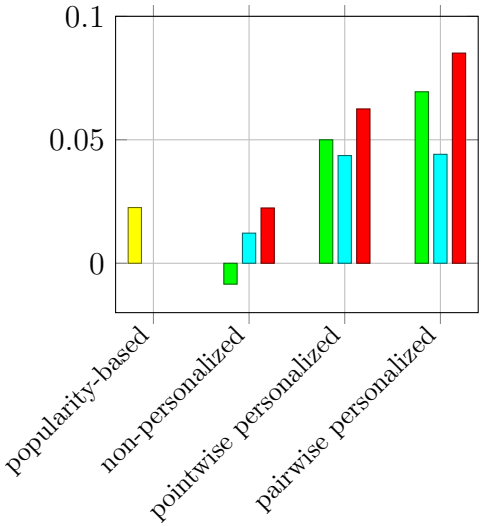
As future work it would be interesting to investigate about the system's *scrutability*, by allowing users to correct the recommender engine reasoning.



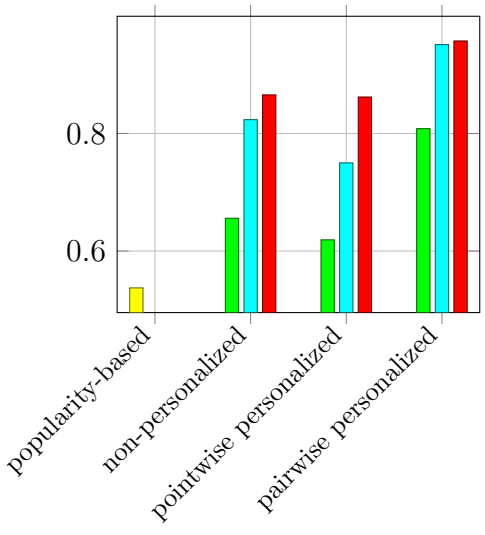
(a) satisfaction



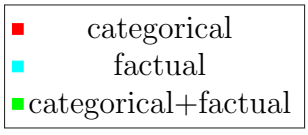
(c) effectiveness

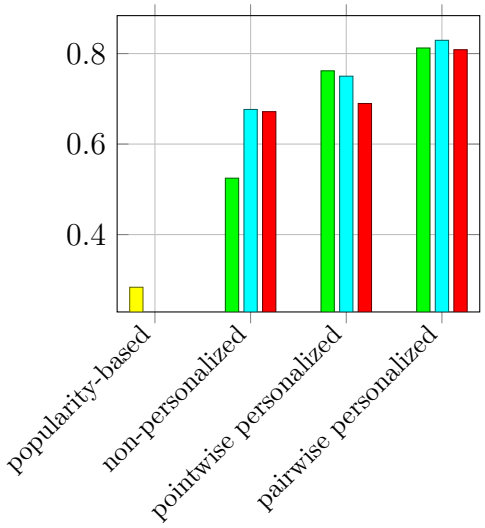


(b) persuasiveness



(d) transparency





(e) trust

Explanations here should be part of a continuous cycle where the user understands how the system is working under the hood and takes control over the type of recommendations made by the engine. This continuous loop could pave the way to a new kind of conversational recommender systems in which the user is allowed to explore and move in the feature space by knowing which features relevant to her are involved in the recommendation process.

Chapter 7

Knowledge Graphs for RecSys

Knowledge-Graphs (KGs) have proven their strength as a source of high-quality information for different tasks such as data integration, search, text summarization, named entity disambiguation and personalization. One of the most prominent examples is the Google Knowledge Graph, which can be used in search engines to further explore the knowledge space related to search results. Another prominent industrial and research field which has been benefiting from the adoption of Knowledge Graphs is that of Recommender Systems. In this paper, we report on an approach to the recommendation that puts together the power of deep learning techniques with the explicit semantic expressiveness of KGs by combining the topological structure of a neural network (configured as an autoencoder) with that of DBpedia and Wikidata. Supported by an extensive experimental evaluation, we show how the selection of the knowledge source that feeds the semantics-aware autoencoder affects the final results in terms of accuracy and diversity of recommended items.

7.1 Introduction

Nowadays, we are overwhelmed by a large amount of available information we can benefit from: e-commerce sites and entertainment web services usually offer thousands of different items among which users are invited to find the ones they need or desire the most. In this direction, recommender sys-

tems (RSs) have proved to be very helpful in suggesting appropriate items to users according to their past choices and behaviors. A typical RS exploits item ratings, either implicit or explicit, from users in a system to predict a list of unseen items which are of potential interest to the user [77]. Over the years, several strategies have been developed to build efficient recommendation algorithms. They are generally divided into three main categories: collaborative filtering (CF) techniques, content-based (CB) methods and hybrid approaches [25]. While CF techniques exclusively rely on the feedback (rating, click, watch, listen) from the users on specific items without considering their description, either structured or unstructured, CB ones exploit the data associated to an item to compute relevant recommendations to a user. One of the main issues to tackle in adopting a CB approach is then getting the right amount of meaningful information about items, which in turns results necessary to model content-aware items and users descriptions. Attributes inferred by gathering data about items rated by users can be used, in principle, to model their profile and their preferences.
More recently, the technological wave related to deep learning techniques and approaches also hit the field of recommender systems. A variety of new approaches based on different configurations of Neural Networks (NNs) have been proposed to compute personalized lists of items to be suggested to the end users [46, 78, 50]. Among them, autoencoders have been proposed as an interesting tool to mimic the user behavior in producing ratings and by exploiting and modeling user preferences on latent item attributes [33]. Autoencoders are a particular configuration of artificial neural networks which turned out to be very effective especially for dimensionality reduction and feature selection tasks [79]. Indeed, in their mirrored structure, neurons of the hidden layers can be interpreted as a projection of the input layer in a different space. In [60], the authors presented semantics-aware autoencoders[1] which leverage the common graph-based structure to encode the semantics and the structure of a knowledge graph to enhance the representational power of the underlying NN. One of the main advantages of such a hybrid structure it that of giving an explicit semantics/label to the latent dimensions of the new space.

Among the various and diverse KG freely available on the Web, for sure DBpedia [2] and Wikidata [80] play a key role due to their encyclopedic nature which makes them the ideal candidates to provide structured descriptions on items in a recommender system. Although there is a partial overlapping among the information sources to build DBpedia and Wikidata, the data they encode is different under various aspects, such as the amount

[1] An implementation is available at: `https://github.com/sisinflab/SEMAUTO-2.0`

of data and the way it is organized [81].

In this paper, we analyze how the selection of different information from DBpedia and Wikidata may affect the results of a recommendation system in terms of accuracy and diversity of results when using a semantics-aware autoencoder. We tested different configurations of the semantics-aware autoencoder used to compute recommendations on three different datasets thus showing how critical may be the selection of the right ontological knowledge in a recommendation task. Main contributions of this work can be summarized as:

- An extensive evaluation of a semantics-aware autoencoder fed by knwoledge coming from DBpedia and Wikidata;
- a comparison with state of the art approaches in terms of accuracy and novelty of results;
- an analysis on how the structure and coverage of information encoded on a KG may affect accuracy and novelty of recommendation.

7.2 Related work

Very few works exist about qualitative studies of Linked Open Data (LOD) knowledge graphs. In [81] authors present an analysis of main Knowledge-Graphs such as DBpedia, Wikidata, YAGO, underlining their differences in coverage, identifying overlapping and complementary parts of KGs. They assert that KGs are not easily interchangeable and each of them has its strengths and weakness for a domain related task. Thus, using a specific KG that is suitable for the task to accomplish leads to better performances of the overall system. Authors in this work made a category-specific analysis, asserting that even if DBpedia and YAGO come from the same source (Wikipedia) and have a quite similar number of instances, there are notable differences in coverage. YAGO has five times the number of events of DBpedia, while DBpedia has four times as many settlements (i.e., cities and town) as YAGO; but Wikidata contains twice as many persons as DBpedia and YAGO. They conclude their investigation by providing a coverage summarization for some popular classes. A comparative survey of some popular KGs is done in [82] in which authors propose a method to find the most suitable KG for a given task setting. To achieve this result, authors identify a set of characteristics they found relevant to describe a KG and then compare different KGs accordingly. Furthermore in [83] they provide a more detailed analysis of quantitative information stored in KGs by using several

statistics such as the number of triples and classes, distribution of classes and corresponding instances, domain and classes coverage. Finally, to select the KG that best fits task requirements, a novel method that takes into account KG's quantitative assessment is proposed.

In the last few years, thanks to increasing computational resources, new techniques based on deep learning have been successfully adopted in the recommendation scenario [84]. This has led to the development of different neural network models, each capable of getting interesting results [85]. In particular, some of them turned out to be more effective for a specific recommendation task than others; e.g. autoencoders have proved their strength in collaborative filtering, outperforming state-of-the-art approaches [32], while Recurrent Neural Networks seem to be more suitable in session-based recommendation [37]. Among the several autoencoders extensions, denoising autoencoders have been efficiently used to address the recommendation problem by improving users' profile learning [33] or getting a smaller and non-linear representation of the User-Item rating matrix [47]. Furthermore, [86] shows how to build a hybrid RS by integrating side information in CF deep learning techniques, alleviating the sparsity problem and improving overall system performances.

LOD are increasingly adopted in recommender systems because they provide more complex structured data that leverages relationships among entities in the graph and moreover they encode somehow semantics behind the data, as shown in [3] and [5]. Recent works leveraged the data encoded in KGs to represent items thus achieving interesting results in recommendation scenarios [45, 87, 3]. Others approach as the one proposed by [88] uses kernel graphs to compute item similarities by matching their local neighborhood graphs. LODs have been also exploited for measuring semantic distances between resources in order to provide top-N recommendations [89]. In [60] a novel method that combines both deep learning techniques and KGs has been presented in which authors model a semantics-aware neural network that explicitly computes user profiles for recommendation tasks. In particular, they focus on cold-start scenarios using DBpedia as a source of information for both users and items modeling. In [90], authors used the aforementioned method to perform experiments in recommendation scenarios by using DBpedia KG on three different datasets and they compared their approach with state-of-the-art algorithms. Furthermore, they investigated the effectiveness of their method that leverages on a KG to provide an explanation in recommendation scenarios [91]. Another approach that uses KGs to represent relationship among users and items is investigated in [**?**], in which authors leveraged on language models to extract features from nodes sequences in RDF graphs.

7.3 Semantics-aware Autoencoders in Recommendation Scenarios

Autoencoders are unsupervised neural networks that learn a function capable of reconstructing the network's input at the output layer. They are built on top of two main components which are the encoder and the decoder. The former is usually responsible of compressing the input data into a lower dimensional representation, while the latter does the opposite job and it then reconstructs the original input data starting from a lower dimensional representation. Like every neural network configuration, autoencoders are structured in layers which contain neurons. Every neuron in layer i is connected through an edge to all the neurons of the following layer $i + 1$. In other words, we have a fully connected network.
In a recommendation scenario, we may use all the items in a catalog as representative of both the input and the output layer. We may then train the NN by using user ratings as inputs to obtain similar values produced by the output layers.
Starting from the observation that both NNs and KGs are directed graphs, in [60] the authors propose to use the topology of the latter to model the former. They keep input and output layers' nodes as representative of the items they want to recommend and substitute anonymous nodes in the hidden layers by labeled resources from a knowledge graph thus inheriting their mutual semantic connections (see Figure 7.1). Differently from the generic definition of autoencoder, we see here that the resulting neural network is no more fully connected as nodes of the input layer are linked to neurons in the hidden layer if and only if a corresponding connection exists in the original knowledge graph. In such a semantics-aware autoencoder, we somehow project the items in a space whose dimensions represent all the entities (features) it is connected to.

A semantics-aware autoencoder is trained with user ratings and it learns how to reconstruct them on the output layer. Considering that such a network is not fully connected, user ratings are propagating only through those nodes that represent features connected in the KG to items rated by the user. According to neural network models, a generic neuron outputs a value that results to be a non-linear function of the weights' summation over incoming edges. Then, in a recommendation scenario it turns out that positively rated items tend to have connected neurons with higher output values than those neurons connected with negatively rated items.

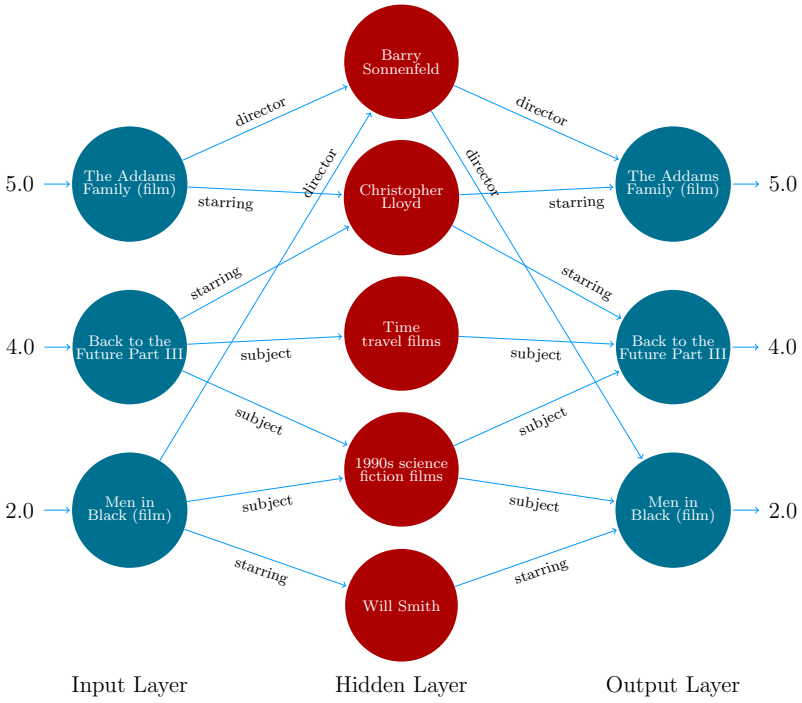


Figure 7.1: Architecture of a semantics-aware autoencoder.

7.3.1 User Profiles

If we train one autoencoder per user, the resulting model may be interpreted as an explicit representation of the user profile on items attributes. As a matter of fact, at the end of training, hidden nodes encode a value that represents the relevance for the user in the node's associated feature. Thus, sets of pairs $\langle feature, value \rangle$ can be defined and used as a representation of the user profile. A semantics-aware autoencoder is, therefore, a model that uses deep learning techniques to extract weighted features from a KG according to user ratings in order to build users profile for recommendation tasks. In particular, given a user u, the weight of a feature c is the summation of the weights $w_k^u(c)$ associated to the edges entering the hidden node representing a KG entity c (see Figure 7.2).
More formally, we have:

$$\omega^u(c) = \sum_{k=1}^{|In(c)|} w_k^u(c)$$

where $In(c)$ is the set of the edges entering the hidden node representing the feature c. As an example, if we consider the excerpt of the network in Figure

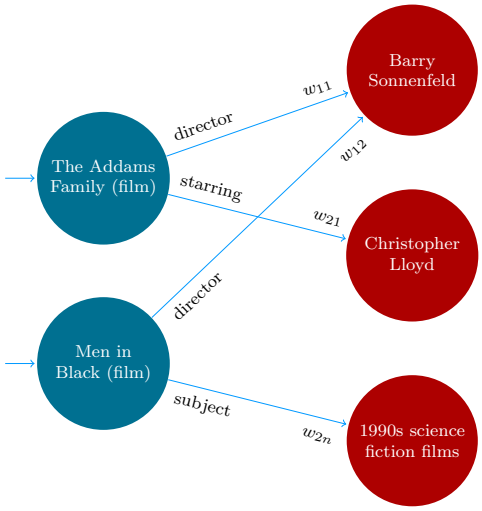


Figure 7.2: An excerpt of the network in Figure 7.1 after the training.

7.2, for `Barry Sonnenfeld` we have:

$$\omega^u(\texttt{Barry Sonnenfeld}) = w_{11} + w_{12}$$

Having weights associated to each resource coming from KG, we can now model a user profile composed by a vector of weighted features. Given S^u as the set of features belonging to all the items rated by u and $S = \bigcup_{u \in U} S^u$ as the set of all features among all the users in the system we have that for each user $u \in U$ and for each feature $c \in S$, the user profile $P(u)$ is represented as:

$$P(u) = \{\langle c, \omega \rangle \mid \omega = \omega^u(c) \text{ if } c \in S^u, \omega = 0 \text{ otherwise}\} \tag{7.1}$$

Recommendation. The vector representing a user profile computed with Equation (7.1) results to be very sparse in the space representing the overall number of features. In other words, many values are set to 0 as the corresponding feature does not belong to any item the user rated in the past. Since this may negatively affect the final result in a recommendation scenario, we reduce the sparseness of the user profile by predicting a value to fill 0-valued features in the vector through the word2vec-like approach as described in [90].

Once we have a less sparse version of the user profile, we produce recommendation performing a user-kNN (see Equation (7.3)) by computing how close are users with each other through a cosine similarity as in Equation (7.2).

$$sim(u, v) = \frac{P(u) \cdot P(v)}{||P(u)|| \cdot ||P(v)||} \tag{7.2}$$

$$\hat{r}(u, i) = \frac{\sum_{j=1}^{k} sim(u, v_j) \cdot r(v_j, i)}{\sum_{j=1}^{k} sim(u, v_j)} \tag{7.3}$$

In Equation (7.3), given an unseen item i from user u, we predict a rating $\hat{r}(u, i)$ for u on i by considering the ratings $r(v_j, i)$ assigned to i by the k most similar user v_j.

7.4 Experiments

As a direct consequence of relying on a knowledge graph, it stands to reason that its structure, as well as the information it encodes, might affect how user profiles are computed. Thus, it is crucial to investigate how KGs structure impacts the recommendation accuracy. We strongly believe that the better the data are engineered and well-curated, the more accurate a recommendation is.

Given the recommendation model previously described, we performed an experimental evaluation to verify the influence of a knowledge source in the final recommendation task. In this paper, we focus on different aspects of DBpedia and Wikidata due to the richness of information they encode in different knowledge domains. DBpedia and Wikidata differ from each other not only by their structure but as pointed out in [81], the choice of a KG is domain dependent because different fields of knowledge are covered in different ways among different KGs.

We first describe the structure of the datasets used in the experiments, then we move on to the evaluation protocol for the recommendation, and finally we discuss the results.

7.4.1 Dataset

We conducted our experiments on three different datasets as summarized in Table 7.1. MovieLens 1M[2] dataset stores information about users-items interactions made on a 5-star scale and relates to the movie domain. Last.fm[3] contains information about music, bands and artists listenings; since in Last.fm for each user we have the number of times a user has listened to a song,

[2] `http://grouplens.org/datasets/movielens/`

[3] `http://www.lastfm.com`

	#users	#items	#ratings	sparsity
MovieLens 1M	6040	3952	1000209	95.81%
Last.fm	1892	17632	92834	99.72%
LibraryThing	7279	37232	626000	99.77%

Table 7.1: Datasets

we infer users' preferences by scaling it within the range $[1, ..., 10]$ (using min-max normalization). Finally in LibraryThing[4], which is a social web application for book cataloging, rates are made on a 10-star scale with reference to books. By relaying on these datasets, we have three different knowledge domains which are covered both by DBpedia and by Wikidata.

In order to map items to resources in DBpedia we adopted a freely available mapping[5] originally presentedin in [92] and then refined in [93]. Thus, we retain 3549 mapped items for MovieLens 1M, 9781 for Last.fm and 9926 for LibraryThing. Starting from DBpedia resources URI we obtained Wikidata entities through `owl:sameAs` links.

7.4.2 Knowledge-Graphs: DBpedia vs Wikidata

In order to evaluate how KGs data may impact the quality of recommendation, we performed experiments using DBpedia and Wikidata. In addition to this, we also evaluated how different kind of information from a KG impacts the recommendation. If we look at the semantic knowledge they encode, we may identify **factual knowledge** where we have facts stated on a specific resource, e.g. `dbr:Men_in_Black_(Film) dbo:director dbr:Barry_Sonnenfeld`, and **ontological** and **categorical** one which encode the semantics of an entity through classes and categories, such as `dbr:Men_in_Black_(Film) dct:subject dbc:Buddy_Film` or `dbr:Men_in_Black_(Film) rdf:type dbo:Film`. In DBpedia, categorical information is reached through the following predicates:

- `http://purl.org/dc/terms/subject`
- `http://www.w3.org/2009/08/skos-reference/skos.html#broader`

The former allows us to explore categorical resources related to an item, while the latter lets us discover a wider category in a hierarchical perspective.

[4] `https://www.librarything.com/`

[5] `https://github.com/sisinflab/LODrecsys-datasets`

As for Wikidata, we considered categorical information encoded through the predicate `https://www.wikidata.org/wiki/Property:P921` labeled as *main subject* and, whenever possible, by `https://www.wikidata.org/wiki/Property:P136` labeled as *genre*. Since DBpedia `skos:broader` is not directly mapped in Wikidata, we used `https://www.wikidata.org/wiki/Property:P279`, labeled as *subclass of*, to identify hierarchical categories as well.

Regarding factual information, we used the approach proposed in [75] to automatically identify those DBpedia predicates (listed in Table 7.2) which turn out to be the most meaningful for a recommendation task; the corresponding Wikidata properties were properly collected through SPARQL queries.

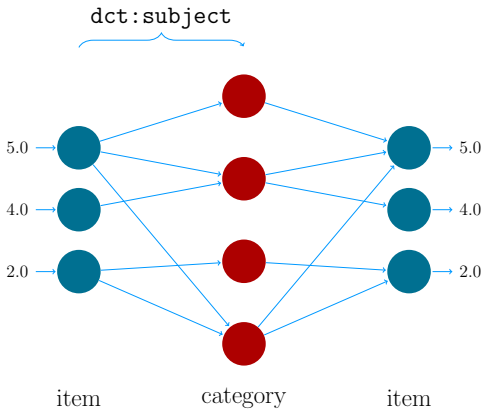
Last.fm	LibraryThing	MovieLens 1M
dbo:genre	dbo:author	dbo:starring
dbo:instrument	dbo:literaryGenre	dbo:director
dbo:occupation	dbo:publisher	dbo:writer
dbo:associatedBand	dbo:mediaType	dbo:producer
dbo:associatedMusicalArtist	dbo:language	dbo:musicComposer
dbo:recordLabel	dbo:country	dbo:distributor
dbo:hometown	dbo:previousWork	dbo:language
dbo:birthPlace	dbo:subsequentWork	dbo:cinematography
dbo:country	dbo:nonFictionSubject	dbo:country
dbo:influencedBy	dbo:series	dbo:editing
dbo:voiceType	dbo:coverArtist	dbp:music
dbo:award	dbo:illustrator	dbp:studio
dbo:bandMember	dbo:translator	dbp:extra
dbo:currentMember	dbp:awards	dbp:screenplay
dbo:pastMember	dbp:writer	dbp:genre

Table 7.2: DBpedia predicates

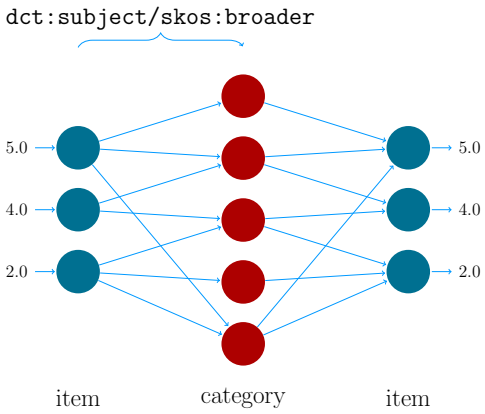
7.4.3 Data Settings

Here we show the different configurations we adopted to inject data from DBpedia and Wikidata in our semantics-aware autoencoder. As stated in Section 7.3, the input and output layers are always composed by resources representing items in our recommendation setting (e.g. movies for MovieLens 1M, books for LibraryThing, songs, bands, etc. for Last.fm). The differences

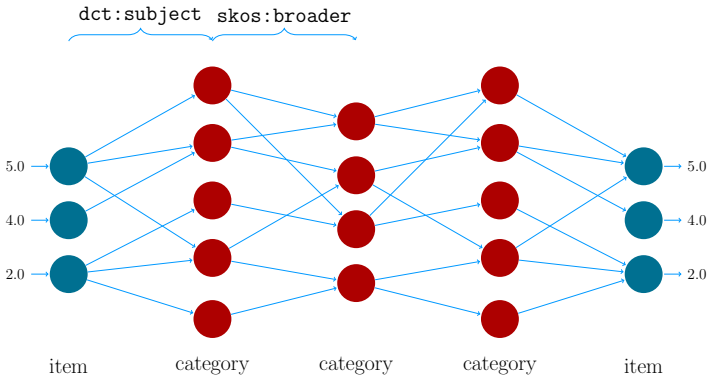
of the configurations we propose mainly rely on the information encoded in the hidden layers. In Figure 7.3 we show only the case for DBpedia, as for Wikidata we have analogous configurations.



(a) Categories reached by the `dct:subject` property encoded in the hidden layer.



(b) Categories reached by the `dct:subject/skos:broader` property chain encoded in the hidden layer.



(c) Subjects and broaders distributed among different hidden layers.

Figure 7.3: The different configurations used in our experiments.

The first configuration (Figure 7.3a) encodes only categories through the `dct:subject` property. Hence, all the nodes in the hidden layer have a one-to-one mapping with a corresponding category in DBpedia. We will refer to this configuration with `S` for *subject*. As categories are structured in a hierarchical way through `skos:broader`, we also tested the following configurations:

`B` We considered in the hidden layer only those categories which are one-

hop distant through the `skos:broader` property from those resources directly connected to an item via `dct:subject` (Figure 7.3b). In this configuration, connections between the items and the hidden layer are represented by the property chain `dct:subject/skos:broader`;

`M` (for Mixed) We put in the same hidden layer both the categories connected via `dct:subject` and `dct:subject/skos:broader`;

`S-B` In this last configuration we considered three hidden layers[6] thus mimicking the actual topology of the knowledge graph in the structure of the NN (see Figure 7.3c);

Finally, we have the `F` (for `Factual`) configuration where the hidden layer is composed by all the resources which are at a distance of one-hop from the input item through the properties in Table 7.2.

7.4.4 Evaluation

Before measuring how different KGs impact on recommendation quality, we first prove the strength of the proposed method by comparing it with some state-of-the-art baselines. Then we quantify how our performances vary depending on the different subset of KG used.

For the evaluation of our approach we adopted the "All Unrated Items" protocol described in [52]: for each user u, a top-N recommendation list is provided by computing a score for every item i not rated by u, whether i appears in the user test set or not. Training and test sets are generated by splitting each dataset with Hold-Out 80/20, which ensures that every user has 80% of their ratings in the training set and the remaining 20% in the test set.
The produced recommendation lists are finally compared with the test set by computing performance metrics. Precision, Recall and F1-score [94] metrics have been chosen to evaluate the accuracy of our model in a top-10 recommendation scenario, using threshold values of 4 for MovieLens 1M and 8 for both LibraryThing and Last.fm.

Accuracy metrics are a valuable way to evaluate the performance of a recommender system. Nonetheless, it has been argued [95] that also diversity should be taken into account when evaluating how good a recommendation engine is. Gini index is an ideal candidate to measure the distribu-

[6]Due to the mirrored structure of an autoencoder, the numebr of layers is always odd

tion/diversity of items across recommendation lists:

$$Gini = \frac{1}{n-1} \cdot \sum_{j=1}^{n} (2j - n - 1) \cdot p(i_j)$$

where $p(i_j)$ is the proportion of user choices for item i_j and $i_1, ...i_n$ is the list of items ordered according to increasing $p(i_j)$. A Gini index value equal to 0 means that all items are chosen equally often, while it is 1 if a single item is always chosen.

Among the several state-of-the-art techniques used in recommender scenarios, we tested the most widely adopted: BPRSLIM, WRMF and a single-layer autoencoder for rating prediction. BPRSLIM [96, 55] is a Sparse Linear Method which leverages Bayesian Personalized Ranking as an objective function. WRMF [56, 57] is a Weighted Regularized Matrix Factorization method which exploits users' implicit feedback to provide recommendations. In their basic version, both strategies rely exclusively on the User-Item matrix in a pure CF approach. They can be hybridized by exploiting side information (SI) [96], i.e. additional data associated with items. We used the implementations of BPRSLIM and WRMF available in MyMediaLite[7] [97] and implemented the autoencoder in TensorFlow[8].

7.5 Results discussion

For all the tested approaches, we performed experiments on MovieLens 1M, Last.fm and LibraryThing.

In Table 7.3 we report the best results[9] we gathered on the three datasets by applying the methods discussed above. As for our SEMAUTO approach, we tested it for a different number of neighbors k (only the best results are shown in the table). Further, we report only results for information found in DBpedia as side information (BPRSLIM + SI and WRMF + SI) because it turned out that DBpedia gains better accuracy than Wikidata in recommendation tasks. We highlighted in bold the best performing approach overall while we underline the best performing configuration for our semantics-aware autoencoder (KG-AUTOENCODER).

We first discuss results obtained in terms of accuracy and then we move on to diversity. As we can see from the table, our semantics-aware autoencoder outperforms all the baselines for Last.fm dataset, while on the other

[7] `http://mymedialite.net`

[8] `https://www.tensorflow.org/`

[9] Full results table at `https://github.com/sisinflab/papers-results/blob/master/ISWC/2019/KG-AUTOENCODER/results.pdf`

	setting	KG	k	F1@10	Prec@10	Recall@10	Gini
LAST.FM							
AUTOENCODER	–	–	–	0.00048	0.00027	0.00240	**0.00190**
BPRSLIM	–	–	–	0.00077	0.00043	0.00400	0.02867
BPRSLIM + SI	–	DBpedia	–	0.00113	0.00064	0.00476	0.05360
WRMF	–	–	–	0.00077	0.00043	0.00400	0.01073
WRMF + SI	–	DBpedia	–	0.00058	0.00032	0.00293	0.00877
SEMAUTO	S	DBpedia	5	0.00151	0.00085	0.00644	0.05783
	S	Wikidata	10	0.00124	0.00069	0.00587	0.03815
	B	DBpedia	10	0.00113	0.00064	0.00484	0.03812
	B	Wikidata	30	0.00086	0.00048	0.00391	0.01430
	M	DBpedia	5	0.00151	0.00085	0.00644	0.05783
	M	Wikidata	5	0.00067	0.00037	0.00320	0.05317
	S-B	DBpedia	5	0.00111	0.00064	0.00422	0.05624
	S-B	Wikidata	5	**0.00172**	**0.00096**	**0.00844**	0.05742
	F	DBpedia	5	0.00169	0.00096	0.00689	0.05878
	F	Wikidata	10	0.00143	0.00080	0.00693	0.03689
LIBRARYTHING							
AUTOENCODER	–	–	–	0.01562	0.01375	0.01808	0.07628
BPRSLIM	–	–	–	0.01874	0.01577	**0.02309**	0.09338
BPRSLIM + SI	–	DBpedia	–	**0.01939**	**0.01685**	0.02284	0.17915
WRMF	–	–	–	0.01142	0.01071	0.01223	0.00864
WRMF + SI	–	DBpedia	–	0.01136	0.01043	0.01247	0.00832
SEMAUTO	S	DBpedia	100	0.01293	0.01168	0.01447	0.01855
	S	Wikidata	100	0.00993	0.00888	0.01125	0.01124
	B	DBpedia	45	0.01264	0.01139	0.01420	0.02475
	B	Wikidata	100	0.00791	0.00741	0.00848	0.00983
	M	DBpedia	45	0.01299	0.01164	0.01469	0.02938
	M	Wikidata	150	0.00867	0.00805	0.00941	**0.00770**
	S-B	DBpedia	100	0.01390	0.01247	0.01570	0.01205
	S-B	Wikidata	100	0.00995	0.00892	0.01123	0.00950
	F	DBpedia	40	0.01468	0.01306	0.01677	0.02888
	F	Wikidata	45	0.01278	0.01153	0.01435	0.02237
MOVIELENS 1M							
AUTOENCODER	–	–	–	**0.22969**	**0.28416**	0.19274	0.04536
BPRSLIM	–	–	–	0.17106	0.19581	0.15187	0.14060
BPRSLIM + SI	–	DBpedia	–	0.14986	0.17113	0.13329	0.17294
WRMF	–	–	–	0.20336	0.25343	0.16981	0.03758
WRMF + SI	–	DBpedia	–	0.20373	0.25371	0.17020	0.03750
SEMAUTO	S	DBpedia	50	0.18582	0.22419	0.15867	0.02298
	S	Wikidata	100	0.16809	0.21619	0.13749	0.01712
	B	DBpedia	45	0.17640	0.21369	0.15019	0.02207
	B	Wikidata	100	0.15487	0.20555	0.12424	0.01611
	M	DBpedia	45	0.18633	0.22430	0.15935	0.02421
	M	Wikidata	100	0.15592	0.20046	0.12757	**0.01378**
	S-B	DBpedia	50	0.22001	0.26616	0.18749	0.03653
	S-B	Wikidata	100	0.15800	0.20394	0.12896	0.01574
	F	DBpedia	50	0.22447	0.26788	**0.19317**	0.04446
	F	Wikidata	50	0.17150	0.21149	0.14423	0.01872

Table 7.3: Experimental results over DBpedia and Wikidata KGs using both factual and semantics information. For BPRSLIM + SI and WRMF + SI in KG column, we indicate only the Knowledge Graph for which we have the best performance when used to get side information.
In setting column we denote: `S` = subjects, `B` = broaders, `M` = merge `S` and `B` in a single hidden layer, `S-B` = `S` and `B` in multiple hidden layers, `F` = factual information.

two, LibraryThing and MovieLens 1M, it performs quite the same as the fully-connected autoencoder does. To explain this result, we may assume that, since hidden nodes in our semantics-aware autoencoder depends on the number of features retrieved from a Knowledge Graph, more features means more neurons. According to the Universal Approximation Theorem, more neurons allows a neural network to approximate better any function hence we computed the ratio of features associated to items[10] as $\frac{average\#features}{average\#items}$ and we found that this approach works better on those datasets having a higher ratio.

Nonetheless, looking at the sparsity values reported in Table 7.1, we can notice that even if the KG-AUTOENCODER obtains quite the same values on LibraryThing and Last.fm datasets, they differ too much in the ratio of average features per item. In fact, the KG-AUTOENCODER performs better on the latter where the ratio is higher and worst in the former where the ratio is lower. Hence, we may assume that this method is less sensitive to the dataset sparsity with respect to how well data are curated in the KG and for that reason we suppose it can be used as an assessment tool for data quality in KGs when used for recommendation tasks.

On the other side, we can observe that factual information (F) brings more knowledge for a recommendation task than ontological/categorical one when the datasets are very sparse (as for MovieLens 1M and LibraryThing). This is a quite interesting result. A possible explanation is that categorical information introduces fewer connections among items descriptions than the factual one. Hence, factual statements result more useful in making denser connections among items (and then users) which are exploited by the latent collaborative part of the semantics-aware autoencoder.

If we just focus on the absolute numbers, one may argue that our approach is not competitive as it is slightly beaten in terms of accuracy by state of the art algorithms such as BPRSLIM and WRMF (although it results the second best performing approach). Nevertheless, we point out that, differently from the other approaches based on matrix factorization (or any deep learning techniques) we compute a meaningful and explicit user profile which contains user preferences on single features. This may result extremely useful in case we want to automatically generate a content-based explanation for the ranking computed with the recommendation list as also shown in [91]. Then, although we rely on a deep learning approach, we can go beyond the pure black-box and provide a human-understandable explanation for a recommendation list.

[10] https://github.com/sisinflab/papers-results/blob/master/ISWC/2019/KG-AUTOENCODER/summary.pdf

As for diversity, we can observe that when using Wikidata, values for Gini index are in general lower than the DBpedia case. This means that using Wikidata we are able to better diversify the items in a catalog. This result somehow reinforces the one obtained in [98]. Lack of diversity in recommendation strongly depends on the so-called "popularity bias". Popular items tend to be recommended more than those in the long tail. This observation leads us to a possible interpretation on diversity results we obtain in our experiments if we consider the popularity of entities in a knowledge graph as the number of connections they have. In DBpedia we have that popular resources (e.g. movies) are more connected to other nodes than unpopular ones. This is not the case with Wikidata where there is a less biased distribution of connections among resources in the graph. Hence, when the adopted knowledge graph suffers from a popularity bias in terms of connections among resources, this is inherited by the recommendation dataset thus affecting the final recommended list of results.

7.6 Conclusion and future work

In this paper, we presented an evaluation of a semantics-aware autoencoder used to predict ratings in a recommendation scenario. We showed how such an approach makes it possible to combine the computational predictive power of NNs (in the form of autoencoders) with the representational power of knowledge graphs such as DBpedia and Wikidata. In particular, we evaluated our autoencoder on different configurations, and we tested and compared its results in terms of accuracy and diversity in the recommendation list. On the one hand, we showed that the proposed approach is able to compete with state of the art approaches with the advantage of, in principle, allowing a system to automatically generate explanations for the computed results. On the other hand, we confirmed that the selection of the right information from the right knowledge graph may affect recommendations both in terms of accuracy and in terms of diversity of results. As an example, Wikidata seems to be a better choice than DBpedia if we look for recommendation lists where the popularity bias is mitigated. We are currently working on a new version of the SEMAUTO that better exploits collaborative information by building a global model encoding preferences from all the users at the same time, as done by state-of-the-art CF approaches.

Chapter 8

Conclusions

The final result of this thesis is an intensive investigation about interpretable neural network models using Knowledge Graphs with a focus on their explainability in recommendation scenarios.

First, we proposed a new not fully-connected architecture for Autoencoder Neural Networks that leverage Knowledge Graph to map the topology of the former with the latter. Subsequently, we evaluated this novel approach against state-of-the-art baselines on different datasets. Then, we performed an A/B test to evaluate the explanations that the Semantics-Aware Autoencoder generates. Moreover, we compared how the use of different features from a Knowledge Graph affects the satisfaction of users. Finally, since the proposed method relies on Knowledge Graphs, we evaluated how different Knowledge Graphs and various features impact the accuracy of recommendations.

Our experiments validate the proposed approach, showing its ability to overcome the neural network interpretability without losing accuracy. Nevertheless, our method turns out to be very effective in generating discriminative user profiles; in fact, the number of suggested items is higher with respect to baselines. Moreover, the proposed approach, since it relies on Knowledge Graph to compute the recommendations and since the effectiveness of recommendations depends on how features are well-engineering in the Knowledge Graphs, it turns out that this method can be used as a tool to evaluate the quality of how well engineered are data in a Knowledge Graph.

As future work, this method can be enhanced in order to leverage the collaborative information within the neural network, instead of using a word2vec-like approach to infer the most likely missing features for each user profile. Moreover, the Semantics-Aware Autoencoder can be used as a stand-alone block to place at the end of a Deep Learning pipeline in order to provide explanations to the users. Concerning the Linked Open Data quality aspect, it can be leveraged to perform a Knowledge Graph summarization since it can weigh the most relevant nodes (features) in a graph, so it can compress a Knowledge Graph preserving the its main entities. Nevertheless, this approach can be investigated even in scenarios different from the recommendation one; for instance, it could be used as an explainable model for decision making.

Bibliography

[1] Lisa Ehrlinger and Wolfram Wöß. Towards a definition of knowledge graphs. 09 2016.

[2] Sören Auer, Christian Bizer, Georgi Kobilarov, Jens Lehmann, Richard Cyganiak, and Zachary Ives. Dbpedia: A nucleus for a web of open data. In *Proceedings of the 6th International The Semantic Web and 2Nd Asian Conference on Asian Semantic Web Conference*, ISWC'07/ASWC'07, pages 722–735, Berlin, Heidelberg, 2007. Springer-Verlag.

[3] Tommaso Di Noia, Roberto Mirizzi, Vito Claudio Ostuni, Davide Romito, and Markus Zanker. Linked open data to support content-based recommender systems. In *Proceedings of the 8th International Conference on Semantic Systems*, I-SEMANTICS '12, pages 1–8, New York, NY, USA, 2012. ACM.

[4] Benjamin Heitmann and Conor Hayes. C.: Using linked data to build open, collaborative recommender systems. In *In: AAAI Spring Symposium: Linked Data Meets Artificial Intelligence'. (2010*, 2010.

[5] Marco de Gemmis, Pasquale Lops, Cataldo Musto, Fedelucio Narducci, and Giovanni Semeraro. *Semantics-Aware Content-Based Recommender Systems*, pages 119–159. 01 2015.

[6] Francesco Ricci, Lior Rokach, and Bracha Shapira. *Recommender Systems Handbook*. Springer Publishing Company, Incorporated, 2nd edition, 2015.

[7] Gediminas Adomavicius and Alexander Tuzhilin. Toward the next generation of recommender systems: A survey of the state-of-the-art and possible extensions. *IEEE Trans. on Knowl. and Data Eng.*, 17(6):734–749, June 2005.

[8] Gerard Salton. *Automatic Text Processing: The Transformation, Analysis, and Retrieval of Information by Computer*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1989.

[9] Robert M. Bell and Yehuda Koren. Scalable collaborative filtering with jointly derived neighborhood interpolation weights. In *Proceedings of the 2007 Seventh IEEE International Conference on Data Mining*, ICDM '07, pages 43–52, Washington, DC, USA, 2007. IEEE Computer Society.

[10] John S. Breese, David Heckerman, and Carl Kadie. Empirical analysis of predictive algorithms for collaborative filtering. In *Proceedings of the Fourteenth Conference on Uncertainty in Artificial Intelligence*, UAI'98, pages 43–52, San Francisco, CA, USA, 1998. Morgan Kaufmann Publishers Inc.

[11] Mukund Deshpande and George Karypis. Item-based top-n recommendation algorithms. *ACM Trans. Inf. Syst.*, 22(1):143–177, January 2004.

[12] Joseph A. Konstan, Bradley N. Miller, David Maltz, Jonathan L. Herlocker, Lee R. Gordon, and John Riedl. Grouplens: Applying collaborative filtering to usenet news. *Commun. ACM*, 40(3):77–87, March 1997.

[13] Will Hill, Larry Stead, Mark Rosenstein, and George Furnas. Recommending and evaluating choices in a virtual community of use. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, CHI '95, pages 194–201, New York, NY, USA, 1995. ACM Press/Addison-Wesley Publishing Co.

[14] Upendra Shardanand and Pattie Maes. Social information filtering: Algorithms for automating “word of mouth”. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, CHI '95, pages 210–217, New York, NY, USA, 1995. ACM Press/Addison-Wesley Publishing Co.

[15] Greg Linden, Brent Smith, and Jeremy York. Amazon.com recommendations: Item-to-item collaborative filtering. *IEEE Internet Computing*, 7(1):76–80, January 2003.

[16] Paul Resnick, Neophytos Iacovou, Mitesh Suchak, Peter Bergstrom, and John Riedl. Grouplens: An open architecture for collaborative filtering of netnews. In *Proceedings of the 1994 ACM Conference on Computer Supported Cooperative Work*, CSCW '94, pages 175–186, New York, NY, USA, 1994. ACM.

[17] Francois Fouss, Alain Pirotte, Jean-Michel Renders, and Marco Saerens. Random-walk computation of similarities between nodes of a graph with application to collaborative recommendation. *IEEE Trans. on Knowl. and Data Eng.*, 19(3):355–369, March 2007.

[18] Nathaniel Good, J. Ben Schafer, Joseph A. Konstan, Al Borchers, Badrul Sarwar, Jon Herlocker, and John Riedl. Combining collaborative filtering with personal agents for better recommendations. In *Proceedings of the Sixteenth National Conference on Artificial Intelligence and the Eleventh Innovative Applications of Artificial Intelligence Conference Innovative Applications of Artificial Intelligence*, AAAI '99/IAAI '99, pages 439–446, Menlo Park, CA, USA, 1999. American Association for Artificial Intelligence.

[19] David M. Blei, Andrew Y. Ng, and Michael I. Jordan. Latent dirichlet allocation. *J. Mach. Learn. Res.*, 3:993–1022, March 2003.

[20] Ruslan Salakhutdinov, Andriy Mnih, and Geoffrey Hinton. Restricted boltzmann machines for collaborative filtering. In *Proceedings of the 24th International Conference on Machine Learning*, ICML '07, pages 791–798, New York, NY, USA, 2007. ACM.

[21] Steffen Rendle. Factorization machines. In *Proceedings of the 2010 IEEE International Conference on Data Mining*, ICDM '10, pages 995–1000, Washington, DC, USA, 2010. IEEE Computer Society.

[22] Scott Deerwester, Susan T. Dumais, George W. Furnas, Thomas K. Landauer, and Richard Harshman. Indexing by latent semantic analysis. *JOURNAL OF THE AMERICAN SOCIETY FOR INFORMATION SCIENCE*, 41(6):391–407, 1990.

[23] Yehuda Koren. Factorization meets the neighborhood: A multifaceted collaborative filtering model. In *Proceedings of the 14th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '08, pages 426–434, New York, NY, USA, 2008. ACM.

[24] Gábor Takács, István Pilászy, Bottyán Németh, and Domonkos Tikk. Major components of the gravity recommendation system. *SIGKDD Explor. Newsl.*, 9(2):80–83, December 2007.

[25] Robin Burke. Hybrid recommender systems: Survey and experiments. *User Modeling and User-Adapted Interaction*, 12(4):331–370, November 2002.

[26] Paolo Cremonesi, Roberto Turrin, Eugenio Lentini, and Matteo Matteucci. An evaluation methodology for collaborative recommender systems. In *Proceedings of the 2008 International Conference on Automated Solutions for Cross Media Content and Multi-channel Distribution*, AXMEDIS '08, pages 224–231, Washington, DC, USA, 2008. IEEE Computer Society.

[27] Alejandro Bellogín, Iván Cantador, and Pablo Castells. A comparative study of heterogeneous item recommendations in social systems. *Information Sciences*, 221:142 – 169, 2013.

[28] G.E. Hinton and R.R. Salakhutdinov. Reducing the dimensionality of data with neural networks. *Science (New York, N.Y.)*, 313:504–7, 08 2006.

[29] Gintare Karolina Dziugaite and Daniel M Roy. Neural network matrix factorization. *arXiv preprint arXiv:1511.06443*, 2015.

[30] Huifeng Guo, Ruiming Tang, Yunming Ye, Zhenguo Li, and Xiuqiang He. Deepfm: a factorization-machine based neural network for ctr prediction. *arXiv preprint arXiv:1703.04247*, 2017.

[31] Heng-Tze Cheng, Levent Koc, Jeremiah Harmsen, Tal Shaked, Tushar Chandra, Hrishi Aradhye, Glen Anderson, Greg Corrado, Wei Chai, Mustafa Ispir, et al. Wide & deep learning for recommender systems. In *Proceedings of the 1st workshop on deep learning for recommender systems*, pages 7–10. ACM, 2016.

[32] Suvash Sedhain, Aditya Krishna Menon, Scott Sanner, and Lexing Xie. Autorec: Autoencoders meet collaborative filtering. In *Proceedings of the 24th International Conference on World Wide Web*, WWW '15 Companion, pages 111–112, New York, NY, USA, 2015. ACM.

[33] Yao Wu, Christopher DuBois, Alice X. Zheng, and Martin Ester. Collaborative denoising auto-encoders for top-n recommender systems. In *Proceedings of the Ninth ACM International Conference on Web Search and Data Mining*, WSDM '16, pages 153–162, New York, NY, USA, 2016. ACM.

[34] Dawen Liang, Rahul G Krishnan, Matthew D Hoffman, and Tony Jebara. Variational autoencoders for collaborative filtering. In *Proceedings of the 2018 World Wide Web Conference*, pages 689–698. International World Wide Web Conferences Steering Committee, 2018.

[35] Yuanxin Ouyang, Wenqi Liu, Wenge Rong, and Zhang Xiong. Autoencoder-based collaborative filtering. In *International Conference on Neural Information Processing*, pages 284–291. Springer, 2014.

[36] Hao Wang, Naiyan Wang, and Dit-Yan Yeung. Collaborative deep learning for recommender systems. In *Proceedings of the 21th ACM SIGKDD international conference on knowledge discovery and data mining*, pages 1235–1244. ACM, 2015.

[37] Balázs Hidasi, Alexandros Karatzoglou, Linas Baltrunas, and Domonkos Tikk. Session-based recommendations with recurrent neural networks. In *6th International Conference on Learning Representations*, 2015.

[38] Shuai Zhang, Lina Yao, Aixin Sun, and Yi Tay. Deep learning based recommender system: A survey and new perspectives. *ACM Computing Surveys (CSUR)*, 52(1):5, 2019.

[39] Marco Tulio Ribeiro, Sameer Singh, and Carlos Guestrin. "why should i trust you?": Explaining the predictions of any classifier. In *Proceedings of the 22Nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '16, pages 1135–1144, New York, NY, USA, 2016. ACM.

[40] Quanshi Zhang, Ying Nian Wu, and Song-Chun Zhu. Interpretable convolutional neural networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 8827–8836, 2018.

[41] Sungyong Seo, Jing Huang, Hao Yang, and Yan Liu. Interpretable convolutional neural networks with dual local and global attention for review rating prediction. In *Proceedings of the Eleventh ACM Conference on Recommender Systems*, RecSys '17, pages 297–305, New York, NY, USA, 2017. ACM.

[42] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Ł ukasz Kaiser, and Illia Polosukhin. Attention is all you need. In I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, editors, *Advances in Neural Information Processing Systems 30*, pages 5998–6008. Curran Associates, Inc., 2017.

[43]

[44] T. Di Noia, V.C. Ostuni, J. Rosati, P. Tomeo, E. Di Sciascio, R. Mirizzi, and C. Bartolini. Building a relatedness graph from linked open data: A

case study in the it domain. *Expert Systems with Applications*, 44:354–366, 2016.

[45] S. Oramas, V.C. Ostuni, T. Di Noia, X. Serra, and E. Di Sciascio. Sound and music recommendation with knowledge graphs. *ACM Transactions on Intelligent Systems and Technology*, 8(2), 2016.

[46] Paul Covington, Jay Adams, and Emre Sargin. Deep neural networks for youtube recommendations. In *Proceedings of the 10th ACM Conference on Recommender Systems*, pages 191–198. ACM, 2016.

[47] Florian Strub, Romaric Gaudel, and Jérémie Mary. Hybrid recommender system based on autoencoders. In *Proceedings of the 1st Workshop on Deep Learning for Recommender Systems*, DLRS 2016, pages 11–16, New York, NY, USA, 2016. ACM.

[48] Hao Wang, Naiyan Wang, and Dit-Yan Yeung. Collaborative deep learning for recommender systems. In *Proceedings of the 21th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '15, pages 1235–1244, New York, NY, USA, 2015. ACM.

[49] Jeroen B. P. Vuurens, Martha Larson, and Arjen P. de Vries. Exploring deep space: Learning personalized ranking in a semantic space. In *Proceedings of the 1st Workshop on Deep Learning for Recommender Systems*, DLRS 2016, pages 23–28, New York, NY, USA, 2016. ACM.

[50] Ali Mamdouh Elkahky, Yang Song, and Xiaodong He. A multi-view deep learning approach for cross domain user modeling in recommendation systems. In *Proceedings of the 24th International Conference on World Wide Web*, WWW '15, pages 278–288, Republic and Canton of Geneva, Switzerland, 2015. International World Wide Web Conferences Steering Committee.

[51] Xavier Glorot and Yoshua Bengio. Understanding the difficulty of training deep feedforward neural networks. In *Proceedings of the thirteenth international conference on artificial intelligence and statistics*, pages 249–256, 2010.

[52] Harald Steck. Evaluation of recommendations: Rating-prediction and ranking. In *Proceedings of the 7th ACM Conference on Recommender Systems*, RecSys '13, pages 213–220, New York, NY, USA, 2013. ACM.

[53] Kalervo Järvelin and Jaana Kekäläinen. Ir evaluation methods for retrieving highly relevant documents. In *Proceedings of the 23rd Annual*

International ACM SIGIR Conference on Research and Development in Information Retrieval, SIGIR '00, pages 41–48, New York, NY, USA, 2000. ACM.

[54] Daniel M Fleder and Kartik Hosanagar. Recommender systems and their impact on sales diversity. In *Proceedings of the 8th ACM conference on Electronic commerce*, pages 192–199. ACM, 2007.

[55] Steffen Rendle, Christoph Freudenthaler, Zeno Gantner, and Lars Schmidt-Thieme. Bpr: Bayesian personalized ranking from implicit feedback. In *Proceedings of the Twenty-Fifth Conference on Uncertainty in Artificial Intelligence*, UAI '09, pages 452–461, Arlington, Virginia, United States, 2009. AUAI Press.

[56] Rong Pan, Yunhong Zhou, Bin Cao, Nathan N. Liu, Rajan Lukose, Martin Scholz, and Qiang Yang. One-class collaborative filtering. In *Proceedings of the 2008 Eighth IEEE International Conference on Data Mining*, ICDM '08, pages 502–511, Washington, DC, USA, 2008. IEEE Computer Society.

[57] Yifan Hu, Yehuda Koren, and Chris Volinsky. Collaborative filtering for implicit feedback datasets. In *Proceedings of the 2008 Eighth IEEE International Conference on Data Mining*, ICDM '08, pages 263–272, Washington, DC, USA, 2008. IEEE Computer Society.

[58] Cataldo Musto, Tiziano Franza, Giovanni Semeraro, Marco de Gemmis, and Pasquale Lops. Deep content-based recommender systems exploiting recurrent neural networks and linked open data. In *Adjunct Publication of the 26th Conference on User Modeling, Adaptation and Personalization*, UMAP '18, pages 239–244, New York, NY, USA, 2018. ACM.

[59] Tommaso Di Noia, Thomas Lukasiewicz, Maria Vanina Martínez, Gerardo I. Simari, and Oana Tifrea-Marciuska. Combining existential rules with the power of cp-theories. In *Proceedings of the Twenty-Fourth International Joint Conference on Artificial Intelligence, IJCAI 2015, Buenos Aires, Argentina, July 25-31, 2015*, pages 2918–2925, 2015.

[60] Vito Bellini, Vito Walter Anelli, Tommaso Di Noia, and Eugenio Di Sciascio. Auto-encoding user ratings via knowledge graphs in recommendation scenarios. In *Proceedings of the 2nd Workshop on Deep Learning for Recommender Systems*, pages 60–66. ACM, 2017.

[61] V. Bellini, A. Schiavone, T. Di Noia, A. Ragone, and E. Di Sciascio. Computing recommendations via a Knowledge Graph-aware Autoencoder. *ArXiv e-prints*, July 2018.

[62] Jonathan L. Herlocker, Joseph A. Konstan, and John Riedl. Explaining collaborative filtering recommendations. In *Proceedings of the 2000 ACM Conference on Computer Supported Cooperative Work*, CSCW '00, pages 241–250, New York, NY, USA, 2000. ACM.

[63] Rashmi Sinha and Kirsten Swearingen. The role of transparency in recommender systems. In *CHI '02 Extended Abstracts on Human Factors in Computing Systems*, CHI EA '02, pages 830–831, New York, NY, USA, 2002. ACM.

[64] Nava Tintarev and Judith Masthoff. *Designing and Evaluating Explanations for Recommender Systems*, pages 479–510. Springer US, Boston, MA, 2011.

[65] Jesse Vig, Shilad Sen, and John Riedl. Tagsplanations: Explaining recommendations using tags. In *Proceedings of the 14th International Conference on Intelligent User Interfaces*, IUI '09, pages 47–56, New York, NY, USA, 2009. ACM.

[66] Nava Tintarev and Judith Masthoff. A survey of explanations in recommender systems. In *Proceedings of the 2007 IEEE 23rd International Conference on Data Engineering Workshop*, ICDEW '07, pages 801–810, Washington, DC, USA, 2007. IEEE Computer Society.

[67] Nava Tintarev and Judith Masthoff. Evaluating the effectiveness of explanations for recommender systems. *User Modeling and User-Adapted Interaction*, 22(4-5):399–439, October 2012.

[68] Panagiotis Symeonidis, Alexandros Nanopoulos, and Yannis Manolopoulos. Moviexplain: A recommender system with explanations. In *Proceedings of the Third ACM Conference on Recommender Systems*, RecSys '09, pages 317–320, New York, NY, USA, 2009. ACM.

[69] Behnoush Abdollahi and Olfa Nasraoui. Explainable restricted boltzmann machines for collaborative filtering. *CoRR*, abs/1606.07129, 2016.

[70] Yongfeng Zhang, Guokun Lai, Min Zhang, Yi Zhang, Yiqun Liu, and Shaoping Ma. Explicit factor models for explainable recommendation based on phrase-level sentiment analysis. In *Proceedings of the 37th International ACM SIGIR Conference on Research & Development*

in Information Retrieval, SIGIR '14, pages 83–92, New York, NY, USA, 2014. ACM.

[71] Cataldo Musto, Fedelucio Narducci, Pasquale Lops, Marco De Gemmis, and Giovanni Semeraro. Explod: A framework for explaining recommendations based on the linked open data cloud. In *Proceedings of the 10th ACM Conference on Recommender Systems*, RecSys '16, pages 151–154, New York, NY, USA, 2016. ACM.

[72] Hao Wang, Xingjian SHI, and Dit-Yan Yeung. Collaborative recurrent autoencoder: Recommend while learning to fill in the blanks. In D. D. Lee, M. Sugiyama, U. V. Luxburg, I. Guyon, and R. Garnett, editors, *Advances in Neural Information Processing Systems 29*, pages 415–423. Curran Associates, Inc., 2016.

[73] Xin Dong, Lei Yu, Zhonghuo Wu, Yuxia Sun, Lingfeng Yuan, and Fangxi Zhang. A hybrid collaborative filtering model with deep structure for recommender systems. In *AAAI*, pages 1309–1315, 2017.

[74] Vivi Nastase and Michael Strube. Decoding wikipedia categories for knowledge acquisition. In *AAAI*, volume 8, pages 1219–1224, 2008.

[75] Azzurra Ragone, Paolo Tomeo, Corrado Magarelli, Tommaso Di Noia, Matteo Palmonari, Andrea Maurino, and Eugenio Di Sciascio. Schema-summarization in linked-data-based feature selection for recommender systems. In *Proceedings of the Symposium on Applied Computing*, SAC '17, pages 330–335, New York, NY, USA, 2017. ACM.

[76] Bart P. Knijnenburg and Martijn C. Willemsen. *Evaluating Recommender Systems with User Experiments*, pages 309–352. Springer US, Boston, MA, 2015.

[77] Francesco Ricci, Lior Rokach, and Bracha Shapira. Introduction to recommender systems handbook. In Francesco Ricci, Lior Rokach, Bracha Shapira, and Paul B. Kantor, editors, *Recommender Systems Handbook*, pages 1–35. Springer, 2011.

[78] Xinxi Wang and Ye Wang. Improving content-based and hybrid music recommendation using deep learning. In *Proceedings of the 22Nd ACM International Conference on Multimedia*, MM '14, pages 627–636, New York, NY, USA, 2014. ACM.

[79] Pascal Vincent, Hugo Larochelle, Yoshua Bengio, and Pierre-Antoine Manzagol. Extracting and composing robust features with denoising

autoencoders. In *Proceedings of the 25th international conference on Machine learning*, pages 1096–1103. ACM, 2008.

[80] Denny Vrandečić and Markus Krötzsch. Wikidata: A free collaborative knowledgebase. *Commun. ACM*, 57(10):78–85, September 2014.

[81] Daniel Ringler and Heiko Paulheim. One knowledge graph to rule them all? analyzing the differences between dbpedia, yago, wikidata & co. In *Joint German/Austrian Conference on Artificial Intelligence (Künstliche Intelligenz)*, pages 366–372. Springer, 2017.

[82] Michael Färber, Basil Ell, Carsten Menne, and Achim Rettinger. A comparative survey of dbpedia, freebase, opencyc, wikidata, and yago. *Semantic Web Journal*, 1:1–5, 2015.

[83] Michael Färber, Frederic Bartscherer, Carsten Menne, and Achim Rettinger. Linked data quality of dbpedia, freebase, opencyc, wikidata, and yago. *Semantic Web*, (Preprint):1–53, 2016.

[84] Heng-Tze Cheng, Levent Koc, Jeremiah Harmsen, Tal Shaked, Tushar Chandra, Hrishi Aradhye, Glen Anderson, Greg Corrado, Wei Chai, Mustafa Ispir, Rohan Anil, Zakaria Haque, Lichan Hong, Vihan Jain, Xiaobing Liu, and Hemal Shah. Wide & deep learning for recommender systems. In *Proceedings of the 1st Workshop on Deep Learning for Recommender Systems*, DLRS 2016, pages 7–10, New York, NY, USA, 2016. ACM.

[85] Ayush Singhal, Pradeep Sinha, and Rakesh Pant. Use of deep learning in modern recommendation system: A summary of recent works. *International Journal of Computer Applications*, 180(7):17–22, Dec 2017.

[86] Xin Dong, Lei Yu, Zhonghuo Wu, Yuxia Sun, Lingfeng Yuan, and Fangxi Zhang. A hybrid collaborative filtering model with deep structure for recommender systems. In *AAAI Conference on Artificial Intelligence*, North America, 2017.

[87] Tommaso Di Noia, Vito Claudio Ostuni, Paolo Tomeo, and Eugenio Di Sciascio. Sprank: Semantic path-based ranking for top-N recommendations using linked open data. *ACM TIST*, 8(1):9:1–9:34, 2016.

[88] Vito Claudio Ostuni, Tommaso Di Noia, Roberto Mirizzi, and Eugenio Di Sciascio. A linked data recommender system using a neighborhood-based graph kernel. In Martin Hepp and Yigal Hoffner, editors, *E-Commerce and Web Technologies*, pages 89–100, Cham, 2014. Springer International Publishing.

[89] Guangyuan Piao and John G. Breslin. Measuring semantic distance for linked open data-enabled recommender systems. In *Proceedings of the 31st Annual ACM Symposium on Applied Computing*, SAC '16, pages 315–320, New York, NY, USA, 2016. ACM.

[90] Vito Bellini, Angelo Schiavone, Tommaso Di Noia, Azzurra Ragone, and Eugenio Di Sciascio. Computing recommendations via a knowledge graph-aware autoencoder. In *Proceedings of the RecSys 2018 Workshop on Knowledge-aware and Conversational Recommender Systems (KaRS) co-located with 12th ACM Conference on Recommender Systems (RecSys 2018), Vancouver, Canada, October 7, 2018.*, 2018.

[91] Vito Bellini, Angelo Schiavone, Tommaso Di Noia, Azzurra Ragone, and Eugenio Di Sciascio. Knowledge-aware autoencoders for explainable recommender systems. In *Proceedings of the 3rd Workshop on Deep Learning for Recommender Systems*, DLRS 2018, pages 24–31, New York, NY, USA, 2018. ACM.

[92] Vito Claudio Ostuni, Tommaso Di Noia, Eugenio Di Sciascio, and Roberto Mirizzi. Top-n recommendations from implicit feedback leveraging linked open data. In *Proceedings of the 7th ACM Conference on Recommender Systems*, RecSys '13, pages 85–92, New York, NY, USA, 2013. ACM.

[93] Vito Walter Anelli, Tommaso Di Noia, Pasquale Lops, and Eugenio Di Sciascio. Feature factorization for top-n recommendation: From item rating to features relevance. In *Proceedings of the 1st Workshop on Intelligent Recommender Systems by Knowledge Transfer & Learning co-located with ACM Conference on Recommender Systems (RecSys 2017), Como, Italy, August 27, 2017.*, pages 16–21, 2017.

[94] Paolo Cremonesi, Yehuda Koren, and Roberto Turrin. Performance of recommender algorithms on top-n recommendation tasks. In *Proceedings of the Fourth ACM Conference on Recommender Systems*, RecSys '10, pages 39–46, New York, NY, USA, 2010. ACM.

[95] Barry Smyth and Paul McClave. Similarity vs. diversity. In *Proceedings of the 4th International Conference on Case-Based Reasoning: Case-Based Reasoning Research and Development*, ICCBR '01, pages 347–361, London, UK, UK, 2001. Springer-Verlag.

[96] Xia Ning and George Karypis. Slim: Sparse linear methods for top-n recommender systems. In *Proceedings of the 2011 IEEE 11th International*

Conference on Data Mining, ICDM '11, pages 497–506, Washington, DC, USA, 2011. IEEE Computer Society.

[97] Zeno Gantner, Steffen Rendle, Christoph Freudenthaler, and Lars Schmidt-Thieme. MyMediaLite: A free recommender system library. In *5th ACM International Conference on Recommender Systems (RecSys 2011)*, 2011.

[98] Phuong T. Nguyen, Paolo Tomeo, Tommaso Di Noia, and Eugenio Di Sciascio. Content-based recommendations via dbpedia and freebase: A case study in the music domain. In *The Semantic Web - ISWC 2015 - 14th International Semantic Web Conference, Bethlehem, PA, USA, October 11-15, 2015, Proceedings, Part I*, pages 605–621, 2015.