Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Rationalizing Recommender Systems

Master's Thesis

Michael Bernhard Bühler

November 29, 2017

Supervisors: Prof. Dr. Andreas Krause
Dr. Sebastian Tschiatschek

Department of Computer Science, ETH Zürich

Abstract

Interpretable Machine Learning models have become increasingly popular in the last few years. Instead of just predicting a value, humans often want to know why a model makes certain predictions. In this thesis we present an interpretable Recommender System model. In order to predict how well a product *B* fits to another product *A*, the model is logically divided into two steps. The first one is to extract sentences from user reviews of product *B*. The idea is that the extracted sentences contain human interpretable information about the connection between *A* and *B*. In the second step the model uses the extracted sentences to predict how well the two products *A* and *B* fit together. Both steps are optimised jointly using a real-world Amazon product data set. We show on an example that the extracted sentences can sometimes indeed explain well why a product *B* fits to a product *A*. We demonstrate that the prediction accuracy can be improved if we include both products *A* and *B* in the sentence extraction process (instead of just product *B*). Furthermore, we explain that it can be beneficial for the prediction accuracy of some models if they extract only a small set of sentences. Compared to the extraction of a large set of sentences, a small set contains less noise.

Contents

Chapter 1

Introduction

1.1 Motivation

Machine Learning has become increasingly popular in the last few years. It can help in predicting the weather, it detects credit card fraud and it arguably is the most important part of self-driving cars. One particular subclass of Machine Learning models is called Recommender Systems. These are models that can suggest items that are of use for a particular user (e.g. Ricci et al., 2011). Recommender Systems are already part of our everyday lives. We interact with them when we are buying a book over the internet and the bookshop suggests three other books to us which are similar to the one we just bought. Other popular examples can be found in social media platforms. Facebook, Spotify & Co. sometimes give us the feeling that they know us better than we know ourself. Thanks to the information they have about us, they can suggest new friends and new music to us (Wang et al., 2015; Jacobson et al., 2016).

Even though Machine Learning tools and techniques made huge improvements in the last 20 years (e.g. Golge, 2014), there are still a lot of unsolved challenges. A major one is the interpretability of Machine Learning algorithms. Even though the predictions they make are often are very good, it sometimes is useful to know why the model gives a particular output. Imaging, for example, that a Machine Learning algorithm assists a doctor by suggesting what treatment option is the best one for a patient. For the doctor it is essential to understand the underlying reason why the algorithm thinks that the given option is the most appropriate one (Lei et al., 2016). Also for the Facebook example above it is interesting to know why Facebook thinks that you know some specific person.

Lei et al., 2016, have built a model with an interpretable output for predicting beer ratings by looking at user reviews. See Figure 1.1 for an overview over their model. In a first step, they extract words and short phrases from

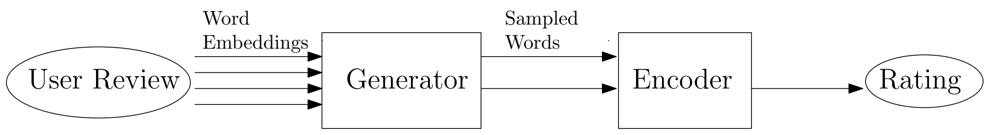

Figure 1.1: The basic structure of the model from Lei et al., 2016. From the word embeddings of a user review, the Generator extracts a few words and phrases. These extracted words are then used in the Encoder to calculate a rating for the beer.

a user review of a given product. Based on the extracted words, the model then predicts a rating for the beer. In this model, the extracted words and phrases serve the purpose of explaining the predicted rating.

The model of Lei et al., 2016, works well, but extracts words and phrases from the user review by looking only at one product. For a Recommender System however, it might be useful to look at other products when trying to extract text. Let us look at the following example review describing a computer mouse:

> This (computer) mouse has a very trendy design and has a lot of additional buttons. Furthermore, the mouse has very low power consumption and was manufactured by the respectable brand Logitech.

Example phrases as it might get extracted by the model of Lei et al. are shown in red and are underlined. Assume now that you just bought a new, very elegant looking keyboard from Logitech on Amazon. The online shop suggests that you should also buy a mouse with the given user review above. In that case, a better set of phrases might be {(computer) mouse, very trendy design, respectable brand Logitech}. This set of text fits more to the specific keyboard you just bought. Hence, to extract text which explains the connection between two products well, information about both products is essential. The model from Lei et al., 2016, only looks at one product.

1.2 Problem Formulation

Data Input We have a set of products $\mathcal{Y}$. For each of these products we have a set of user reviews available. These user reviews describe if and why the users think that some product is good or bad. Furthermore, we assume that we find other information in the user reviews which describes the product in some way (e.g. what it does, how it works, what kind of other products are similar to it, etc.).

Besides of the user reviews, we have a set containing pairs of products from $\mathcal{Y}$. Each such pair (we call it a link) means that the two products in the link fit well together. For example there might be a link between two products if they are frequently bought together or if they are often looked at during the same user session. We call the first product in each of these pairs the context product and the second product the target product.

A more formal description of the inputs to our model can be found in Section 3.1.

Output We want to develop an algorithm which predicts if a target product is a good recommendation for a context product. Furthermore, the algorithm should explain its predictions by extracting a subset of sentences from the user reviews of the target product. The extracted sentences should depend on the context product and explain well why the target product is a good recommendation for the context product or not.

1.3 Contributions of this Thesis

In this thesis, we build and implement a deep learning model for an interpretable Recommender System. As a main part of the model we extract sentences from user reviews. These sentences explain why the model predicts the way it does. We extend on the model from Lei et al. (see Figure 1.1) by making the text extraction step explicitly dependent on the context product. We train our model on real-world data which has been scrawled from the Amazon website (McAuley et al., 2015b; He and McAuley, 2016).

We demonstrate that including the context product into the sentence extraction procedure can improve the accuracy in predicting whether a product is a good recommendation for another product or not. We also explain that sometimes a smaller amount of extracted sentences can make it easier for the model to make good recommendations. This allows to work with a very concise set of sentences without loosing much prediction accuracy. Furthermore, we show on a concrete example that our model is able to extract interpretable sentences to explain why two products fit well together.

1.4 Related Work

In the past, several different approaches have been used for building Recommender Systems. Collaborative Filtering methods (e.g. Linden et al., 2003; Herlocker et al., 2004) try to aggregate information about many users or products (for example user ratings) to predict new products for some users. Content-based recommendations, on the other hand, use metadata stored with each product and user profile to make recommendations (e.g. Lops et al., 2011). Among the content-based recommendation methods, some people have used user reviews to model user opinions. These opinions were then used to recommend new products (e.g. Chen and Wang, 2013; Pero and Horváth, 2013). Other people tried to use user reviews to characterise the products (e.g. Aciar et al., 2007). McAuley et al., 2015a, used user reviews among other data to construct a product graph which shows relatedness between products. This work is interesting because our model uses the same Amazon-scrawled data. A more detailed overview over review-based Recommender Systems can be found in the book from Chen et al., 2015.

In the last few years, research focused more and more on making Recommender Systems not only accurate, but also interpretable. One approach is to co-cluster users and products. Users and products in the same co-cluster can then be used for interpretable product recommendations (Heckel and Vlachos, 2016; Wu et al., 2016).

Extracting text to make Machine Learning algorithms interpretable has become popular as well. In Arras et al., 2017, the authors classify documents using a layer-wise relevance propagation algorithm. To explain why the algorithm classifies the way it does, it outputs words from the text it wants to classify. Denil et al., 2014, built a convolutional neural network to classify documents. By using visualization techniques from Computer Vision, their model extracts sentences from the text which were important for the classification step. Another work, Pappas and Popescu-Belis, 2017, constructed an attention based model for document classification and showed that the attention layer selects informative words.

As already explained in Section 1.1, Lei et al., 2016, extract word and phrases from user reviews and use only the extracted text to make predictions. In our thesis, we build upon this model and let the extraction step depend on a context product. Furthermore, our model extracts full sentences from several reviews.

Paulus, 2017, built a deep learning model which uses a Determinantal Point Process (DPP) to extract a small set of words. This set is then used to make predictions. In contrast to our work, they do not try to make the model dependent on a context, but focus on how to learn DPPs efficiently using pretraining and variation reduction techniques.

Chapter 2

Background

This chapter gives an introduction to some topics which are important to understand the model in Chapter 3. The first section of this chapter covers a mathematical framework to handle diversity and attraction between objects in a set. We will use this for representing the context in our model and for sampling sentences. The second section gives an introduction to Deep Sets. These are neural network layers which can deal with sets as input.

2.1 Probabilistic Sub- and Supermodular Models

In this section we will present two mathematical frameworks for modelling diversity or attraction between items in a set.

We begin by introducing the notions of submodular and supermodular set functions (Edmonds, 1970). We then use these definitions to define probabilistic submodular and supermodular models (Djolonga and Krause, 2014).

Definition 2.1 *Let $\mathcal{Y}$ be a set of items and $F : 2^{\mathcal{Y}} \to \mathbb{R}$ be a set function, where $2^{\mathcal{Y}}$ denotes the powerset of $\mathcal{Y}$. For all $i \in \mathcal{Y}$ and $A \subseteq B \subseteq \mathcal{Y} \setminus \{i\}$, F is called submodular iff*

$$F(A \cup \{i\}) - F(A) \geq F(B \cup \{i\}) - F(B).$$

Similarly, F is called supermodular iff

$$F(A \cup \{i\}) - F(A) \leq F(B \cup \{i\}) - F(B).$$

Let $Pr(Y)$ be a probability distribution over subsets $Y \subseteq \mathcal{Y}$. $Pr(Y)$ is called log-submodular if it can be written as $Pr(Y) \propto \exp(F(Y))$, where F is submodular. Analogously, $Pr(Y) \propto \exp(F(Y))$ is called log-supermodular if F is supermodular.

Because of their diminishing returns property, log-submodular models are natural candidates for modeling coverage and diversity in sets (Tschiatschek et al., 2016). Analogously, log-supermodular models are good for modelling attractive dependencies between items in a set (Djolonga et al., 2016).

In the following we describe one example of a log-submodular model (L-Ensembles) and one example of a log-supermodular model (FLIC).

2.1.1 L-Ensembles

To sample sentences in our model, we will use a mathematical object called L-Ensemble (Borodin and Rains, 2005). An L-Ensemble is a subclass of the better known Determinantal Point Processes (DPP) (Macchi, 1975; Borodin and Olshanski, 2000) and is described by a positive semi-definite matrix L. DPPs (and hence L-Ensembles) are well suited for sampling sets with diverse items. In the following, we follow the introduction into L-Ensembles from Kulesza and Taskar, 2012.

Without loss of generality we assume that we have a ground set $\mathcal{Y}$ of the form $\mathcal{Y} := \{1, 2, \ldots, N\}$, where N is the number of items in the ground set.

Definition 2.2 *Let L be a real-valued, symmetric matrix. A probability distribution $\mathcal{P}_L$ is called an L-Ensemble if for all $Y \subseteq \mathcal{Y}$*

$$\mathcal{P}_L(Y) \propto \det(L_Y),$$

where $L_Y = [L_{ij}]_{i,j \in Y}$ denotes the restriction of L to the entries indexed by elements of Y. Furthermore we define $\det(L_\varnothing) = 1$, where $\varnothing$ denotes the empty set.

From this definition follows that L must be positive semi-definite because otherwise the probability for some set Y would be negative. Intuitively, the ensemble matrix L encodes which elements are likely to be sampled together and which elements are not. Assume for example that $Y = \{1, 2\}$. Then $\mathcal{P}_L(Y) \propto \det(L_{\{1,2\}}) = L_{11} \cdot L_{22} - L_{12}^2$, where L_{ij} denotes the element in the i-th row and j-th column of L. The bigger an on-diagonal element L_{ii} is, the higher is the probability that the i-th element is in the sampled set. On the other hand, the off-diagonal elements L_{12} can be interpreted as repulsion weights. The bigger they are, the more unlikely is it to have both items together in the sampled set.

We list the most important results concerning L-Ensembles, which we need for this thesis, in Appendix A. In the following, we restate the most important insights in natural language. L-Ensembles are log-submodular models, which means they are good candidates to model diversity between items. Furthermore, L-Ensembles allow an efficient (i.e. polynomial in N) calculation of the partition function, which allows to efficiently calculate $\mathcal{P}_L(Y)$ for all $Y \subseteq \mathcal{Y}$ in closed form. Furthermore, it is possible to efficiently do operations like sampling from an L-Ensemble and to calculate the expected set size $\mathbb{E}[|\mathbf{Y}|]$.

2.1.2 Facility Location Complements Model (FLIC)

For this section assume you have a ground set $\mathcal{Y}$ and a set $\overline{Y}$ of subsets from $\mathcal{Y}$ (i.e. $\overline{Y} \subseteq 2^{\mathcal{Y}}$, where $2^{\mathcal{Y}}$ is the powerset of $\mathcal{Y}$). Furthermore, assume that all items in the sets $Y \in \overline{Y}$ share some similarities. For the purpose of a real-world example, assume that $\mathcal{Y}$ is the set of all books you can buy on Amazon and that $\overline{Y}$ are the set of books which are often bought together. You are interested in a probability distribution $\mathcal{P}_{FLIC}$ on subset of $\mathcal{Y}$. $\mathcal{P}_{FLIC}$ should have the property that sets $Y \in \overline{Y}$ should receive high probabilities because of the assumption that they are similar. If, on the other hand, you have sets $Y \subseteq \mathcal{Y}$ consisting of dissimilar items, then $\mathcal{P}_{FLIC}$ should assign low probability to these sets.

One way to model the problem above mathematically is by using the Facility Location Complements Model (Djolonga et al., 2016):

Definition 2.3 *Without loss of generality, let $\mathcal{Y} := \{1, 2, \ldots, N\}$ be the set of natural numbers $\leq N$. Let K be a positive integer. Furthermore, let $u \in \mathbb{R}^N$ be a real-valued vector and $A \in \mathbb{R}^{K \times N}$ be a matrix with non-negative real values. For all sets $Y \subseteq \mathcal{Y}$, the Facility Location Complements Model (FLIC) is the probability distribution given by*

$$\mathcal{P}_{FLIC}(Y; u, A) \propto \exp\left(\sum_{i \in Y} u_i - \sum_{k=1}^{K}\left(\max_{i \in Y} A_{ki} - \sum_{i \in Y} A_{ki}\right)\right).$$

The model is parametrized by the vector u and the matrix A. Intuitively, the i-th component of the vector u (i.e. u_i) measures the importance of the i-th item. If you increase u_i while fixing the matrix A, the sets containing the i-th item will get higher probabilities. The matrix A captures attractive dependencies among items in $\mathcal{Y}$. The i-th column of A can be seen as a topic vector of the i-th item. If two items are often together in sets from Y (Y was defined at the beginning of this section), then they usually have similar topic vectors after the model is trained. If, on the other hand, two items rarely are in the same set in Y, then they usually have very different topic vectors. In our model, we will make use of this property to define vector representations for our context products (see Chapter 3).

From what we just said follows that K is the number of dimensions in the topic vectors.

2.2 Deep Sets

Deep Sets (Zaheer et al., 2017) are deep neural network architectures which can handle a set of real-valued vectors as input. This is different from RNN's and similar architectures which handle sequences instead of sets as input (e.g. Pollack, 1987).

In the following we will shortly sketch the architecture of Deep Sets. First we define what property an algorithm working on sets should possess. Then we show how a deep learning layer must look like in order to fulfil this property (Subsection 2.2.1). Afterwards we present the full Deep Set architecture (Subsection 2.2.2).

2.2.1 Permutation Equivariant Layers

Let $\mathcal{X}$ be a space containing sequences of real-valued vectors. We want to define a property for functions working on $\mathcal{X}$ to make it look like the input is a set. This property is captured in the notion of a permutation invariant function (David et al., 1966). For what follows, we also need the definition of a permutation equivariant function (e.g. Ravanbakhsh et al., 2016).

Definition 2.4 *A function $f : \mathcal{X} \to \mathbb{R}^n$ is permutation invariant if*

$$f\left((x_1, \dots, x_m)\right) = f((x_{\sigma(1)}, \dots, x_{\sigma(m)}))$$

for any permutation σ.
Furthermore, a function $f : \mathcal{X} \to \mathbb{R}^m$ is permutation equivariant if

$$f((x_1, \dots, x_m))_{\sigma(i)} = f((x_{\sigma(1)}, \dots, x_{\sigma(m)}))_i$$

for all $i \in \{1, \dots, m\}$ and any permutation σ. Here $f(S)_i$ denotes the i-th entry of the function f evaluated on the sequence S. m is the length of the sequence.

A permutation invariant function allows an arbitrary permutation of the input sequence and still returns the same result. This behaviour is exactly what we expect from a function where the input represents a set. Intuitively, a permutation equivariant function f doesn't care if we first apply a permutation to the sequence of input vectors and then apply f or if we first apply f to the sequence and then apply the permutation. The result is the same.

We will now show how to construct a permutation equivariant deep learning layer (Ravanbakhsh et al., 2016). Section 2.2.2 then shows how to use permutation equivariant layers to construct permutation invariant functions.

Definition 2.5 *Let $X \in \mathbb{R}^{N \times D}$ be a matrix representing a set of N D-dimensional real vectors stacked in the rows. Let Λ, $\Gamma \in \mathbb{R}^{D \times D'}$ and let σ be a non-linear*

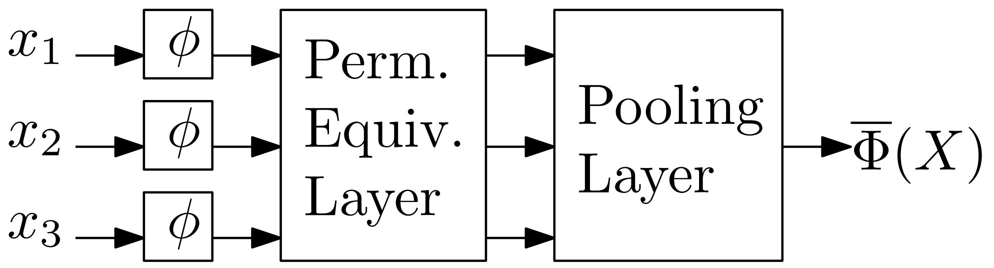

Figure 2.1: The basic structure of the Deep Set architecture with 3 items in the set. $\overline{\Phi}(X)$ is invariant with respect to the order of the input vectors x_1, x_2 and x_3 (permutation invariance).

activation function applied pointwise at its input. 11^T *is a matrix of dimensions* $N \times N$ *filled with ones. The function*

$$y = \sigma(X\Lambda + 11^T X\Gamma)$$

is called a permutation equivariant layer.

The output of this function is a real-valued matrix with dimensions $N \times D'$. Each element of the input (i.e. each row in the matrix X) has a corresponding row in y. We can hence see the permutation equivariant layer as a transformation of the input vectors.

Intuitively, the first term in the argument of σ (i.e. $X\Lambda$) looks at the items independently (i.e. no information flows from one element to the other elements in the set). The second term (i.e. $11^T X\Gamma$) then aggregates information from all items in the set, but in a way that every row adds exactly the same quantity. Because all the items add the same quantity in the second term, the function preserves its permutation equivariance property. For a formal proof of this see Ravanbakhsh et al., 2016.

2.2.2 Architecture

In the last subsection we have seen how to construct permutation equivariant layers for neural network architectures. In this section we show how Deep Sets uses them to build permutation invariant functions.

Assume we have a set $X := \{x_1, x_2, \ldots, x_N\}$ with $x_i \in \mathbb{R}^D$ for all $i \in \{1, 2, \ldots, N\}$. In the following, D, D', and D'' are positive integers denoting the dimensions in vectors and matrices.

See Figure 2.1. On the leftmost side we enter the real-valued vectors x_i for $i \in \{1, 2, \ldots, N\}$ into the neural network. We apply the same transformation $\phi : \mathbb{R}^D \to \mathbb{R}^{D'}$ to each of these vectors. ϕ can for example be a standard feed-forward neural network layer with weight sharing, but can also be something else. Afterwards we apply zero or more permutation equivariant layers to the matrix $X' \in \mathbb{R}^{N \times D'}$, which is constructed by taking the vectors $\phi(x_i)$ as its rows. The output of the permutation equivariant layers is then a matrix with dimensions $N \times D''$. The pooling layer then max or sum pools this matrix along the first axis to construct a vector $\overline{\Phi}(X)$ of size D''.

The vector $\overline{\Phi}(X)$ can be seen as an embedding vector of the input set. According to the construction, the whole process gives a permutation invariant function (see also Zaheer et al., 2017).

The Deep Set architecture is flexible and can be adapted to specific use cases. The user can for example decide how to implement the function ϕ and how the permutation equivariant layers should look like in detail.

Chapter 3

Rationalizing Recommender Systems

In this chapter we propose a model to solve the problem formulated in Section 1.2. First, we explain what kind of data we use and introduce the notation used in the remainder of this thesis. Section 3.2 then shows our model from a bird's eye view and explains the overall model functionality. In the remaining sections of this chapter we explain the components of the model in more detail.

3.1 Data Sets and Definitions

This section explains the data sets our model expects as input.

Let $\mathcal{Y}$ be a set of products. This can for example be the set of all books which can be bought on Amazon. In this chapter we will present a model which predicts how good two products from $\mathcal{Y}$ fit together.

We have a data set $\mathcal{D} := \{(c_1, t_1), (c_2, t_2), \ldots, (c_{|\mathcal{D}|}, t_{|\mathcal{D}|})\}$ with $c_i, t_i \in \mathcal{Y}$ for all $i \in \{1, 2, \ldots, |\mathcal{D}|\}$. We will call the first product in each pair $(c, t) \in \mathcal{D}$ the context product and the second product the target product. A pair $(c, t) \in \mathcal{D}$ is called a link. For all links $(c, t) \in \mathcal{D}$ we assume that the target product t is a good recommendation given the context product c. So if a user is interested in product c, then the probability that he is also interested in t should be high.

Additionally, we have a data set $\mathcal{D}_C := \{(c_1, t_1), (c_2, t_2), \ldots, (c_{|\mathcal{D}_C|}, t_{|\mathcal{D}_C|})\}$ with $c_i, t_i \in \mathcal{Y}$ for all $i \in \{1, 2, \ldots, |\mathcal{D}_C|\}$. This data set contains pairs of products which are usually not bought together. We assume for all $(c, t) \in \mathcal{D}_C$ that the product t is not a good recommendation for the product c. The terms context product, target product and link are defined analogously as for the data set $\mathcal{D}$.

Furthermore, we define the set of all links $\mathcal{L} := \mathcal{D} \cup \mathcal{D}_C$.

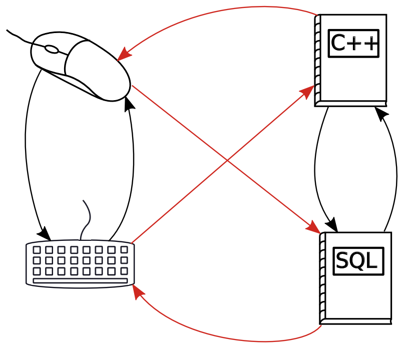

Figure 3.1: This example consists of a set $\mathcal{Y} = \{Mouse, Keyboard, Book1, Book2\}$. The data set $\mathcal{D} := \{(Keyboard, Mouse), (Mouse, Keyboard), (Book1, Book2), (Book2, Book1)\}$ is a set of links, where the target products are good recommendations for the corresponding context products. The data set $\mathcal{D}_C := \{(Keyboard, Book2), (Mouse, Book1), (Book1, Keyboard), (Book2, Mouse)\}$ consists of pairs of products which are usually not bought together.

In Figure 3.1 we illustrate an example product graph for four products.

We map each product to a topic vector, such that similar products have similar vectors and dissimilar products have very different vectors. With similar products we mean products which are often bought together. Furthermore, two vectors are similar if they have a high cosine similarity. We use the mapping from products to vectors for the representation of the context product. We represent the mapping with the function $\mathcal{C} : \mathcal{Y} \rightarrow \mathbb{R}^K$, where K is the dimension of the topic vectors. We will see in Section 3.4 how $\mathcal{C}$ can be constructed.

To each product $y \in \mathcal{Y}$ we associate a set of sentences. These sentences can for example be extracted from user reviews in a preprocessing step. We model this by the function $\mathcal{R} : \mathcal{Y} \rightarrow 2^{\mathcal{A}}$, where $\mathcal{A}$ denotes the set of all sentences in the English language, and $2^{\mathcal{A}}$ denotes the powerset of $\mathcal{A}$. Hence, for all products $y \in \mathcal{Y}$, we can access a set of sentences by calling $\mathcal{R}(y)$. To make our model work, we assume that $\mathcal{R}(y)$ describes y well. More precisely: if we have a link $(c, t) \in \mathcal{L}$, we assume that it is possible to predict from $\mathcal{R}(t)$ and $\mathcal{C}(c)$ if the product t is a good recommendation for the product c or not.

3.2 Overview of the Model

This chapter gives a high-level overview of our proposed model. In later sections we will then focus more on specific parts of the model.

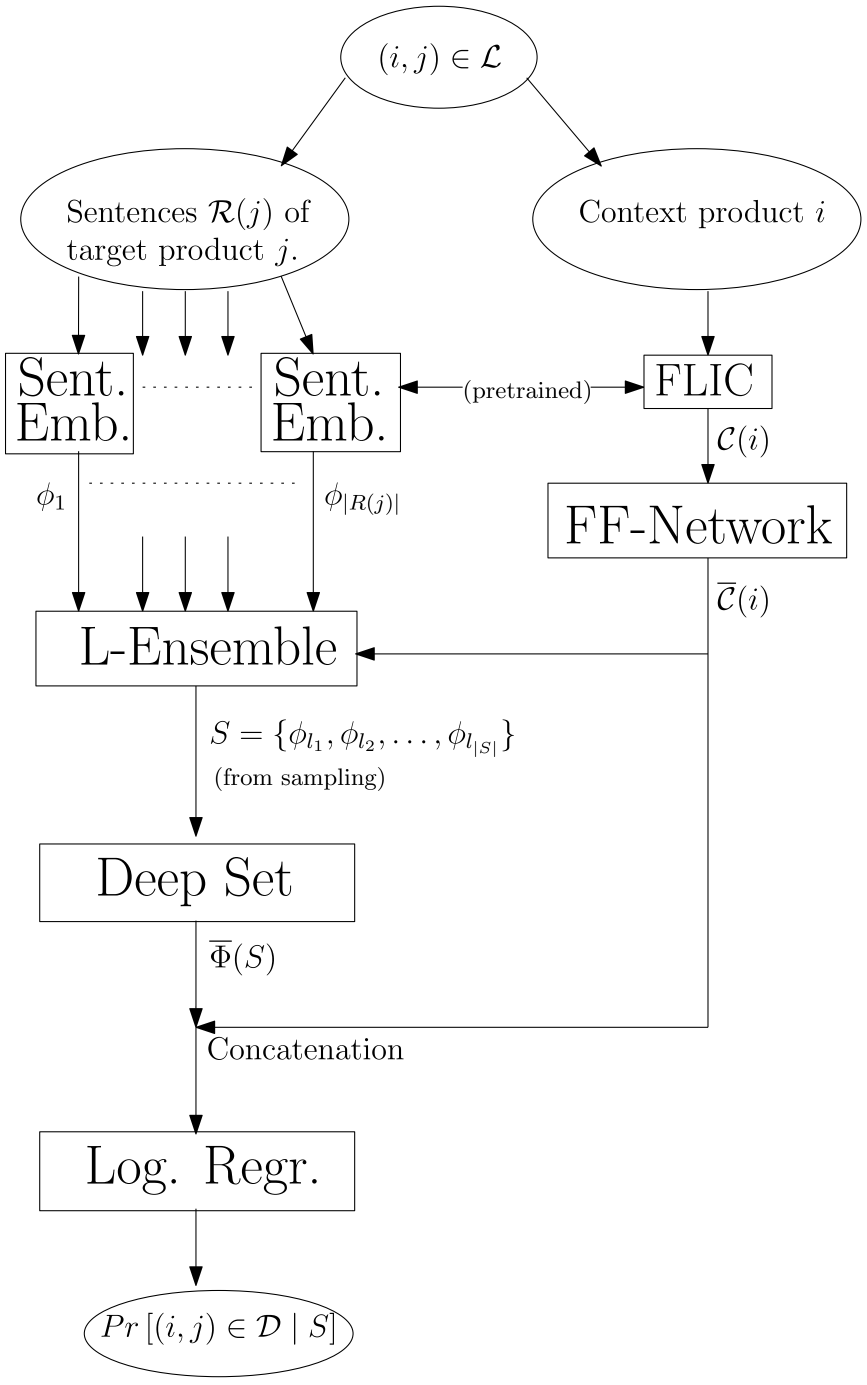

Figure 3.2: The Rationalizing Recommender Systems model.

We depict the model architecture in Figure 3.2. The model takes a link $(i, j) \in \mathcal{L}$ as its input. In a first step, it processes the context product i and the target product j separately. Each sentence in $\mathcal{R}(j)$ of the target product j gets embedded into a real-valued vector (see Section 3.3). The set $\Phi := \{\phi_1, \phi_2, \dots, \phi_{|\mathcal{R}(j)|}\}$ resulting from the sentence embeddings is now our representation of j.

In Section 3.4 we describe how to process the context product using the Facility Location Diversity Model (FLIC) to produce a topic vector $\mathcal{C}(i)$. A further transformation via a standard feed-forward neural network layer gives us a transformed context vector $\overline{\mathcal{C}}(i)$. This vector represents the context product for the remaining part of the model.

Both the model for the sentence embeddings of the target product and the FLIC model for the context product are pretrained.

In a further step our model samples a subset $S \subseteq \Phi$ from the sentence embeddings. It does this using an L-Ensemble, which we introduced in Section 2.1.1. Important is that we make the L-Ensemble matrix dependent on both the set Φ and the context topic vector $\overline{\mathcal{C}}(i)$. The details can be found in Section 3.5.

The Deep Set part of our model then transforms the set S of sampled sentence embeddings into a new vector representation. The idea of this step is to reduce the set S into a single vector $\overline{\Phi}(S)$ (see Section 3.6). Afterwards we concatenate the vectors $\overline{\Phi}(S)$ and $\overline{\mathcal{C}}(i)$ and apply Logistic Regression to it (Section 3.7). The output can be interpreted as the probability that the link (i, j) is in the data set $\mathcal{D}$. Ideally, the model should output 1 if $(i, j) \in \mathcal{D}$ and 0 if $(i, j) \in \mathcal{D}_C$.

For training the model we optimise the model parameters with respect to the expected cross entropy error (Section 3.8). In Section 3.9 we show how to approximate the gradient for the complex cost function.

3.3 Sentence Embeddings

In this section we apply a sentence embedding model to all sentences in $\mathcal{R}(j)$, where j is a target product. Unfortunately, there are fundamental problems when trying to learn these sentence embeddings from our review data set:

- We will see in Section 3.5 that it is important for our model to have a diverse set of sentence embeddings in $\mathcal{R}(j)$. We want that semantically different sentences in $\mathcal{R}(j)$ receive different embedding vectors. Unfortunately, we don't have enough data to enforce this property. The only information about sentence relatedness is to look if they are in the same review or not. Hence, most sentence embedding models tend to give sentences in the same review similar embedding vectors. This is exactly what we do not want.

- A good sentence embedding model is difficult to learn. In our case, training our model together with a sentence embedding model is even more difficult because our complex model has several layers on top of the sentence embedding model. Hence the training would become very slow.

Because of these reasons, we decided to use the pretrained Infersent model from Conneau et al., 2017, to embed our sentences. We chose their sentence embedding model because it seems to perform very good in transfer learning. The authors use the Stanford Natural Language Inference (SNLI) Corpus to train the model in a supervised manner on a Recognizing Textual Entailment task. The model takes a sequence of GloVe word embeddings as input (for GloVe see Pennington et al., 2014). As a first step, the model uses a Bidirectional Long Short Term Memory model to construct latent vectors for all word embeddings. To construct the sentence embedding vector, the model then uses max pooling over these latent vectors.

3.4 Context Product

In this section we have a look at how to construct the vectors $\overline{\mathcal{C}}(i)$ for $i \in \mathcal{Y}$. $\overline{\mathcal{C}}(i)$ is then our representation for the context product in the model.

The first step uses the FLIC model, which we introduced in Section 2.1.2, to calculate $\mathcal{C}(i)$ for a context product $i \in \mathcal{Y}$. In the next section it will become clear that we want similar products to have similar topic vectors and different products to have different topic vectors. As we have mentioned in the background chapter, FLIC topic vectors naturally have this property (see also Djolonga et al., 2016).

To train the FLIC model, we can use a different data set than for the rest of the model. Let us call this data set Y. It should consist of subsets of $\mathcal{Y}$ and have the property that for sets $y \in Y$ products in y are often bought together. We will see in the experiment section how we can construct such a data set.

After training FLIC on the set Y, we discard the vector u and keep only the matrix A (see Section 2.1.2 for the definition of u and A). We now define $\mathcal{C}(i)$ to be the topic vector of product i in the trained FLIC model. This topic vector is given by some column of the matrix A.

Unfortunately, we can not use $\mathcal{C}(i)$ directly. In the L-Ensemble we will calculate dot products between the context vector and the sentence embedding vectors to get a measure of relatedness between the context and the sentences. To make this work, we need an additional transformation $\overline{\mathcal{C}(i)}$ which brings the context topic vector in the correct space to compare it to the sentence embedding vectors. We implement $\overline{\mathcal{C}(i)}$ as a standard feed-forward neural network layer:

$$\overline{\mathcal{C}(i)} := \tanh(W \cdot \mathcal{C}(i)),$$

where $W \in \mathbb{R}^{D \times K}$ is a matrix of weights. K is the dimension of the FLIC topic vectors and D is the dimension of the sentence embedding vectors (see the earlier sections of this chapter). tanh is the hyperbolic tangent function.

3.5 L-Ensemble & Sampling

The core component of our model is the L-Ensemble. It is responsible for sampling a subset S from the set of embedded review sentences $\Phi = \{\phi_1, \phi_2, \ldots, \phi_{|\mathcal{R}(j)|}\}$. We will construct an ensemble matrix L which depends not only on Φ, but also on the context vector $\overline{\mathcal{C}(i)}$. In the following subsection we explain how to construct the L-matrix (see Definition 2.2). Using this matrix we can then efficiently sample sets $S \subseteq \Phi$ (see Theorem A.3). Furthermore, we can efficiently calculate log-probabilities (see Equation A.1) and expectations (Corollary A.4).

One can see directly from Equation A.1 that there is a non-zero probability to sample the empty set from an L-Ensemble. This case is problematic for our model because we use the sampled set in further layers. It is not clear how to handle the empty set. We solve this problem using a rejection sampling approach. See Subsection 3.5.2 for details.

3.5.1 The Ensemble Matrix L

In this subsection we show how to construct the L-matrix which we need to sample from the L-Ensemble.

Remember that we represent the vector of the context product with $\overline{\mathcal{C}}(i)$ and the set of embedded sentences from the target product with $\Phi = \{\phi_1, \phi_2, \ldots, \phi_{|\mathcal{R}(j)|}\}$.

We define the element in the m-th row and n-th column of the Ensemble matrix L as

$$L_{mn} := scale \cdot \phi_m^T \phi_n \cdot \tanh(\phi_m^T \overline{\mathcal{C}}(i)) \cdot \tanh(\phi_n^T \overline{\mathcal{C}}(i)), \tag{3.1}$$

where $scale \in \mathbb{R}_+ \setminus \{0\}$ is a positive scalar value. Note that L is positive semidefinite, which is an important condition for L-Ensembles.

Let us try to see why this way of constructing L is reasonable. An entry L_{mn} is big if ϕ_m and ϕ_n are similar to each other and when both ϕ_m and ϕ_n are similar to the context vector $\overline{C}(i)$. Similarity is measured by the dot product. For the probability of a single item m in the set we have

$$\mathcal{P}_L(\{m\}) \propto \det(L_{\{m\}}) = L_{mm} = scale \cdot \|\phi_m\|_2^2 \cdot \tanh^2(\phi_m^T \overline{C}(i)).$$

Hence, the probability to sample a single item set $\{m\}$ is high if ϕ_m is approximately parallel to the context vector $\overline{C}(i)$. On the other hand, if ϕ_m is orthogonal to $\overline{C}(i)$, the probability to sample $\{m\}$ is close to 0.

For the probability of sampling a set of 2 items $\{m, n\}$ we get

$$\begin{aligned}\mathcal{P}_L(\{m,n\}) &\propto \det(L_{\{m,n\}}) = L_{mm} \cdot L_{nn} - L_{mn}^2 \\ &= scale^2 \Bigg(\|\phi_m\|_2^2 \cdot \|\phi_n\|_2^2 \cdot \tanh^2(\phi_m^T \overline{C}(i)) \cdot \tanh^2(\phi_n^T \overline{C}(i)) \\ &\quad - \left(\phi_m^T \phi_n \cdot \tanh(\phi_m^T \overline{C}(i)) \cdot \tanh(\phi_n^T \overline{C}(i)) \right)^2 \Bigg).\end{aligned}$$

So, the probability for the set $\{m, n\}$ is high if both ϕ_m and ϕ_n are similar to the context, but ϕ_m and ϕ_n are not similar to each other.

The positive factor *scale* is a way to control the amount of elements in the sets which are sampled from the L-Ensemble. If *scale* is close to 0, then we almost always sample the empty set (because we defined $\det(L_\varnothing) = 1$ (see Definition 2.2)). If *scale* gets much more bigger than 1, then the L-Ensemble will prefer sets S with a lot of elements because $\det(scale \cdot L) = scale^k \cdot \det(L)$, where $k \times k$ are the dimensions of the matrix L.

You can find more information about the intuition of L-Ensembles in Chapter 2.2.1 (Geometry) from Kulesza and Taskar, 2012.

3.5.2 Avoiding to Sample Empty Sets

Using the matrix L, which we constructed in the last subsection, we can now sample and calculate log-probabilities of sets efficiently. Unfortunately, we have $\mathcal{P}_L(\varnothing) > 0$. Hence, it is possible to sample the empty set from an L-Ensemble. An empty set $\varnothing$ is problematic because our model uses $\varnothing$ in further layers and no meaningful predictions can be made from an empty set of sentence embeddings.

We solve the problem by introducing a new sampling scheme as follows: As long as we sample the empty set from $\mathcal{P}_L$, we sample again. At some point we will sample a non-empty set and we will keep it. This is a simple example of the more general rejection sampling approach (Neumann, 1951).

Remember that Φ is our set of sentence embeddings. Furthermore, let $\overline{\mathcal{P}}_L$ be the distribution belonging to the new sampling scheme as defined in the last paragraph. We are interested in the probabilities $\overline{\mathcal{P}}_L(S)$ for all $S \subseteq \Phi \setminus \varnothing$. Note that the probability for the set S can be expressed by the following recursive formula.

$$\overline{\mathcal{P}}_L(S) = \mathcal{P}_L(S) + \mathcal{P}_L(\varnothing) \cdot \overline{\mathcal{P}}_L(S)$$

The first term $\mathcal{P}_L(S)$ is the probability that we sample S in our first try. The second term $\mathcal{P}_L(\varnothing) \cdot \overline{\mathcal{P}}_L(S)$ is the probability that we sample the empty set and need to sample again. Plugging in the formulas for $\mathcal{P}_L(S)$ and $\mathcal{P}_L(\varnothing)$ (Equation A.1 and Definition 2.2) and solving for $\overline{\mathcal{P}}_L(S)$ gives

$$\overline{\mathcal{P}}_L(S) = \frac{\det(L+I)}{\det(L+I)-1} \cdot \mathcal{P}_L(S)$$

for all sets $S \subseteq \Phi \setminus \varnothing$. I is the identity matrix with the same dimensions as L. For the expected set size of the new distribution (i.e. $\mathbb{E}_{S\sim\overline{\mathcal{P}}_L}[|S|]$), we get

$$\begin{aligned}
\mathbb{E}_{S\sim\overline{\mathcal{P}}_L}[|S|] &= \sum_{S\subseteq\Phi\setminus\varnothing} \overline{\mathcal{P}}_L(S) \cdot |S| \\
&= \frac{\det(L+I)}{\det(L+I)-1} \cdot \sum_{S\subseteq\Phi} \mathcal{P}_L(S) \cdot |S| \\
&= \frac{\det(L+I)}{\det(L+I)-1} \cdot \mathbb{E}_{S\sim\mathcal{P}_L}[|S|]
\end{aligned}$$

Hence, to calculate the probabilities and the expected set size of the new distribution $\overline{\mathcal{P}}_L$, it is enough to multiply the corresponding values from $\mathcal{P}_L$ with a constant factor given by the matrix L.

3.6 Deep Set

In the last step we sampled a set $S \subseteq \Phi$ of sentence embeddings. In this section we present the Deep Set architecture we use to transform this set of sentence embeddings to a single vector.

Let $S = \{\phi_{l_1}, \phi_{l_2}, \dots, \phi_{l_{|S|}}\}$ be the sampled set of sentence embedding vectors from the L-Ensemble. Furthermore, let D be the dimension of the embedding vectors (i.e. $\phi_{l_i} \in \mathbb{R}^D$ for all $i \in \{1, 2, \dots, |S|\}$). In Figure 3.3 you see our Deep Set architecture. We first apply a permutation equivariant layer to the set S. The output is a matrix of dimensions $|S| \times D'$. We then apply the max pooling operator over the first dimension of the matrix to get a vector $\overline{\Phi}(S)$ of dimension D'. $\overline{\Phi}(S)$ is now our representation of the target product.

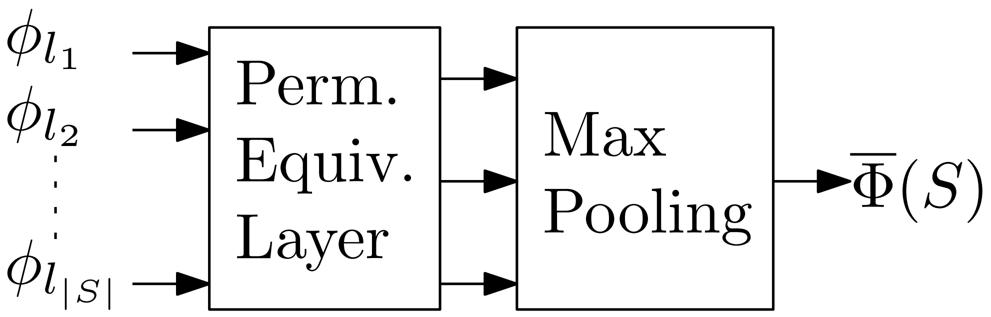

Figure 3.3: The basic structure of our Deep Set architecture. We apply one permutation equivariant layer and then a max pooling layer to the sentence embeddings.

3.7 Logistic Regression

In this section we show how the classification step works. Remember that we have as input a vector $\overline{\Phi}(S)$ which represents the target product j. The context is represented by the vector $\overline{C}(i)$. We apply logistic regression on the concatenated vectors $[\overline{\Phi}(S), \overline{C}(i)]$. As a result we get an estimate for the probability that the link (i, j) is in the data set $\mathcal{D}$:

$$Pr[(i,j) \in \mathcal{D} \mid S] := \sigma(w \cdot [\overline{\Phi}(S), \overline{C}(i)]),$$

where

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

is the sigmoid activation function. If $\overline{\Phi}(S) \in \mathbb{R}^D$ and $\overline{C}(i) \in \mathbb{R}^K$, then w is a weight vector of dimension $D + K$.

Given a set $S \subseteq \Phi$, we have now an expression for the probability that the link (i, j) is in the data set $\mathcal{D}$. Ideally, we want that our model outputs $Pr[(i,j) \in \mathcal{D} \mid S] \approx 1$ if $(i,j) \in \mathcal{D}$ and $Pr[(i,j) \in \mathcal{D} \mid S] \approx 0$ if $(i,j) \in \mathcal{D}_C$.

3.8 Cost Function

In this section we present the cost function which we use to train the model. As we have two classes ($\mathcal{D}$ and $\mathcal{D}_C$) and because we used the logistic regression layer in the last section, a natural choice for the cost function is the average cross entropy error (Rubinstein, 1999). Because our method depends on the sampled sets S, our cost function calculates the expected average cross-entropy error, where the expectation goes over the sets $S \sim \overline{\mathcal{P}}_L$.

For notational simplicity let us define $cost(i, j, S)$ to be the cross entropy cost of a specific set of sentence embeddings S with respect to the link $(i, j) \in \mathcal{L}$:

$$\begin{aligned} cost(i, j, S) := &- \log\left(Pr[(i, j) \in \mathcal{D} \mid S]\right) \cdot \mathbb{I}_{(i,j)\in\mathcal{D}} \\ &- \log\left(1 - Pr\left[(i, j) \in \mathcal{D} \mid S\right]\right) \cdot \left(1 - \mathbb{I}_{(i,j)\in\mathcal{D}}\right) \end{aligned}$$

$\mathbb{I}_{(i,j)\in\mathcal{D}}$ denotes the indicator function which is 1 if $(i, j) \in \mathcal{D}$ and 0 otherwise.

The cost of one link $(i, j) \in \mathcal{L}$ is the expected cross entropy cost plus a regularizing term which penalizes more if the expected number of sentences in the samples is far from what we want:

$$Cost(i, j) := \mathbb{E}_{S\sim\overline{\mathcal{P}}_L} cost(i, j, S) + \lambda \cdot (\mathbb{E}_{S\sim\overline{\mathcal{P}}_L}|S| - s)^2 \tag{3.2}$$

The expectation goes over the sampled sets S from the modified L-Ensemble $\overline{P}_L$. s and λ are two hyperparameters of the system. s is the number of sentences which the model should sample in average. λ is a regularizing parameter. The higher λ, the more you penalize a wrong expected set size. Good values for s are in the range $\{1, 2, \ldots, 10\}$ and λ should be around 0.01 or 0.001. Note that s is only a soft way to tell the model how much sentences it should sample. If it thinks that sampling more or less sentences gives much better accuracy, it can still do this.

The total cost in the data set is then the mean cost over all links.

$$COST := \frac{1}{|\mathcal{L}|} \sum_{(i,j)\in\mathcal{L}} Cost(i, j) \tag{3.3}$$

3.9 Backpropagating Through the Expectation

In this section we show how the gradient of the cost function presented in the last section can be calculated. In particular we explain how we can make a backpropagation step through the expectation in the cost function. This allows to approximate the expectation and to use gradient descend methods to optimize the model.

Note that we only have to learn parameters in 3 parts of the model. In the following let Θ_{FF} denote the set of weights to learn in the FF-Network (Section 3.4). Similarly, let Θ_{DS} be the set of weights in the Deep Set layer (Section 3.6) and Θ_{LR} stand for the set of weights in the Logistic Regression layer (Section 3.7).

We only present the results of the gradient calculations here. The calculations itself can be found in Appendix B.

Let $(i,j) \in \mathcal{L}$. For $\theta_{ff} \in \Theta_{FF}$ we have

$$\begin{aligned}\frac{\partial Cost(i,j)}{\partial \theta_{ff}} &= \mathbb{E}_{S\sim\overline{\mathcal{P}}_L}\left[cost(i,j,S)\cdot\frac{\partial \log \overline{\mathcal{P}}_L(S)}{\partial \theta_{ff}} + \frac{\partial cost(i,j,S)}{\partial \theta_{ff}}\right] \\ &\quad + 2\lambda\left(\mathbb{E}_{S\sim\overline{\mathcal{P}}_L}|S| - s\right)\cdot\mathbb{E}_{S\sim\overline{\mathcal{P}}_L}\left[\frac{\partial \log \overline{\mathcal{P}}_L(S)}{\partial \theta_{ff}}\cdot|S|\right].\end{aligned}$$

Furthermore, for $\theta \in \Theta_{DS} \cup \Theta_{LR}$, we have

$$\frac{\partial Cost(i,j)}{\partial \theta} = \mathbb{E}_{S\sim\overline{\mathcal{P}}_L}\left[\frac{\partial cost(i,j,S)}{\partial \theta}\right].$$

These two equations suggest a way to optimise this complicated cost function. Analogue to the work from Lei et al., 2016, we can use the Monte Carlo method to approximate the expectations in the cost function (Metropolis and Ulam, 1949). We sample sets $S \sim \overline{\mathcal{P}}_L$ and average over the respective costs to approximate the expectations. If done like this, the partial derivatives are only needed for $cost(i,j,S)$ and $\log \overline{\mathcal{P}}_L(S)$. Both these derivatives can be calculated without problems.

Chapter 4

Empirical Results

We ran some experiments to see how our model performs on a real-world data set. Section 4.1 explains how we constructed the necessary data sets to train our model. In Section 4.2 we describe our experimentation setup. Furthermore, this section presents the baseline models which we used to evaluate our model. We then compare our model with the introduced baseline models in Section 4.3. Section 4.4 looks at how the model behaves if we sample more or less sentences in the L-Ensemble. Section 4.5 demonstrates that making samples depending on the context product increases the model performance. Whereas in the other sections we focus on evaluating the model on a set of new links, Section 4.6 looks at how the model performs when it is exposed to a completely new set of target products. With new target products we mean products which never appeared as target products during training. Hence, the model has not seen the user review sentences of these products during training. In the last section of this chapter we discuss how meaningful the sentences are which the model outputs in the L-Ensemble.

4.1 Data Sets

This section explains how we constructed the data sets for training our model. The first subsection shortly presents the real-world Amazon-scrawled data set. Subsections 4.1.2 - 4.1.4 then explain how we constructed the data sets from it which our model from Chapter 3 needs.

4.1.1 Real-World Amazon Data

As our data source we use crawled Amazon product data, which McAuley et al. have made available for download [1] (He and McAuley, 2016; McAuley

[1]http://jmcauley.ucsd.edu/data/amazon/ (accessed: 13. November 2017)

et al., 2015b). The data set consists of over 140 million user reviews for products in more than 20 categories (e.g. Books, Electronics, Kindle Applications, Magazine Subscriptions). Among other information, some of these reviews have a *helpfulness* tag which we use to measure how useful reviews are. Furthermore, metadata for all products exists. This metadata contains for each product four lists of related products. The lists are called *also_bought*, *also_viewed*, *bought_together* and *buy_after_viewing*. From this metadata we can build suitable data sets $\mathcal{D}$ and $\mathcal{D}_C$ for our problem (see Subsection 4.1.4).

We split all user reviews and metadata into separate data sets according to the category of the products. For this thesis we mainly used two categories. The first category, which we will call the *Kindle* data set, contains applications (not eBooks) for the Kindle eReader. These applications can for example be games, weather applications, apps for productivity, etc. The second category is called *Magazine Subscriptions*. After buying such a product, you will receive every few weeks a new issue of your chosen Magazine.

In the following subsections we show how we processed the scrawled Amazon data to construct data sets for these two categories. Table 4.1 shows the resulting data set sizes.

	Kindle	*Magazine Subscriptions*
Number of different products ($\|\mathcal{Y}\|$)	459	515
Number of links in $\mathcal{D}$ ($\|\mathcal{D}\|$)	24600	8400
Number of links in $\mathcal{D}_C$ ($\|\mathcal{D}_C\|$)	24600	8400

Table 4.1: The data set sizes

4.1.2 FLIC Topic Vectors

In this section we show how the FLIC topic vectors $\mathcal{C}(y)$ for a context product $y \in \mathcal{Y}$ can be constructed.

Let $\mathcal{Y}$ be a set of product (e.g. the set of products in our *Kindle* data set). Remember from Section 2.1.2 that the FLIC model works on a dataset $\overline{Y} \subseteq 2^{\mathcal{Y}}$, where $2^{\mathcal{Y}}$ is the power set of $\mathcal{Y}$. We constructed $\overline{Y}$ in the following way:

- For each $y \in \mathcal{Y}$, we merged the four lists *also_bought*, *also_viewed*, *bought_together* and *buy_after_viewing* and transformed it to a set by deleting duplicate products. Let us call this resulting set of products S_y. To get good FLIC topic vectors, we assume that the product y is similar to all products in S_y.
- We removed all products y from $\mathcal{Y}$ for which $|S_y| < 5$ (i.e. we removed all products that are not similar to at least five other products).

- As a next step, we sampled 10 random subsets from $S_y \cup \{y\}$ for all $y \in \mathcal{Y}$. We made sure that each subset consists of at least three products and that no two subsets are the same. The exact sampling procedure can be found in the programming code.
- The set $\overline{Y}$ is the union of all sets constructed in the last step for all products.

We trained the FLIC model using the Noise Contrastive Estimation Algorithm (Gutmann and Hyvärinen, 2010; Djolonga et al., 2016). The noise data set $\mathcal{N}$ needed for this algorithm has been constructed by taking random subsets from $\mathcal{Y}$. Each product $y \in \mathcal{Y}$ appears in a specific set in $\mathcal{N}$ with its empirical frequency of being in a set in $\overline{Y}$.

4.1.3 The Review Data Set

In this subsection we look at how to construct the set of sentence embeddings $\mathcal{R}(y)$ for a product $y \in \mathcal{Y}$. For this let $Re(y)$ be the set of user reviews for product y.

- The L-Ensemble in our model does not scale well with respect to the number of sentence embeddings it receives as its input. Hence, we used the *helpfulness* tags associated with the reviews and the lengths of the reviews to rank all of them. According to this ranking, we selected the top five reviews from $Re(y)$ to get a smaller set $\overline{Re}(y)$ of reviews.
- For each $y \in \mathcal{Y}$ we split the reviews $\overline{Re}(y)$ into a set of sentences and removed some special characters (e.g. *, ~).

After the last step each product has between 15 and 100 sentences. These sentences represent $\mathcal{R}(y)$ and can be put into the sentence embedding model from Section 3.3.

4.1.4 The Link Data Sets $\mathcal{D}$ and $\mathcal{D}_C$

In this section we show how to construct the link data sets from the Amazon data. For this let $\mathcal{Y}$ and S_y for all $y \in \mathcal{Y}$ be defined as in Section 4.1.2. We do the following:

- $\mathcal{D}$ was constructed by taking all possible links from S_y for all y (i.e. $\mathcal{D} := \{(y, s) \mid y \in \mathcal{Y} \text{ and } s \in S_y\}$).
- To construct $\mathcal{D}_C$, we sampled for each product $y \in \mathcal{Y}$ links which are not in $\mathcal{D}$. To do this we first defined the set $\overline{S_y} := \mathcal{Y} \setminus (S_y \cup \{y\})$. This set contains all products not in S_y (excluding y). From this set we sampled a subset $\tilde{S}_y$ of cardinality $|S_y|$ u.a.r.. $\mathcal{D}_C$ is then given by $\mathcal{D}_C := \{(y, s) \mid y \in \mathcal{Y} \text{ and } s \in \tilde{S}_y\}$.

Constructing the data sets this way ensures that $\mathcal{D}$ and $\mathcal{D}_C$ have the same cardinality and are disjoint.

4.2 Experimentation Setup

We split the links in the data set $\mathcal{L} = \mathcal{D} \cup \mathcal{D}_C$ into 60% training set, 20% validation set and 20% test set. The training set was used to update the model parameters during training. For this we used the Adam optimizer (Kingma and Ba, 2014). The score on the validation set was used to decide when to reduce the learning rate and when to stop training (early stopping). We started with a learning rate of 10^{-3} and divided it by 4 when the cost on the validation set did not decrease over the course of 3 epochs. We stopped the training when the learning rate became smaller than 10^{-5}.

We used the test set for all results and plots presented in this chapter. Furthermore, each experiment was performed using 5-fold cross-validation. Hence, each data point and box in the boxplots has been constructed using five points from the cross-validation run.

As a measure for prediction performance we chose accuracy. We define this as the percentage of all links $(i, j) \in \mathcal{L}$ which the model classifies correctly into the classes $\mathcal{D}$ and $\mathcal{D}_C$. A bit more formal: Let $(i, j) \in \mathcal{L}$ be a link and $\{S_1, S_2, \ldots, S_N\}$ denote the sets of sentence embeddings which the model samples (see Chapter 3). If the model outputs $\frac{1}{N} \sum_{k=1}^{N} Pr[(i, j) \in \mathcal{D} | S_k] > 0.5$, then the model classifies the link (i, j) as a link from $\mathcal{D}$. Otherwise it classifies it as a link from $\mathcal{D}_C$. Accuracy is then the percentage of correctly classified links:

$$Accuracy := \frac{1}{|\mathcal{L}|} \left[\sum_{(i,j) \in \mathcal{D}} \mathbb{I}_{\text{prob}>0.5} + \sum_{(i,j) \in \mathcal{D}_C} \mathbb{I}_{\text{prob} \leq 0.5} \right],$$

where

$$\text{prob} := \frac{1}{N} \sum_{k=1}^{N} Pr[(i, j) \in \mathcal{D} | S_k]$$

$\mathbb{I}$ is the indicator function as defined in Section 3.8.

4.2.1 Hyperparameters

There are a lot of hyperparameters in our model. We only explain the most important ones here. More details can be found in the code.

We pretrained the FLIC topic vectors such that each product $y \in \mathcal{Y}$ had an associated vector $\mathcal{C}(y)$ of length 200. Furthermore, our pretrained sentence embedding vectors have 256 dimensions.

Because of the complexity of our model (and especially the L-Ensemble), training the model is difficult and slow. We achieved the best results with only one hidden layer in the *FF-Network* and one permutation equivariant layer in the *Deep Set* part. We chose 256 dimensions for the output of the permutation equivariant layer.

In order to approximate the expectations in the cost function using the Monte Carlo method, we average over the costs of 10 sampled sets of sentence embeddings for each link during training. To take more sets would reduce the variance of the empirical expectations, but is computationally too expensive.

For most experiments we want to sample concise sets of sentences. We set the hyperparameter s, which is an approximation of the expected number of sentences in each sampled set, to 3. Hence, in average we want our model to sample three sentences per set.

Table 4.2 summarises the values of the most important hyperparameters.

Dimension FLIC	200
Dimension sentence embeddings	256
Dimension perm. equiv. layer	256
Percentage validation data	20%
Percentage test data	20%
Number of sampled sets per link	10
The expected set size s	3
Set size regularizer λ	0.005

Table 4.2: The values of the most important hyperparameters in our model.

4.2.2 Baselines

This subsection covers the baselines we used to validate our model.

Independent Sentences Model

This model is very similar to our model, but we restrict the entries of the ensemble matrix L to its diagonal elements:

$$L_{mn} := \begin{cases} scale \cdot \phi_m^T \phi_n \cdot \tanh(\phi_m^T \overline{C}(i)) \cdot \tanh(\phi_n^T \overline{C}(i)) & \text{if } m = n \\ 0 & \text{otherwise} \end{cases}$$

So all non-diagonal elements are 0. Assume without loss of generality that we sample subsets S from $\mathcal{Y} = \{1, 2, \ldots, N\}$. The probability of S is then given by

$$\overline{\mathcal{P}}_L(S) \propto \det(L_S) = \prod_{y \in S} L_{yy}$$

Hence all items in the sets are sampled independently from each other. This means that the model doesn't sample sets with diverse items. It just chooses all the sentences which are similar to the context.

Uniformly Sampled Sentences Model

In this model we assume that all sentences in $\mathcal{R}(j)$ for a product $j \in \mathcal{Y}$ are equally important. The model is similar to ours, but here we sample sets S by adding all sentence embeddings $r \in \mathcal{R}(j)$ u.a.r. with probability $Pr[r \in S] := \frac{s}{|\mathcal{R}(j)|}$ to S. s is the average number of sentences which our model should sample (see Section 3.8).

The sampled sentences are then processed as in our main model (i.e. we use Deep Sets, Logistic Regression and then the average cross entropy cost).

As this model samples the sentences uniformly at random, it can be used to show that sampling a good subset of sentences can make a difference in prediction accuracy.

Logistic Regression Baseline

We decided to add a very simple baseline. It concatenates the two FLIC topic vectors of the context and target product and applies Logistic Regression to it. Hence, for a link $(i, j) \in \mathcal{L}$ the model calculates

$$Pr[(i,j) \in \mathcal{D}] := \sigma(w \cdot [\mathcal{C}(i)], \mathcal{C}(j)]),$$

where

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

is the sigmoid activation function. w is a weight vector of size two times the dimension of the FLIC topic vectors.

Note that this model has less free parameters than our other models. On the other hand, it doesn't need to sample sentences, but only outputs a final score.

In Section 4.5 we will use slightly different models than for the rest of this chapter. We will present those models directly in that section.

4.3 Comparing the Baseline Models

In this section we compare our model with the three baseline models which we introduced in the last section.

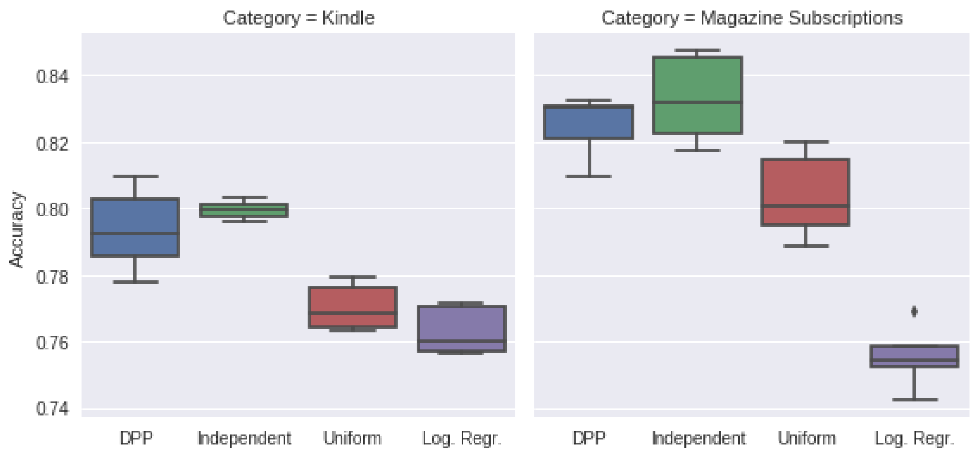

Figure 4.1: The prediction accuracy of the four models in the categories *Kindle* (on the left) and *Magazine Subscriptions*. The DPP model and the Independent model perform best. The Uniformely Sampled Sentences model performs less good and the Logistic Regression baseline is worst.

Look at Figure 4.1. We see two boxplots, one for the *Kindle* data set and one for the *Magazine Subscriptions* data set. Each boxplots shows the prediction accuracies on the test set for the four models DPP (the model described in Chapter 3), Independent (the first baseline model with the independent sentences), Uniform (the second baseline model, where the sentences have been chosen u.a.r.) and the Logistic Regression baseline. Remember from Section 4.2 that each box in the boxplot has been constructed by using the results from a 5-fold cross-validation run. Hence, it is possible to see all five data points from the cross-validation run on the boxplots. Each of the horizontal lines in the boxes represents one data point.

We see that the DPP model and the model with the independently sampled sentences perform best in both data sets. The model with the uniformely sampled sentences gives much lower accuracy. This suggests that a good selection of sentences in the L-Ensemble is indeed important for good predictions.

It is interesting that the DPP model doesn't perform better than the model with the independent sentences. This means that diversity in the sets of sampled sentences is not important for making accurate predictions in our data sets. Indeed, one can think about cases, where it is better to sample two similar sentences instead of two different ones. When the model receives two similar sentences which fit both well to the context product, the model might be persuaded that the target product is a good recommendation for the context product. If, on the other hand, the model samples two different sentences, where one sentence is only partially similar to the context,

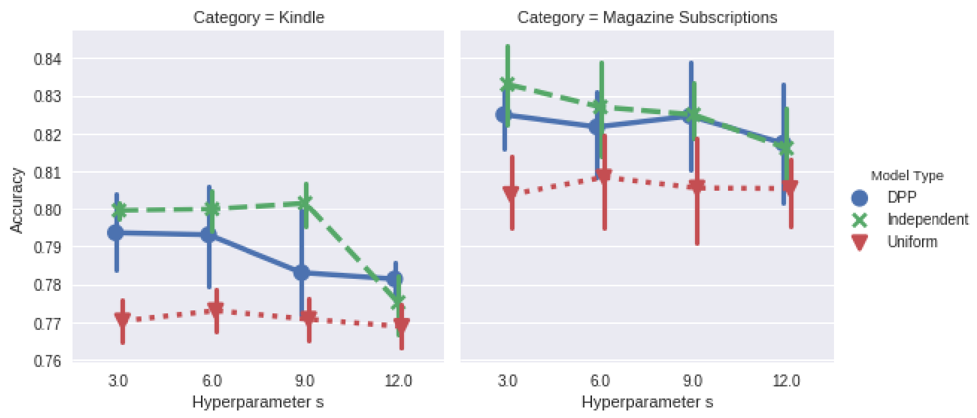

Figure 4.2: These plots show the accuracy of the predictions with respect to the hyperparameter s. s stands for the average number of sentences which our model should sample (see Section 3.8). On the plots we can see that the accuracies of the DPP and the Independent model slightly decrease when s gets bigger.

it might be less sure about the connection between the context and the target product. See Section 4.7 for an example of this. Ideally, we want that $\mathcal{R}(j)$ consists of a set of sentences which all describe the product $j \in \mathcal{Y}$ well. In practice, however, there are a lot of products j, where the set of sentences $\mathcal{R}(j)$ only explains one or even no property of the product. In that case, sampling diverse and still relevant sentences is very difficult. Hence, the accuracy stays the same or even drops when the model tries to enforce diversity in the sampled sets.

The Logistic Regression baseline performs worst. This is understandable as it has a smaller amount of weights to optimise than the other models.

4.4 Influence of the Number of Sampled Sentences on the Prediction Accuracy

In this section we look at how the number of sentences in the sampled sets influences the prediction accuracy.

Remember from Section 3.8 that the hyperparameter s denotes the average number of sentences which we want the model to sample for each set in the L-Ensemble. Intuitively, we expect that a bigger amount of sentences in the sampled set should help the model to more easily decide whether two products fit well together or not. In Figure 4.2, however, we see that this intuition doesn't hold. If s increases, the accuracy stays approximately the same or even decreases. In the following paragraph we explain on an example why this behaviour happens.

Assume you bought a very trendy looking keyboard from Logitech on Amazon. Now Amazon wants to sell you a computer mouse with the following set of sentences $\mathcal{R}(\text{mouse})$:

This computer mouse has a very trendy design.
It is from the respectable brand Logitech.
The mouse is quite new.
One of the worst computer mice I have ever seen.
Why is it allowed to sell a bad product like this?
This mouse is great.

Only the first two sentences (the one in red and underlined) fit to the trendy keyboard you bought. Let us conceptually apply the model from Chapter 3 on the link (keyboard, mouse). If we want to sample a set of three products, a good set of sentences to describe the relationship between the keyboard and the mouse is {*This computer mouse has a very trendy design., It is from the respectable brand Logitech., The mouse is quite new.*}. Using this subset of sentences to predict, the model most likely says that the mouse fits well to the keyboard. Assume now that you want to sample a set of six sentences from $\mathcal{R}(\text{mouse})$. In that case the sampled set might be the full set of sentences. So the model uses all sentences to predict if the mouse fits good to the keyboard or not. Hence, the model receives two relevant input sentences (i.e. the sentences in red and underlined) and four irrelevant ones. This might be too much noise for the model to handle and it might predict that the mouse is not a good recommendation for the keyboard.

The last paragraph was just an example, but the general principle holds. The more sentences we want to sample, the higher is the average number of noisy sentences in the sampled sets. If the number of noisy sentences in the samples is too high, the prediction accuracy will go down. This hypothesis is confirmed by the fact that the model tends to sample less sentences than it should when s is high (see Table 4.3). By sampling less sentences, the model needs to deal with less noise in the sampled sets and hence achieves better accuracy.

What is the underlying reason that the model can not handle too many noisy sentences? If the model processes a link $(i, j) \in \mathcal{D}_C$, then the products i and j are not similar. Hence, the set $\mathcal{R}(j)$ can not describe the product i well. That is why the model learns that sets with a lot of noisy sentences often belong to the class $\mathcal{D}_C$.

To make better use of a higher number of extracted sentences, a model could try to weight the effect of the different sentences on the predictions (attention based mechanism). We let this open for a future work to explore.

Hyperparameter s (from Section 3.8)	3	6	9	12
DPP	3.6	6.0	8.2	10.4
Independent	3.2	5.6	7.4	9.8

Table 4.3: The entries in this table show the empirically measured average number of sentences that the model samples when it should sample s sentences. We used the *Kindle* data set for these measurements. It is noticeable that for large s the empirical average set size is much lower than s itself.

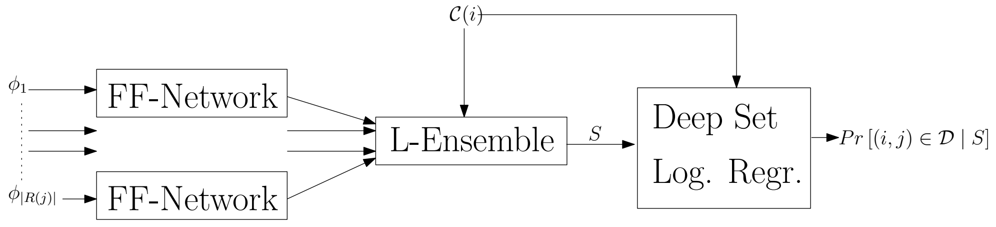

Figure 4.3: The modified model architecture we use for this section. Instead of applying the FF-Network to the context topic vector $\mathcal{C}(i)$, we apply it to the set of sentence embeddings. The rest of the model is the same as in the main model (see Figure 3.2).

4.5 Context-dependent Sampling

In this section we answer the question if prediction accuracy decreases when we make the sampling step independent of the context product.

Unfortunately, our model is not well suited to answer this question. Note that the only parameters we train before the sampling step are in the FF-Network (see Section 3.2). Hence, if we make the L-Ensemble independent of the context product, we can not train to sample good sets anymore because we don't have any parameters to train. Because of this, we slightly change the architecture of the model we are considering. We introduce two new models for this section:

Using Context model See Figure 4.3. We modify the main model from Chapter 3 in two points. The first one is that we apply the FF-Network to all embeddings in the set of sentence embeddings $\{\phi_1, \ldots, \phi_{|\mathcal{R}(j)|}\}$ and not to $\mathcal{C}(i)$. This modification allows the training of the sentence embeddings such that the L-Ensemble outputs good sets even when we will later remove the context vector $\mathcal{C}(i)$ from it (see the *No Context* model below). The second change is that we normalize all sentence embedding vectors to one (i.e. $\|\phi_k\|_2 = 1$ for all $k \in \{1, 2, \ldots, |\mathcal{R}(j)|\}$. This allows us to take the magnitude of the sentence embedding vectors out of the game when sampling. The model from Chapter 3 has the property that sentence embeddings with

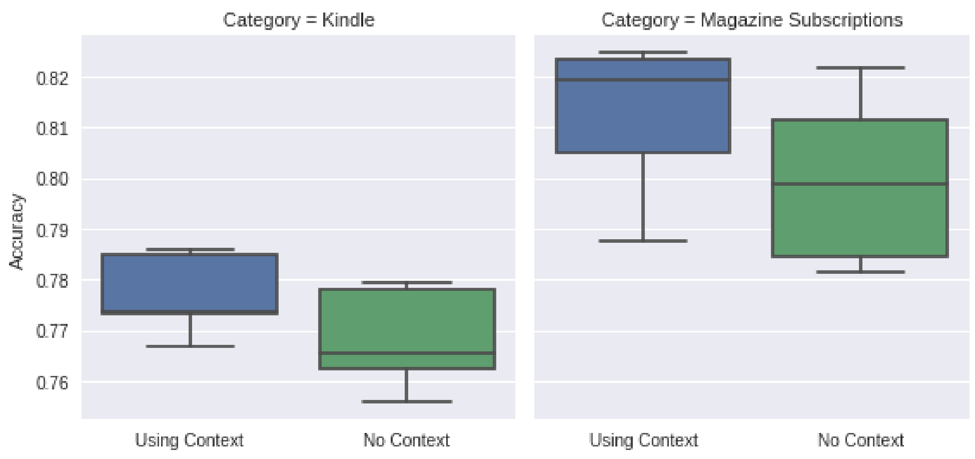

Figure 4.4: We compare the two new models from this section. On both datasets, using the context to sample sets of sentences in the L-Ensemble seems to give better accuracy compared to not using the context.

a bigger norm get sampled more frequently, which we don't want for the models considered in this section.

No Context model This model is the same as the *Using Context* model with one small modification. We set the entries of the ensemble matrix L to be

$$L_{mn} := scale \cdot \phi_m^T \phi_n \tag{4.1}$$

This modification makes the ensemble matrix L independent of the context vector $\mathcal{C}(i)$.

We can now compare the two models we introduced in this section. See Figure 4.4. In both data sets, the *Using Context* model seems to perform a bit better than the *No Context* model. Hence, sampling sentences which depend on the context product might increase the prediction accuracy. We did not perform a hypothesis test to confirm this because our sample sizes were too small.

4.6 New Products

So far we evaluated the models on test sets which contain new links. However, even though the links were new, the model has already seen all products and user reviews from the test set during training. In this section we look at how well our model performs when it is exposed to a set of target products which the model has not seen during training. Hence, we basically answer the question of how well the model handles new user reviews.

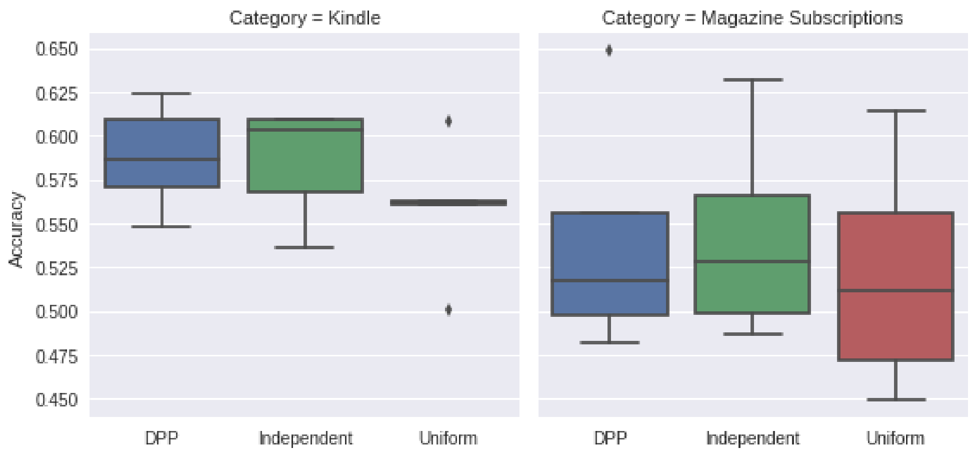

Figure 4.5: The accuracies of the models on the test set with new target products (i.e. the review sentences were not seen during training). In the *Kindle* data set, the models are able to apply what they learned on new products. In the *Magazine Subscription* data set, the models do not perform good.

In the last few sections we constructed the data for the cross-validation loop by partitioning the link data $\mathcal{L}$. In this section we split the set of products $\mathcal{Y}$. For each fold of the cross-validation loop we remove 20% of the products. These products appear neither in the training nor in the validation set as target products. Instead, the links which have the removed products as target products form the test set. Note that the removed products can still appear as context products in the training set. This is fine because the model does not see the reviews of these products during training.

See Figure 4.5. Using the new test set, our model achieves about 60% prediction accuracy on the *Kindle* data set. This is much higher than 50%, which is the accuracy for random guessing. So the model is able to apply what it learned on new target products with unseen reviews. As expected, however, the accuracy is much lower than the one in Section 4.3. There, the target products in the test set were already seen during training.

The performance of the model on the *Magazine Subscriptions* data set is much weaker. The reason for this is that the number of links in that data set is smaller than in the *Kindle* data set. Hence, the training time is shorter to reduce overfitting. We conclude that the model can sometimes learn to apply its knowledge on new, unseen reviews, but that it needs a lot of data and training time to do this well. So the model is able to apply what it learned on new target products with new unseen reviews.

4.7 Quality of the Sampled Sentences

One goal of our model is to output an interpretable set of sentences, which explains why the model predicts as it does. In this section we look at this using an example.

We are considering the following three products from the *Kindle* data set. Note that you can have a look at each of these products on the Amazon website by clicking on the link in the footnotes.

1. The first product is a Solitaire game[2]. We will use this product as the target product in the links we are considering. We call it *Solitaire 1*.
2. The second product is another Solitaire game[3]. This product is very similar to *Solitaire 1*. Because of this, we will call this second product *Solitaire 2*. After training our model, it is very sure that the link (*Solitaire 2*, *Solitaire 1*) is in the data set $\mathcal{D}$ (and not in the data set $\mathcal{D}_C$).
3. The third product is a game book[4], i.e. an interactive book, where the reader chooses actions while reading the book and hence influences the story. We call this product *Game Book*. This product is clearly different from the first two Solitaire games and our trained model is sure that the link (*Game Book*, *Solitaire 1*) belongs to the data set $\mathcal{D}_C$.

In a first step, we evaluate our model from Chapter 3 on the link (*Solitaire 2*, *Solitaire 1*). Remember that $\mathcal{R}(\textit{Solitaire 1})$ denotes the set of sentence embeddings belonging to the product *Solitaire 1*. We sample 100 sets of sentence embeddings in the L-Ensemble and count how many times each sentence appears. Note that this ignores any dependencies between the sentences in the sampled sets. This is fine because diversity between the sentences is not very important for the data set (see Section 4.3).

When looking at the link (*Solitaire 2*, *Solitaire 1*), the following four sentences from $\mathcal{R}(\textit{Solitaire 1})$ are the ones which our model samples most:

1) Not as well designed as the EA Solitaire game for the Kindle keyboard, waiting for them to release a Paperwhite version.
2) I enjoyed the many choices on the menu for different types of Solitaire games.
3) The bonus is I am learning to play a variety of Solitaire games versus the one game I knew.
4) The reason I bought this suite was for the Klondike solitaire.

It is interesting to note that all these sentences contain the word *Solitaire* (we colored it red and underlined it). Our trained model recognizes this and

[2] https://www.amazon.com/dp/B009ZI12NM (accessed: 29. November 2017)
[3] https://www.amazon.com/dp/B006IZMWWA (accessed: 29. November 2017)
[4] https://www.amazon.com/dp/B00B0RGGF6 (accessed: 29. November 2017)

suggests sets containing these sentences as the explanation why *Solitaire 1* is a good recommendation for the context product *Solitaire 2*.

Let us now look at the output from our model, when it looks at the link (*Game Book*, *Solitaire 1*). The model most often samples the following four sentences:

1) In spite of its being in shades of gray, it's easy to differentiate between the red and black suits.
2) I enjoyed the many choices on the menu for different types of Solitaire games.
3) I didn't have any trouble distinguishing the black vs red.
4) (I didn't try it on any other type of Kindle so don't know how it does on them).

As the products *Game Book* and *Solitaire 1* are very different products, it is not easy for the model to sample good sentences. That is why the set of sentences looks almost random and there is no clear connection to the *Game Book* product. This behaviour of the model is of course to expect because there are no sentences which describe the connection between *Game Book* and *Solitaire 1*.

To summarise this section, we can say that our model sometimes is able to extract meaningful sentences for links in the data set $\mathcal{D}$ and chooses random sentences for links in $\mathcal{D}_C$. Nevertheless, we need to say that the sampled sets are not always as nice as in the example from this section. There are a lot of links, for which it is very difficult to understand as a human why the model samples some specific sentences to explain its predictions.
Another issue with our sentence sampling approach is that the model has a bias towards sampling some sentences more often than others (irrespective of the context product). The reason for this is that the entries L_{mn} of the ensemble matrix depend on the magnitude of the untrained sentence embedding vectors ϕ_m and ϕ_n (compare Equation 3.1). For further research we recommend to normalize the sentence embedding vectors to remove this bias.

Chapter 5

Conclusion

In this thesis we proposed a new interpretable Recommender System model. Given a context product and a target product, the model predicts in two steps if the target product is a good recommendation for the context product or not. The first step is to use an L-Ensemble to sample a subset from the set of sentences which describes the target product. The second step then uses the extracted sentences to predict if the target product fits well to the context product or not. We evaluated the model on real-world Amazon product data. We used user reviews for the set of sentences which describes the target product. Experiments suggest that a good selection of the sentences in the L-Ensemble is important for accurate predictions. It is also helpful to include the context product into the sampling procedure. In this way the model chooses sentences which are similar to the context. Furthermore, we have demonstrated that the model often extracts sentences which describe well why the target product is a good recommendation for the context product. We have also shown that our model can sometimes apply the text understanding it learned during training to unseen user reviews on test time.

The Amazon user review data partially violates an important assumption of our model. In particular, some user reviews do not necessarily describe the products very well. Often, only few sentences per product were good descriptions of the product and all others were not useful. Hence, it would be interesting to see how the model performs when it is applied to a different data set with better product descriptions. We expect that the results found in this thesis would be more significant.

We demonstrated that the model performance does not necessarily improve when more sentences are sampled in the L-Ensemble. One way to improve the prediction accuracy for more sampled sentences could be to introduce an attention-based mechanism which weights some sentences as more important than others. We leave this for future research.

Diversity in the sampled sets from the *L*-Ensemble doesn't seem to be important for our data sets. Considering this, we can replace the runtime inefficient *L*-Ensemble with something simpler, for example a model which samples the sentences independently. Other modifications of the model are also possible. One can try to sample the text on a finer granularity (e.g. on the word or phrases level). This way of sampling is more difficult as the words depend on each other. On the other hand, one might achieve conciser and better interpretable text outputs.

Appendix A

Some Important Results about L-Ensembles

This appendix provides a summary of the most important theorems and algorithms about L-Ensembles which we need for this thesis. All proofs for these results can be found in the work form Kulesza and Taskar, 2012.

Without loss of generality we assume in the following that we have a ground set $\mathcal{Y}$ of the form $\mathcal{Y} := \{1, 2, \dots, N\}$, where N is the number of items in the ground set. We are interested in drawing subsets $S \subseteq \mathcal{Y}$ according to the L-Ensemble given by $\mathcal{P}_L$ (see Definition 2.2).

Lemma A.1 *(Theorem 2.1. in Kulesza and Taskar, 2012)*
For any $A \subseteq \mathcal{Y}$,

$$\sum_{A \subseteq S \subseteq \mathcal{Y}} det(L_S) = \det(L + I_{\bar{A}}),$$

where $I_{\bar{A}}$ is the diagonal matrix with ones in the diagonal positions corresponding to elements of $\bar{A} = \mathcal{Y} - A$ and zeros everywhere else.

From this lemma follows directly that we can calculate the partition function of $\mathcal{P}$ efficiently. Hence, the probability of a set S is given by

$$\mathcal{P}_L(S) = \frac{\det(L_S)}{\det(L + I)}, \tag{A.1}$$

where I is the N x N identity matrix.

Lemma A.2 *(Equation 74 of page 22 in Kulesza and Taskar, 2012)*
P_L is log-submodular, that is,

$$\log \mathcal{P}_L(S \cup \{i\}) - \log \mathcal{P}_L(S) \geq \log \mathcal{P}(S' \cup \{i\}) - \log \mathcal{P}_L(S')$$

whenever $S \subseteq S' \subseteq \mathcal{Y} \setminus \{i\}$.

Hence, *L*-Ensembles are natural candidates if we want to sample sets with diverse items.

It is possible to sample efficiently from L-Ensembles.

Theorem A.3 *(Theorem 2.3 in Kulesza and Taskar, 2012)*
Let $L = \sum_{n=1}^{N} \lambda_n v_n v_n^T$ be an orthonormal eigendecomposition of a positive semi-definite matrix L. Then Algorithm 1 samples $S \sim \mathcal{P}_L$.

Algorithm 1 Sampling from an *L*-Ensemble *(Algorithm 1 on page 16 in Kulesza and Taskar, 2012)*

```
1: Input: Set $\mathcal{Y}$, eigendecomposition $\{(v_n, \lambda_n)\}_{n=1}^{N}$ of $L$
2: $\mathcal{J} \leftarrow \emptyset$
3: for $n = 1, 2, \dots, N$ do
4:     $\mathcal{J} \leftarrow \mathcal{J} \cup \{n\}$ with probability $\frac{\lambda_n}{\lambda_n + 1}$.
5: end for
6: $V \leftarrow \{v_n\}_{n \in \mathcal{J}}$
7: $S \leftarrow \emptyset$
8: while $|V| > 0$ do
9:     Select $i$ from $\mathcal{Y}$ with $Pr(i) = \frac{1}{|V|} \sum_{v \in V} (v^T e_i)^2$
10:    $S \leftarrow S \cup \{i\}$
11:    $V \leftarrow V_\perp$, an orthonormal basis for the subspace of $V$ orthogonal to $e_i$.
12: end while
13: Output: $S$
```

Algorithm 1 samples in two steps. In the first step a random set of eigenvectors gets selected. In the second step, one adds one element after another to S based on the selected eigenvectors. The orthonormal basis in line 11 is a bit tricky because one can not directly use an algorithm like Gram-Schmidt to get the result. In general the resulting space would not be orthogonal to e_i. Instead we recommend to first compute the subspace of V orthogonal to e_i and then to use an algorithm like the QR-decomposition to calculate the orthonormal basis.

The last thing we need for our model is the expectation of the expected set size.

Corollary A.4 *(Equation 34 on page 12 in Kulesza and Taskar, 2012)*
Let $\lambda_1, \lambda_2, \dots, \lambda_N$ be the eigenvalues of L and let $\mathbf{Y}$ denote a random subset drawn from $\mathcal{Y}$ according to $\mathcal{P}_L$. Then $|\mathbf{Y}|$ is distributed as the number of successes in N Bernoulli trials, where trial n succeeds with probability $\frac{\lambda_n}{\lambda_n + 1}$. The expected

cardinality of $\mathbf{Y}$ *is*

$$\mathbb{E}[|\mathbf{Y}|] = \sum_{n=1}^{N} \frac{\lambda_n}{\lambda_n + 1}.$$

Hence, the expected set size can be calculated very efficiently when an eigenvalue decomposition of L is given.

Appendix B

Gradient Calculations

In this appendix we derive the gradient results from Section 3.9. As in Section 3.9, let Θ_{FF} denote the set of weights to learn in the FF-Network (Section 3.4). Similarly let Θ_{DS} be the set of weights in the Deep Set layer (Section 3.6) and Θ_{LR} stand for the set of weights in the Logistic Regression layer (Section 3.7).

The calculation of the gradients follows to some degree what Lei et al., 2016, did for their model. In particular, we will use the following equivalence several times:

$$\frac{\partial \overline{\mathcal{P}}_L(S)}{\partial \theta} = \frac{\partial \overline{\mathcal{P}}_L(S)}{\partial \theta} \cdot \frac{\overline{\mathcal{P}}_L(S)}{\overline{\mathcal{P}}_L(S)} = \frac{\partial \log \overline{\mathcal{P}}_L(S)}{\partial \theta} \cdot \overline{\mathcal{P}}_L(S) \tag{B.1}$$

This calculation follows directly from the chain rule for computing derivatives.

We first calculate $\frac{\partial Cost(i,j)}{\partial \theta_{ff}}$ for $(i,j) \in \mathcal{L}$ and $\theta_{ff} \in \Theta_{FF}$. For this let us split up $Cost(i,i)$ in two parts A and B:

$$Cost(i,j) = \underbrace{\mathbb{E}_{S \sim \overline{\mathcal{P}}_L}[cost(i,j,S)]}_{A} + \underbrace{\lambda \cdot (\mathbb{E}_{S \sim \overline{\mathcal{P}}_L}[|S|] - s)^2}_{B}$$

We calculate

$$\frac{\partial A}{\partial \theta_{ff}} = \frac{\partial \mathbb{E}_{S\sim\overline{\mathcal{P}}_L}[cost(i,j,S)]}{\partial \theta_{ff}} = \frac{\partial \sum_{S\subseteq\Phi\setminus\varnothing} \overline{\mathcal{P}}_L(S)\cdot cost(i,j,S)}{\partial \theta_{ff}} \tag{1}$$

$$= \sum_{S\subseteq\Phi\setminus\varnothing}\left[\frac{\partial \overline{\mathcal{P}}_L(S)}{\partial \theta_{ff}}\cdot cost(i,j,S) + \overline{\mathcal{P}}_L(S)\cdot\frac{\partial cost(i,j,S)}{\partial \theta_{ff}}\right] \tag{2}$$

$$= \sum_{S\subseteq\Phi\setminus\varnothing}\left[\frac{\partial \log \overline{\mathcal{P}}_L(S)}{\partial \theta_{ff}}\cdot \overline{\mathcal{P}}_L(S)\cdot cost(i,j,S) + \overline{\mathcal{P}}_L(S)\cdot\frac{\partial cost(i,j,S)}{\partial \theta_{ff}}\right] \tag{3}$$

$$= \mathbb{E}_{S\sim\overline{\mathcal{P}}_L}\left[\frac{\partial \log \overline{\mathcal{P}}_L(S)}{\partial \theta_{ff}}\cdot cost(i,j,S) + \frac{\partial cost(i,j,S)}{\partial \theta_{ff}}\right] \tag{4}$$

In (1) and (4) we used the definition of the expectation. In (2) we used linearity of expectation and of differentiation and the product rule for derivatives. In (3) we used Equation B.1. Further we have

$$\frac{\partial B}{\partial \theta_{ff}} = \frac{\partial \lambda(\mathbb{E}_{S\sim\overline{\mathcal{P}}_L}[|S|]-s)^2}{\partial \theta_{ff}}$$

$$= 2\lambda\left(\mathbb{E}_{S\sim\overline{\mathcal{P}}_L}[|S|]-s\right)\cdot\frac{\partial \mathbb{E}_{S\sim\overline{\mathcal{P}}_L}[|S|]}{\partial \theta_{ff}} \tag{1}$$

$$= 2\lambda\left(\mathbb{E}_{S\sim\overline{\mathcal{P}}_L}[|S|]-s\right)\cdot\frac{\partial \sum_{S\subseteq\Phi\setminus\varnothing}|S|\cdot\overline{\mathcal{P}}_L(S)}{\partial \theta_{ff}} \tag{2}$$

$$= 2\lambda\left(\mathbb{E}_{S\sim\overline{\mathcal{P}}_L}[|S|]-s\right)\cdot\sum_{S\subseteq\Phi\setminus\varnothing}|S|\cdot\frac{\partial \overline{\mathcal{P}}_L(S)}{\partial \theta_{ff}} \tag{3}$$

$$= 2\lambda\left(\mathbb{E}_{S\sim\overline{\mathcal{P}}_L}[|S|]-s\right)\cdot\sum_{S\subseteq\Phi\setminus\varnothing}|S|\cdot\frac{\partial \log\overline{\mathcal{P}}_L(S)}{\partial \theta_{ff}}\cdot\overline{\mathcal{P}}_L(S) \tag{4}$$

$$= 2\lambda\left(\mathbb{E}_{S\sim\overline{\mathcal{P}}_L}[|S|]-s\right)\cdot\mathbb{E}_{S\sim\overline{\mathcal{P}}_L}\left[|S|\cdot\frac{\partial \log\overline{\mathcal{P}}_L(S)}{\partial \theta_{ff}}\right] \tag{5}$$

In (1) we used the chain rule and in (2), (5) the definition of the expectation. For (3) we again needed the linearity of the expectation and of the differentiation operator and for (4) Equation B.1. If we add $\frac{\partial A}{\partial \theta_{ff}}$ and $\frac{\partial B}{\partial \theta_{ff}}$ together, we get

$$\frac{\partial Cost(i,j)}{\partial \theta_{ff}} = \mathbb{E}_{S\sim\overline{\mathcal{P}}_L}\left[cost(i,j,S)\cdot\frac{\partial \log\overline{\mathcal{P}}_L(S)}{\partial \theta_{ff}} + \frac{\partial cost(i,j,S)}{\partial \theta_{ff}}\right]$$
$$+ 2\lambda\left(\mathbb{E}_{S\sim\overline{\mathcal{P}}_L}|S|-s\right)\cdot\mathbb{E}_{S\sim\overline{\mathcal{P}}_L}\left[\frac{\partial \log\overline{\mathcal{P}}_L(S)}{\partial \theta_{ff}}\cdot|S|\right]$$

Let us now calculate $\frac{\partial Cost(i,j)}{\partial \theta}$ for $(i,j) \in \mathcal{L}$ and $\theta \in \Theta_{DS} \cup \Theta_{LR}$. This case is much easier to calculate because $\overline{\mathcal{P}}_L(S)$ does not depend on θ.

$$\frac{\partial A}{\partial \theta} = \frac{\partial \mathbb{E}_{S \sim \overline{\mathcal{P}}_L}[cost(i,j,S)]}{\partial \theta} = \mathbb{E}_{S \sim \overline{\mathcal{P}}_L}\left[\frac{\partial cost(i,j,S)}{\partial \theta}\right]$$

This holds because of the linearity of expectation and of differentiation and because $\overline{\mathcal{P}}_L(S)$ doesn't depend on θ. Furthermore,

$$\frac{\partial B}{\partial \theta} = \frac{\partial \lambda(\mathbb{E}_{S \sim \overline{\mathcal{P}}_L}[|S|] - s)^2}{\partial \theta} = 0$$

because $\mathbb{E}_{S \sim \overline{\mathcal{P}}_L}$ does not depend on θ and λ, s are constants. Hence, we have

$$\frac{\partial Cost(i,j)}{\partial \theta} = \mathbb{E}_{S \sim \overline{\mathcal{P}}_L}\left[\frac{\partial cost(i,j,S)}{\partial \theta}\right].$$

Bibliography

Aciar, S. et al. (2007). “Informed Recommender: Basing Recommendations on Consumer Product Reviews”. In: *IEEE Intelligent Systems* 22.3, pp. 39–47. ISSN: 1541-1672. DOI: 10.1109/MIS.2007.55.

Arras, Leila et al. (2017). “”What is relevant in a text document?”: An interpretable machine learning approach”. In: *PLOS ONE* 12.8, pp. 1–23. DOI: 10.1371/journal.pone.0181142. URL: https://doi.org/10.1371/journal.pone.0181142.

Askira-Gelman, Irit (1998). “Knowledge discovery: Comprehensibility of the results”. In: *In Proc. HICSS, Maui, HI* 5, 247–255 vol.5.

Borodin, Alexei and Grigori Olshanski (2000). “Distributions on Partitions, Point Processes, and the Hypergeometric Kernel”. In: *Communications in Mathematical Physics* 211.2, pp. 335–358. ISSN: 1432-0916. DOI: 10.1007/s002200050815. URL: https://doi.org/10.1007/s002200050815.

Borodin, Alexei and Eric M. Rains (2005). “Eynard–Mehta Theorem, Schur Process, and their Pfaffian Analogs”. In: *Journal of Statistical Physics* 121.3, pp. 291–317. ISSN: 1572-9613. DOI: 10.1007/s10955-005-7583-z. URL: https://doi.org/10.1007/s10955-005-7583-z.

Chen, Li, Guanliang Chen, and Feng Wang (2015). “Recommender systems based on user reviews: the state of the art”. In: *User Modeling and User-Adapted Interaction* 25.2, pp. 99–154. DOI: 10.1007/s11257-015-9155-5. URL: https://doi.org/10.1007/s11257-015-9155-5.

Chen, Li and Feng Wang (2013). “Preference-based clustering reviews for augmenting e-commerce recommendation”. In: *Knowledge-Based Systems* 50.Supplement C, pp. 44–59. ISSN: 0950-7051. DOI: https://doi.org/10.1016/j.knosys.2013.05.006. URL: http://www.sciencedirect.com/science/article/pii/S095070511300155X.

Conneau, Alexis et al. (2017). “Supervised Learning of Universal Sentence Representations from Natural Language Inference Data”. In: *CoRR* abs/1705.02364. arXiv: 1705.02364. URL: http://arxiv.org/abs/1705.02364.

David, F. N., M. S. Kendall, and D. E. Barton (1966). "Symmetric Functions and Allied Tables". In: University Press.

Denil, Misha, Alban Demiraj, and Nando de Freitas (2014). "Extraction of Salient Sentences from Labelled Documents". In: *CoRR* abs/1412.6815. arXiv: 1412.6815. URL: http://arxiv.org/abs/1412.6815.

Djolonga, Josip and Andreas Krause (2014). "From MAP to Marginals: Variational Inference in Bayesian Submodular Models". In: *Advances in Neural Information Processing Systems 27*. Ed. by Z. Ghahramani et al. Curran Associates, Inc., pp. 244–252. URL: http://papers.nips.cc/paper/5492-from-map-to-marginals-variational-inference-in-bayesian-submodular-models.pdf.

Djolonga, Josip, Sebastian Tschiatschek, and Andreas Krause (2016). "Variational Inference in Mixed Probabilistic Submodular Models". In: *NIPS*.

Edmonds, Jack (1970). "Submodular Functions, Matroids, and Certain Polyhedra". In: *Combinatorial structures and their applications, pp. 69–87*.

Golge, Eren (2014). *Machine Learning History*. http://www.erogol.com/brief-history-machine-learning/. Accessed: 30.10.2017.

Gutmann, Michael and Aapo Hyvärinen (2010). "Noise-contrastive estimation: A new estimation principle for unnormalized statistical models". In: *Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics*. Ed. by Yee Whye Teh and Mike Titterington. Vol. 9. Proceedings of Machine Learning Research. Chia Laguna Resort, Sardinia, Italy: PMLR, pp. 297–304. URL: http://proceedings.mlr.press/v9/gutmann10a.html.

He, Ruining and Julian McAuley (2016). "Ups and Downs: Modeling the Visual Evolution of Fashion Trends with One-Class Collaborative Filtering". In: *CoRR* abs/1602.01585. arXiv: 1602.01585. URL: http://arxiv.org/abs/1602.01585.

Heckel, Reinhard and Michail Vlachos (2016). "Interpretable recommendations via overlapping co-clusters". In: *CoRR* abs/1604.02071. arXiv: 1604.02071. URL: http://arxiv.org/abs/1604.02071.

Herlocker, Jonathan L. et al. (2004). "Evaluating Collaborative Filtering Recommender Systems". In: *ACM Trans. Inf. Syst.* 22.1, pp. 5–53. ISSN: 1046-8188. DOI: 10.1145/963770.963772. URL: http://doi.acm.org/10.1145/963770.963772.

Jacobson, Kurt et al. (2016). "Music Personalization at Spotify". In: *Proceedings of the 10th ACM Conference on Recommender Systems*. RecSys '16. Boston, Massachusetts, USA: ACM, pp. 373–373. ISBN: 978-1-4503-4035-9. DOI: 10.1145/2959100.2959120. URL: http://doi.acm.org/10.1145/2959100.2959120.

Kingma, Diederik P. and Jimmy Ba (2014). "Adam: A Method for Stochastic Optimization". In: *CoRR* abs/1412.6980. arXiv: 1412.6980. URL: http://arxiv.org/abs/1412.6980.

Kulesza, Alex and Ben Taskar (2012). “Determinantal Point Processes for Machine Learning”. In: *Foundations and Trends® in Machine Learning* 5.2–3, pp. 123–286. ISSN: 1935-8237. DOI: 10.1561/2200000044. URL: http://dx.doi.org/10.1561/2200000044.

Lei, Tao, Regina Barzilay, and Tommi S. Jaakkola (2016). “Rationalizing Neural Predictions”. In: *CoRR* abs/1606.04155. arXiv: 1606.04155. URL: http://arxiv.org/abs/1606.04155.

Linden, Greg, Brent Smith, and Jeremy York (2003). “Amazon.com recommendations: Item-to-item collaborative filtering”. In: *IEEE Internet Computing*.

Lops, Pasquale, Marco de Gemmis, and Giovanni Semeraro (2011). “Content-based Recommender Systems: State of the Art and Trends”. In: *Recommender Systems Handbook*. Ed. by Francesco Ricci et al. Boston, MA: Springer US, pp. 73–105. ISBN: 978-0-387-85820-3. DOI: 10.1007/978-0-387-85820-3_3. URL: https://doi.org/10.1007/978-0-387-85820-3_3.

Macchi, Odile (1975). “The coincidence approach to stochastic point processes”. In: *Advances in Applied Probability* 7.1, pp. 83–122. DOI: 10.1017/S0001867800040313.

McAuley, Julian, Rahul Pandey, and Jure Leskovec (2015a). “Inferring Networks of Substitutable and Complementary Products”. In: *CoRR* abs/1506.08839. arXiv: 1506.08839. URL: http://arxiv.org/abs/1506.08839.

McAuley, Julian et al. (2015b). “Image-based Recommendations on Styles and Substitutes”. In: *CoRR* abs/1506.04757. arXiv: 1506.04757. URL: http://arxiv.org/abs/1506.04757.

Metropolis, Nicholas and S. Ulam (1949). “The Monte Carlo Method”. In: *Journal of the American Statistical Association* 44.247, pp. 335–341. ISSN: 01621459. URL: http://www.jstor.org/stable/2280232.

Neumann, John von (1951). “Various Techniques Used in Connection with Random Digits”. In: *Notes by G E Forsythe, National Bureau of Standards Applied Math Series, 12 (1951) pp 36-38*.

Pappas, Nikolaos and Andrei Popescu-Belis (2017). “Multilingual Hierarchical Attention Networks for Document Classification”. In: *CoRR* abs/1707.00896. arXiv: 1707.00896. URL: http://arxiv.org/abs/1707.00896.

Paulus, Max Benedikt (2017). “Learning Determinantal Point Processes From Weak Supervision”.

Pennington, Jeffrey, Richard Socher, and Christopher D. Manning (2014). “GloVe: Global Vectors for Word Representation”. In: *Empirical Methods in Natural Language Processing (EMNLP)*, pp. 1532–1543. URL: http://www.aclweb.org/anthology/D14-1162.

Pero, Štefan and Tomáš Horváth (2013). “Opinion-Driven Matrix Factorization for Rating Prediction”. In: *User Modeling, Adaptation, and Personal-*

ization: 21th International Conference, UMAP 2013, Rome, Italy, June 10-14, 2013 Proceedings. Ed. by Sandra Carberry et al. Berlin, Heidelberg: Springer Berlin Heidelberg, pp. 1–13. ISBN: 978-3-642-38844-6. DOI: 10.1007/978-3-642-38844-6_1. URL: https://doi.org/10.1007/978-3-642-38844-6_1.

Pollack, J.B. (1987). "On connectionist models of natural language processing".

Ravanbakhsh, Siamak, Jeff Schneider, and Barnabas Poczos (2016). "Deep Learning with Sets and Point Clouds".

Ricci, Francesco, Lior Rokach, and Bracha Shapira (2011). "Introduction to Recommender Systems Handbook". In: *Recommender Systems Handbook*. Ed. by Francesco Ricci et al. Boston, MA: Springer US, pp. 1–35. ISBN: 978-0-387-85820-3. DOI: 10.1007/978-0-387-85820-3_1. URL: https://doi.org/10.1007/978-0-387-85820-3_1.

Rubinstein, Reuven (1999). "The Cross-Entropy Method for Combinatorial and Continuous Optimization". In: *Methodology And Computing In Applied Probability* 1.2, pp. 127–190. ISSN: 1573-7713. DOI: 10.1023/A:1010091220143. URL: https://doi.org/10.1023/A:1010091220143.

Tschiatschek, Sebastian, Josip Djolonga, and Andreas Krause (2016). "Learning Probabilistic Submodular Diversity Models Via Noise Contrastive Estimation". In: *Proceedings of the 19th International Conference on Artificial Intelligence and Statistics*. Ed. by Arthur Gretton and Christian C. Robert. Vol. 51. Proceedings of Machine Learning Research. Cadiz, Spain: PMLR, pp. 770–779. URL: http://proceedings.mlr.press/v51/tschiatschek16.html.

Vellido, Alfredo, José D. Martín-Guerrero, and Paulo J. G. Lisboa (2012). "Making machine learning models interpretable". In: *In Proc. European Symposium on Artificial Neural Networks, Computational Intelligence and Machine Learning*.

Wang, Z. et al. (2015). "Friendbook: A Semantic-Based Friend Recommendation System for Social Networks". In: *IEEE Transactions on Mobile Computing* 14.3, pp. 538–551. ISSN: 1536-1233. DOI: 10.1109/TMC.2014.2322373.

Wu, Yao et al. (2016). "CCCF: Improving Collaborative Filtering via Scalable User-Item Co-Clustering". In: *Proceedings of the Ninth ACM International Conference on Web Search and Data Mining*. WSDM '16. San Francisco, California, USA: ACM, pp. 73–82. ISBN: 978-1-4503-3716-8. DOI: 10.1145/2835776.2835836. URL: http://doi.acm.org/10.1145/2835776.2835836.

Zaheer, Manzil et al. (2017). "Deep Sets". In: *CoRR* abs/1703.06114. arXiv: 1703.06114. URL: http://arxiv.org/abs/1703.06114.

ETH

Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Declaration of originality

The signed declaration of originality is a component of every semester paper, Bachelor's thesis, Master's thesis and any other degree paper undertaken during the course of studies, including the respective electronic versions.

Lecturers may also require a declaration of originality for other written papers compiled for their courses.

I hereby confirm that I am the sole author of the written work here enclosed and that I have compiled it in my own words. Parts excepted are corrections of form and content by the supervisor.

Title of work (in block letters):

Rationalizing Recommender Systems

Authored by (in block letters):
For papers written by groups the names of all authors are required.

Name(s):	**First name(s):**
Bühler	Michael

With my signature I confirm that
- I have committed none of the forms of plagiarism described in the 'Citation etiquette' information sheet.
- I have documented all methods, data and processes truthfully.
- I have not manipulated any data.
- I have mentioned all persons who were significant facilitators of the work.

I am aware that the work may be screened electronically for plagiarism.

Place, date	**Signature(s)**
Zürich, 29.11.2017	M. Bühler

For papers written by groups the names of all authors are required. Their signatures collectively guarantee the entire content of the written paper.