Improving Graph Neural Networks for Multimodal Data

(Usprawnienie Modeli Grafowych Sieci Neuronowych do Działania z Danymi Wielomodalnymi)

Kacper Puchalski

Praca magisterska

Promotor: dr hab. Piotr Wnuk-Lipiński

Uniwersytet Wrocławski
Wydział Matematyki i Informatyki
Instytut Informatyki

3 stycznia 2025

Abstract

These days, session based recommendation is common across many e-commerce platforms and other websites where user behind the session is anonymous, yet it aims to predict his further interactions. There have already been many studies trying to solve the problem, one particular class of them models each session as graph and applies a Graph Neural Network (GNN) to capture session's preference and return list of recommendations. In this thesis we propose on how to improve efficiency of those, by identifying information regarding multimodal nature of sessions. With use of Gaussian Mixtures we analyse user's behaviour and provide insight into what we assume is multimodality in case of session data. Suggested solution introduces method of applying extracted knowledge into the GNN at its core by modifying adjacency matrix of each session graph, greatly improving quality of recommendations.

Obecnie rekomendacje oparte na sesji są popularne w sklepach internetowych jak i na innych platformach jak aplikacje streamingowe. Problem polega na przewidzeniu przyszłych interakcji anonimowego użytkownika stojącego za sesją. Istnieje już wiele prac starających się rozwiązać ten problem, z których część oparta jest na Grafowych Sieciach Neuronowych (GNN), gdzie każda sesja jest przedstawiana jako osobny graf. W tej pracy proponujemy jak poprawić ich działanie, identyfikując informacje o wielomodalnej naturze sesji. Za pomocą Mikstur Gausowskich analizujemy zachowanie użytkowników i sugerujemy czym jest wielomodalność w przypadku sesji. Przedstawiona metoda pokazuje jak możemy wykorzystać tą wiedzę do poprawy rekomendacji, poprzez modyfikację macierzy sąsiedztwa wykorzystywanej przez GNN będącym sercem systemu rekomendującego.

Contents

Chapter 1

Introduction

Raise in use of Recommender Systems over the last decade was inevitable. With growing number of online stores, streaming services and consumption of short-form content, correct recommendation of next product is now more important than ever. Most Recommender Systems work based on user's sequence of products, those might be items bought or viewed, songs listened, movies watched and so on, derived from which further ones are suggested, usually in some form of a list. If the system is working properly, that means recommendations are good enough, the user is likely to spend more time in the app (or website), therefore raisng potential profit for the buisness behind the application.

In most scenarios we have loads of information about both products and users. Use of platforms like Netflix requires completing registration process, that meaning we get to know user's gender, age, nationality etc. Same goes for products, movies in that case, are well described, belong to some genres, have specific duration, cast, narrative perspective and more. Possession of such detailed data greatly increases chances to personalize, therefore improve, predictions of a system. We would call that category a sequence prediction problem.

On the other hand, user's data is not always available. An example of such would be someone shopping online without creating the account. In that scenario, we have to base our predictions solely on current session. Lack of information about user's previous preferences makes our task much harder, especially when the session is short. A Recommendation System tackling this problem is called Session Based. With limited information, capturing user interest requires more finesse than in case of fully described sequence. What is even more challenging, in session based problem we usually work with *clicks* rather than more serious interaction (buys or opinions), therefore no knowledge if item was liked or disliked by the user is available. We discuss present solutions in Chapter 2.

Sequence Based Recommendation is often misunderstood with Session Based Recommendation, that is why we need to classify the differences in detail. Work

presented in [14] provides a great survey on the topic, following are the most important keypoints. Sequence data is a list of following user-item interactions, ordered by timestamps, over undefined time period. That is, sequence for one user could last for a week, whilst for another over a year, there is no clear boundary. Tracking for such long time is possible thanks to user logging into the same account every time he uses a service, rather than being anonymous. Session is defined in similar manner, although being much shorter in terms of items viewed and timespan. We assume it is ordered as well. In most cases user is not know, so session is representing a single visit to a service, once a user reconnects a new session is established. If the user is know, as in Figure 1.1a, we can split the sequence into separate sessions by periods of inactivity. *Diginetica* utilizes it, setting one hour of inactivity as splitting criterion. Both Sequence and Session Based RS aim to recommend unknown part of the series, either a single interaction or set of them, based on historical preferences. In this thesis we attend task of next item recommendation, which corresponds to edge prediction in a graph.

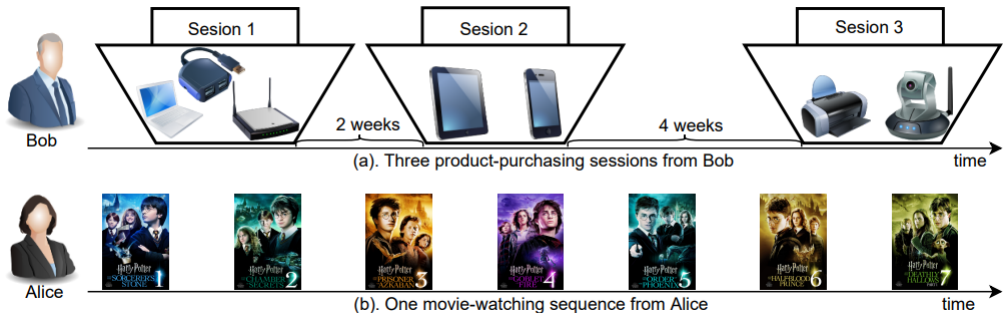

Figure 1.1: Session vs Sequence data. Note significant time window between each movie interaction, while sessions are spanning over (usually) few hours. Schema from [14].

The main goal of this thesis is to improve performance of Session Based Recommender System, mainly on multimodal sessions. To make it precise, we do not work with data of different types (ie. parsing audio, image and text in single model), rather we assume single type, possibly coming from mixture of multiple distributions. We think of multimodal sessions as those when user's focus over searched items changes rapidly, or in a weird, what seems to be an unpredictable manner. We indetify those distributions with use of Expectation Maximization algorithm [2].

In the first part of our work we compare notion of multi-categorical sessions with multimodality. While doing so we propose to apply Gaussian Mixture Model on session data, with that introducing classification method of sessions rather than products. We use resulting partition to identify and put more focus onto types of session where original model struggles.

Later in the thesis we present main contribution of our work. We provide an algorithm that generates edge features in a session graph for any pair of two items.

To accomplish this we do not rely on item metadata, but use latent product embeddings which can be obtained merely from previously trained model. Moreover, we suggest new approach to define type of each product that is not dependent on information regarding item categories. Combining that with simple randomization techniques, we create a new method of data augmentation for GNNs on Session Based Recommendation task.

In Chapter 2 we address related work in the field of Session Based Recommender Systems, Chapter 3 clarifies the problem at hand while Chapter 4 presents datasets and algorithms used for experiments. Chapter 5 introduces naive and relatively simple approaches one could take to improve model performance on multimodal sessions. Finally, in Chapter 6 we provide a novel method of data augmentation for Graph Neural Networks, that is augmenting an adjacency matrix. Chapter 7 presents results for SRGNN [16] and TAGNN [19] as base models, while Chapter 8 concludes the thesis and provides ideas for further work.

Chapter 2

Related Work

2.1 Session Based Recommender Systems

We present short overview of most common methods used for Session Based Recommendation. Our work focuses solely on GNN models, for which we give further background in Chapter 4.

2.1.1 Basic Algorithms

Starting from the least complex solutions for Session Based Recommendation, we need to mention works based on Pattern Mining. Simply speaking, algorithms in question process available data looking for repeating sequences of items. Work of [12] follows those principles. Given training set of sessions sequences per user, we search for patterns of sessions occurring one after another. During evaluation, for single session we look for pattern in which such session occurs and recommend items from following sessions. That is, supposing there exists a pattern $P_i = \{\cdots, s_K, s_L, \cdots\}$, we select items from session s_L for recommendations for session s_K.

Another simple yet successful approach is to use K Nearest Neighbours method with slight modifications. We start with embedding each item as a binary vector of dimension equal to number of witnessed sessions, where every element is labeled 1 if an item belongs to corresponding session, 0 otherwise. With that we may apply for example cosine similarity as in [7], to look for the neighborhood of a product for which we want a list of recommendations.

Last but not least is class of solutions adopting factorization models. Observed matrix of user-item interactions is factorized so latent representation of items are available, like in [9]. With those we can estimate new interactions that is return recommendations for previously unseen sessions. Major drawback is that information about user's is not always available, which means every session needs to be treated as separate user. This results in sparse matrix to begin with, on which factorization

methods struggle.

2.1.2 Deep Learning Models

One of the most well known algorithms for Session Based Recommendation based on Deep Learning is *GRU4Rec* [4]. The name speaks for itself, as it is making use of Gated Recurrent Unit for modelling session context. During interference, at *Kth* step model takes as input together with previous hidden state, an embedding of *Kth* item from a sequence. After passing over whole session, obtained hidden state serves as a vector presenting session context from which recommendation for next product are drawn.

Moving away from RNNs, approaches grounded on generative models focus on using session context to generate next interactions rather than guessing only next clicks. More natural usage is to complete current session, by generating potential session [15] that fully encapsulates user's current behaviour, therefore maximize its chances to satisfy his intentions. Different yet still worth mention are solutions based on Reinforcment Learning. Here model interprets user actions, depending if he interacted with one of recommended items and via trial and error learns his preferences [21]. For better performance this approach needs more data regarding user-item interactions than just clicks, as RL system excels with more information about possible actions.

2.1.3 Novel Techniques

Generally speaking, Session Based Recommendation is very similar to task of predicting next word in Natural Language Processing, as both focus on returning next most probable token after analysing some sequence. Therefore, using *transformers* as base for recommendation system should result in great success. Work of [8] focuses on that, introducing *Sequential Masked Modeling* with use of encoder only transformer architecture.

Recommendation Systems often make use of *Contrastive Learning*, [11] improves performance of currently present solutions. They present *Self-Contrastive Learning*, simplifying its application and eliminating previously necessary construction of positive and negative samples.

2.2 Data Augmentation in Graph Neural Networks

Data Augmentation is known technique to improve generalization used across field of Machine Learning algorithms. It is the most valuable in case of Deep Learning solutions, as with expressive power of the model it's tendency to overfit raises as well.

When it comes to graphs, augmentation usually focuses on creating an augmented version of original graph, that changes each epoch or in some other manner. Those operations modify the graph slightly, while conserving major structure. Many are described in [3], below we recall some of the most important ones.

Edge modification, most often either adding or removing edges from input graph, is a widely adopted method. Strictly speaking, it takes an adjacency matrix and removes or adds entries from it. This could be done either randomly or with help of precalculated probability distribution based on the input graph. If the edges contain additional information, then instead of removing whole edge only features might be influenced. That would work more like classic feature augmentation ie. adding gaussian noise. While beneficial, it does not apply to instance where no extra information about input graph is available.

Similarly node modification might be a useful method. It brings more drastic implications on base graph, as any edge attached to the removed node is deleted as well. Furthermore, addition of a new node brings with it creation of extra edges, as isolated vertices usually do not bring much information. Once again, this process is usually somewhat random, based on predefined probability distribution.

Hopefully, not every classic Deep Learning method is of no use. Dropout [13] finds same applications as in other algorithms. No matter the wider spectrum, GNNs base on simple message propagation mechanism which reassembles work of standard hidden or convolutional layer, in which dropout might be applied.

2.2.1 Manipulating Adjacency Matrix

Introduction of new structure of adjacency matrix is another way to interfere with graph structure, without modifying it directly. Some of the algorithms to do so are described in [20]. As the matrix is usually binary (discrete) expanding it into continuos space should greatly change behaviours of most GNN models.

Learning new graph structure which corresponds to substitution of original adjacency matrix, is an idea given in [5]. They introduce second loss function based solely on difference between original and new, generated graph. With minimization of that term, model is able to represent augmented version of input graph, that is more appropriate for underlying task.

2.3 Augmentation in Recommender Systems

In case of sessions graphs we cannot really apply same principles, as it would greatly change nature of data. An example of such is given in Figure 2.1. Work described in [18] proposes some ideas regarding augmenting sessions. Method similar to *Noise Injection*, that is injecting more or less random item into the session, is used and

described in Chapter 5.
For the sake of notation, we will refer to *sequence* as *session* for the rest of this paper, as both focus on next item prediction.

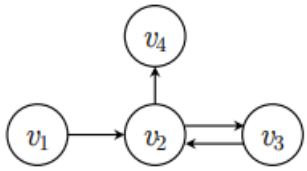

Figure 2.1: Graph representing 5 click session $V_1 \rightarrow V_2 \rightarrow V_3 \rightarrow V_2 \rightarrow V_4$. If node V_2 was removed, the 5-click session would break into three single clicks representing three separate sessions, or a 3-click session $V_1 \rightarrow V_3 \rightarrow V_4$ still loosing relatively much information. Same problem preserves with edge removal. Therefore different manner of augmentation is necessary.

Chapter 3

Problem Definition

In Session Based Recommender System we aim to predict next user-item interaction using only current session history, without any access to user's historical data. Let set $I = \{i_1, i_2, ..., i_n\}$ present all unique items in service, of which every one belongs to at least single session. Anonymous session is represented as a list of items $s = [i_{s,1}, i_{s,2}, \cdots, i_{s,l}]$ of length l, ordered chronologically. The goal here is to include next interaction $i_{s,l+1}$ in a list of recommended items for that session. Our session based recommendation model taking as input sequence s, outputs probabilities $\hat{y}$ for every $item \in I$. We treat those as items scores for session s, out of which items with K highest corresponding values are selected for returned recommendation list.

3.1 Training Process

Most often session based recommendation may be interpreted as multiclass classification problem, where a *label* for each session is the *Id* of ground truth next item interaction. For every training sample (session) system returns score vector $\hat{y}$ which is compared to one-hot encoding y of target product. Loss function is applied on both vectors $L(y, \hat{y})$ in order to calculate the gradients and apply back propagation.

Table 3.1: Most common notations in Session Based Recommender Systems

Notation	*Description*
I	Set of items (or vertices in graph) $I = \{i_1, i_2, ..., i_n\}$
S	Set of anomynous sessions $S = \{s_1, s_2, ..., s_m\}$
s_i	Single finite session $s_i = [i_{s1}, i_{s2}, \cdots]$, repetitions are possible
G_{s_i}	Session graph of session s_i
A_{s_i}	Adjacency matrix of session graph for session s_i of dimension $len(s_i) \times 2len(s_i)$
$\hat{y}$	Score vector of dimension $\|I\|$ returned by model for single input session

Therefore, model is trained to maximize precision of yielded recommendations. In our work we aim to improve this process and with it quality of recommendations, by introducing information about session's multimodality into model input.

3.2 Evaluation

Most algorithms in the field compare themselves with help of metrics @20. This means model in question returns list of recommendations consisting of 20 items. As per usual, we do the same and compare against two highly expressive metrics

1. **Precision@20** - Binary metric, if target item is included in list of 20 best recommendations returned for given session. Used in every paper up to date, very intuitive.
2. **Mean Reciprocal Rank @20** - Position of target amongst list of 20 recommended items. Represents how close to the top of recommendations it is, higher is better. If target was not included (false prediction), value is 0.

Chapter 4

Methodology

4.1 Algorithms

SRGNN

Sessions-based Recommendation with Graph Neural Networks [16] at the time of release achieved SOTA level results. With it's relatively simple construction, it leaves room for further improvements and modification. Plasticity of the model makes it very easy to experiment with, while being resilient enough not to crumble in case of wrong assumptions. That is why this architecture will serve us as a black box in most experiments described in this thesis. Below is a short description of this model.

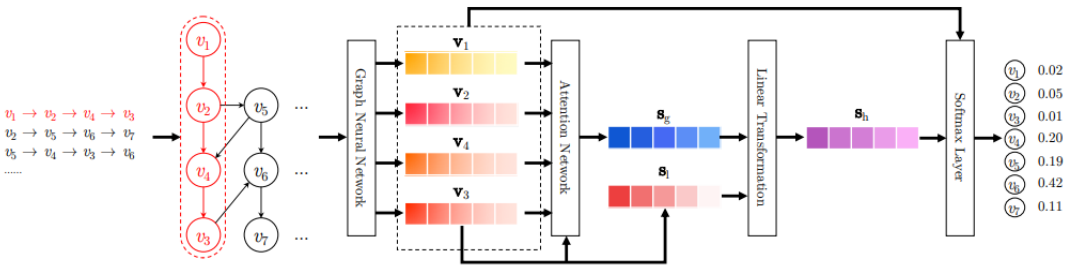

Figure 4.1: Brief workflow of the SRGNN method. Modelling every session as a graph, proceeded by a gated GNN followed with attention layer. Session embedding is achieved by combination of nodes embeddings, while final prediction vector is result of dot product calculated with raw item embeddings. Diagram from [16].

Basis of SRGNN is construction of session graphs. Each session S may be represented as directed graph on (I_S, E_S) where $I_S \subseteq I$ and edges $\forall_{k=1}^{l-1}(i_k, i_{k+1}) \in E_S$ corresponding to user interacting with item $k+1$ after k. As some items might repeat in a single session, adjacency matrix of resulting graph is normalized by every node's outdegree. Graph Neural Network interacting with the graph projects every item in session into latent vector of predetermined dimension. With those at hand, a whole

session is presented as a combination of them.

$$\begin{aligned} m_k^{i+1} &= \sum_{v \in N(k)} Message(h_k^i, h_v^i) \\ h_k^{i+1} &= Update(h_k^i, m_k^{i+1}) \end{aligned} \tag{4.1}$$

The GNN used for information propagation is a variant of vanilla graph neural network 4.1, namely Gated Graph Sequence Neural Network [6]. Information propagation happens in similar manner to normal GNN, although with introduction of *update* and *reset* gates which reassembles GRU-like updates introduced in [1]. Final node vector is obtained after fully propagating through graph neighbourhood, limited to a prespecified depth.

Session embedding is a combination of user's long and short term preference, calculated from latent vectors yielded by GNN in previous step. The global embedding of a sequence is an aggregation of all items present, preceded with soft attention as in 4.2, where **q,** W_1**,** W_2**, bias,** W_3 are all learnable parameters of the model.

$$\begin{aligned} s_g &= \sum_{i=1}^{l} \alpha_i \cdot v_i \\ \alpha_i &= q^T \sigma(W_1 \cdot v_l + W_2 \cdot v_i + bias) \end{aligned} \tag{4.2}$$

Further, final session embedding is computed as a concatenation of last item clicked, as it usually has the most correspondence to user's current interest, and prementioned global embedding, such that $s = W_3 \cdot [v_l : s_g]$. Score, or chances, of item i_j being next item in session, is simply a dot product of item raw embedding and resulted session embedding $score_i = s^T \cdot i_j$. Items with top K scores are returned as list of recommendations for input session.

Item interaction probabilities are achieved by applying *softmax* on item scores $\hat{y} = softmax(scores)$. Loss function is a simple Cross-Entropy 4.3 over two vectors, $\hat{y}$ and one-hot y of true target item, where $y_{real} = 1$.

In conclusion, given training sample in form (s_j, i_j) where s_j is a sequence representing items and i_j the next clicked product, system returns probabilities $\hat{y}$ of any item from I occurring as the next click of s_j. This is then fed to loss function, based on which gradients are computed so that backpropagation is possible. Training dataset is split into actual training data and disjoint validation set. Every epoch order of train samples is reshuffled, which are then packed into batches to improve convergence. Process continues for multiple epochs until stopping criteria, based on value of loss function averaged over validation samples, is met. That finishes description of SRGNN.

$$L(y, \hat{y}) = -\sum_{i=1}^{n} y_i \cdot log(\hat{y_i}) + (1 - y_i) \cdot log(1 - \hat{y_i}) \tag{4.3}$$

TAGNN

Major upgrade based directly on SRGNN, Target Attentive Graph Neural Network [19], works in similar manner as it was designed with same purpose in mind. Difference takes place in calculation of session embedding. SRGNN took only two components into consideration, while TAGNN introduces a third one with the other being the same 4.4.

$$\begin{aligned} s_t &= W_3 \cdot [s_{target,t} : s_{global} : s_{local}] \\ \hat{z}_t &= s_t^T \cdot v_t \end{aligned} \tag{4.4}$$

The $s_{target,t}$ stands for attention weight calculated for every pair between item t (which is a potential *target*) in dataset and an item in current session. Please note that s_t is different for every item, and final scores are computed with use of it, instead of global and local embeddings which were the same for all target items.

4.2 Data

Basis for every session based dataset are the sessions, so no matter the source, core of the data is given in form *(SessionId, ItemId, timestamp)*. With that in mind, we split the dataset intro train&test parts by *SessionId*, that means taking whole sessions into account so we omit danger of information leakage during testing, meaning testing on latter half of each session while training over its beginning. As every session $S = [i_{s,1}, i_{s,2}, ..., i_{s,l}]$ consists of l items, we may generate from it more train (or test) samples by taking some prefix $[i_{s,1}, ..., i_{s,k}]$ where $k < l$ and $i_t = i_{s,k+1}$ as target and interpreting that as a new sample. An example is shown in Table 4.1 Such procedure further enriches our dataset, however not without impact.

Creating sessions based on some prefix leads to tail heavy distribution, regarding session length. By this manner, for every session present in raw data, we naturally get extra samples of length 1, 2, ..., l. Potential trouble might arise, if a rare item would be included at early stage in such session. Naturally, we would magnify its importance across dataset. However, other items would be multiplied in occurrences as well, so proportion of sessions in which such infrequent product occurs would still be relatively small. In case of singular items, we omit the problem by preprocessing data as described later in this section.

What is more, this causes longer sessions to be more common across dataset. Luckily, as those rarely occur normally, this balances the data a little bit while still

Table 4.1: Example of creating train (or test) samples from single recorded session of length 6. Items listed in square brackets are data fed as an input to our model, whilst *target* is an item we aim to include in returned recommendation list, preferably at the first place.

S=$[i_1, i_2, i_3, i_4, i_5, i_6]$	$[i_1, i_2, i_3, i_4, i_5] \xrightarrow{target} i_6$
	$[i_1, i_2, i_3, i_4] \xrightarrow{target} i_5$
	$[i_1, i_2, i_3] \xrightarrow{target} i_4$
	$[i_1, i_2] \xrightarrow{target} i_3$
	$[i_1] \xrightarrow{target} i_2$

preserving dominance of short sessions. First recommendations are usually the most crucial in order to keep user interested, as well as hardest to guess correctly given lack of information about user's preferences. Therefore, we do not bother with that as putting a little bit more emphasis on lasting session should help with tackling overfitting the model on short-term ones.

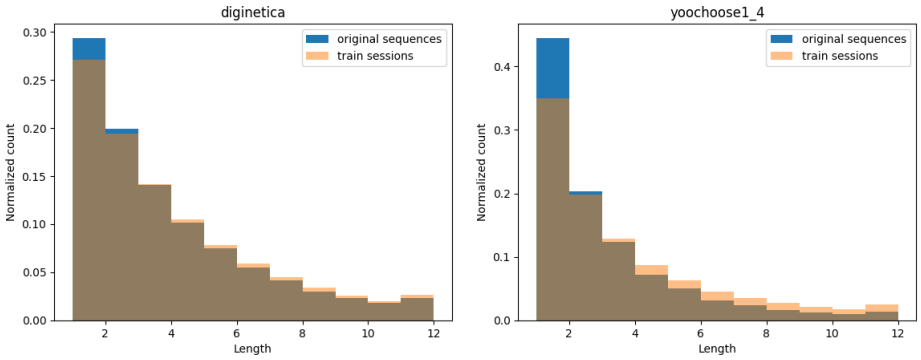

Figure 4.2: Change in length distribution after splitting original sessions into prefix-like samples. Drift (proportional to maximal length of original sequence) could be observed, due to few (under 10%) longer sessions largely increasing that part of data. A single original sequence of length= 60 would spawn 50 more samples of length ≥ 10, but just a single one of length= 1.

We benchmark our results on two most common datasets in the field, **Diginetica** and **YooChoose**. Both come from real world e-commerce websites, which guarantees credibility of the data. To make sure we did not overfit our hyperparameters on those two, we selected third dataset. We choose a small subset of training data from **Otto Rec-Sys** challange dataset and split it into train and test sets taking first 3 weeks of sessions into train set and last week as test, similarly as with YooChoose.

For all described datasets, unless specified otherwise, we follow process described in [16]. We filter out sessions of length one, as recommendation for those would reassemble *cold start* problem [17]. Furthermore, all items that occur less than 5 times are discarded.

Table 4.2: Summary statistics about datasets. Otto Rec-Sys does not contain metadata regarding item categories.

Statistic	YooChoose 1/64	Diginetica	Otto Rec-Sys
# of clicks	557'248	982'961	236'850
# of training sessions	369'859	719'470	134'573
# of test sessions	55'898	60'858	62'687
# of items	37'483	43'097	18'102
Average length	6.16	5.12	5.98
# of categories	336	995	$\sim$
Average # of categories per session	3.74	1.38	$\sim$

Problem we are addressing in this thesis is well exemplified by following session from *YooChoose*. Six distinct items respectively of category $[3, 3, 3, 3, 5, 5]$. Swiftly adjusting recommendations from category 3 onto category 5 is troublesome for most models, yet it might be improved with help of knowledge regarding multimodality.

4.3 Gaussian Mixtures and User Interest

Tackling multimodality is the core of our work, therefore we shall explain what we aim to solve. Firstly, we assume that every session is a group of items sampled from some unknown probability distribution, likely to be determined by underlying user (his preference). As long as session would focus on only one type of products, we do not have to bother at all, that means we consider it to be unimodal or *normal* ie. not multimodal. However, a nature of a session may change rapidly in e-commerce, when for example user is collecting presents for his close ones switching between their hobbies.

While changing interest in rather steady manner should not be a problem, as thanks to GRU-like structure of the model we would quickly get rid of no longer relevant information, user constantly swapping between two or more types of products is much more troublesome. If that is the case, the session reported could look like sequence which was sampled from two (or more) unrelated distributions over the same latent space, or rather a single multimodal probability distribution.

We test if probability function over products hidden space could be modelled with a mixture of gaussian models, such that every sequence is sampling consecutive items from it. We categorize items under the mixture with use of Expectation Maximization algorithm [2]. We start with experimentally selecting k, number of components, then fitting μ of every gaussoid as random data point from item (or session) embeddings latent space. Later we run EM for numerous iterations until convergence. Every iteration may be split into two steps 4.5, in first one *Expecta-*

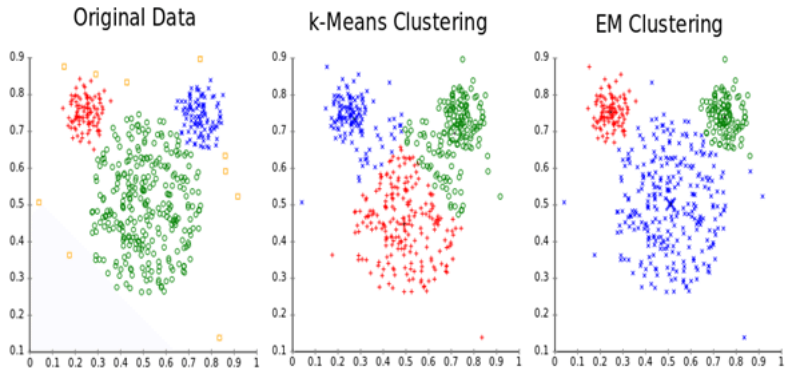

Figure 4.3: Comparison of clustering with K-Means and Expectation Maximization on *Mouse* dataset, data is representing known comic figure - Mickey Mouse. Gaussian Mixtures clearly give better separation.

tion, we estimate probabilities of every item x coming from gaussoid z given current parameters (θ, μ, σ), via log likelihood function. Secondly, in *Maximization* step, we update parameters to maximize log likelihood function representing correctness of current assignment. Such approach helps us show that indeed, session multimodality could have some dependency with the mixture.

$$\begin{aligned} w_j^i &= \frac{P(x^i|z^i;\theta,\mu,\sigma)\cdot P(z^i=j)}{\sum_{l=1}^{k} P(x^i|z^i=l)\cdot P(z^i=l)} \\ \theta_j &= \frac{1}{n}\sum_{i=1} nw_j^i \\ \mu_j &= \frac{\sum_{i=1} nw_j^i \cdot x^i}{\sum_{i=1} nw_j^i} \\ l(\mu,\sigma,\theta) &= \sum_{i=1}^{n} log \sum_{j=1}^{k} P(x^i;\mu_j,\sigma_j,\theta_j) \end{aligned} \tag{4.5}$$

Chapter 5

Preliminary Studies

In this chapter we discuss how can we classify session as *multimodal*. The most tempting method, is to just state that any session with more than single unique category of items is multimodal. In wide spectrum of things, this might be correct if the categories would be very general and well separated, like clothes, agd, sports, etc. We investigate the matter further and provide a different understanding.

In the first section, we dive into available data and compare quality of recommendations through the lens of item categories. To do so, we introduce minor change to data preprocessing to serve as an extra filter. Later we check how elementary data augmentation influences our system, focusing mainly on number of various categories in single session. Thirdly, we put our attention on improving initialization of item embeddings to introduce additional knowledge into the model, before starting training process. Finally we propose an approach to cluster sessions based on their embedding. With such separation of data, we create an ensemble of finetuned versions of our model which serves as improved version of recommender system.

5.1 Investigating Categories

To begin with, we created our own training data with some deviations from process described in [16]. For *Diginetica* procedure was the same, however for *Yoochoose* we discard those clicks labeled with *special* category, indicating that the product was on special offer. This extra filtration will leave us with sequences consisting of interaction with better separated categories, which should more clearly represent user's intent. Rest of the preprocessing is done in the same way.

To distinguish unimodal and multimodal sessions we need to compare them between each other, therefore we need some kind of session embedding. Luckily, our model architecture already provides one. We calculate it by passing input data all the way through the model skipping the last part where dot product is calculated between the session vector and item candidates. That is, session embedding is attained in the

same way as described in Section **Generating Session Embedding** of Chapter 4 in Figure 4.1.

Now we investigate model's results through the lens of categories.

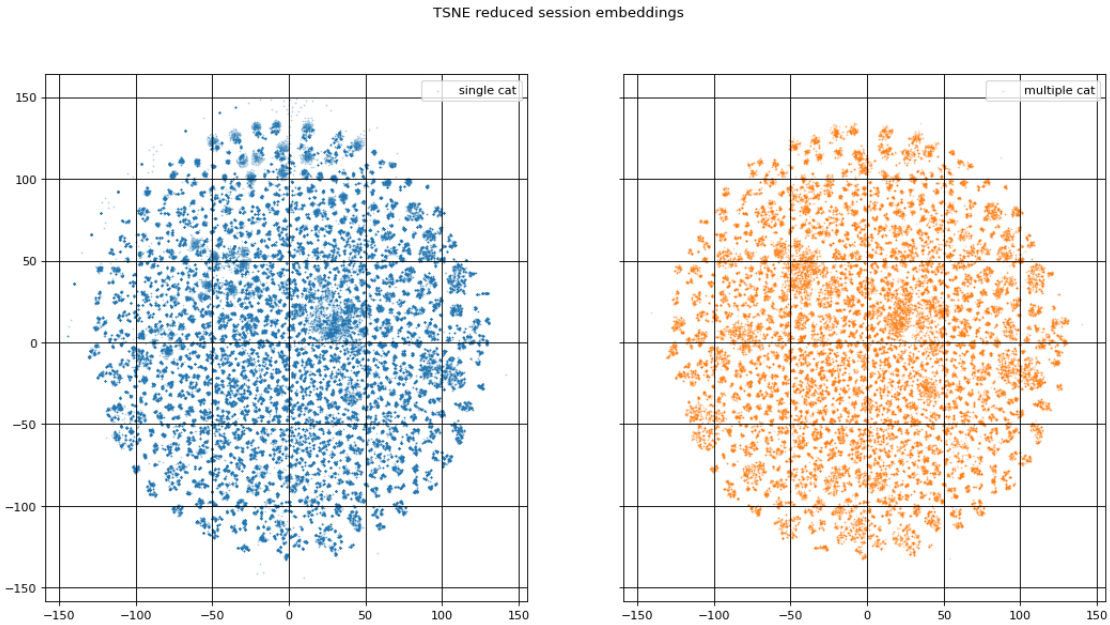

Figure 5.1: Comparison of embeddings for sessions consisting of only single category and those of multiple, for *Yoochoose* dataset with custom preprocessing. Data is coming from test sessions, based on already trained model on dimension= 100, projected with TSNE into 2d space.

Analysing model metrics on sessions with single versus multiple categories does seem to show a contrast regarding model performance, see Table 5.1. Most likely reason is substantial difference in number of multi-categorical sessions within datasets. *Yoochoose* averages over 3.5 distinct item categories per session, which even when discarding the *special* category, is more than twice as much as in *Diginetica*. This corresponds to amount of training sequences with either single or multiple categories, where the latter is minority in case of *Diginetica*. Therefore, the disproportion in evaluation scores of recommender system is significantly higher than with *Yoochoose*.

On the other hand, distribution of session embeddings (presented on Figure 5.1) for same partition does not show such visible difference. Same goes for comparison of session embeddings for correctly guessed recommendations and otherwise (Figure 5.2).

5.2 Naive Modifications

We experimented with graph augmentation technique known as *vertex injection*, applied to every session separately. Scrutinizing, we (randomly, with $p = 0.5$) inject

Yoochoose Custom	Overall	One Category	Multiple Categories
hit (%)	55.87	56.27	55.15
mrr (%)	25.27	26.06	23.86
Diginetica			
hit (%)	50.71	52.87	46.80
mrr (%)	17.76	19.61	14.39

Table 5.1: Results grouped by number of categories in a session. Decrease in model scores is clearly visible, which might point to interpreting multimodality as simply presence of multiple categories.

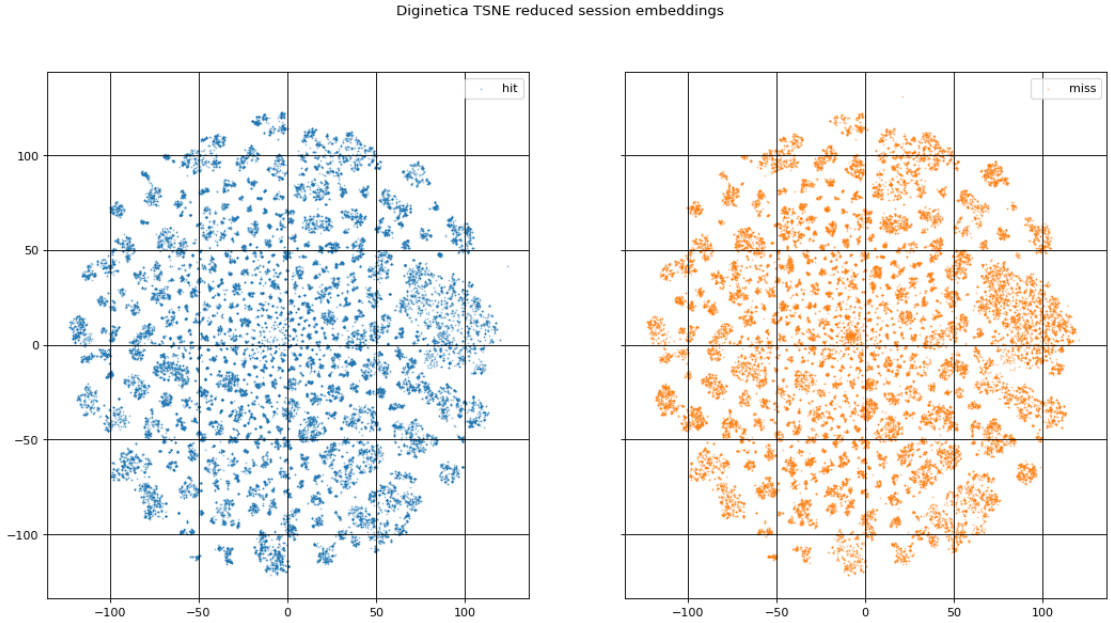

Figure 5.2: Comparison of embeddings for sessions with correct recommendations and opposite, for *Diginetica* dataset. Lack of noticeable differences.

an extra item at random position for every session that is uni-categorical. Item selected is of different category than those in session. The goal is to increase number of multi-categorical sessions in dataset.

Model trained on such data achieved much worse metrics than the base one, with $Hit = 44.41\%$ and $MRR = 20.27\%$ for *Yoochoose*. This is a decrease of over 10 percentage points on Hit and 5 on MRR. Such result is fully understandable, as most of the sessions are short and adding to it an extra random item introduces lots of noise and blurs relevant information.

5.3 Improving Initial Embeddings

Our model is highly dependent on its item embeddings, therefore initializing them with something else than normal distribution should improve resulting metrics. Session is nothing more than a sequence of undefined length, which strongly reassembles sequences of tokens found in NLP tasks. Noticing that, we decided to try pretraining item embeddings with separate module which treats items just like words in text corpus. We used Word2Vec [10] to calculate them, given its simplicity and low computational requirements. Later we use them as initial embeddings for main GNN model. Resulting difference in training process is shown in Figure 5.3.

Despite smaller train loss for first few batches, model without help of *W2V* quickly achieves lower loss than other two. Most likely assignment of items into latent space returned by *W2V* is significantly different than one needed to solve our recommendation task. Even with a relatively high learning rate at the beginning of training, precalculated embeddings take much more time to be adapted by the underlying GNN, while randomly generated ones can be swiftly tuned.

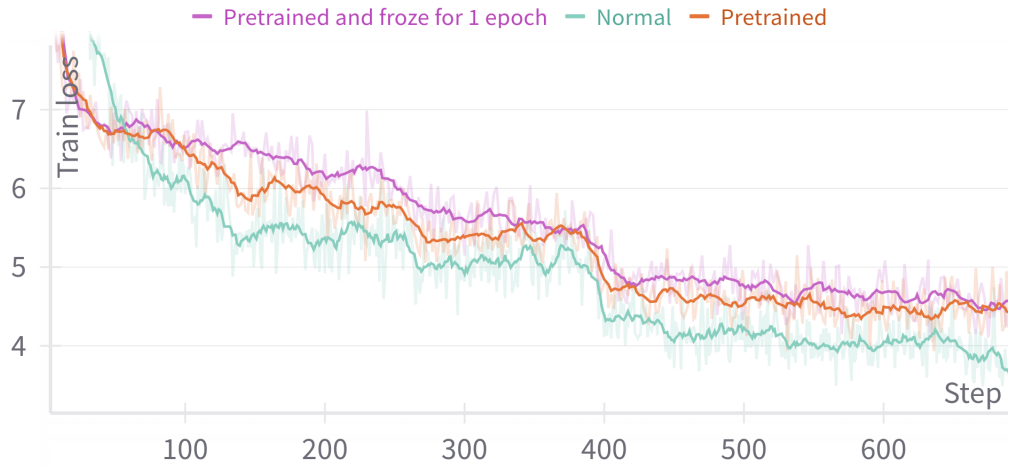

Figure 5.3: Comparison of train loss given normaly initialized item embeddings, precalculated with Word2Vector and optionally frozen for first epoch (130 steps) of training.

5.4 Gaussian Clustering

We are not sure if one can categorize sessions as multimodal with use of only product categories, so we will use session embeddings. Such representation yields us vectors in continuous space of prespecified dimension, so we can apply unsupervised clustering methods. With those, we want to group sessions into two separate clusters, either multimodal or not. We experimented with *k-Means* and *Expectation Maximization* algorithms. Comparison between them are shown in Figure 5.4. While *k-Means*

partition does seem more reasonable, keep in mind we are trying to improve on categorization achieved by original product categories with probably similar partitioning. Labelling achieved by *GMM* has up to 64% compatibility with that of single vs multi-categorical session. Moreover, as we increase number of gaussoids, the clusterization becomes more clear with some clusters denoising data and rest giving rather good separation.

5.4.1 Switch Case

Thanks to session classification we could run relatively simple experiment. Categorizing every session, we would then create new model for every cluster. That is, splitting both training and testing set into separate sets for each group, then train new model on data from single cluster only.

Describing the process in detail, we start with training single SRGNN model, keeping validation set unseen. Once training terminates, we keep the sessions from validation set and move on to training the *Gaussian Mixture Model*. Starting with calculating session embeddings on just trained GNN, we get set of vectors corresponding to validation data. We use control dataset here, in order to avoid potential overfitting that could influence our *GMM*. Next we apply EM [2] algorithm on said group of vectors (points in d-dimensional space) to get our clustering. Algorithm 1 further clarifies the process of splitting data into new train and test sets.

Algorithm 1 Splitting data into separate session clusters.

Require: Train, Val, Test sets, $\#clusters$
  GlobalModel ← TrainModel(Train, Val)
  ValSessionEmb ← CalculateSessionEmbeddings(Val, GlobalModel)
  GMModel ← TrainGMM(ValSessionEmb, # clusters)
  for $session \in$ Train **do**
    idx ← $Label(session$, GMModel)
    $train_{idx}.append(session)$
  for $session \in Test$ **do**
    idx ← $Label($session, GMModel)
    $test_{idx}.append(session)$
  return $(train_1, test_1), \cdots, (train_{\#cluster}, test_{\#clusters})$

Once the clustering is done, we train new Recommender System on every cluster separately. This is simply done by finetuning the model on which the clustering was calculated in the first place. Thanks to that, together with tweaking the hyperparameters, we do not lose information but only improve on subset of session whilst avoiding overfitting.

With that we can create an ensemble structure that applies correct model to each class. During evaluation, for each session we calculate its embedding with

hit difference (%)	mrr difference (%)	# sessions
-0.6	-1.15	522
+0.73	+1.62	433
+0.52	+0.83	2877
+0.14	-0.17	603
-0.28	-0.76	655
+0.62	+0.8	500
-0.16	+1.5	601
+0.68	-0.62	648
+2.05	**+1.65**	**6839**

Table 5.2: Difference in hit and mrr on *Diginetica* test data, for Switch Case approach. For comparison we selected clusters where original model metrics were below average. Positive value means finetuned version achieved higher score. Bottom row summarizes the benefit. Last column represents number of test sessions that belong to respective cluster.

GlobalModel, categorize the embedding with *GaussianMixtureModel* and finally, run the predictions with model associated with that class only, as is shown in Algorithm 2.

Algorithm 2 Prediction with use of cluster models.

Require: single session
SessionEmb ← CalculateSessionEmbeddings(session, GlobalModel)
SesssionCluster ← Label(session, GMModel)
return $model_{SesssionCluster}.recommend(session)$

Results are shown in Table 5.2. Surprisingly, some finetuned models achieve worse performance than original model, most likely due to overfitting as corresponding clusters are relatively small. Another phenomenon is that some clusters increased in *Hit* while simultaneously decreased in *MRR* or vice versa. Nonetheless improvement is notable and promotes use of Gaussian Mixtures. However, due to it's extra computational demands, this solution is rather impractical in real-world applications.

Finishing the chapter, we have shown that baseline SRGNN indeed struggles in case of multi-categorical sessions in comparison to uni-categorical ones. Moreover, basic augmentation technique with goal to improve model performance on sessions with more than one product category does not work, therefore another solution is necessary. Our algorithm described in Section 5.4.1 represents straightforward way of solving the problem. We managed to group sessions with help of *GMM* and improve on those clusters, where original model produced underwhelming recommendations.

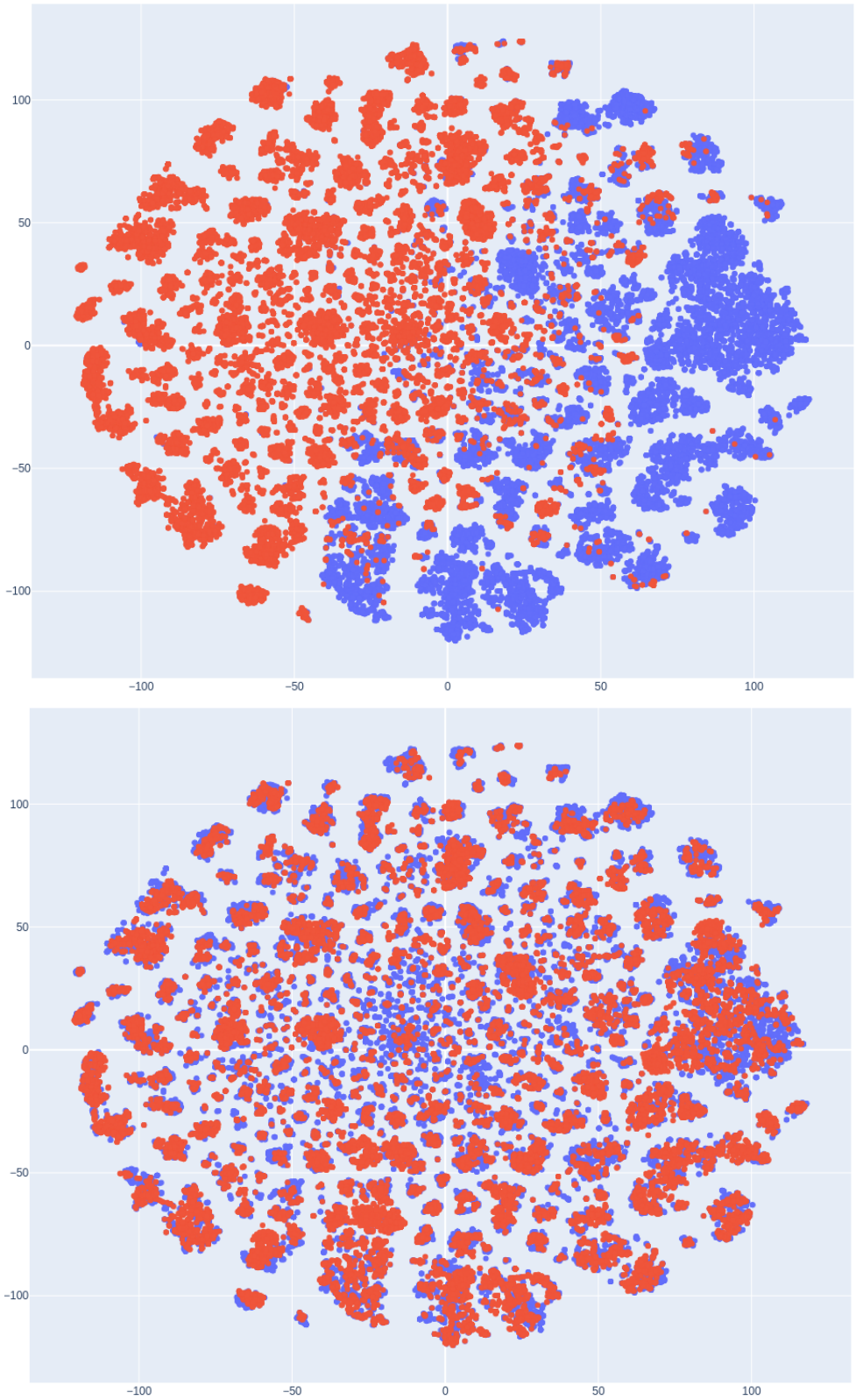

Figure 5.4: Comparison of sessions embedding clustering for *KMeans* (upper) and *GMM* (lower), calculated on dimension= 100. Data is coming from *Diginetica* test sessions, based on already trained model, projected with TSNE into 2d space.

Chapter 6

Proposed Method

Experiments described in Chapter 5 support idea of using Gaussian Mixtures, although applying it on sessions did not yield any commercial value. Repeating the process on item embeddings might give birth to something more applicable. Our goal in this chapter is to achieve new product classification, representing type of an item better than original categories.

With such classification we then reintroduce the idea that, if session contains items with different labels then it is consider multimodal. Enriched with that new knowledge, we propose a novel form of augmentation in GNNs. That is injecting extra information, as well as noise, into adjacency matrix used at the core of the model.

6.1 Idea

We think of session as a sequence of items sampled from some underlying discrete distribution. Our basic assumption is that the global distribution is multimodal, a mix of numerous gaussoids. Given unknown user's preference, region of sampling might be limited to only one or few modes, rather than whole space.

If our hypothesis is true, we might then partition item's space with *Gaussian Mixture Model* and use resulting clustering to improve quality of recommendations. To begin with, we would consider session multimodal, if its items were sampled from more than one gaussoid present in our mixture.

More specific, a session is a sequence of items, that is consequently sampled points from the hidden space with some unknown, probably multimodal, distribution. Supposing such distribution has $\mathbf{K}$ modes, every item in a unimodal session would have probability score of $[p_1, p_2, \cdots, p_K]$ with single highly dominant value, representing *type* of items, or the mode, from which all items in session were sampled.

On the other hand, a multimodal session should have items with two or more

significantly high probability scores corresponding to those multiple modes. While that could mean different types of items occurring interchangeably, which is not common, we make a different assumption. Assuming a session is multimodal, underlying probability scores change over time.

Suppose following example, at the start of session items only of i-th type are being viewed, which means that p_i holds by far the highest value. Then, at some point in time, user changes his interest to j-th class of products, value of p_i decreases and p_j becomes the most dominant score in the distribution. That means, we should start recommending items that belong to j-th subpopulation across our latent space.

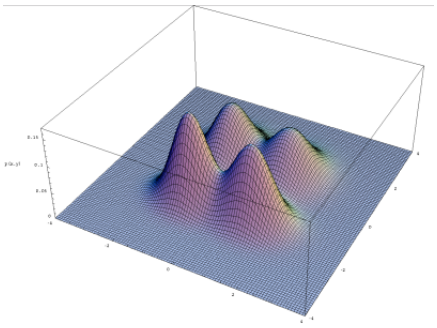

Figure 6.1: Example of probability density function of multimodal distribution in 2 dimensional space. In picture, a mixture of 4 different gaussoids.

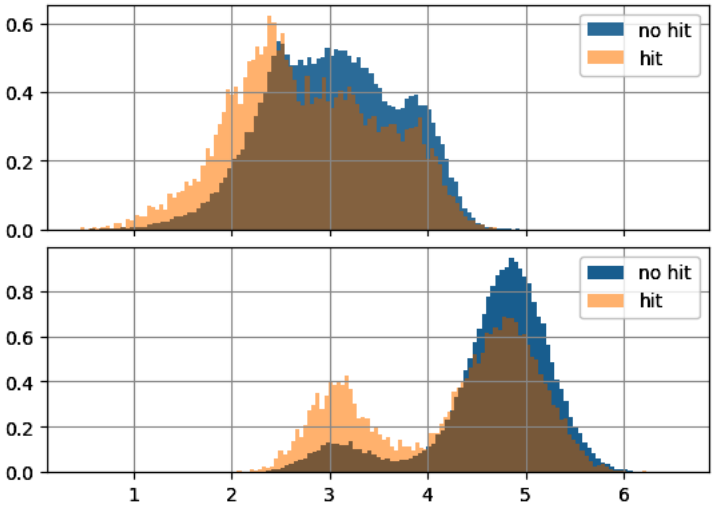

Figure 6.2: Average (upper) and maximum (lower) euclidean distance in embedding space, between following items in test sessions in *Yoochoose* dataset. We could see the difference between sessions with correct recommendation and missed ones. Items for sequences with good recommendations tend to be closer together than in fail cases. Furthermore, distribution of maximum distance looks similar to Bimodal one.

6.2 Items Classification

The basis for this clustering, is once again firstly training a Recommender System. Therefore we proceed with that, using *SRGNN* as baseline. After finishing training the model, we capture its item embeddings. With those, we train GMM to obtain new categorization. We find $\#clusters = 8$ give the best results, that being number of modes fit to data.

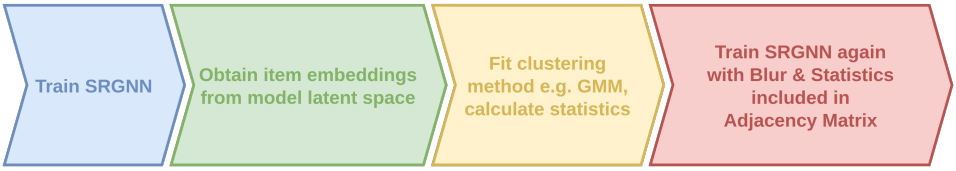

Figure 6.3: High level overview of full training process for a single model with augmented Adjacency Matrix.

6.3 Adjacency Matrix Augmentation

Here we propose few methods of augmenting adjacency matrix, not only in random manner, but also with help of extra information gained from training the recommender system.

6.3.1 Random Noise

What seems to be the simplest augmentation method out there, just adding some gaussian blur to available data. Keep in mind, we are not modifying edges or nodes in session graph, as described in [18], we modify only the matrix with some noise.

After final normalization of matrix is done in order to equalize importance of every vertex per its *indegree* and *outdegree*, we add some gaussian noise with probability $p = 0.5$ to the matrix. To do so, we sample noise matrix $\mathbf{N}$

$$\begin{aligned} M^{k\times k} &\sim \mathcal{U}(0,1) \\ N^{k\times k} &\sim \mathcal{N}(\mu,\sigma) \\ M &:= M \leq p \\ A &:= A + M \otimes N \end{aligned} \tag{6.1}$$

Where $A^{k\times k}$ is the adjacency matrix, $M^{k\times k}$ is binary mask denoting which cells of input matrix are to be modified, $\otimes$ stands for element-wise multiplication and p, μ, σ are hyperparameters of the model. We set $\mu = 0.01, \sigma = 0.01$, so that probability of negatives in the output matrix is small, but not negligible. We often use this blurring technique with combination to methods described below.

Algorithm 3 Pseudocode for our method of improving SRGNN with augmenting adjacency matrix. We omit writing hyperparametrs and others alike for readability.

Require: Train sessions $S = \{s_1, s_2, \cdots\}$, I =set of unique items

A ← Matrix(S) ▷ Binary adjacency matrix $\forall s \in S$
BaseModel ← Train(S, A)
E ← BaseModel.embedding(I) ▷ Obtain Item Embeddings
GMM ← FitGMM(E)
ItemLabels ← GMM.label(E)
C ← GMM.means ▷ Means of every mixture component in GMM

function AUGMENT MATRIX($session = [i_1, i_2, \cdots, i_n]$, p augment probability)
    $A^s \leftarrow 0^{|I|\times|I|}$ ▷ Initialize empty adjacency matrix
    for $j \leftarrow 1$ to $n-1$ **do**
        if $U(0,1) \leq p$ **then**
            distance $\leftarrow ||C[ItemLabels[i_j]] - C[ItemLabels[i_{j+1}]]||_2$ ▷ 6.3.3
            $A^s_{i_j, i_{j+1}} \leftarrow 1/distance$
        else
            $A^s_{i_j, i_{j+1}} \leftarrow 1$ ▷ Default behaviour as in SRGNN [16]
    $A^s \leftarrow [A^s : (A^s)^T]$ ▷ Concatenation, distinguish *in* and *out* edges
    $A^s \leftarrow$ AddNoise(A^s) ▷ Procedure described in Section 6.3.1
    return A^s
AugModel ← InitializeWeights()
for *epoch* untill *EarlyStopping* **do**
    $A_{aug} \leftarrow$ AugmentMatrix(S) ▷ Apply augmenation separately $\forall s \in S$
    AugModel.TrainEpoch(S, A_{aug})
return AugModel

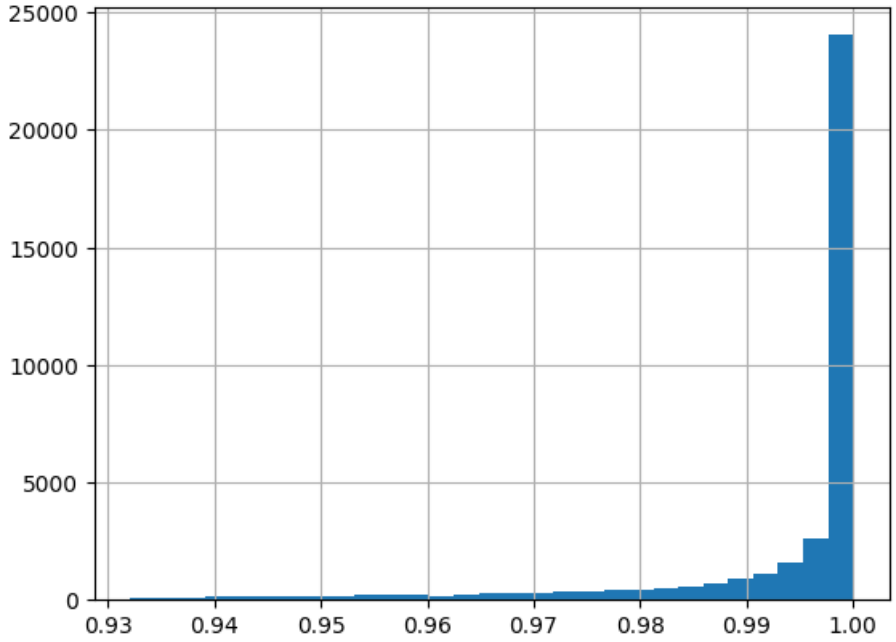

Figure 6.4: Distribution of (maximal) cluster probability for each item from *Diginetica* dataset, on 8 modes. Skewness towards 1 means that mixture is clearly separated, over 90% of items are classified with high certainty.

Every following method is based on separate, previously trained model.

6.3.2 Item to Item Distance

Collecting item embeddings from baseline model yields us with euclidean space. With that, for every item pair we may calculate distance separating them. This value might be thought of as edge weight in our graph, representing how similar the items are to each other. Furthermore, this might be interpreted as likelihood of user interacting with the second item, after being interested into the first one.

With those assumptions, we modify adjacency in following manner. For every edge $i \to j$, instead of inserting 1 into the matrix, we calculate

$$A_{i,j} = \frac{1}{\sqrt{\sum_{k=0}^{d}(emb_i[k] - emb_j[k])^2}} \tag{6.2}$$

Where emb_i is latent vector for embedding of item i, respectively for j. We divide by the distance, so that matrix entries for items close to each other would have larger values. That corresponds to more information being passed for those connections by underlying GNN. After procedure, we apply normalization as in baseline SRGNN.

6.3.3 GMM Clustering

Given our new classification, we modify the adjacency matrix with some extra information. Instead of creating a binary links, we substitute them with cluster-to-cluster relation, that being euclidean distance between means of gaussoids to which each respective item was assigned. So for each edge $i \to j$ in session graph, we calculate

$$A_{i,j} = \frac{1}{\sqrt{\sum_{k=0}^{d}(\mu_i[k] - \mu_j[k])^2}} \tag{6.3}$$

Where μ are vectors representing modes of gaussoids to which items i, j were assigned. The division is self explanatory, as the further centers are, the lower the weight of the connection. If both items belong to same cluster, we set $A_{i,j}$ to $max(1, 2 * max(A))$, so that weight of that edge would be the largest in the matrix, as the items are most likely similar to each other, and nonzero in case of whole session belongs to a single cluster.

Once again, after calculating new matrix, we apply normalization as in [16], that is normalizing the matrix for every node, separately by summed weight of its inward and outward edges.

Given soft classification of GMM, we test an extra clustering method. KMeans offer hard boundary between clusters based on euclidean distance, which may result in more see through separation of items latent space. Augmentation methods works in the same way as one based on GMM, however we use cluster centers returned by KMeans algorithm in place of gaussoid modes. Equation stays the same.

6.3.4 Category Clustering

To show our clustering with use of Gaussian Mixtures is indeed better than baseline product categories, we conduct similar method to previous one. Here, a cluster is simply the category an item is assigned to. Cluster centre would then be a vector representing an average over embeddings of all items in that particular category. So for every connection $i \to j$

$$\begin{aligned} \vec{Cat}_K &= \frac{1}{|K|}\sum_{i \in K} \vec{emb}_i \\ A_{i,j} &= \frac{1}{\sqrt{\sum_{k=0}^{d}(Cat_{cat(i)}[k] - Cat_{cat(j)}[k])^2}} \end{aligned} \tag{6.4}$$

Where $\vec{Cat}_K$ denotes *centre* of category K and $cat(i)$ stands for category of item i. Similarly to previous method, if both i, j belong to the same category we follow it up with setting $A_{i,j}$ to $max(1, 2 * max(A))$ due to same reason. After all, we apply normalization as described above.

Algorithm 3 presents an overview of our method. In provided pseudocode any step after training *BaseModel* is our contribution. This version includes modifications listed in 6.3.3 and 6.3.1, however clustering and distance measure can be substituted with any mentioned in this chapter. With proposed changes augmented model should be able to extract more information about relation between items, thanks to previously trained recommender system without risk of overfitting.

Chapter 7

Results

7.1 Based on SRGNN

We use SRGNN [16] as baseline architecture in our experiments and we compare against the results of original model. We have completed 5 separate (training base SRGNN, clustering items and then training model with modified adjacency matrix) full pipeline runs for each of described methods and compared average metrics, to avoid impact of nondeterministic calculations and make sure they are actually statistically significant. Test metrics are calculated on normal, non augmented, binary adjacency matrix for all models. As per usual, augmentation took place only during training.

On every evaluated dataset our method outperforms basic SRGNN, results are presented in Table 7.1. Naturally, largest improvement is noted on *Otto RecSys* dataset. It is the smallest one of three we selected, hence extra information gained from relevance measure between items, included in adjacency matrix, is of highest value. Same applies to blurring which reduces risk of overfitting on rather little amount of training data.

In contrast, same logic does not explain difference in observed gain between *Yoochoose 1/64* and *Diginetica*. *Yoochoose* contains less training data, yet the improvement is not as impressive. Most probable cause lays within the source of data. While items of *Diginetica* are of 'normal' categories, those of *Yoochoose* may be labelled with a *Special* category, indicating that user interacted with it while browsing through special offer or discounted products. Nature of ordinary search session is rather different than one of a sale hunter, resulting in need of separate embeddings for single item dependent on the fact if its price is cut or not. Lack of such distinction may lead to loss of information, reducing usefulness of the embedding. As our method is fully dependent on quality of product embeddings, this likely degrades its effectiveness on *Yoochoose* dataset.

	Diginetica		YooChoose 1/64		Otto RecSys	
	hit@20	mrr@20	hit@20	mrr@20	hit@20	mrr@20
SRGNN [16]	50.71	17.75	69.94	30.68	31.42	20.49
Blur	51.11	17.75	69.97	30.68	31.49	20.54
i2i+ Blur	**51.18**	18.05	**70.09**	30.69	32.05	21.16
GMM + Blur	51.12	18.01	70.02	**30.76**	32.02	21.18
KMeans + Blur	51.12	**18.06**	70.04	30.69	**32.05**	**21.34**
Categories + Blur	51.08	18.02	70.02	30.75	∼	∼
Improvement(%)	0.93	1.69	0.21	0.26	2.01	4.15

Table 7.1: Results for algorithms mentioned in 6.3. Metric values might differ for original SRGNN, as we submit those we were able to reproduce ourselves. Lack of result with Category clustering for *Otto RecSys* is due to no item metadata.

7.2 Gaussian Mixture Clusters

Firstly, what seems to be the most natural idea given our suggestions, we investigate results for augmentation backed up by item classification with use of Gaussian Mixture Model. However, on most datasets *Item2Item* augmentation turned out to work the best which discredits the assumption, that *GMM* augmentation would outperform different approaches. Clearly, underlying problem of modelling user's preferences is beyond expressive power of just μ of normal distributions. Moreover, *KMeans* distance method achieves higher metrics as well. We provide an understanding on why *KMeans* might be a better solution, later we elaborate on Item2Item approach as well.

Gaussian Mixtures offer rather soft classification, while *KMeans* grouping provides a hard boundary between separate clusters, which seems to be advantageous. Even when using only 8 modes, *GMM* classification tends to be very unbalanced with some clusters containing over 10'000 items when other consist of just few hundreds, given 43098 products (in *Diginetica*). Our algorithm introduces extra information into adjacency matrix if and only if items from at least two different clusters were viewed during session, therefore such an imbalance might result in hardly any changes to the training process (bigger number of clusters did not solve the imbalance problem). On the other hand, when applying *KMeans* resulting assignment is well balanced with no. items in every cluster varying from 4'000 to 7'000. With that partition, sessions with more than single cluster will be much more common, hence magnifying the impact of augmentation.

Another cause might be spacing and complexity of the clustering algorithms. Using information about centers of *KMeans* fully utilizes underlying algorithm, as both the clustering process and resulting labelling is based only on those centers. Contrary, same cannot be said for GMM. Mean of every normal distribution is just a part of its definition, rest being contained in covariance matrix which we do not

	min	max
GMM	0.150	0.427
KMeans	0.347	0.546

Table 7.2: Minimal and maximal distance between pair of cluster centers for *GMM* (means of associated gaussoids) and *KMeans*. Data based on single run on *Diginetica*

make use of in our process. Possible modifications to include covariance matrix are mentioned in Section 8.1. Moreover, modes of *GMM* are much more squeezed in when compared to centers of *KMeans* (Table 7.2). This, with discarding covariance matrix, results in assigning weight as if the items were much closer to each other than in reality.

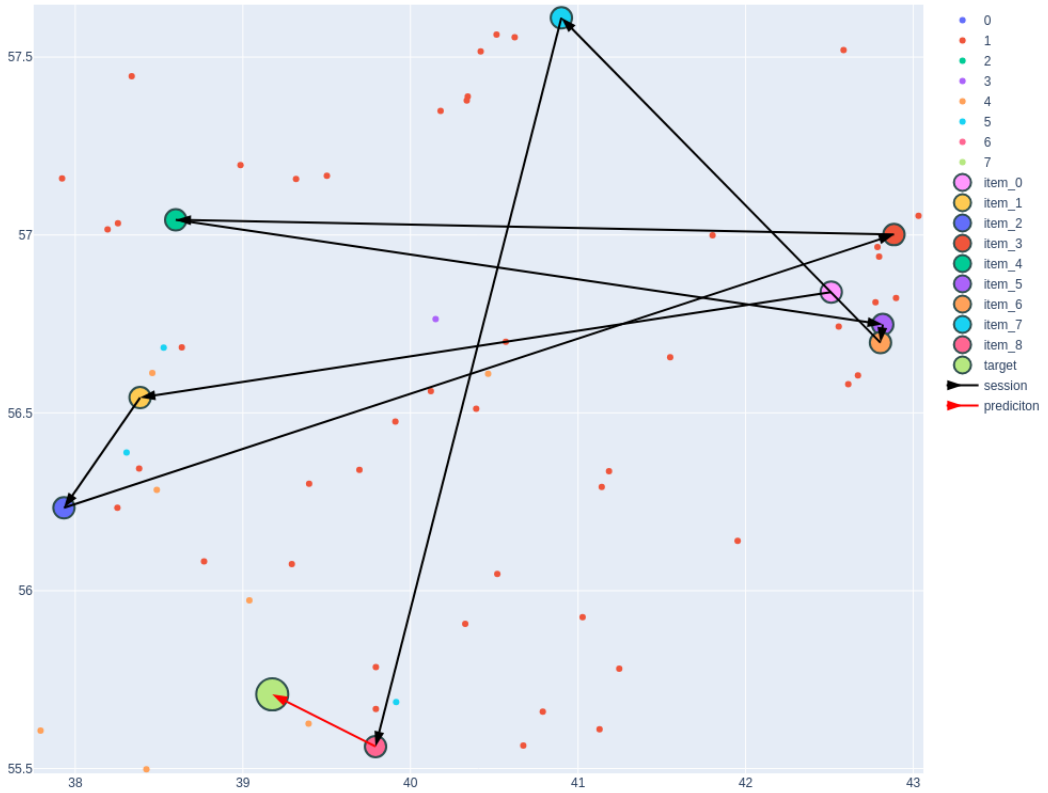

Figure 7.1: Small region of items latent space, parsed with TSNE into two dimensions. Highlighted points connected with arrows represent single session, in that case coming fully from mode number 1. Colour of other visible points stands for their respectful modes.

7.3 Item to Item Distance

Best results regarding *Hit* were achieved with inducing extra information about euclidean distance between vertices for each edge in session graph. Such information is without doubt much more precise than separation between respective cluster centers, which is a likely reason for better metrics of the recommender system. In this section we analyse on what samples improved model gives better recommendations and why does it happen.

Given multimodal session, we know that at some point user's interest changes dramatically. That means type, not necessary category, of items viewed changes as well. What seems to be a good assumption, is that items of different types would be far away from each other in the latent space. This is what Item2Item augmentation makes use of.

When point of session's focus changes rapidly, we want to update our recommendations just as fast. When session graph adjacency matrix is binary, model cannot easily distinguish between outdated and currently relevant interactions. Introducing edge weight, that is inverse of distance, between each following clicks, decreases importance of items far away in the embedding space. Therefore, their presence even if it's majority, has smaller influence on session local embedding which then will not constraint the model from yielding recommendations closer to most recently viewed products.

Most importantly, weight of items close to each other will still be high, as it represents their pairwise relevance and similarity. Of course, this corresponds to magnitude of information being passed in every GNN step when calculating their embeddings. For a pair of items with small weight, their embeddings will have little dependency on each other in the message propagation step.

Noise added to the whole matrix forces GNN to gather information from every item present in session, although just a tiny part of it. That enriches embedding of every vertex with a more global picture of user's behaviour, before further transformations. With that, product embeddings achieved from session graph, are more codependent on each other which further improves representative power of session global embedding and with it, quality of recommendations.

7.3.1 Example

We selected random session from test set of *Diginetica*. This particular one is of length 10 and consists of 7 unique products, out of which two occur more than once. All of them belong to the same category. Both basic SRGNN and modified implementations recommend correct target item, however our scores 1 on MRR where basic gets a 0.2 score. That means augmented model returns target item at the first position in list of recommendations while basic at 5th position. In Table 7.3 we

Base Matrix

0	0	0	0	0	0	0	0	0
1	0	0	0	0	**0.333**	**0.333**	0	**0.333**
2	0	0	0	0	0	0	0	0
3	0	0	0	0	0	0	**1.0**	0
4	0	**0.5**	0	**0.5**	0	0	0	0
5	0	**1.0**	0	0	0	0	0	0
6	0	0	**1.0**	0	0	0	0	0
7	0	0	0	0	**1.0**	0	0	0

Augmented Matrix

0	0.011	0.02	0	-0.002	0	0	0.001	0
1	0	0.027	0	-0.018	**0.352**	**0.326**	0	**0.345**
2	0	0.008	0	-0.007	0	0	0	0
3	0	-0.002	0.019	0.011	-0.002	0	**1.007**	0.004
4	0.022	**0.524**	0	**0.48**	0	0.02	0	0.015
5	0	**1.014**	0	0.018	0.011	0.002	0	0
6	-0.007	0.003	**1.011**	0	0.028	-0.005	0.004	0
7	0	0	0	0.009	**1.001**	0	0	0.012

Table 7.3: Part representing outward edges (for better visibility) of Normal and Augmented Adjacency Matrix. Wieghts presented stand for outgoing edges. Augmentation applied was Item2Item distance + noise with $\mu = 0.01, \sigma = 0.01$.

	$1 \to 4$	$1 \to 5$	$1 \to 7$
Base	1.078	1.226	1.118
Augmented	1.143	1.440	1.476

Table 7.4: Euclidean distance in items latent space, of base and augmented model. Distances between item #1 and those clicked right after it. Session in question is from *Diginetica* test set.

present latter half of Adjacency Matrix, that is part representing edges going **out** of every node. In this session item #1 was clicked 3 times, that is why corresponding row in basic matrix has 3 nonzero entries all equal to 1/3, similarly item #4 was visited twice. Item #2 has no entries, as this is the last item in the session clicked only once, so our goal for this session is to correctly predict an edge from #2 to target product. Item number 0 represents special token, used for padding sessions up to equal length when compressing them into batches. It is appended after last interaction but not taken into account, that is why in the original matrix values of 0th row and column are equal to zero.

Weights for connections outgoing from item #1 are not equal, not only because of random noise as it might seem. Values were calculated with respect to Equation 6.2, therefore they are inversely proportional to distances in base model latent space, which are listed in Table 7.4.

Separation between items in base model does not seem to correlate with distance in augmented version, as the placement of products in latent space is quite different. What is more, sparsity of the hidden space is significantly distinct, as maximal boundary distance in base space is 7.862 and 10.257 in augmented.

We take closer look at the recommendations returned by proposed methods. Taking as input single session mentioned beforehand, we calculate euclidean distance between embedding of true target item and every recommendation returned by the models. Such comparison shows how *relevant* our proposals are. Results are shown in Table 7.5. Equal values across rows mean that items returned up to this point were the same, however possibly in different order. Backing up our understanding, cumulative distance of recommendation list is smaller for any modification comparing to base model. We provide an extra comparison on how recommendations are spread out accros space in Section B.2 in the Appendix.

7.4 Diginetica SERP Relevance

As a proof of concept, we make use of *Search Engine Result Page* information included in *Diginetica* dataset. Each session in data has associated its query that began the session. Query brings with itself a list of items ranked according to SERP, the default recommendations made by e-commerce platform. This ranking is most

Method	Target @	Cumulative Distance					···	@20
Base	5	1.077	2.137	3.425	4.664	5.643	6.527	21.613
i2i	1	1.077	2.137	3.115	4.404	5.288	6.46	21.036
Categories	2	0.884	1.961	3.021	4.309	5.576	6.815	21.23
GMM	8	1.059	2.298	3.375	4.259	5.548	6.815	21.009
KMeans	2	0.884	1.961	3.021	4.259	5.454	6.432	21.267

Table 7.5: Cumulative sum of distance between recommended item and target product in latent space of base model. We omit listing the target item itself, its position in recommendation list is shown in separate column.

likely influenced by some extra information regarding items we do not have access to (like full item name), but we may use those as a sanity check for our understanding of the problem. What might seem obvious, as examined from data, products viewed during sessions do not fully align with those of SERP.

Investigating SERP products in the latent space, we observe they create few (usually single) tight clusters of points in latent space. Moreover, most of the points from SERP belong to the same *GMM* mode as session items (Figure 7.2). That means, both latent space and *GMM* clustering complies, at least partly, with grouping suggested by Search Engine.

7.5 Other Models with Augmentation

Our method depends solely on the latent space into which items are embedded, so details of GNN architecture as well as rest of the model do not matter that much, as long as our starting embedding space is euclidean. Therefore, we claim that method described in Chapter 6 could be model agnostic.

We provide empirical prove for the concept, we repeat our experiments on TAGNN [19], an improved version of SRGNN. Of course, adjacency matrix of session graph is used at the core of GNN applied in the algorithm. Pipeline of workflow is the same, we train base TAGNN model, obtain item embeddings, calculate Item2Item distances, fit *Gaussian Mixtures* or *KMeans*, and train second model from scratch with augmentation of adjacency matrix.

Results shown in Table 7.6 indicate that indeed, model metrics did improve by even higher margin to one of SRGNN. Further tuning of hyperparameters could yield even better score. In contrast to experiments on SRGNN, augmentation based on Gaussian Mixtures tends to outperform other methods.

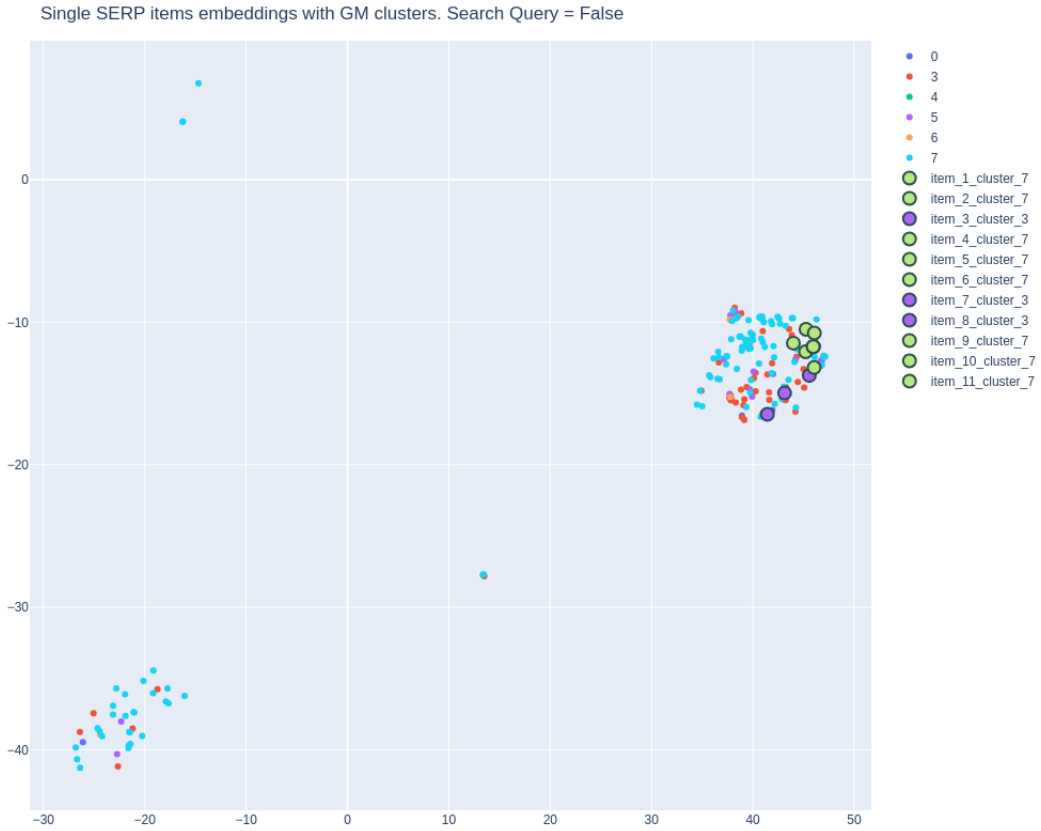

Figure 7.2: SERP item embeddings parsed with TSNE (trained on all products) into two dimensions, for a single test session. Items from session itself are highlighted. Color of points represent their respective *GMM* cluster. Here 87% of points belong to 2 most common (in that SERP) *GMM* modes, number 3 and 7, as well as all items from session.

	Diginetica		YooChoose 1/64		Otto RecSys	
	hit@20	mrr@20	hit@20	mrr@20	hit@20	mrr@20
TAGNN [19]	50.34	17.66	70.50	30.76	34.10	21.43
i2i+ Blur	51.04	**18.17**	70.51	30.79	35.04	22.71
GMM + Blur	**51.13**	18.05	**70.78**	**31.04**	35.02	**22.82**
KMeans + Blur	51.11	18.08	70.72	30.86	**35.08**	22.77
Categories + Blur	51.08	18.00	70.52	30.62	∼	∼
Improvement(%)	1.57	2.21	0.40	0.91	2.87	6.25

Table 7.6: Our results for augmentation methods mentioned in 6.3, combined with TAGNN architecture. As we did not manage to fully reproduce the results, values might differ from original paper.

Chapter 8

Conclusions

In this thesis we have proposed and justified means by which we interpret multi-modality in case of session data. We compared it to notion of multi-categorical sessions, and ran various experiments with goal to distinguish the two. Starting with simple approaches, we tested how changing nature of data influences the model. Later we successfully tried to classify the sessions with *Gaussian Mixture*, using it to finetune model on those kind of sessions where it achieved below average performance. In the end we have presented a novel way to augment training of GNNs in Session Based Recommendation.

Our solution focuses on extracting information from already trained Recommender System and utilizing it, to create an improved version. We did that by using latent space of item embeddings as a source of knowledge about relation between products. Then we suggested new, as opposed to item categories contained in metadata, classification of items based on the hidden space, with help of clustering algorithm like *KMeans* and *GMM*. We put our attention on how to use such classification and include this information in model training.

Following, we proposed few different methods on how to calculate item similarity in Chapter 6, which we then used to obtain weight of a connection between pair of products. With this we have presented course of action that introduces edge features into session graphs. As opposed to utilizing binary adjacency matrix and including those in message propagation step, when an edge occurs instead of filling it with one, we inject (with some probability) the weight directly into the adjacency matrix at corresponding position. Combined with blurring technique, this procedure helps the system by decreasing risk of overfitting, as well as reducing importance of connections between products irrelevant to each other, therefore improving final recommendations.

8.1 Further Work

Enhancing Adjacency Matrix with additional information about relation between items in a session yields reasonable benefit. For now, we have tested how introducing edge weights into the matrix influences the model. More experiments could be done, substituting our *Item2Item* distance with different features. Utilizing timestamp of respective session clicks is one option, that is assigning edge weight proportional to, or parsed with nonlinear function based on time passed between two following clicks.

Moreover, we could introduce a convolutional layer into the algorithm, so that Adjacency Matrix would consist of several connection weight channels (eg. first *Item2Item* distance, second related to time component, etc.). With convolutions, even using 1×1 kernel, we could transform it, maintaining all information, back into single channel matrix that would be passed on to GNN.

Another major point should be closer examination of why KMeans outperforms clustering done with Gaussian Mixtures. We gave few suggestions of why that happens in Section 7.2, although other causes probably exist.

Further idea that seems worth to investigate, is that KMeans encapsulate all information in cluster center, which is enough to improve the model. In GMM case, only part of knowledge is in the μ of each mode. Major part is hidden within covariance matrix, which currently is not included in augmentation. Notice that covariance matrix for every mode is of shape $hiddenSize \times hiddenSize$, so with help of convolutional layers, or even some simpler aggregations, we can transform it into vector of dimension *hiddenSize*. Given that embedding, we could concatenate it with session local and global embeddings, before applying last transformation $s = W_3 \cdot [s_l : s_g : s_{cov}]$. As this score is representing preferences for next item, we could select covariance matrix of Gaussian Mixture mode corresponding to last item viewed in a session.

Bibliography

[1] Kyunghyun Cho, Bart van Merrienboer, Çaglar Gülçehre, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. Learning phrase representations using RNN encoder-decoder for statistical machine translation. *CoRR*, abs/1406.1078, 2014.

[2] A. P. Dempster, N. M. Laird, and D. B. Rubin. Maximum likelihood from incomplete data via the EM algorithm. *Journal of the Royal Statistical Society: Series B*, 39:1–38, 1977.

[3] Kaize Ding, Zhe Xu, Hanghang Tong, and Huan Liu. Data augmentation for deep graph learning: A survey. *SIGKDD Explor.*, 24(2):61–77, 2022.

[4] Balázs Hidasi, Alexandros Karatzoglou, Linas Baltrunas, and Domonkos Tikk. Session-based recommendations with recurrent neural networks. In *4th International Conference on Learning Representations, ICLR 2016, San Juan, Puerto Rico, May 2-4, 2016, Conference Track Proceedings*, 2016.

[5] Bo Jiang, Ziyan Zhang, Doudou Lin, Jin Tang, and Bin Luo. Semi-supervised learning with graph learning-convolutional networks. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2019.

[6] Yujia Li, Daniel Tarlow, Marc Brockschmidt, and Richard S. Zemel. Gated graph sequence neural networks. In *4th International Conference on Learning Representations, ICLR 2016, San Juan, Puerto Rico, May 2-4, 2016, Conference Track Proceedings*, 2016.

[7] Malte Ludewig and Dietmar Jannach. Evaluation of session-based recommendation algorithms. *User Modeling and User-Adapted Interaction*, 28(4–5):331–390, 2018.

[8] Anis Redjdal, Luis Pinto, and Michel Desmarais. Optimizing encoder-only transformers for session-based recommendation systems. *CoRR*, abs/2410.11150, 2024.

[9] Steffen Rendle, Christoph Freudenthaler, and Lars Schmidt-Thieme. Factorizing personalized markov chains for next-basket recommendation. In *Proceedings of*

the 19th International Conference on World Wide Web, WWW 2010, Raleigh, North Carolina, USA, April 26-30, 2010, pages 811–820. ACM, 2010.

[10] Xin Rong. word2vec parameter learning explained. *CoRR*, abs/1411.2738, 2014.

[11] Zhengxiang Shi, Xi Wang, and Aldo Lipani. Self contrastive learning for session-based recommendation. *CoRR*, abs/2306.01266, 2023.

[12] Wei Song and Kai Yang. Personalized recommendation based on weighted sequence similarity. In *Practical Applications of Intelligent Systems*, pages 657–666, Berlin, Heidelberg, 2014. Springer Berlin Heidelberg.

[13] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: a simple way to prevent neural networks from overfitting. *J. Mach. Learn. Res.*, 15(1):1929–1958, 2014.

[14] Shoujin Wang, Longbing Cao, and Yan Wang. A survey on session-based recommender systems. *CoRR*, abs/1902.04864, 2019.

[15] Shoujin Wang, Liang Hu, Yan Wang, Quan Z. Sheng, Mehmet Orgun, and Longbing Cao. Intention2basket: A neural intention-driven approach for dynamic next-basket planning. In *Proceedings of the Twenty-Ninth International Joint Conference on Artificial Intelligence, IJCAI-20*, pages 2333–2339. International Joint Conferences on Artificial Intelligence Organization, 2020. Main track.

[16] Shu Wu, Yuyuan Tang, Yanqiao Zhu, Liang Wang, Xing Xie, and Tieniu Tan. Session-based Recommendation with Graph Neural Networks. In *Proceedings of the Twenty-Third AAAI Conference on Artificial Intelligence*, volume 33 of *AAAI '19*, pages 346–353, 2019.

[17] Xiaolong Xu, Hongsheng Dong, Lianyong Qi, Xuyun Zhang, Haolong Xiang, Xiaoyu Xia, Yanwei Xu, and Wanchun Dou. Cmclrec: Cross-modal contrastive learning for user cold-start sequential recommendation. In *Proceedings of the 47th International ACM SIGIR Conference on Research and Development in Information Retrieval*, SIGIR '24, page 1589–1598, New York, NY, USA, 2024. Association for Computing Machinery.

[18] Joo yeong Song and Bongwon Suh. Data augmentation strategies for improving sequential recommender systems. *CoRR*, abs/2203.14037, 2022.

[19] Feng Yu, Yanqiao Zhu, Qiang Liu, Shu Wu, Liang Wang, and Tieniu Tan. Tagnn: Target attentive graph neural networks for session-based recommendation. In *Proceedings of the 43rd international ACM SIGIR conference on research and development in information retrieval*, pages 1921–1924, 2020.

[20] Tong Zhao, Wei Jin, Yozen Liu, Yingheng Wang, Gang Liu, Stephan Günnemann, Neil Shah, and Meng Jiang. Graph data augmentation for graph machine learning: A survey. *IEEE Data Eng. Bull.*, 46(2):140–165, 2023.

[21] Xiangyu Zhao, Liang Zhang, Zhuoye Ding, Dawei Yin, Yihong Zhao, and Jiliang Tang. Deep reinforcement learning for list-wise recommendations. *CoRR*, abs/1801.00209, 2018.

Appendix A

A.1 Source Code

Together with this paper, we include zipped folder called "ThesisCodes". Moreover, all the code may be accessed at *github* repository. We kept track of experiments with use of *Weights& Biases*, our project is available here. Major part of the code is based on one provided by [16], however we updated and simplified it with newer version of *Pytorch* and *Pytorch Lightning*. We attach an extra *requirements.txt* file with python libraries and their versions used in our experiments.

A.2 Hyperparameters and Training

For training of baseline **SRGNN** we adapted most hyperparameters from [16] with slight modifications. The dimension of latent space was set to $d = 100$, starting learning rate of 0.001 with Adam optimizer, weight initialization done with normal distribution ($\mu = 0; \sigma = 0.1$), batch size of 100 and weight decay penalty term of 10^{-5}. Our changes from original were to decay learning rate by 0.1 every two epochs instead of three, use GNN propagation step equal to 2 and introduce early stopping condition based on improvement of validation loss with patience of five epochs.

For training of GMM as well as $KMeans$ we settled on 8 clusters. Gaussian Mixtures were trained with fully separate covariance matrices, equal weights of each component and initial μ as centroids of precalculated $KMeans$.

Our method used $p = 0.5$ as probability of applying augmentation, that is setting value in adjacency matrix to our distance measure (respectively for selected approach) instead of a binary entry. Noise addition also used $p_{noise} = 0.5$ as probability to blur adjacency matrix with gaussian noise sampled from ($\mu = 0.01, \sigma = 0.01$). Moreover, we decreased weight penalty to 10^{-6} as our augmentation itself acts as a precaution to overfitting and decreased the patience of early stopping to 4 epochs. Rest of hyperparameters was kept the same. Training took up to 20 epochs for both base and our method.

Appendix B

B.1 Embeddings

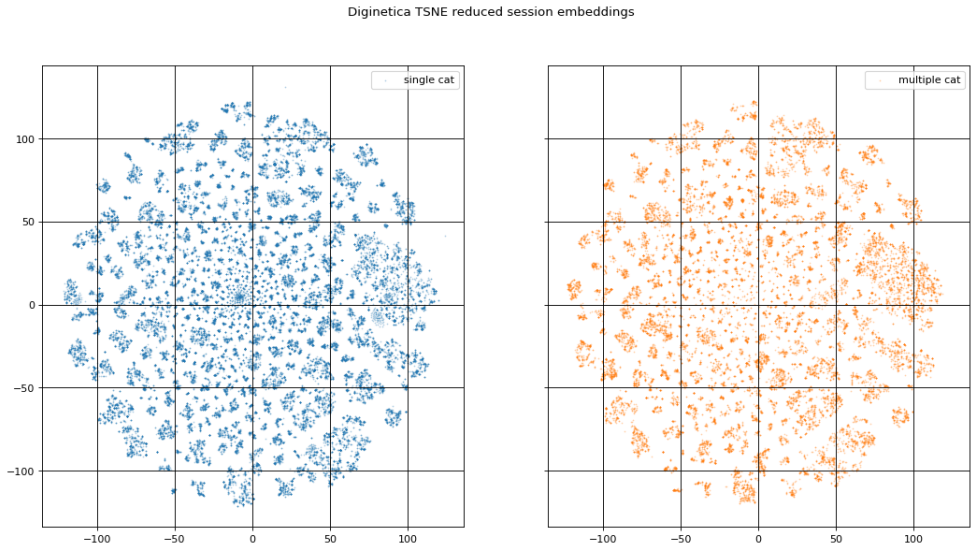

Figure B.1: Comparison of embeddings for sessions consisting of only single category and those of multiple, for *Diginetica* dataset. Data is coming from test sessions, based on already trained model on dimension= 100, processed with TSNE into 2d space.

B.2 Recommendations

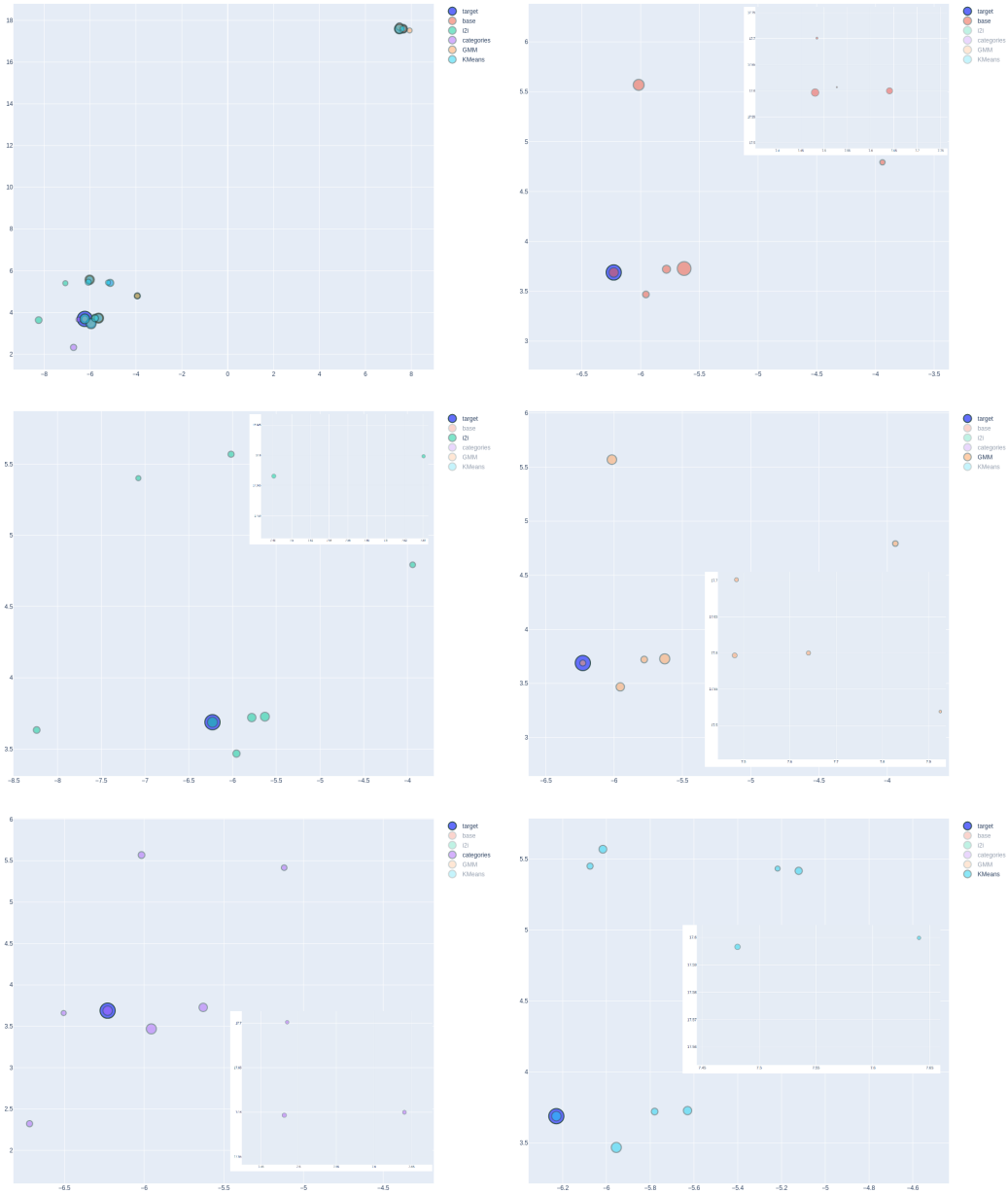

Figure B.2: Comparison of embedding for model recommendation, following different augmentation methods. Size of marker represents position of an item in returned recommendation list, bigger point - higher position. Inserted part on the plots represents section from upper right corner of the first plot. Input session was from *Diginetica*.