Variational Autoencoders with Gaussian Mixture Prior for Recommender Systems

by

Kristof BOUCHER CHARBONNEAU

THESIS PRESENTED TO ÉCOLE DE TECHNOLOGIE SUPÉRIEURE
IN PARTIAL FULFILLMENT OF A MASTER'S DEGREE
WITH THESIS IN INFORMATION TECHNOLOGY ENGINEERING
M.A.Sc.

MONTREAL, JUNE 29, 2020

ÉCOLE DE TECHNOLOGIE SUPÉRIEURE
UNIVERSITÉ DU QUÉBEC

ACKNOWLEDGEMENTS

I would like to express my thanks to my supervisors Patrick Cardinal and Marco Pedersoli for their constant support and continued encouragement. Even during busy periods, I am glad they were always making time for me. It was not always easy, events out of our control happened, but I am truly happy that we were able to make this co-supervision work.

I am also grateful to Radio-Canada for sponsoring this work, especially my manager Stephane Caron, who supported this work by providing me with all the necessary resources. I would also like to thank my colleague and dear friend Vincent Barnabé-Lortie. Without your ideas, advice and support, this work would not have been possible.

Many thanks also go out to my family (Lynda, Gilles, René, Linda, Mathieu, Valérie) for their reassuring words that helped me during many occasions. I would like to dedicate this thesis to David. Your constant support and love helped me successfully finish this dissertation. Your comments and proofreading were much appreciated.

Finally, although his involvement was more indirect, I would like to thank my supervisor Dr. Osamu Shouno-san. Based at the Honda Research Institute in Japan, Dr. Shouno-san pushed me in the right direction and helped discover my true passion.

Auto-encodeurs variationnels avec un mélange gaussien comme à priori pour les systèmes de recommandation

Kristof BOUCHER CHARBONNEAU

RÉSUMÉ

Les systèmes de recommandations sont utilisés dans plusieurs domaines, en passant par les engins de recommandations aux plateformes de divertissements comme la musique en ligne ou la vidéo sur demande. Ils sont une composante essentielle lorsqu'une source considérable d'information est disponible aux usagés. Ces systèmes sont en mesure de proposer la bonne information au bon moment. La plupart des techniques ont besoin de récupérer l'impression des usagés sur l'information consommée afin d'émettre la prochaine recommandation. Cependant, cette collecte d'information n'est pas toujours simple, complète et fiable. Une alternative à cette collecte d'information directe est de plutôt recueillir le témoin indiquant l'interaction d'un utilisateur avec un contenu.

Radio-Canada, le diffuseur officiel du Canada, a collecté ce type d'information sur sa plateforme de vidéos en continu appelé « Tou.TV ». La société d'État nous a demandé de créer un nouveau système de recommandation qui utilise ces données afin de remplacer le système existant. Cette thèse présente deux architectures hybrides fondées sur les auto-encodeurs variationnels (VAE) avec un à priori basé sur les mélanges de gausiennes afin de mieux comprendre l'espace caché. Nous appliquons ces deux modèles sur les données de Tou.TV, mais également sur l'ensemble de données appelé « MovieLens-20M ». Les modèles présentés apportent des changements simples aux VAE et permettent d'obtenir des résultats compétitifs par rapport aux systèmes populaires.

Mots-clés : Système de recommandation, données implicites, apprentissage machine, apprentissage profond, Auto-encodeur variationnel, mixture de Gaussiennes.

Variational Autoencoders with Gaussian Mixture prior for recommender systems

Kristof BOUCHER CHARBONNEAU

ABSTRACT

Recommender systems are used everywhere, from search engines to entertainment websites like video or audio streaming platforms. They are essential when the amount of information available is substantial by providing users with the right information at the right time. The standard approach involves gathering the impressions of users based on content to create the next recommendation. However, collecting these impressions is costly and is not always accurate. A different alternative is to simply gather the binary interactions between a user and the content.

Radio-Canada, the official French broadcaster in Canada, has collected these types of interactions for its video streaming platform called "Tou.TV". They asked us to create a novel recommender system using temporal and contextual signals. In this thesis, we present two hybrid systems based on a variational autoencoder (VAE) architecture with a Gaussian mixture prior to better understand the latent space with multiple Gaussian distributions. We apply these systems on the Tou.TV dataset, but also demonstrate the efficacy of our approach on the popular dataset called MovieLens-20M. These systems with the simple enhancements proposed on the traditional VAE architecture are able to outperform popular off-the-shelf models.

Keywords: Recommender systems, implicit data, machine learning, deep learning, variational autoencoder, gaussian mixture.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

INTRODUCTION

0.1 Motivation

In an era when more content is available electronically every day, it has become harder for consumers to find interesting media such as books, movies, TV shows, songs and textual information. The shear volume of information is overwhelming and provides the user with an infinite number of content possibilities. Navigating the sea of information without any personalization makes it harder to find the relevant content when thousands of results are involved. Now this may seem mundane when the purpose is for leisure, but companies are failing to provide the right content to their user without any personalization mechanisms, and this might represent an important revenue loss. At the other end of the spectrum, without a personalized recommender system, researchers might not be able to find the relevant information in regards to their research. Due to the aforementioned examples, this led to the creation of hyper-personalized recommender systems in various industries. These systems are based on users' personal profile like tastes, past history and behaviours to provide a subset of tailored content. They are omnipresent and represent an integral aspect of popular products and services consumed by everyday users. They learn from different signals to provide a small subset of results that are highly personalized.

Moreover, these systems also improve catalog discoverability, serendipity and complement searching. Netflix estimates that 75% of new content consumed by their users is through a recommendation algorithm (Amatriain & Basilico, 2012), while 60% of YouTube videos accessed from the homepage comes from a recommender system (Davidson *et al.*, 2010). On popular e-commerce platforms like Amazon, users can navigate through different suggestions based on a combination of their own purchase history and other's similar history to find the best deal and narrow down the best items to buy next (Linden *et al.*, 2003). Companies also benefit by investing in recommender systems to provide a better user experience and increase their profit margins. At Amazon, 35% of their revenue comes from personalized recommendations (MacKenzie *et al.*, 2013), while Netflix stated that they saved more than 1$B per year due to these systems (Gomez-Uribe & Hunt, 2015). At Spotify, a popular music streaming service, they evaluate the impact of their recommender system to have helped increase their monthly users from 75 million to 100 million (Leonard, 2016).

These companies are able to attract many users and reach their financial goals with these hyper-personalized systems. They also collect a considerable amount of information about their users such as content preferences, usage patterns and interactions to update and provide a more tailored experience. Two types of data are usually used when creating a recommendation algorithm. The first type can be a binary or discreet value representing the interaction between a user and the items. In the literature, this type of data is often called "implicit interaction" and generally represents an action of a user (e.g., a click, a view, an item bought. . .). The second type is denoted as "explicit interaction" and is usually correlated with the "interest" (e.g., an ordinal scale denoting satisfaction). Along with these two types of data, metadata representing the content (e.g., the title of a movie, the content of a book, the values of each pixel in an image . . .) can also be used as an additional source of knowledge in a recommender system. In this thesis, we denote three types of algorithms. The first is collaborative filtering (CF) which creates neighbourhoods of similar data. The second is content-based filtering (CBF) which consider similarities between contents. And lastly, hybrid methods (HB) which combines two or more recommendations or use an expressive model to encode high-dimensional information. Each of these techniques are described in the next chapter along with the two data types.

In recent years, tackling problems with machine learning (ML) has become increasingly popular due to the availability of data and graphical processing units (GPU). This enables practitioners and researchers to train deep architectures, a technique also known as Deep Learning (DL). Such techniques are especially promising since they can automatically extract significant features from unstructured data like images, texts and sounds. This removes the need for handcrafted features, and enhances traditional recommender models like the latent-factor model or a similarity-based one (Oord *et al.*, 2013; Wang *et al.*, 2014). In more recent work like Tan *et al.* (2016), the authors exclusively use a Recurrent Neural Network to encode the temporality found in a user's sessions or in (Liang *et al.*, 2018a), where the authors encode the implicit interactions in a latent model called Variational Auto-Encoder (VAE) to provide recommendations.

In this work, we aim to improve traditional DL techniques and apply them to recommender systems for implicit and sparse data. We propose a non-temporal and a temporal model, both using a VAE to encode latent variables. We use this probability latent-variable model due to the following: (1) its ability to automatically learn noise with its Gaussian distribution (Vincent *et al.*, 2010), (2) its ability to represent the statistical strength between users and items (Liang *et al.*, 2018a) and (3) it's ability to learn from multimodal data and provide recommendations from a unified architecture.

The non-temporal model uses a standard neural network to create the latent representation used by the VAE. The temporal version use a Recurrent Neural Network (RNN) to create the latent representation. This model creates a temporal embedding that represents the long-term dynamics of the sequential data, characterizing the evolution of taste across time for a user. This idea was inspired by Moore *et al.* (2013) where the authors show that taste shares similar characteristics with dynamical systems.

Finally, we argue that using a standard Gaussian distribution used by traditional VAE (Kingma & Welling, 2013b) might not be expressive enough to model the taste complexities of each user. Furthermore, it limits the model to only learn unimodal representation. Hence, we assume that observed data comes from a multimodal prior distribution. We proposes to use a Gaussian Mixture Model (GMM) as the prior distribution of the model (Bishop, 2006).

0.2 Contributions

Traditional linear factor models for CF (Gopalan *et al.*, 2015; Ma *et al.*, 2011; Vozalis & Margaritis, 2007) are still dominating the industry of recommender systems due to their simplicity and quality of recommendations. However, as the amount of data increases, these approaches have several weaknesses such as data sparsity, cold start and scaleability issues (Chen *et al.*, 2018). Furthermore, current research using newer types of models like neural networks are still mainly focused on explicit datasets (Sedhain *et al.*, 2015; Salakhutdinov *et al.*, 2007; Zheng *et al.*, 2016).

The models presented in this thesis focus on implicit data and are built on the latest work by (Liang *et al.*, 2018a) where the model learns a latent representation from implicit data with a VAE. This model provides recommendations by sampling the multidimensional latent representation which comes from a standard Gaussian prior. Although their work produces impressive results, they assume that the data comes from a single Gaussian distribution, and does not consider temporality, which is often known when recording the interaction. Based on the previous work, we create two types of models for implicit sparse data which creates a distribution of probability across all available items and their users given their interaction history. In both models, a VAE with a GMM prior (GMM-VAE) is used to learn a multidimensional latent representation. The first is a non-temporal model and considers only implicit data. The second model uses temporal data where the latent space produced by the VAE is included in the hidden state of an RNN. This is to learn the complex variability of the data, such as the evolution of a user's taste. Our results show that the GMM used in the VAE is able to effectively capture more information existing in implicit data than a single standard Gaussian distribution. Our contributions are as follows:

1. We analyze quantitatively and qualitatively the impact of the GMM in regards to implicit recommendation tasks. We note a considerable gain in performance by averaging the prior's GMM into a single Gaussian distribution over traditional VAE models. But we get overall

better results by keeping the GMM but using a different estimator over other state-of-the-art algorithms.

2. We create a new model by combining the latent variables produced by the GMM-VAE into the hidden state of an RNN similar to Chung *et al.* (2015). But unlike the previous work, we also demonstrate the importance of latent variables in the RNN dynamics for accurate recommendations.

This work was commissioned by Radio-Canada, the official francophone national broadcaster in Canada. The models presented were developed for their video streaming platform called Tout.TV. The description of their dataset is included but will not be released to the public due to privacy concerns. Both models are publicly available.

0.3 Organization

This thesis is organized in four chapters:

- In Chapter 2, we provide an introduction on the relevant work in recommender system. This ranges from traditional methods like CF, to more advance models like neural networks along with the metrics used to evaluate such systems. We also give a brief explanation on supervised and unsupervised learning methods.

- In Chapter 3, we describe the structure of the non-temporal and temporal model. More formally, we show how to use a GMM in a VAE instead of a standard Gaussian distribution. This chapters also details how we can combine the latent representation into the recurrent hidden state of an RNN.

- In Chapter 4, we analyze quantitative and qualitative impact of using a GMM for implicit recommendation tasks and image generations. We also describe the dataset provided by Radio-Canada and describe comparisons against state-of-the-art models.

- In Chapter 5, we conclude this thesis and give an overview of our methods and their limitations as well as future work.

This chapter presented the advantages of using a recommender system to provide users with a better experience, improve systems, and increase revenues. We also summarized our contributions and described our VAE models that use a different type of prior to provide recommendations. The next chapter gives an overview of key findings and concepts on recommender systems and machine learning that were used to develop the two models presented in chapter 3.

CHAPTER 1

LITERATURE REVIEW

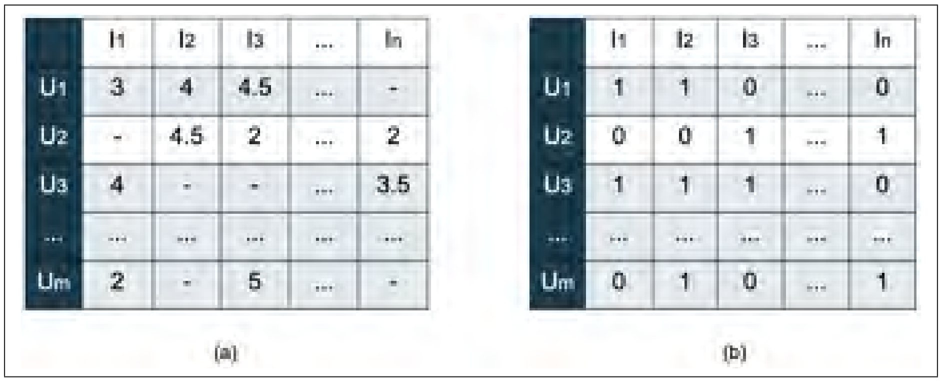
In this chapter, we describe the essential recommendation algorithms and machine learning techniques used in this work. First, we give a brief overview of recommender systems for implicit feedback that inspired our work. We then describe some of the common techniques found in machine learning, namely algorithms in supervised learning like neural networks and RNN, and unsupervised learning like clustering concepts, GMM and VAE.

1.1 Collaborative Filtering

Collaborative Filtering (CF) is one of the most widely used algorithms in the industry due to its wide range of possible applications, from e-commerce websites to social media platforms (Chen *et al.*, 2018). Traditional CF algorithms use the relationships between the users and their items in order to identify new user-item associations. This is done by considering the user explicit or implicit interactions.

Figure 1.1 illustrates two user-item matrices for both explicit and implicit interactions typically used in CF algorithms. Explicit interactions Figure 1.1(a) are usually gathered following an ordinal scale denoting the appreciation of users for specific content. Each value gathered, real or discreet, provides information about the magnitude of the appreciation. Missing values represents items that are not seen by users. They should not automatically be correlated with negative information unlike lower-ranked values on ordinal scales.

While explicit interactions provide clear information about the strength of the relationships between the users and their items, they're often not available in datasets (Chen *et al.*, 2018). This is because the user did not provide any feedback or the system did not implement the ranking functionality. Implicit interactions Figure 1.1(b) are binary values representing interactions between a user and an item, like a view for a video-streaming platform or an item bought for an e-commerce website. It's usually easier to collect such interactions since most of the systems are designed to automatically log actions of the users. While this information is easy to collect, it's necessary to qualify and quantify the binary values gathered: does the positive binary value actually signify a positive feedback (e.g., did the user actually enjoyed the video they were watching, or did they click on the video by mistake) or is this positive binary value actually noise that pollutes the data? Furthermore, unlike explicit data, the implicit binary feedback does not provide any information about the strength of the user-item relationship. The majority of the literature is focused on developing recommender systems for explicit feedback (Hu *et al.*, 2008), probably due to the convenience of this type of interaction. Because of the practical aspect of implicit data (i.e., simple data collection) and the constraints imposed by Radio-Canada, this thesis will focus on exploring algorithms using implicit information.



	I_1	I_2	I_3	...	I_n
U_1	3	4	4.5	...	-
U_2	-	4.5	2	...	2
U_3	4	-	-	...	3.5
...	...	...	...	...	...
U_m	2	-	5	...	-

(a)

	I_1	I_2	I_3	...	I_n
U_1	1	1	0	...	0
U_2	0	0	1	...	1
U_3	1	1	1	...	0
...	...	...	...	...	...
U_m	0	1	0	...	1

(b)

Figure 1.1 Different types of signals handled by recommender systems. (a) corresponds to explicit data with values following an ordinal scale. (b) represents the implicit value where 1 is the interaction between a user and an item, and 0 is the lack of information between a user and an item.

1.1.1 Memory-based CF

Memory-based methods consider the relationships between the users and the items. It creates neighbourhoods that share similar information like similar history for users, or similar metadata for items. It then finds the nearest neighbours, predicts their correlation and provides an ordered list of inter-similarity. Memory-based CF can be subdivided into two more methods: user-based CF and item-based CF.

1.1.1.1 User-Based CF Recommendation

The idea behind user-based CF methods is to find similar patterns between users (i.e., users sharing similar interests) and predict a probability distribution of potential related items in the case of implicit data. Similar to the Nearest Neighbour algorithm (Bhatia & Vandana, 2010), the similarity between users is obtained by comparing their history using a vector-distance metric such as cosine similarity. The Equation 1.1 represents the cosine similarity, and $\vec{r_u}$ and $\vec{r_v}$ are two vectors of implicit values containing the interactions of user u and user v.

$$sim_{uv} = cos(\vec{r_u}, \vec{r_v}) = \frac{\sum_{i \in I_{uv}} r_{u,i} r_{vi}}{\sqrt{\sum_{i \in I_u} r_{ui}^2} \sqrt{\sum_{i \in I_v} r_{vi}^2}} \tag{1.1}$$

Where I_u and I_v are the set of items observed by user u and user v, respectively and I_{uv} denotes the set of items observed by both user u and user v. The cosine similarity between two history vectors is the dot product between the two vectors divided by the product of their norms. Geometrically, it measures the cosine of the angle between the inner product space of two non-zero vectors. In this formulation, the cosine similarity is simply the occurrence and co-occurrence counts: a user u is said to co-occur with user v for every item i they both share.

Once the similarity between users is found, it's necessary to select the closest users before performing the final predictions. The list of closest users is constructed using a k-nearest neighbour technique by selecting k closest users or by considering a threshold θ and satisfying the condition $sim(\vec{r_u}, \vec{r_v}) > \theta$.

Usually, Top-N recommendation are used for implicit feedback information and involve predicting a user's preference for each item in a point-wise way. Each item will have a probability between 0 and 1, the list is then sorted in a descending order and N items will be selected for the recommendation. Let M_u be the list of users that are closest to the user u, $\overline{r_v}$ the average count of positive items for user v, the predicted rating for the user u and item i is:

$$\hat{r}_{u,i} = \vec{r_u} + \frac{\sum_{v \in M_u} sim_{uv}(r_{vi} - \overline{r_v})}{\sum_{v \in M_u} |sim_{uv}|} \tag{1.2}$$

1.1.1.2 Item-Based CF Recommendation

In the item-based recommendation scenario, we capture the similarity of implicit feedback between the items to be recommended. The steps are similar to the user-based method shown previously: we need to find the similarity between items, find the nearest neighbours and predict the probability of the next items to recommend. The similarity between item i and item j is given by the cosine vector technique:

$$sim_{ij} = cos(\vec{r_i}, \vec{r_j}) = \frac{\sum_{u \in U_{ij}} r_{iu} r_{ju}}{\sqrt{\sum_{u \in U_j} r_{iu}^2} \sqrt{\sum_{u \in I_j} r_{ju}^2}} \tag{1.3}$$

Where the set U_i and the set U_j corresponds to the set of users who have interacted with item i and item j, respectively. U_{ij} denotes the set of users who have interacted with both item i and item j. Again, the cosine similarity between two items can be seen as an occurrence and co-occurrence model: an item i co-occurs with an item j every time they both occur with user u. Similar to the user-based model described in the previous section, the closest items for the current item i are found using the k-nearest neighbours or the threshold method.

The final predictions are gathered similarly to the user-item CF prediction described in the previous section where all predictions are made in a point-wise way. A probability distribution is created and indicates the similarity between one item and all the other possible items, with the current item probability set to 0. The list is then sorted in descending order and N items will be selected for the recommendation. Let $\hat{r}_{ij}$ the similarity score between item i and item j with a value between 0 and 1, $\overline{r_j}$ the average count of item j and M_i the similar neighbour set to i:

$$\hat{r}_{ij} = \vec{r_i} + \frac{\sum_{j \in M_i} sim_{ij}(r_{ij} - \overline{r_j})}{\sum_{j \in M_i} |sim_{ij}|} \tag{1.4}$$

1.1.2 Model-based CF

Memory-based CF algorithms are easy to understand and simple to implement but are not well suited for large datasets. This is because of the computation of $\hat{r}$ that uses the entire database every time to make a prediction which is memory inefficient, not very scalable and thus can be considerably slow. Unlike the previous technology, model-based CF algorithms usually requires a learning phase that extract information from the dataset to create a “model”. This is used to make a prediction without the need to use the entirety of the database every time. Learning a prior model makes prediction quickly and more scalable for real-time recommendations.

Among the techniques available, latent-factor models are very popular and provide competitive results Chen *et al.* (2018). Latent-factor models factorize the implicit rating matrix into two low-ranked matrices: the user and item features. To produce these low-ranked matrices, matrix factorization (MF) techniques like Alternative Least Square (ALS) (Hu *et al.*, 2008) or optimization via Stochastic Gradient Descent (SGD) (Jorge *et al.*, 2017) are generally used as well as deep learning techniques (Zhang *et al.*, 2019).

1.1.2.1 Matrix Factorization

Recommender systems for implicit data often deal with sparse matrices. Traditional algorithms for MF that uses Singular Value Decomposition (SVD) are not defined for sparse matrices and cannot be applied for implicit problems. Many alternatives have been proposed to deal with such matrices, we show two of these techniques.

The MF problem is described as follows:

$$R = UV^T \tag{1.5}$$

Where R is the implicit rating matrix with m users and n items $R_{m \times n}$. MF decomposes R into two matrices sharing a common k-dimensional latent feature space. U is the k-dimensional latent feature matrix for the users $U_{m \times k}$ and V is the k-dimensional latent feature matrix for the items $V_{n \times k}$. The common latent feature space describes the users and items such that R is approximated by $\hat{R} = UV^T$. Figure 1.2 illustrates the MF problem. The final prediction for a user u and item i is the dot product between the two latent matrices:

$$\hat{R}_{ui} = U_u \cdot V_i \tag{1.6}$$

The goal of MF is to factorize the original matrix into two latent matrices such that the dot product between the two minimizes the deviation between the approximation and the original matrix. The model size is defined by the number of features k available. The number of features is a hyper parameter and should be tuned depending on the problem. A small k might not capture enough information about the data, while a larger k might lead to overfitting. The objective function is given by:

$$L = \min ||R - \hat{R}|| \tag{1.7}$$

This minimization problem can be solved by using iterative optimization techniques such as SGD to estimate the parameters of the model. In this setting, the objective function becomes:

$$L = \min ||R - \hat{R}|| = \min_{U,V} \sum_{(u,i) \in D} (r_{u,i} - U_u V_i^T)^2 + \lambda_u (||U_u||)^2 + \lambda_i (||V_i||)^2 \tag{1.8}$$

Where λ_u and λ_i are necessary to regularize the model and control overfitting by penalizing weights with high magnitudes which leads to poor generalizations. SGD continuously updates the parameters of the model according to the direction of the gradient until convergence of the objective function or a predefined number of iterations. This algorithm was used in many different applications (Takács *et al.*, 2008; Koren, 2008; Paterek, 2007; Takáes *et al.*, 2009) and has many benefits when dealing with large datasets that cannot be held in memory. This is done by performing the parameter updates on each training samples unlike vanilla gradient descent, which needs to see the entire dataset before making the update.

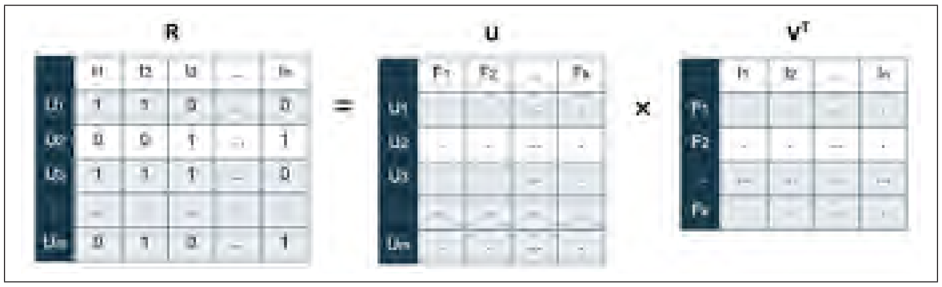


Figure 1.2 A representation of the matrix factorization algorithm where the user-item interaction matrix R is approximated with the user-factor matrix U and item-factor matrix V. Common latent factors between U and V are learned during the training stage.

Formally, SGD computes the gradient of each parameter θ for each training example x^i and label y^i:

$$\theta = \theta - \eta \cdot \nabla_\theta J(x^i, y^i; \theta) \tag{1.9}$$

Where θ is the current parameter to update, η is a factor controlling the magnitude of the gradient value and is lower or equal to 1. $\nabla_\theta J(x^i, y^i; \theta)$ is the gradient of the loss function J for the training example x^i and label y^i.

Following our problem on minimizing L described above, SGD starts by randomly initializing the parameters of the matrix U and matrix V. Then, given a training set with a tuple in the form (u, i, r) corresponding to the active binary signal r for user u and item i, the algorithm will iteratively update the values of U_u and V_i following the opposite direction of the gradient of the error until the stopping criterion is met. The error is the difference between the true value and the predicted value $e_{ui} = r_{u,i} - \hat{r}_{u,i}$ which can be further simplified to $e_{ui} = 1 - \hat{r}_{u,i}$ due to $r_{u,i}$ being binary. The update for the user and item matrix are given by Jorge *et al.* (2017):

$$U_u \leftarrow U_u - \eta \frac{\partial L}{\partial U_u} \tag{1.10}$$

$$V_i \leftarrow V_i - \eta \frac{\partial L}{\partial V_i} \tag{1.11}$$

$$U_u \leftarrow U_u - \eta(e_u i V_i - \lambda_u U_u) \tag{1.12}$$

$$V_i \leftarrow V_i - \eta(e_u i U_u - \lambda_i V_i) \tag{1.13}$$

An alternative method for matrix factorization is the ALS algorithm (Hu *et al.*, 2008). In this setting, the authors consider the implicit data problem that is not bound to the binary domain. Hence, the positive interaction $r_{u,i}$ can represent the number of times an item i was purchased by user u in the case of an e-commerce website, and the frequency a content i was watched by a user u in the case of a streaming platform. It's also possible to use this algorithm with implicit binary data by disregarding the extra information that higher values can provide (e.g., a higher preference for a content).

An important distinction between ALS and other algorithms is the notion of "confidence" which grows linearly with the value of $r_{u,i}$ and measure the confidence of observing p_{ui}. Here, p_{ui} is the binarized version of R. If the value for a user u and item i is higher than 1, p_{ui} is equal to 1, otherwise 0. The value of the confidence is given by:

$$c_{ui} = 1 + \alpha r_{u,i} \tag{1.14}$$

Hence, there's a minimal confidence value for all item-user pairs, but as the observation increases for $r_{u,i}$, the confidence of observing p_{ui} also increases. In the implicit binary data problem, the confidence value is the same for all observed interaction in the dataset. Again, the preferences are assumed to be the inner product between the user and item matrix $p_{ui} = U_u^T V_i$. The common latent space shared by U and V can be optimized by minimizing this cost function:

$$L = \min_{m,n} \sum_{u,i} c_{ui}(p_{ui} - U_u^T V_i)^2 + \lambda(\sum_u ||U_u|| + \sum_i ||V_i||)^2 \tag{1.15}$$

Assuming a single regularization factor λ for simplicity, the following cost function considers the entire user-item dataset taking into account all the varying confidence levels. Unlike the previous method where SGD was used to find the optimal parameters of U and V for a sparse binary implicit dataset, optimizing this objective function can be slow and costly due to the $m \times n$ domain.

By alternatively fixing the user-factor matrix U and item-factor matrix V, the cost function becomes quadratic and the global minimum can be computed. The analytic expressions for U and V that minimize the cost function are given by the authors:

$$U_u = (V^T C^u V + \lambda I)^{-1} Y^T C^u p(u) \tag{1.16}$$

$$V_i = (U^T C^i U + \lambda I)^{-1} U^T C^i p(i) \tag{1.17}$$

C^u and C^i is the $m \times m$ diagonal matrix for user u where $C^u_{ii} = c_{ui}$ and the $n \times n$ diagonal matrix for item i where $C^i_{uu} = c_{ui}$, respectively. $p(u) \in \mathbb{R}^n$ is the preference vector of u and $p(i) \in \mathbb{R}^m$ is the preference vector of i.

We refer interested readers to the original article for further information on the optimization process and additional computational tricks.

1.1.3 Challenges

In standard scenarios where the users or the items have sufficient interactions, CF-based techniques usually perform well and provide diverse and interesting recommendations. These models require users to have at least one interaction, otherwise CF cannot come up with recommendations. Furthermore, most systems require a minimal number of interactions to perform well. This scenario is called the “cold-start” problem. As discussed previously, when the number of users and items increases, the scale of the data also increases exponentially. The sparsity of the data can become challenging and in some cases it’s a problem depending on the algorithm used. CF is based on the assumption that users with similar tastes or preferences share similar interaction histories, and can become inaccurate if there are too many users with rare or unique tastes. This unique preference problem is called the “grey sheep” issue.

1.2 Content-Based Filtering

Unlike CF methods, which make neighbourhoods of common tastes and preferences only based on the interaction histories, content-based filtering (CBF) makes recommendations based on the metadata of the items. Nowadays, most systems contain some form of prior knowledge on the items. In an e-commerce setting, the metadata includes the item title, description, brand, categories, while on a music streaming platform, metadata can be the song name, artist, album, genre and release date.

Most of the CBF systems use the available information on the items, along with some engineered features to provide the final recommendation. For example, in a music recommendation system, some might analyze the audio files to determine features such as loudness, tempo and timbre (O'Bryant, 2017). Typical CBF systems consist of three different parts: feature extractions, user profiles creation and recommendations.

1. Feature extractions is the first part of the system where available items are aggregated and analyzed to produce a matrix of items as rows and features as columns. Several algorithms have been proposed to extract relevant information on different modalities and transform them in a more abstract and useful format (Seyednezhad *et al.*, 2018). One example of this transformation is the Vector Space Model (VSM) where textual information is transformed into a spatial representation. Text documents become an *n*-dimensional space where each dimension is a term shared across documents. A commonly used technique to extract relevant terms from documents is the Term Frequency Inverse Document Frequency (TF-IDF) (Ramos, 2003). The rationale of this technique is to specify frequently found terms in a document (TF) that are rarely in other documents (IDF). Terms that are common in a smaller group of documents tend to have a higher TF-IDF value than common terms shared across all documents. When computing TF, we can normalize all terms occurring in a document by its maximum TF and avoid biases for longer or shorter documents (e.g.,

longer documents tend to repeat the same word):

$$\text{TF}(t_i, d_j) = \frac{f_{i,j}}{\max\limits_{1 \leq k \leq |T|} (f_k, j)} \tag{1.18}$$

Where t_i is the i-th term in the vocabulary $T = \{t_1, t_2, \ldots, t_m\}$, d_j the j-th document for the collection $D = \{d_1, d_2, \ldots, d_n\}$, $f_{i,j}$ the frequency of the i-th term in the j-th document (i.e., term count in the document) and $\max\limits_{1 \leq k \leq |T|} (f_k, j)$ the maximum frequency of all the terms in the j-th document. The second term, IDF is the log of all the documents D given the number of documents where the i-th term is present:

$$\text{IDF}(t_i) = \log \frac{|D|}{|\{d_j : t_i \in d_j\}|} \tag{1.19}$$

The final equations of TF-IDF is the product of the two terms:

$$\text{TF-IDF}(t_i, d_j) = \text{TF}(t_i, d_j) \cdot \text{IDF}(t_i) \tag{1.20}$$

2. Creating the user profile is done by aggregating the pre-processed information in the previous step into a single matrix of interacted items and extracted features. The final item-feature matrix is used by the recommender algorithm and is continuously updated with new interactions.

3. Recommendations are based on the pre-processed features contained in the user profile and are made by using a similarity measure between the available items and the profile. In a VSM where the spatial representation of the documents is done by TF-IDF, the cosine similarity between the normalized terms in the documents is commonly used.

$$w_{i,j} = \frac{\text{TF-IDF}(t_i, d_j)}{\sqrt{\sum_{k=1}^{|T|} \text{TF-IDF}(t_k, d_j)}} \tag{1.21}$$

where $w_{i,j} \in [0,1]$ is the weights for the i-th term in the j-th document. The cosine similarity between two documents is as follows :

$$\text{sim}(d_i, d_j) = \frac{\sum_k w_{k,i} \cdot w_{k,j}}{||w_{k,i}|| \cdot ||w_{k,j}||} \tag{1.22}$$

1.2.1 Challenges

CBF methods perform really well when recommendations need to be fitted exactly to a user's tastes or profiles. They also do not suffer from scalability problems since the dimensions of the item-feature matrix are fixed. However, CBF-based systems usually create a "filter-bubble" around the user by always recommending items that will be closely related to each other. Serendipity is then affected and discoverability of new items is low. Depending on the context, this can be an important problem, e.g.: in a newspaper's website, a CBF recommender system might always suggest articles with the same political views. This might not be ideal in an election period when the user needs to be informed of the ideas of the different parties. A "cold start" problem usually appears when not enough interactions are provided. One of the solutions is to use similar characteristics between items, but finding the best features that optimize the similarity between items may be difficult to calculate.

1.3 Hybrid system

Hybrid recommender systems usually combine the prediction of two or more models to make the final prediction, or use more complex models that are able to encode more information into the user-item matrix. These hybrid systems are closely related to the field of assemble analysis in standard classification tasks. They aim to improve the following, (1) recommendation accuracy, (2) discoverability, (3) mitigate the "cold-start" problem, (4) data sparsity, (5) popularity bias and (7) scaleability problem to name just a few. A taxonomy of these systems was created by Burke (2002) and has 6 different categories.

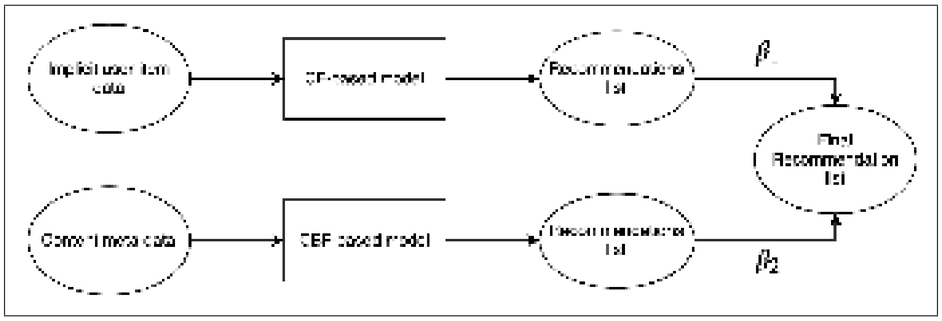


Figure 1.3 A combination of a CBF algorithm and CF algorithm in a weighted setting to make the final recommendation. CF models suffer from the “cold-start” problem and are not able to produce recommendations to new users. CBF methods are less affected by the "cold-start” problems but are not as accurate as CF methods in standard scenarios when discoverability is desired. By combining the prediction of these two systems, it’s possible to balance the strengths and weaknesses of these algorithms and generate more accurate recommendations.

1.3.1 Weighted

In this setting, all models produce a probability score that is combined using a weighted linear function to produce the final output (Çano & Morisio, 2017). Let k be the k-th recommender model, β_k the weight of the recommendation of model k contributing to the final prediction:

$$r_{u,i} = \sum_k \beta_k * r_{ui}^k \tag{1.23}$$

One example of such system is P-Tango which combined CF and CBF models to recommend newspaper articles (De Campos *et al.*, 2010). One particularity of this system is that the weights are set on a per-user basis, this produces the optimal combination of the two models to make the best recommendation. This also reduces problems like the “grey sheep” and the “filter-bubble”.

1.3.2 Feature combination

In this setting, these systems usually aggregates all outputs of previous systems into a single algorithm to make new recommendations. In Bedi *et al.* (2013), the authors presents a three-step book recommender system. The first step involves a CF algorithm that finds the preferred books. The second step creates book categories for each type of users, and in the final step they use the generated features in a CBF system.

1.3.3 Cascade

Similarly to a weighted setting, cascade hybrid recommender systems first produce coarse ranking candidates, while subsequent systems work only on the items that need additional refinement. This technique is more efficient than the weighted method since the subsequent models only use the previous sets of ranked items, which limits the domain of computation. This class of hybrid system is order-dependant. The ranked list generated by the first system and used by the second might be different if the second system was the first. In Lampropoulos *et al.* (2012), a music recommender system has two levels. The first extracts the audio features from a song submitted by a user using a Support Vector Machine (SVM). The second level combines the extracted features with the user's rating of the audio query, and a Bayesian Network (Horný & Lin, 2014) is used to provide the recommendation.

1.3.4 Switching

These hybrid systems will behave differently according to certain criteria. They might use a CBF model in the presence of a "cold-start" scenario and then a CF model when there are sufficient interactions available. In Billsus *et al.* (2000), the authors present "DailyLearner", a new system that distinguishes between a short-term CBF and long-term CBF recommender. It first uses the short-term model and considers the newest stories by extracting textual features using TF-IDF to find the nearest neighbours. If there are no nearest neighbours, the system uses the long-term model which considers the user profile.

1.3.5 Feature augmentation

This hybrid system uses the models sequentially by using the output of the previous model for the input of the next one. For example, one model might give similar items as outputs and can be used as augmented item attributes for the next recommender system. A CBF book recommender called Libra (Mooney & Roy, 2000) uses two models, the first creates features representing similar authors and items and the second uses these features along with others to create the final recommendation.

1.3.6 Meta level

This class of hybrid models are similar to feature augmentation systems, but instead of using the output of one model as input, it considers the entire model as input for the second one. Fab (Balabanović & Shoham, 1997), one of the first website recommender systems uses four components (or "agent") to recommend the most relevant websites. The first uses a CBF to build user profiles based on areas of interest. The second system searches for available websites using the interests of the users from their profile. The third filters the website to avoid duplicates, and the fourth system use a CF model to collect the most relevant websites.

1.3.7 Mixed

A mixed hybrid model uses the recommendations of two or more systems to produce the final list of recommendations (with recommendations from all systems presented at the same time). In a recommendation TV program (Yoshii *et al.*, 2008), the authors use a CF and a CBF system to produce a list of recommendations. The first system represents the interests of users similar to the active user. The items that are more probable to be liked by the users are represented by the second system.

1.3.8 Challenges

Using more than one recommender system has many advantages and can achieve high performances by leveraging their benefits while alleviating problems and challenges arising by only using one type of recommender system. By combining CBF to CF, it's possible to handle the "cold-start" scenarios as well as breaking from the "filter-bubble" and the "grey sheep" problem. There are multiple advantages in using a hybrid architecture but it requires a far greater effort since more than one system need to be engineered and tuned depending on the class of hybrid system considered.

1.4 Measuring performance

Several metrics are used to evaluate the efficiency of recommender systems and are used to compare different solutions for the same problem. These metrics evaluate the accuracy and the rank of the prediction, as well as the coverage and the diversity of the recommendations.

The Mean Absolute Error (MAE) is used to calculate the recommender system's accuracy and measure the average error in the predicted set against the true set of items. This metric weights the outliers or large error terms equally to the other predictions. Let W the items of a user u, $r_{u,i}$ and $\hat{r}_{u,i}$ the true value (binary value in the case of implicit datasets) and predicted value:

$$\text{MAE} = \frac{\sum_{(u,i)\in W} |r_{u,i} - \hat{r}_{u,i}|}{|W|} \tag{1.24}$$

The Root Mean Squared Error (RMSE) is derived from the MSE and evaluates the absolute difference between the predicted recommendations and the true recommendation. Unlike MAE, RMSE is prone to be affected by outliers or large errors since these terms will be squared.

$$\text{RMSE} = \sqrt{\frac{\sum_{(u,i)\in W} (r_{u,i} - \hat{r}_{u,i})^2}{|W|}} \tag{1.25}$$

One challenge faced when using sparse data is the long-tail problem. As described by Aggarwal (2016), it often appears in recommender system scenarios due to the small number of interactions in the user-item matrix. This affects the accuracy measurement since popular items appear more frequently and will often be recommended compared to sparsely interacted items. It's possible to give items specific weights when calculating the MAE and RMSE to penalize popular items that are wrongly recommended.

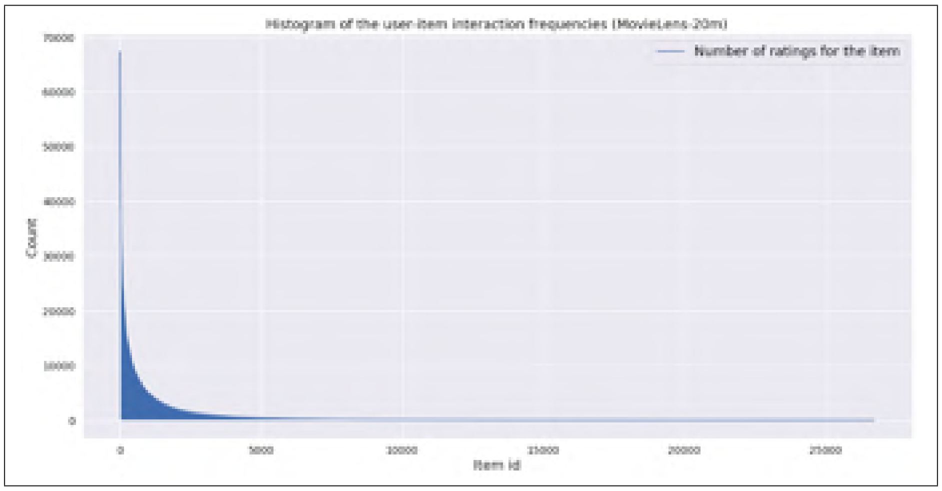


Figure 1.4 Long tail problems in the MovieLens-20M (Harper & Konstan, 2015) recommender dataset. The first 5000 movies have the most interactions, while the remaining movies have very few interactions and represent the "niche" movies.

It's often important to evaluate a subset of recommended items instead of the complete set of recommendations like MAE and RMSE. Precision@k, Recall@k and F1@k measures the ratio of the recommended items in the top-k recommendation to the actual true set. A higher value is preferred:

$$\text{Precision@k} = \frac{|\text{items in recommendation set} \cap \text{top-k recommended items}|}{k} \tag{1.26}$$

$$\text{Recall@k} = \frac{|\text{items in recommendation set} \cap \text{top-k recommended items}|}{|\text{items in recommendation set}|} \tag{1.27}$$

While the Precision@k indicates the proportion of the best k recommended items that are in the top-k recommendation, the Recall@k indicates the proportion of the best k recommended items that are in the true recommended item sets. The F1@k is the harmonic mean of the Precision@k and Recall@k:

$$\text{F1@k} = 2 \times \frac{\text{Precision@k} \times \text{Recall@k}}{\text{Precision@k} + \text{Recall@k}} \tag{1.28}$$

These metrics are ideal when the rank of the recommended items are not considered. When the rank is important, three main metrics are used: Mean Reciprocal Rank (MRR), Discounted Cumulative Gain (DCG) and Normalized Discounted Cumulative Gain (NDCG).

MRR@k is the multiplicative inverse position of the first relevant item among the first k. Let $rank_i$ the true rank of item i in the set:

$$\text{RR}_i@k = \frac{1}{\text{rank}_i} \tag{1.29}$$

$$\text{MRR@k} = \frac{\sum_{u \in U} \text{RR}_i@k}{|U|} \tag{1.30}$$

DCG assumes a logarithm decay in the user's interests and measures the gain of an item based on its position in the top-k list. The IDCG represents the ideal (I) scenario if the recommendation list is sorted and the NDCG is the normalized version of the DCG. Let $|REL - k|$ the list of items ordered by their relevancy until position k:

$$\text{DCG}@k = \sum_{i=1}^{k} \frac{2^{r_{u,i}} - 1}{log_2(i+1)} \tag{1.31}$$

$$\text{IDCG}@k = \sum_{i=1}^{|\text{REL}-k|} \frac{2^{r_{u,i}} - 1}{log_2(i+1)} \tag{1.32}$$

$$\text{NDCG}@k = \frac{\text{DCG}@k}{\text{IDCG}@k} \tag{1.33}$$

Measuring the accuracy of predicted items is useful when comparing multiple systems. It's also often important to measure the ability of the system to discover long tails in the data.
In other words, how much of the possible items in the dataset is the system able to recommend, or how much coverage does the system provide?

$$\text{Coverage@k} = \frac{|\text{top-k unique recommended items}|}{|I|} \tag{1.34}$$

1.5 Machine Learning

Machine learning (ML) is an application of artificial intelligence (AI) that provides systems the ability to acquire new knowledge based on data from the real world without being explicitly programmed. ML techniques enables systems to solve problems via self-learning, like pattern recognition, language modelling, image generation, classification and prediction. Deep Learning (DL) is the recent evolution of ML and is specific to Neural Networks (NN) that use many layers stacked together to create Deep Neural Networks (DNN). ML techniques are not recent and were introduced in the early 50s with statistical methods, Bayesian methods and probabilistic inference. Nowadays, Artificial Neural Network (ANN), especially Recurrent Neural Network (RNN) and Convolutional Neural Network (CNN) are popular models. In this chapter we explain the different machine learning strategies as well as some powerful variants of ANNs.

1.5.1 Supervised Learning

Learning is useful when we want to perform an intricate task that is hard to program with traditional techniques. For example, in speaker identification where we try to associate a raw waveform to its author or in image classification where the task is to identify the objects inside a picture. Supervised learning requires to know the real labels Y associated with each samples X used by the ML model to learn the statistical distribution of the input dataset $D = (x^{(1)}, y^{(1)}), (x^{(2)}, y^{(2)}), \ldots, (x^{(n)}, y^{(n)})$. Knowing the real label provides the model the capability to correct its prediction error, similar to a teacher or supervisor that knows the final answer and is able to correct their students.

Given a set of data, the label corresponding to each sample $x^{(i)}$ is known. The output error, or error signal, usually corresponds to the distance between the model output and the true label. More formally, let X be an input space and $\hat{Y}$ an output space. The goal of supervised learning is to learn a function $f : X \rightarrow \hat{Y}$ whose test error minimizes the distance between the true labels Y and predicted labels $\hat{Y}$ given an input sample $x^{(i)}$: $\text{test}(f) = L(f(x^{(i)}), y^{(i)})$. Where:

$L(f(x^{(i)}), y^{(i)})$ is the function computing the distance between the prediction and the ground truth.

During the training phase, we iteratively use all the training examples to correct the parameters of the model via Gradient Descent (GD) optimization or memory-based optimization.

With memory-based optimization, the model uses all the data seen in the training phase to perform the prediction. One implementation of such algorithms is the K-Nearest Neighbours (Guo *et al.*, 2003) that captures the idea of "similarity" and involves computing the distance between points in a n-dimensional space. The model calculates the distance, usually the Euclidian distance, between each samples from the training set in memory and the query. These distances are then sorted and the k-samples with the smallest distances are chosen. In a regression setting, the output is usually the mean of the k-labels. In a classification setting, the mode of the k-labels is usually used.

In memory-based optimization, or non-parametric methods, we seek to create models that directly or indirectly use the training data and do not make strong assumptions about the data distribution. The number of parameters usually grows with the amount of data used by the model. With parametric methods, we assume a specific distribution that summarizes the training dataset and learns its parameters. Unlike the previous method, the number of parameters used by the model is fixed.

An example of a parametric method is the parametric density estimation technique where we try to estimate the density of the observed data with a chosen probability distribution. The parameters of the distribution are computed with the goal of recovering the underlying probability density function. An example of such technique is by using a Gaussian distribution to estimates the density of the observed dataset. This distribution has two parameters: the mean and the variance and can be estimated by measuring the sample mean and sample variance as shown in Figure 1.5. This is a naive example where the parameters can be evaluated directly. Unfortunately, when the underlying distribution is more complex, finding the right set of parameters is not trivial.

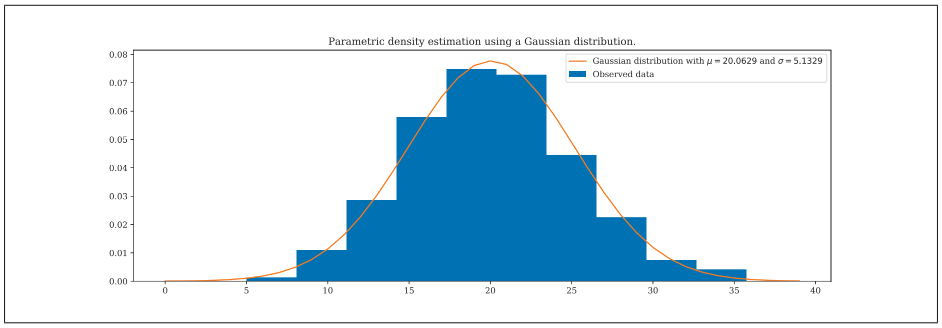


Figure 1.5 Parametric density estimation using a Gaussian distribution to estimates the underlying distribution. The mean and variance of the distribution is found by computing the sample mean and sample variance of the observed data.

Inferring the right parameters can be done by different algorithms. A popular choice is the GD algorithm. At each operation, the gradient of the error is applied to the parameters of each operation to correct their values and minimize future errors. This is similar to SGD used to compute the matrix update of U and V presented in the last chapter. Recall that U_u and V_i were iteratively updated for user u (Equation 1.10) and item i (Equation 1.11). Notice that the update process between both matrices is similar. We define the iterative process by $\nabla F(\theta_t)$ the gradient of the function F with parameters θ at index t and η the learning rate. The parameters are updated at each iteration: $\theta_{t+1} \leftarrow \theta_t - \nabla F(\theta_t) \cdot \eta$.

The previous formulation (Equation 1.10, 1.11) is recovered when the gradient of the function is taken w.r.t to U or V. This recursive computation, i.e., the forward pass when the model output its predictions and the backward pass when the model correct its parameters using GD, is executed until the network has converged or has reached a stopping criterion. Without the optimization process, no parameters are updated and the network would not be able to learn the underlying distribution generating the dataset.

1.5.1.1 Neural Network

Neural Networks (NN) are a family of models that were inspired by the way human brain processes information to deduce and find patterns. Early algorithms were intended to model how biological learning took place (Goodfellow *et al.*, 2016). This enables computers to learn the underlying statistical distribution of the data and predict unseen data. To their simplest form, NN use perceptrons which work with several inputs $X = x_1, x_2, \ldots, x_n$ and produce one binary output y. Each perceptron uses a set of parameters, or weights, to express the "importance" of each input. The weighted sum over each input gives the final output according to the weights and optional bias parameters.

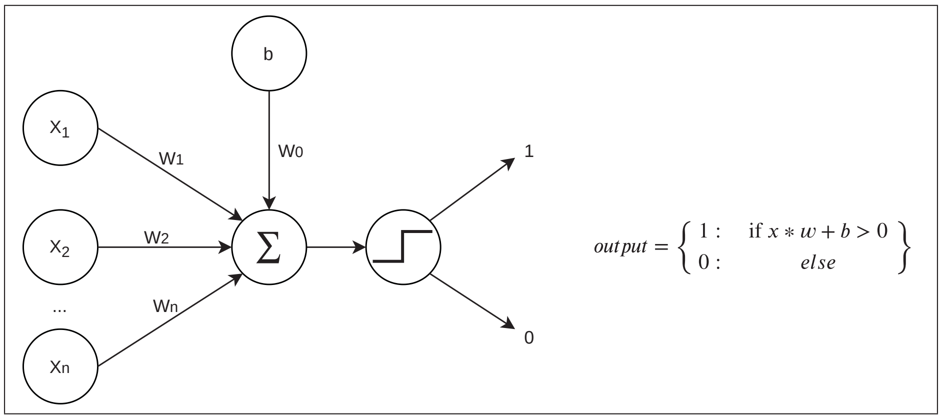

b
X1
X2
...
Xn
W1
W2
Wn
W0
Σ
1
0
$output = \left\{ \begin{array}{ll} 1: & \text{if } x * w + b > 0 \\ 0: & else \end{array} \right\}$

Figure 1.6 A single perceptron. The figure shows a perceptron with N inputs X and parameters W with optional bias b.

A single perceptron is able to linearly separate the input space. By adding a second layer of perceptrons, it's possible to express more complex functions by increasing the complexity of the separation in the input space. If multiple perceptrons are added in the output layer, it's possible to extend the model to work on classification tasks with multiple categories. In such settings, the output is usually a probability distribution over the possible categories. To predict the probability associated with a multinoulli distribution, the softmax function is often used:

$$\text{softmax}(x) = \frac{\exp(x_i)}{\sum_{i=1}^{n} \exp(x_i)} \tag{1.35}$$

The linear characteristics of perceptrons makes them hard to train and not very practical. Not all problems are always linearly separable and require a considerable quantity of perceptrons to solve the problem. By introducing nonlinear activations, the input space can be separated non-linearly. A popular choice is the sigmoid activation function.

$$\text{sigmoid}(x) = \frac{1}{1 + \exp(-x)} \tag{1.36}$$

Furthermore, without an activation function, the output domain is unbounded $[-\inf, \inf]$. With a sigmoid function, the output of the activation will always be between $[0, 1]$. Unfortunately, this function suffers from the vanishing gradient problem if the value of the inputs are far from zero, i.e., the gradient at then end of the sigmoid function is small.

An alternative to the sigmoid function is the tanh function. Unlike the first, this function has an output domain of $[-1, 1]$ and has a steeper gradient.

$$\tanh(x) = \frac{2}{1 + \exp(-2x)} - 1 \tag{1.37}$$

Like sigmoid, this activation function also suffers from the vanishing gradient problem. A popular alternative to sigmoid and tanh is the rectifier linear unit (ReLU). Unlike the previous activation function, ReLU is not bound. The values are set to zero if they're negative: $[0, \inf]$.

$$\text{relu}(x) = \max(0, x) \tag{1.38}$$

With sigmoid and tanh, the activation is dense because almost all the neurons are used to describe the output. However, some neurons might not be useful and should be discarded. By setting the values to zero, ReLU makes the activation sparse and the network efficient by using fewer values in its output. A consequence of negating negative values is the "dying ReLU" problem. Negative neuron's value is set to zero and has a gradient equal to zero. Their weights will not be adjusted and the neurons will not respond to new inputs. A solution to this problem is to use a factor β to decrease the magnitude of the value instead of setting it to zero. This activation function is called the Leaky ReLU:

$$\text{LeakyReLU}(x) = \begin{cases} x & \text{if } x > 0, \\ \beta x & \text{otherwise.} \end{cases} \tag{1.39}$$

Increasing the number of parameters available gives the network the ability to solve more complicated challenges, such as image recognition, speech synthesis and temporal prediction. However, a situation when the network is too closely fitted to the training data and is unable to generalize on unseen data might arise. This situation is called overfitting and is a fundamental problem with NN. Some solutions exist to mitigate this issue. A popular technique is called "dropout" (Srivastava *et al.*, 2014) and consist of randomly deactivating (i.e., set the value to zero) some neurons in a layer. This is able to sever the links that might appear between neurons which might lead to poor generalizations.

Finally, monitoring the learning process of the network is done with the loss function, also called the cost function. Typically, this function measures how well the network predicted a value given the expected output. For example, the Mean Squared Error (MSE) is generally used when comparing the distance in a vector space. In this setting, the goal of the network is to minimize the distance between the prediction and the true value until convergence or if the learning process is stopped. The process of minimizing the distance between the ground truth and the output of the network is called optimization and is usually done via GD in the case of supervised learning and reinforcement learning. Expectation-Maximization, Bayesian optimization or other learning algorithms are used for unsupervised learning.

1.5.1.2 Recurrent Neural Network

Recurrent Neural Networks (RNN) are a variant of NN specialized in processing sequential information via their nonlinear dynamic systems that allows past information to persist (Sutskever & Ilya, 2013). They're especially useful to learn the underlying pattern in long-term data. Unlike traditional NN, their weights are shared across time and are reused along with the new input to make the prediction.

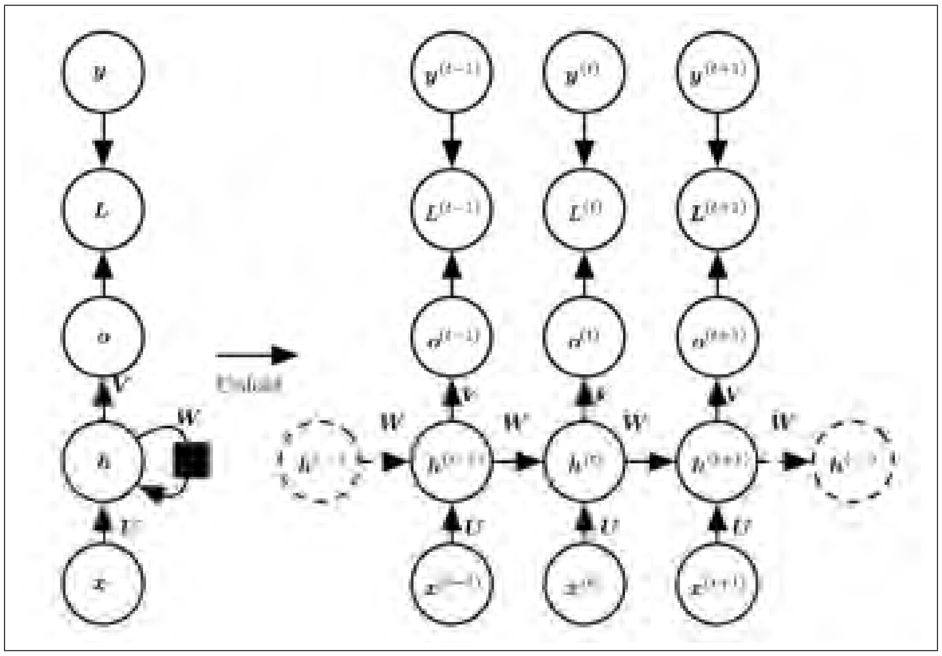


Figure 1.7 Unfolding the RNN (Goodfellow *et al.*, 2016). The general directed graph is represented on the left side of the image into separate time steps. From the bottom to the top: an input $x^{(t)}$ is used by the network with a weight matrix U along with the previous hidden state $h^{(t-1)}$ and weight matrix W to create the next hidden state $h^{(t+1)}$. The output $o^{(t)}$ is the non-normalized log probabilities used by the loss function and training target $y^{(t)}$ to evaluate the quality of the prediction.

Given an input sequence $X = x_1, x_2, \dots, x_T$ and a sequence of hidden state matrices $h_1, h_2, \dots, h_T$ where a state matrix h corresponds to a time step t, the next state is given by considering the parameters of the model, the previous hidden state and the current input:

$$h^{(t)} = f(h^{(t-1)}, x^{(t)}; \theta) \tag{1.40}$$

Where f is a non-linear activation function. When predicting the future given the past, the state $h^{(t)}$ is a lousy summary of the past sequence of inputs up to t (Goodfellow *et al.*, 2016).

The loss function will generally determine which information should be kept or removed from the hidden state of the network, e.g., when predicting the next word in a sentence, the previous words might not be all relevant and should be removed from the memory to allow new, more important information. The cost function of RNNs usually concatenate the sum of the losses at each time steps before summing them.

Modelling long-range temporal dependencies is often difficult (Y. *et al.*, 1994; Martens J., 2011; S. & J., 1997) due to the iterative nonlinear system of RNNs. During the optimization process, the derivative of the weights at each time steps are multiplied together. Thus, if the values are considerably smaller than one, the gradient decreases exponentially as it is propagated. This problem is called the “vanishing gradient problem” and was first described by S. & J. (1997). It’s an undesirable effect since it makes RNNs unable to learn and exploit long-term information. Explosions of the gradient are another factor that contributes to the difficulty of training RNNs. This happens when the gradient is too large early in the optimization process and grows exponentially as they are multiplied together. Exploding gradients can be solved by constraining the gradient’s value to a threshold unlike vanishing gradient that requires a special architecture.

Long-Short Term Memory (LSTM) (S. & J., 1997) was introduced to effectively mitigate the vanishing gradient problem. Theoretically, traditional RNN models are able to remember the complete long-term dependencies. Unfortunately, storing all the temporal information has proven to be a difficult task and might not always be the ideal solution. LSTM introduced the concept of "gates" where every gate has their own task and control the flow of information. Let W the weight matrix of the inputs x at time step t, U the weight matrix of the hidden state h and b the bias of the gate. $\bigodot$ is the element wise product and C is the LSTM cell state.

The LSTM gates, hidden state and cell state are defined as follows:

$$f_t = \text{sigmoid}(W_f x_t + U_f h_{t-1} + b_f) \tag{1.41}$$

$$i_t = \text{sigmoid}(W_i x_t + U_i h_{t-1} + b_i) \tag{1.42}$$

$$o_t = \text{sigmoid}(W_o x_t + U_o h_{t-1} + b_o) \tag{1.43}$$

$$\widetilde{c_t} = \tanh(W_c x_t + U_c h_{t-1} + b_c) \tag{1.44}$$

$$c_t = f_t \odot c_{t-1} + i_t \odot \widetilde{c_t} \tag{1.45}$$

$$h_t = o_t \odot \tanh(c_t) \tag{1.46}$$

The forget gate f sets the weight of the information between 0 or 1 via the sigmoid activation function and controls the amount of information that should be discarded. The input gate i controls the amount of information that should be remembered and stored in the cell state. The output gate o filters the information based on the cell state C. LSTM solves the problem of vanishing gradient with a connection between the forget gate and the gradient computation by removing the exponentially fast decaying factor involved in the original RNN formulation.

Traditional unidirectional LSTM networks are able to preserve information of the past from inputs that have already been seen by using their hidden state. However, when information from the past and the future is required to understand the context and make the best prediction, unidirectional networks might fall short. Bidirectional RNNs (BRNNs) (Schuster & Paliwal, 1997) are able to understand the future dependencies in the input by analyzing the data in two ways. The first way is the same as a unidirectional architecture, from the past t_0 to future $t_{|T|-1}$. The second way is to analyze the input from the future $t_{|T|-1}$ to past t_0. Hence, unlike traditional RNNs, BRNNs also analyze the input backwards. The results of the last forward hidden state and last backward hidden state is concatenated $h = [h_{\rightarrow}, h_{\leftarrow}]$. The final vector contains the information from both past and future.

In their simplest form, BRNNs suffer from the same problems as their unidirectional counterparts. Using two LSTMs (BiLSTMs) for both the forward and backward pass is a solution.

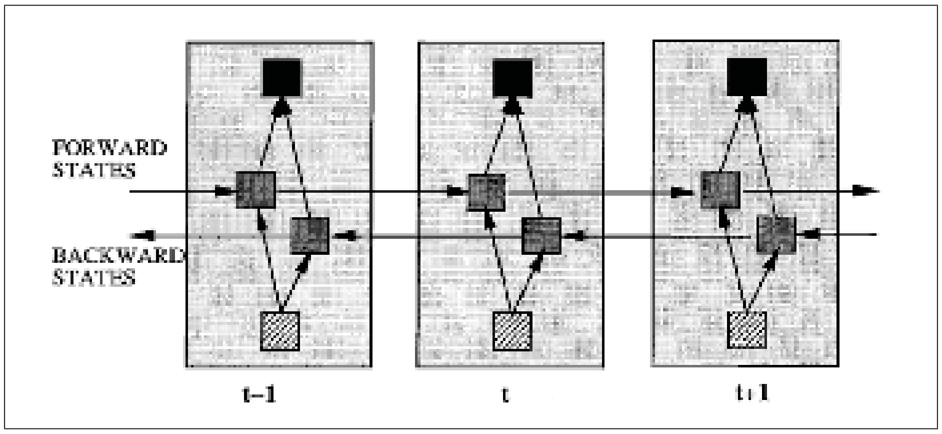


Figure 1.8 The forward state and backward state are shared across time steps (Schuster & Paliwal, 1997). The output of a BRNN is a concatenation between last states.

1.5.2 Unsupervised Learning

In this strategy, the ground truth of the training data is not available. The goal of unsupervised learning is to capture the underlying patterns and irregularities in the input data without knowing the error signal to evaluate the potential solution. Unsupervised learning involves three different tasks: clustering, density estimation and dimensionality reduction.

1.5.2.1 Clustering and dimensionality reduction

Clustering and dimensionality reduction are two of the many applications of unsupervised learning. Recall that unsupervised algorithms are those that only use the features without knowing their labels. While clustering is able to uncover similar patterns in the dataset and group them into clusters, dimensionality reduction defines a simpler representation of the data by compressing as much information as possible into a lower-dimensional feature space.

Clusters are usually created in multiple steps and involve measuring the distance between samples, grouping the samples into several clusters based on their distance and estimating the quality of the groups (i.e., the intra-cluster cohesion and inter-cluster separation). Each cluster is a collection of data that is similar within each other and dissimilar to data in other clusters.

After creating the clusters, labels can be assigned to each samples in the groups and then used in supervised algorithms.

A well-known representation learning algorithm and commonly used partitioning method is called k-means (Ali & Kadhum, 2017). The algorithm creates k clusters from the training set by iteratively measuring the distance of the data to the cluster's centre and assigning them to their closest cluster. The cluster position is then updated based on the mean of all training examples until convergence or a stopping criterion is reached. Let k the number of clusters, $D = \{x^{(1)}, x^{(2)}, \dots, x^{(n)}\}$ a dataset containing n examples and $\vec{A} = \{\vec{\mu}^{(1)}, \vec{\mu}^{(2)}, \dots, \vec{\mu}^{(k)}\}$ a set of k centroids. In the initial step the centroids $\vec{\mu}$ are initialized randomly. In the first step of the iteration, a distance measure is used to find the closest cluster to each data. The Euclidian distance is often used $||x^{(i)} - \mu^{(j)}||_2$. The centroid vector $\vec{A}$ is created based on the mean of all training examples. The final objective function is given by:

$$J = \sum_{i=1}^{n} \sum_{j=1}^{k} ||x^{(i)} - \mu^{(j)}||_2^2 \tag{1.47}$$

The k-mean algorithm is really efficient in clustering large datasets and is useful in discovering patterns that are often hard to find and provide a fast alternative at manual labelling each sample in a dataset. However, the algorithm doesn't necessarily provide the global optimum. Depending on the initial position of the centroids, the algorithm might find a different solution (Steinley, 2006). Furthermore, it's only possible to evaluate the properties of the clustering like the distance between the member of a cluster, but it's not possible to measure how well the cluster assignment correspond to the real world (Goodfellow *et al.*, 2016). Finally, estimating the parameter k often requires prior knowledge of the problem or requires an extensive parameter search to find the optimal value.

While the goal of clustering methods is to create groups that share similar characteristics, dimensionality reduction methods will learn a representation that has a lower dimensionality than the original inputs with the goal of preserving the most important information and minimizing the noise. They assume that the data lies along some low-dimensional manifold and the simplified lower dimensions still preserves the essential information of the original dimensions.

A popular method for compressing the dimensionality of the data is the Principal Component Analysis (PCA) algorithm. This method learns a single best (assuming a least square error) lower-dimensionality representation of the data whose elements have no linear correlation with each other (Goodfellow *et al.*, 2016). Let $X \in \mathbb{R}^{n \times m}$ a m-dimensional dataset of n samples, the goal of PCA is to project X into a lower-dimensional subspace of size z: $Y \in \mathbb{R}^{n \times z}$. PCA finds the orthogonal directions explaining most of the variance of the data (Sorzano *et al.*, 2014), the process can be simplified in five steps:

1. The mean of every dimension is measured: $\bar{X} \in \mathbb{R}^m$.

2. Find the squared covariance matrix $A \in \mathbb{R}^{n \times n}$ of X given by the formula below:

$$Cov(X,Y) = \frac{1}{n-1}\sum_{i=1}^{n}(X_i - \bar{X})(Y_i - \bar{Y})^T \tag{1.48}$$

   The covariance matrix usually contains useful information about the data. A high variance on the diagonal is linked to high variability in the features itself. For example, a positive covariance between two features usually indicates a similar variability in their directions, and a null covariance between two features indicates no predictable relationship.

3. From the covariance matrix A , find the eigenvector and the corresponding eigenvalues. An important property of eigenvectors is the invariant direction. Applying a linear transformation might translate their position in the space but not their direction. Eigenvalues are the scaling factor of eigenvectors. Hence, the eigenvalues might tell us about the magnitude of the transformation and the eigenvectors might tell us about the direction of the distortion. If v is a vector and λ is a scalar that satisfies:

$$Av = \lambda v \tag{1.49}$$

Then v is an eigenvector and λ is an eigenvalue of A. To find the eigenvalues, we rewrite the equation to:

$$Av - \lambda v = 0 \tag{1.50}$$

$$v(A - \lambda I) = 0 \tag{1.51}$$

Where I is the identity matrix with the same dimensions as A. Assuming non-null eigenvectors, the eigenvectors are given by solving the following equation:

$$det(A - \lambda I) = 0 \tag{1.52}$$

The eigenvectors are found iteratively by using the eigenvalues from the previous equation into the Equation 1.49.

4. The eigenvectors are sorted by decreasing eigenvalues and the top-k eigenvectors with the highest eigenvalues are selected to form the lower-dimensional feature subspace $W \in \mathbb{R}^{m \times k}$, where k is a hyper-parameter that defines the number of components to keep.

5. Finally, the original data X is linearly transformed with the subspace matrix W:

$$Y = X \times W^T \tag{1.53}$$

The equation above projects the original data using the k principal components into a new lower-dimensional subspace.

Reducing the dimensionality has many benefits such as a reduced computation time and space complexity, simpler model to observe and interpret and remove the need to compute extraneous parameters. An important property of PCA is its ability to find a representation where elements are mutually uncorrelated, an attempt at a disentangled representation (Goodfellow *et al.*, 2016). A disentangled representation encodes each dimension into disjoint, independent factors of its representation that do not vary together and generally leads to an understandable and exploitable structure (Higgins *et al.*, 2018).

1.5.2.2 Density Estimation and Probability Mass Estimation

Density estimation involves understanding and capturing the structure of the true probability density function (in a continuous setting) or a probability mass function (in a discrete setting) on the space of the given data (Goodfellow *et al.*, 2016). The probability density and probability mass describes the relationship between the data and their probability while the probability distribution is the overall shape of all the probability functions.

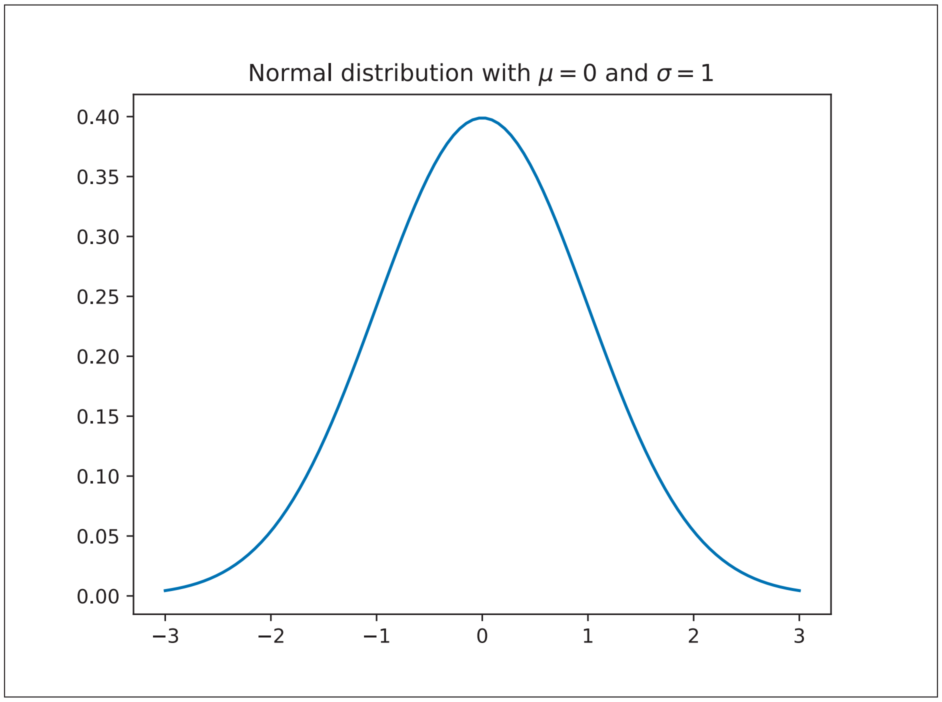


Figure 1.9 A standard Gaussian distribution with a mean of 0 and a standard deviation of 1.

Capturing the structure of a probability distribution is done by parametric and non-parametric techniques. It's possible to perform the computation on the learned probability distribution $p(X)$ to estimate new samples $p(x_{n+1})$ that are not part of the original data $X = x_0, x_1, \ldots, x_n$. Parametric estimation requires an assumption of the original data distribution and models the probability distribution $p(X; \theta)$ through a set of parameters θ (e.g., the data distribution is assumed to be normal, the parameters are the mean and the variance). Non-parametric estimation does not require such assumption and instead uses all of the data as if they were the parameter of the distribution to estimate new samples. It tries to find similar characteristics from the known data distribution and the new sample. Histogram and kernels are well-known non-parametric estimators (Cao, 2002).

When modelling distributions in a parametric way, the Gaussian distribution is an important assumption because it fits many natural phenomena. This is simple with only two parameters and is easy to explain and to understand. Gaussians have interesting analytical properties such as their symmetric (the left side of the distribution mirrors the right side), unimodal (one region with a high probability mass), asymptotic (sample size grow indefinitely), and the mean, median and mode are all equal. However, they are quite limited when modelling data from the real world (Bishop & Nasrabadi, 2007) because they're unable to capture complex distributions, like multimodal distributions with multiple high probability masses.

Latent variable models (LVMs), such as mixture models, formulate the model in terms of latent variables and observable variables, usually denoted by z and x respectively. The latent variables are not observed directly and are learned during the training process. In a mixture distribution, $z \in \{1, 2, ..., K\}$ represents a discrete latent state, $p(z)$ a discrete prior and $p(x|z)$ the likelihood:

$$p(z, x) = p(z)p(x|z) \tag{1.54}$$

First, z is sampled then the observable x is sampled from a distribution which depends on z. The prior $p(z)$ is usually a multinomial distribution $Mult(\pi)$ and the likelihood $p(x|z)$ can take any parametric form. For the likelihood, we use $p(x|z = k) = p_k(x)$, where p_k represent the base distribution k for the observations. We call this a mixture model because we're mixing K base distributions together:

$$p(x|\theta) = \sum_{k=1}^{K} \pi_k p(x|\theta) \tag{1.55}$$

Where π_k is called a mixing coefficient and is a probability distribution that controls the magnitude of each component in the mixture:

$$\sum_{k=1}^{K} \pi_k = 1 \tag{1.56}$$

A common mixture model is the Gaussian Mixture Model which assumes Gaussian distributions as based distributions. Each based distribution is a Gaussian distribution with their own mean μ_k and covariance Σ_k:

$$p(x) = \sum_{k=1}^{K} \pi_k \mathcal{N}(x|\mu_k, \Sigma_k) \tag{1.57}$$

A possible way of setting the optimal parameters for π, μ and Σ is by EM. Maximum likelihood cannot be used because the parameters no longer have a closed-form analytical solution due to the presence of the summation in the logarithm (Bishop & Nasrabadi, 2007):

$$\ln p(X|\pi, \mu, \Sigma) = \sum_{n=1}^{N} \ln(\sum_{k=1}^{K} \pi_k \mathcal{N}(x_n|\mu_k, \Sigma_k)) \tag{1.58}$$

EM is a two-steps iterative maximum likelihood method often used to estimate the best set of parameters. Theoretically, increases in the likelihood function is guaranteed as the algorithm improves the parameters estimation until the parameters estimates become stable (Abbi *et al.*, 2008). However, achieving stability in the estimation of the parameters is not always simple and might require a considerable number of iterations. Hence, heuristics like a maximum number of iterations or a value threshold are often used as stopping criterion. We refer readers who are interested to learn more about EM to Chapter 9 of Bishop & Nasrabadi (2007).

GMMs are often used for clustering purposes and unlike the k-means algorithm which provides only one label for each sample (in a hard clustering setting). Meanwhile, GMMs represent uncertainty and provide a distribution of probabilities in the label space for all samples. Effectively capturing the inherent multimodality of the data.

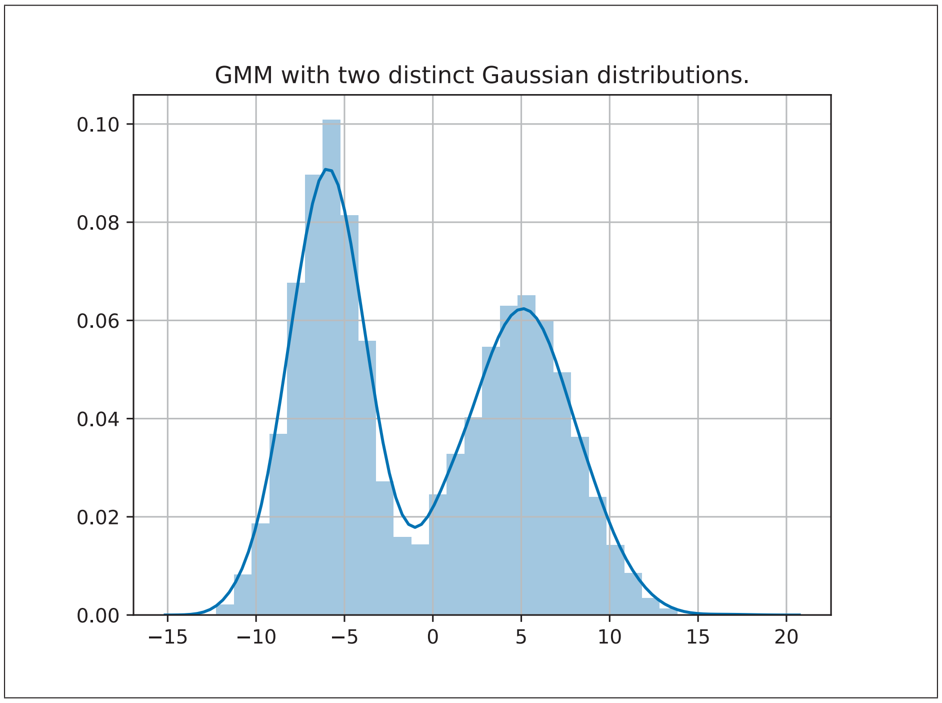


Figure 1.10 GMM with two Gaussian distributions having different means and standard deviations.

1.5.2.3 Autoencoder and Variational Autoencoder

Autoencoders (AE) are NNs that learn to compress their input X into a k dimensional latent space h and reconstruct the original input from the latent space. More formally, the latent distribution is constructed from the input $h = f(x)$ and the output is reconstructed from the hidden space $\hat{X} = g(h)$. AE are designed to learn a noisy copy of the original data by learning useful properties (Goodfellow *et al.*, 2016). One of the techniques to learn a noisy copy is by constraining the hidden representation to a smaller dimension than X. This way, the network is constrained to learn only important features. These models are made of two components: an encoder that learns the mapping between the input and the latent space $f_e = f(h|x)$ and a decoder that learns to recreate an estimation of the original data $f_d = g(x|h)$ (Figure 1.11).

AE captures the most important features present in the data and are used in many applications such as data denoising, feature extraction, dimensionality reduction, image recognition, recommender

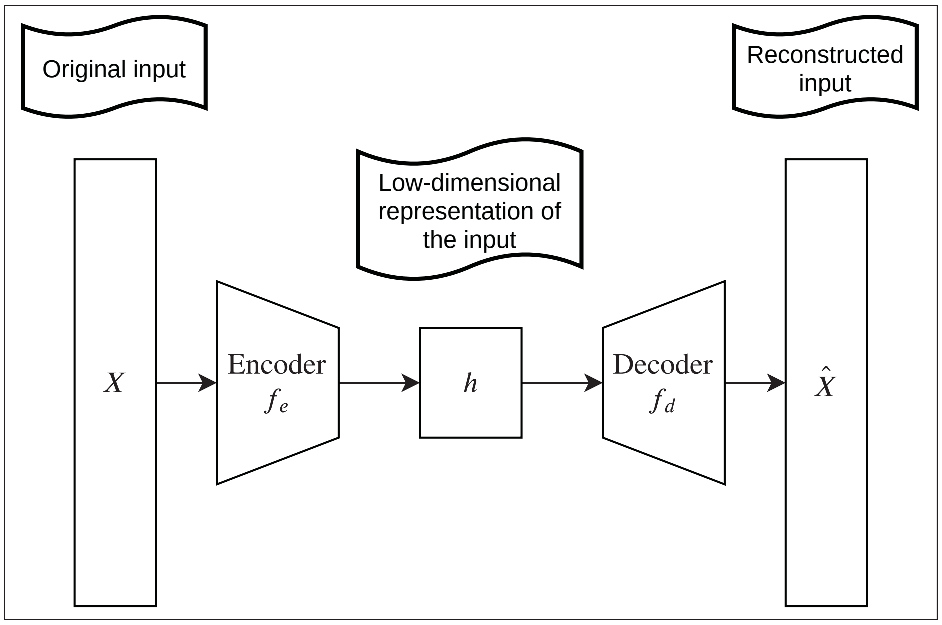


Figure 1.11 AE architecture. The encoder network f_e learns a low-dimensional representation h of the original data X. The decoder network f_d uses the latent representation h to reconstruct the original input $\hat{X}$.

system, etc. Their objective is to minimize the reconstruction error between the original input and the reconstructed input $\hat{X}$:

$$L = |X - \hat{X}| \tag{1.59}$$

The optimization is similar to general NNs and is usually done via SGD.

Variational Autoencoders (VAE) (Kingma & Welling, 2013a; Rezende *et al.*, 2014) like AE are directed models which assumes a concentration of the data around a low-dimensional manifold, learn the structure of the manifold and can reconstruct the original input. They are generative models and estimate the Probability Density Function (PDF) of the training data to generate new data. But unlike AE, VAE uses learned approximate inference and generate new data by sampling the learned latent space. More formally, this method assumes that the data is generated by a random process involving a latent variable z drawn from a prior distribution $p(z)$ and a data x coming from a conditional distribution $p(x|z)$ (Kingma & Welling, 2013a). Decomposing the conditional and the prior distribution leads to: $p(x, z) = p(x|z)p(z)$. The conditional distribution depends on the problem, a Gaussian assumption is generally used in VAE. During inference, i.e., finding good values for the posterior, Bayes' rule is used:

$$p(z|x) = \frac{p(x|z)p(z)}{p(x)} \tag{1.60}$$

The data distribution $p(x)$, called the evidence, is generated by marginalizing each data point in the training set under the entire generative process:

$$P(x) = \int P(x|z)P(z)dz \tag{1.61}$$

Unfortunately, exact inference of the evidence is intractable due to the integral that evaluates all configuration of latent variables and would require an exponential time to compute. Therefore, the posterior $p(z|x)$ need to be approximated. To solve this problem, an approximate inference technique is used and the exact inference problem is treated like an optimization problem, called variational inference (VI).

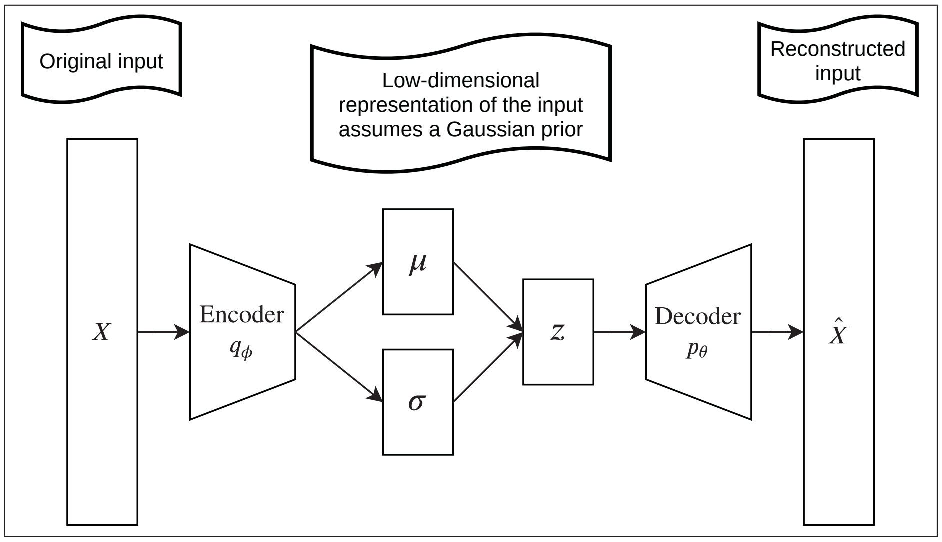


Figure 1.12 VAE architecture with a Gaussian prior. The data X is used by the encoder q_ϕ which outputs the parameters of the Gaussian distribution, z is sampled following the mean and the standard deviation and the decoder p_θ reconstructs the original inputs $\hat{X}$.

VI is a method that approximates probability densities through optimization and tends to be faster than classical methods like Monte Carlo sampling (Blei *et al.*, 2017). To solve the problem described above, VI approximates the posterior with a family of distributions $q_\lambda(z|x)$ to be close to the target distribution $p(z|x)$, where λ corresponds to the variational parameters of q. Closeness between q and p is measured using the Kullback-Leibler (KL) divergence:

$$\mathrm{KL}(q_\lambda(z|x) \parallel p(z|x)) = \mathbb{E}_q[\log q_\lambda(z|x)] - \mathbb{E}_q[\log p(x, z)] + \log p(x) \tag{1.62}$$

The goal is then to find the best candidates of variational parameters λ, the ones closest in terms of KL:

$$q^*_\lambda(z|x) = \arg\min_\lambda \mathrm{KL}(q_\lambda(z|x) \parallel p(z|x)) \tag{1.63}$$

Again, the dependence on the evidence is still present which makes the optimal candidates impossible to compute directly. Instead, a lower bound on $\log p(x)$ is computed. This alternative objective is called the evidence lower bound (ELBO):

$$\mathrm{ELBO}(\lambda) = \mathbb{E}_q[\log p(x, z)] - \mathbb{E}_q[\log q_\lambda(z|x)] \tag{1.64}$$

Maximizing the ELBO is equivalent to minimizing the KL because the KL is always greater or equals to zero by Jensen's inequality (Blei *et al.*, 2017). By combining the ELBO and the KL, we can rewrite the evidence to:

$$\log p(x) = \mathrm{ELBO}(\lambda) + \mathrm{KL}(q_\lambda(z|x) \parallel p(z|x)) \tag{1.65}$$

In a VAE model, latent variables z are local and are not shared between data. Samples of z are usually drawn from a standard centred Gaussian distribution $p(z) = \mathcal{N}(0, I)$ where I is the identity matrix. The ELBO can then be decomposed into an independent sum of data points and optimized using SGD w.r.t to the variational parameters λ.

In a VAE, the approximate inference network, or encoder, $q_\theta(z|x)$ is parametrized with θ and output samples of z.

The generative neural network, or decoder, with parameters ϕ takes the latent variables z as input and outputs the parameters of the data distribution $p_\phi(x|z)$. The VAE is trained by maximizing the ELBO associated with every data point of x:

$$\text{ELBO}(\theta, \phi) = \mathop{\mathbb{E}}_{z \sim q_\phi(z|x)}[\log p_\theta(x|z)] - KL((q_\phi(z|x) \parallel p_\theta(z))) \tag{1.66}$$

In the VAE model, a Gaussian assumption is generally used for q_θ and the latent code z is approximated from this distribution. However, it's not possible to back propagate the error through this layer during the optimization phase because sampling from a Gaussian in which the parameters are the outputs of the encoder $\lambda = (\mu, \Sigma)$ is not possible. The sampling operation has no gradient. However, the reparameterization trick (Kingma & Welling, 2013a) makes the network differentiable by moving the non-differentiable operations outside of the network. Therefore, the sampling operation becomes:

$$z = \mu(X) + \Sigma(X)\epsilon \tag{1.67}$$

Where $\epsilon \sim \mathcal{N}(0, 1)$. The sampling process is now outside the network, the gradient won't flow through it during the optimization process.

As long as the latent variable z is continuous, the samples drawn from the approximated posterior are deterministic and the gradient can be computed. Maximizing the ELBO with the Gaussian assumption encourages the posterior to place high probability on many z values that could have generated x, instead of a single more likely value (Goodfellow *et al.*, 2016) which encourage the diversity during the generation. While the first term of 1.66 measures the reconstruction log-likelihood similar to traditional AE models, the second term minimizes the distance between the approximated posterior distribution $q_\theta(z|x)$ and the Gaussian prior $p(z)$. During the optimization process, the network learns to map the independent normally-distributed z to the correct latent variables needed by the model (Doersch, 2016).

VAE models are one of the many generative modelling techniques. They're simple to implement and generally lead to excellent results and are widely used in image modelling and generation. Their assumptions are weak and can be easily extended to many tasks.

Several variations of this architecture were proposed. More recently, Higgins *et al.* (2017) posits to change the magnitude of the variational term in the ELBO with a coefficient β .

$$ELBO_{\beta}(\theta, \phi) = \mathop{\mathbb{E}}_{z \sim q_{\phi}(z|x)} [\log p_{\theta}(x|z)] - \beta KL((q_{\phi}(z|x) \parallel p_{\theta}(z))) \tag{1.68}$$

The goal is the same as an original VAE, where the probability of generating real data is maximized while keeping a small "distance" between the prior and posterior. When $\beta = 1$, the original VAE is found. A higher beta $\beta > 1$ encourages more efficient encoding by constraining the latent space and limiting the representation capacity of the code z. It's said that the latent representation is disentangled if each variable in the latent code is only sensitive to one single generative factor. A disentangled latent space is ideal since it provides good interpretability and makes generating new samples with specific attributes a possibility.

CHAPTER 2

METHODOLOGY

In this chapter, we describe the two recommender models for implicit interactions. The first model use non-temporal, two-dimensional binary data that describes the interaction between a user and the items. The second model uses temporal data enhanced with media metadata. First, we describe the general architecture shared by both models, namely a VAE with a GMM as the prior distribution. Second, we show a variant of the first architecture for sequential data that can includes extra information like item metadata and interaction context.

2.1 VAE with GMM prior

Our work on VAE in recommender systems were inspired by the authors of Liang *et al.* (2018a) where they used a VAE with a standard Gaussian prior to model the recommendation space. Instead of using a standard Gaussian prior, we use a GMM prior to enable the VAE to learn the multimodality nature of the data. We also use a different objective function to accommodate this new distribution. We call this model the GMM-VAE. Our hypothesis is that human behaviour described by interaction data is inherently multimodal: tastes cannot be described by a single modality, but is a combination of several signals. We assume that the observed data is generated from a multimodal prior distribution and there's an inference model that can be constructed and trained with traditional optimization techniques. The GMM prior will model each cluster capturing the pairwise correlation among the binary responses from the data within the cluster. Building upon the CF literature, we believe that a user can be fragmented across different neighbourhoods of users based on a specific subset of the interaction data. For example, on a movie streaming platform, if a user's history is mainly comprised of content classified as "action" with a small amount of "drama" content, a unimodal system might group this user with users having a similar "action" oriented history. In this case, this user will have more "action" content than "drama" content in their recommendation list. With a multimodal model, it's possible to understand the different type of signals from the user's history. Understanding the underlying connection between the "drama" content and the "action" content might become possible. The system would then group this user more intelligently with other users. This naive example assumes different modalities coming from the categories. In reality, implicit binary signals do not have knowledge of the categories, but will instead analyze the latent structure in the data, similar to unsupervised clustering.

Similar to traditional VAE models with isotropic prior distribution, the VAE with GMM prior can be broken down into two parts: the inference and the generative network. These two processes are represented in the figure below.

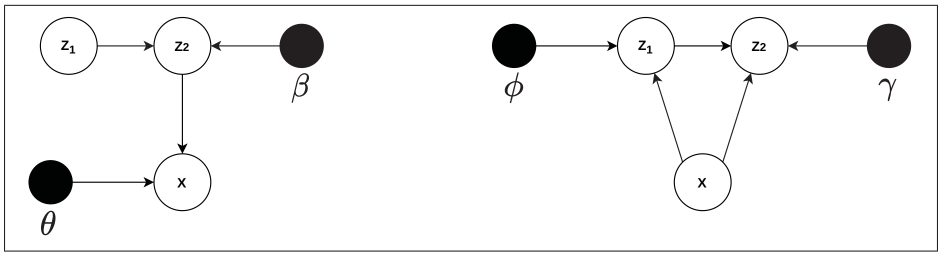


Figure 2.1 Generative VAE model with GMM prior on the left, and inference model on the right.

The generative model $p_\theta(x, z_1, z_2) = p_\theta(x|z_1, z_2)p_\beta(z_2|z1)p_(z_1)$ presented in this thesis and shown in Figure 2.1 describes an observed variable x conditionally generated by random variables z_1 and z_2. Unlike regular VAE, where there is only one vector of means and variances, the GMM-VAE outputs K vectors of mean and variance. The model will choose the set of mean and variance following the component's categorical distribution. The components will be used to sample the latent code from the Gaussian distribution. The generative process is given by:

$$z_1 \sim \text{Cat}(\pi) \tag{2.1}$$

$$z_2|z_1 \sim \prod_{k=1}^{K} \mathcal{N}\left(\mu_{z_2,k}(z_1, \beta), \Sigma^2_{z_2,k}(z_1, \beta)\right) \tag{2.2}$$

$$x|z_1, z_2 \sim \mathcal{N}\left(\mu_x(z_2, \theta), \Sigma^2_x(z_2, \theta)\right) \tag{2.3}$$

We define K as the number of components in the mixture. z_1 follows a categorical distribution $\text{Cat}(\pi)$ that dictates the probability of generating the data x under different Gaussian distributions. We initialize this categorical distribution to be uniformly distributed $\pi = \frac{1}{K}$. Because the model is a GMM, the network outputs K components. We select the most probable components ($\mu_{z_2,k}$ and $\Sigma^2_{z_2,k}$) based on z_1 and parameters β to generate the latent space z_2. The intuition behind this hierarchical latent variable model is that each Gaussian encodes specific data attributes into different clusters. The Gaussians' components are chosen based on the probabilities of z_1 and are used to generate the latent space z_2, which holds all the necessary information to synthesize the data.

At inference, the ELBO defined by traditional VAE models with a standard Gaussian distribution can be optimized directly. However, introducing a GMM makes the KL term intractable (Durrieu *et al.*, 2012). Considering a single Gaussian distribution for $p(x)$ and $q(x)$, a GMM distribution with K components, recalling the definition of the KL divergence between two distributions:

$$\text{KL}(q||p) = \int p(x) \log \left(\frac{p(x)}{q(x)} \right) dx \tag{2.4}$$

$$= \int p(x) \log (p(x)) dx - \int p(x) \log (q(x)) dx \tag{2.5}$$

The first integral has a close form solution and can be solved easily:

$$\int p(x) \log p(x) dx = \int \frac{1}{\sqrt{2\pi\sigma^2}} \exp \left(-\frac{(x-\mu)^2}{2\sigma^2} \right) \log \left(\frac{1}{\sqrt{2\pi\sigma^2}} \exp - \frac{(x-\mu)^2}{2\sigma^2} \right) dx \tag{2.6}$$

$$= -\frac{1 + \log \left(2\pi\sigma^2\right)}{2} \tag{2.7}$$

There's no closed-form solution for the second integral due to the summation in the logarithm:

$$\int p(x) \log (q(x)) = \int p(x) \log \left(\sum_i^k \pi_i \mathcal{N}(\mu_i, \sigma_i) \right) dx \tag{2.8}$$

To overcome this difficulty, we propose two different methods that can be used when modelling the prior and posterior with a GMM distribution. Quantitative results for each of these methods are presented in the next chapter. The first method approximates the KL divergence using a Monte Carlo (MC) estimator. The second method approximates the two GMMs into a single Gaussian distribution.

1. In an MC setting, the KL is expressed as the expectation of the logarithm of the posterior over the prior: $\mathrm{KL}(q||p) = \mathbb{E}_x[\log \frac{q(x)}{p(x)}]$. The estimation of the expectation is done by drawing n independent samples from the PDF q and computing the expectation $\mathrm{D}_{\mathrm{MC}}(q||p) = \frac{1}{n} \sum_i \log \frac{q(x_i)}{p(x_i)}$. Although drawing independent samples for the expectation increases the variance of the estimator, by the law of large numbers the expectation $\mathrm{D}_{\mathrm{MC}}(q||p)$ converges to $\mathrm{KL}(q||p)$ when $n \to \infty$. This method also produces an unbiased estimation because a single Monte Carlo estimate is a sample of the same stochastic function given by the logarithm of the ratio between q and p.

   The interpretation of the bias and variance trade-off in an RS setting is akin to the exploration and exploitation trade-off. A high variance signifies that the agent will tend to explore the space of recommendations instead of recommending high likelihood items. This is opposed to a small variance which maximizes the exploitation. Keeping a low bias is favourable because the recommendations are centred around likely items.

2. While $\mathrm{KL}(q||p)$ is not defined when both distributions are GMs, the solution is trivial when p and q are single Gaussian distributions. Therefore, we propose to use a weighted approximation of the means μ and variances Σ of each GMM. The weights are defined by the categorical distribution of the component vector π.

$$\hat{\mu} = \sum_{k=1}^{K} \pi_k \mu_k \tag{2.9}$$

$$\hat{\Sigma} = \sum_{k=1}^{K} \pi_k \Sigma_k \tag{2.10}$$

   A single Gaussian distribution for each GMM distribution $\hat{p} = \mathcal{N}(\hat{\mu}_p, \hat{\Sigma}_p)$ and $\hat{q} = \mathcal{N}(\hat{\mu}_q, \hat{\Sigma}_q)$ is created using the approximated means and variances and the KL between the new distribution can be found analytically:

$$\begin{aligned}
\mathrm{KL}(\hat{q}||\hat{p}) &= \int \hat{q}(x) \log \frac{\hat{q}(x)}{\hat{p}(x)} d(x) \\
&= \int \frac{1}{2} \log \frac{|\hat{\Sigma}_p|}{|\hat{\Sigma}_q|} - \frac{1}{2}(x - \hat{\mu}_q)^T \hat{\Sigma}_q^{-1}(x - \hat{\mu}_q) \\
&\quad + \frac{1}{2}(x - \hat{\mu}_p)^T \hat{\Sigma}_p^{-1}(x - \hat{\mu}_p) d(x) \\
&= \frac{1}{2} \left[\log \frac{|\hat{\Sigma}_p|}{|\hat{\Sigma}_q|} - d + \mathrm{tr}\{\hat{\Sigma}_p^{-1}\hat{\Sigma}_q\} + (\hat{\mu}_q - \hat{\mu}_p)^T \hat{\Sigma}_p^{-1}(\hat{\mu}_q - \hat{\mu}_p) \right]
\end{aligned} \tag{2.11}$$

2.1.1 Non-temporal Model

The non-temporal model is the main model presented in this thesis and uses a VAE with a GMM prior. Given a matrix of implicit interactions $R \in \mathcal{R}^{m \times n}$ where m is the number of users and n the items, we create a confidence matrix similar to Hu *et al.* (2008). The input of the network is the confidence matrix $C = 1 + \alpha R$ where $\alpha = 40$. Because unobserved values (i.e., zeros) in the implicit matrix aren't necessarily correlated with a negative behaviour, the confidence matrix puts minimal confidence on those values while putting a higher mass on seen values.

Then, a dropout layer acts as a regularizer, similar to Denoising Autoencoder (Vincent *et al.*, 2008). Regularization on the inputs was found to be important to fight overfitting of the model. Afterwards, the noisy input is used by a NN to extract important dependencies from the data. The output of the MLP-based encoder h is projected to create all the components of the GMM π, μ and Σ.

$$\pi = \mathrm{softmax}(W_\pi h + b_\pi) \tag{2.12}$$

$$\mu = W_\mu h + b_\mu \tag{2.13}$$

$$\Sigma = \mathrm{softplus}(W_\Sigma h + b_\Sigma) \tag{2.14}$$

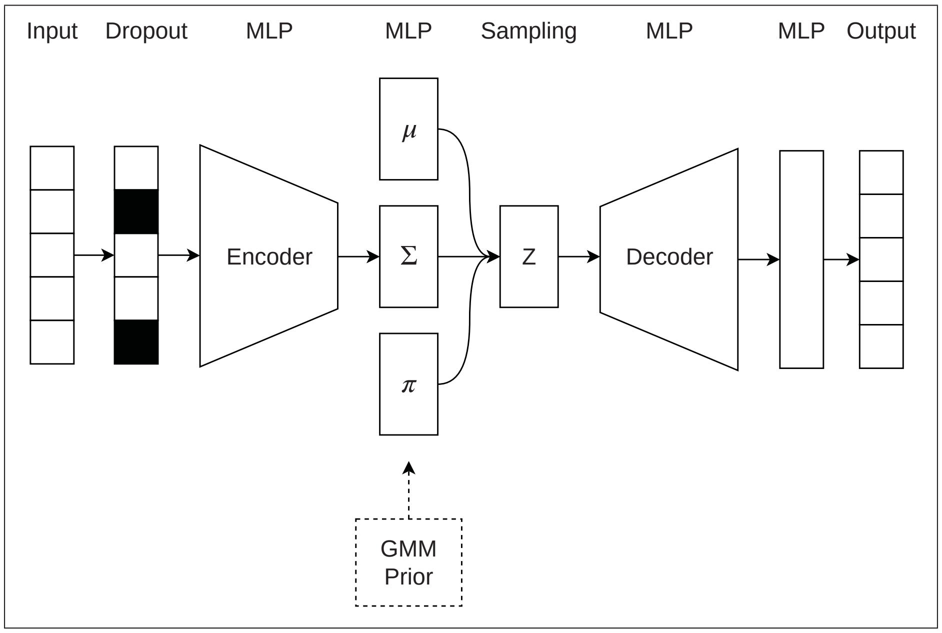


Figure 2.2 Architecture of the non-temporal model. An input X is given to the network. The first layer is a dropout layer followed by the encoder, latent space z, decoder and reconstructed input.

Where $\text{softplus}(x) = \log(1 + e^x)$ is a smoothed version of ReLU. The latent space z is sampled using these components and is the input of the MLP-based decoder whose goal is to reconstruct the original data.

Generating new recommendations is done by feeding the history of a user to the network and retrieving the corresponding GMM components. The decoder reconstructs the output from the sampled code. Items previously found in the history of the user are removed from the prediction and the vector is sorted, similar to a Top-N recommendation.

2.1.2 Temporal Model

The temporal model can generate new recommendations based on the user history and item features. This model is analogous to CF and CBF models and unlike the non-temporal model, user features can be embedded as well. Consequently, recommendations are made not only by comparing similar history across users and related metadata between content, but contextual information can be considered to provide the best recommendations at the right time.

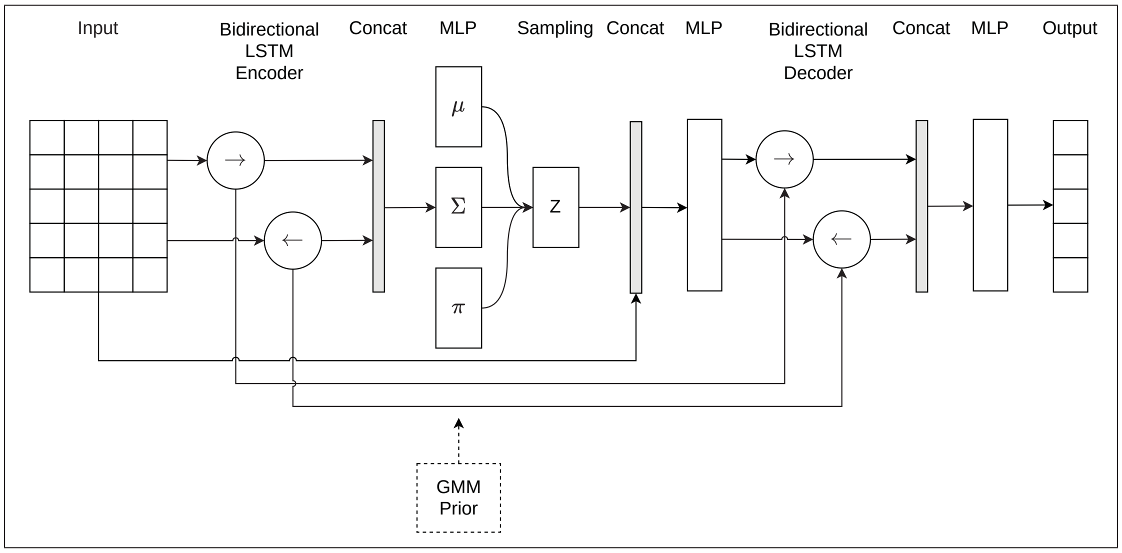


Figure 2.3 Architecture of the autoregressive temporal model. The encoder is a bidirectional LSTM that extracts the temporal dependencies. The GMM parameters come from a projection of the concatenation of the last hidden state from the forward and backward cells. The bidirectional LSTM-based decoder reconstructs the input by using the latent space and the last state of each encoder network. Finally, a classification network uses the reconstructed outputs to predict the most likely item.

In this setting, GMM-VAE is extended with an MLP-based classifier network that outputs a probability distribution over the items to recommend at each time steps. Bidirectional LSTM cells are used in both the encoder and the decoder of the model. The RNN extracts the underlying temporal information from the input data $X = \{x_1^1, x_2^1, \dots x_T^n\}$ where T is the number of time steps corresponding to the t action in the history of a user and n the number of users. The bidirectional LSTM encoder takes the feature vector as an input, and outputs the concatenation of hidden states $h^e = [h_{t-1}^f, h_{t-1}^b]$. Where $h_{t-1}^f \in \mathcal{R}^{n \times e}$ is the last hidden state of the forward LSTM, $h_{t-1}^b \in \mathcal{R}^{n \times e}$ is the last hidden states of the backward LSTM and e is the hidden size.

The latent code z is sampled by using the projected GMM components coming from h^e in a similar way to what has been done with the non-temporal model. Using this process, the latent code is conditioned on the feature vector and is not a deterministic output.

The decoder is autoregressive, meaning that it samples output features conditionally to a given latent code z. The previous features vector x_{i-1} is concatenated with the code and is the input of the decoder $\bar{x} = W_z[x_{i-1}, z] + b_z$ along with the last state from the encoder network $[h_0^d, c_0^d] = \tanh(W_{hc}h^e + b_{hc})$. At each time step, the previous feature vector x_{i-1} is concatenated with the latent code z. The output is a reconstruction of the next feature vector $\hat{x}_i$. We define x_0 to a null vector and only appears at the very first time step of the decoder. The output of the decoder for each time step is used by a classification network that outputs a probability distribution over all available items $\hat{y}_i = \text{softmax}(W_{\text{cls}}\hat{x}_i + b_{\text{cls}})$. The model optimizes the VAE and the classification network independently. The VAE network is optimized using the VAE cost while the classification network uses the softmax classification with cross-entropy. This loss is the distance between the predictions of the model $\hat{y}$ and the true values y.

$$\mathcal{L}_{\text{cls}} = -\sum_i y_i \log(\hat{y}_i) \tag{2.15}$$

Sampling new recommendations are done via conditional generations. The easiest way to do it would be to retrain the network using only a specific user history, but this method is not very practical when there are several users involved or the dimension of the feature space is large. Fortunately, this can be done without retraining by “priming” (Graves, 2013) the network and generating subsequent recommendations with the past history still in memory. First, the encoder is fed with the feature history of a user, a latent code z is sampled and used along with the null input for the very first time step in the decoder and the last states of the encoder. Generating more than one time step is done by using the output of the decoder with the same latent code z as its input. This process is analogous to a synthetic user interaction. If one time step was generated, the N most likely content from the distribution is returned, which is similar to Top-N recommendations. In the case of sampling multiple time steps, the most likely content at each time step is used.

The two models that were proposed in this chapter extend the VAE architecture for non-temporal and temporal data. The traditional Gaussian assumption was replaced with a mixture of Gaussians to capture the underlying modalities in the data. Two methods to optimize this architecture were also introduced. In the following chapter, we evaluate these two models on the dataset provided by Radio-Canada and on a popular dataset used in the recommender system community.

CHAPTER 3

EVALUATION

In this chapter, we present how our recommender system is used on Radio-Canada's streaming platform "Tou.TV". We also present the application of our system on a popular dataset called "MovieLens-20M". We describe how the data was collected, transformed and used in both of the models. We then explain the training process and the optimal hyper-parameters. Finally, we perform a quantitative and qualitative analysis of the performance of the models against existing baselines.

3.1 Task Description

3.1.1 Recommender system for Tou.TV

Radio-Canada (RC) is a Canadian federal Crown corporation and the official French public broadcaster. Throughout the years, RC as expanded its services and now offers a video-streaming website. TV shows and movies are available for free and customers have the possibility to subscribe to their service and unlock premium content. With more than 250,000 ± 62,500 daily visits on average, Tou.TV is a popular platform to engage with French content.

The current RS is CBF-based and uses handcrafted tags that characterize the genre of the content. When a new item is added to the catalogue, tags are manually assigned. This system is relatively recent and covers only 12.79% of the catalogue with only the new content tagged. Recommendations are not based on the user history, but are made between content and are shown at the bottom of the page of the TV show or movie.

In order to provide a personalized experience and increase customer satisfaction, RC would like to use a contextual RS to suggest relevant recommendations at the right time. They would like the final system to have three quality attributes. The first is for the RS to offer personalized suggestions based on the history and the context of the visit, like the period of the day, day of

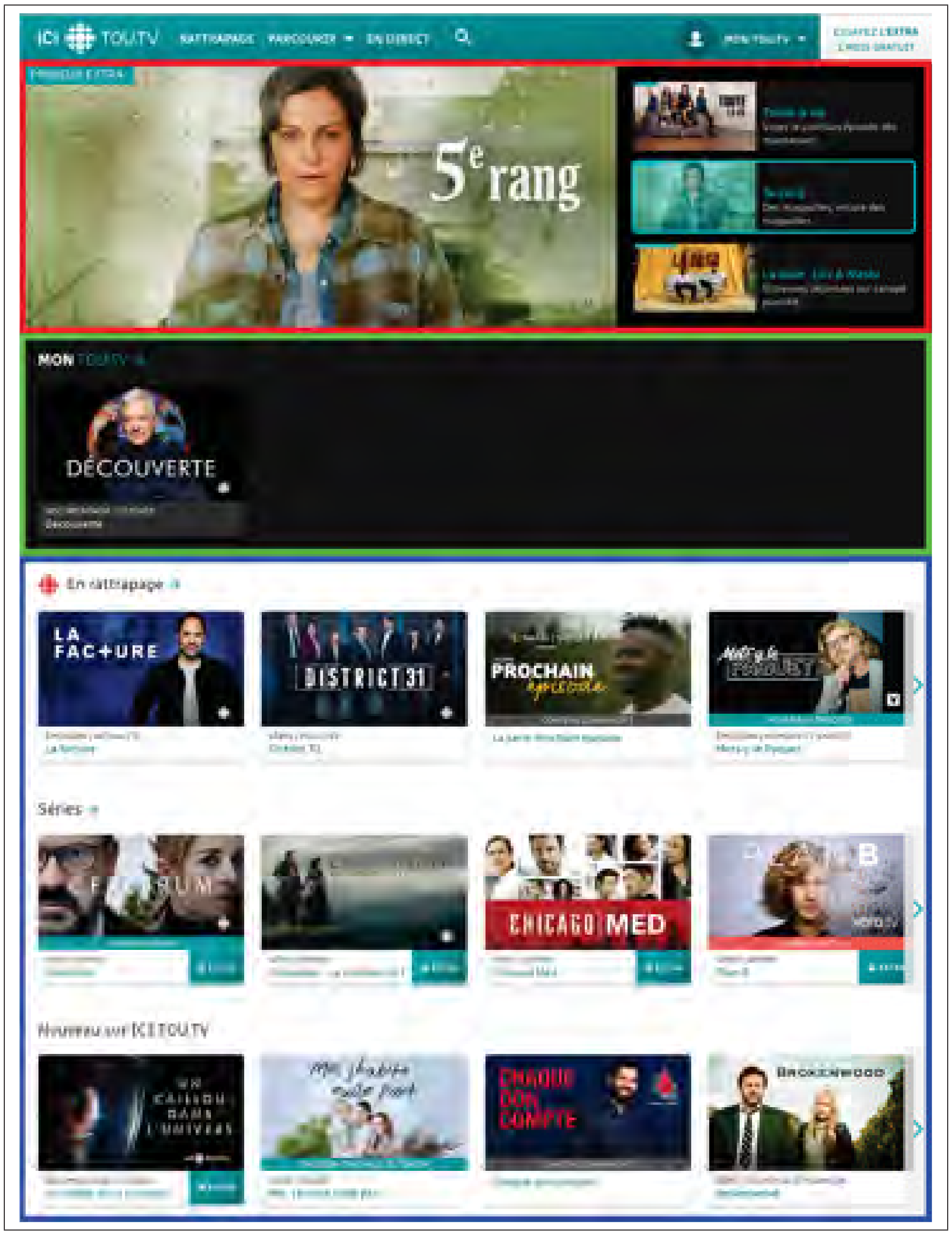


Figure 3.1 Homepage of Tou.TV. The top region (in red) is an editorial lineup that is the same for all users. The middle section (in green) corresponds to content that were started but not finished. The bottom section (in blue) contains a mix of handpicked and auto-generated content.

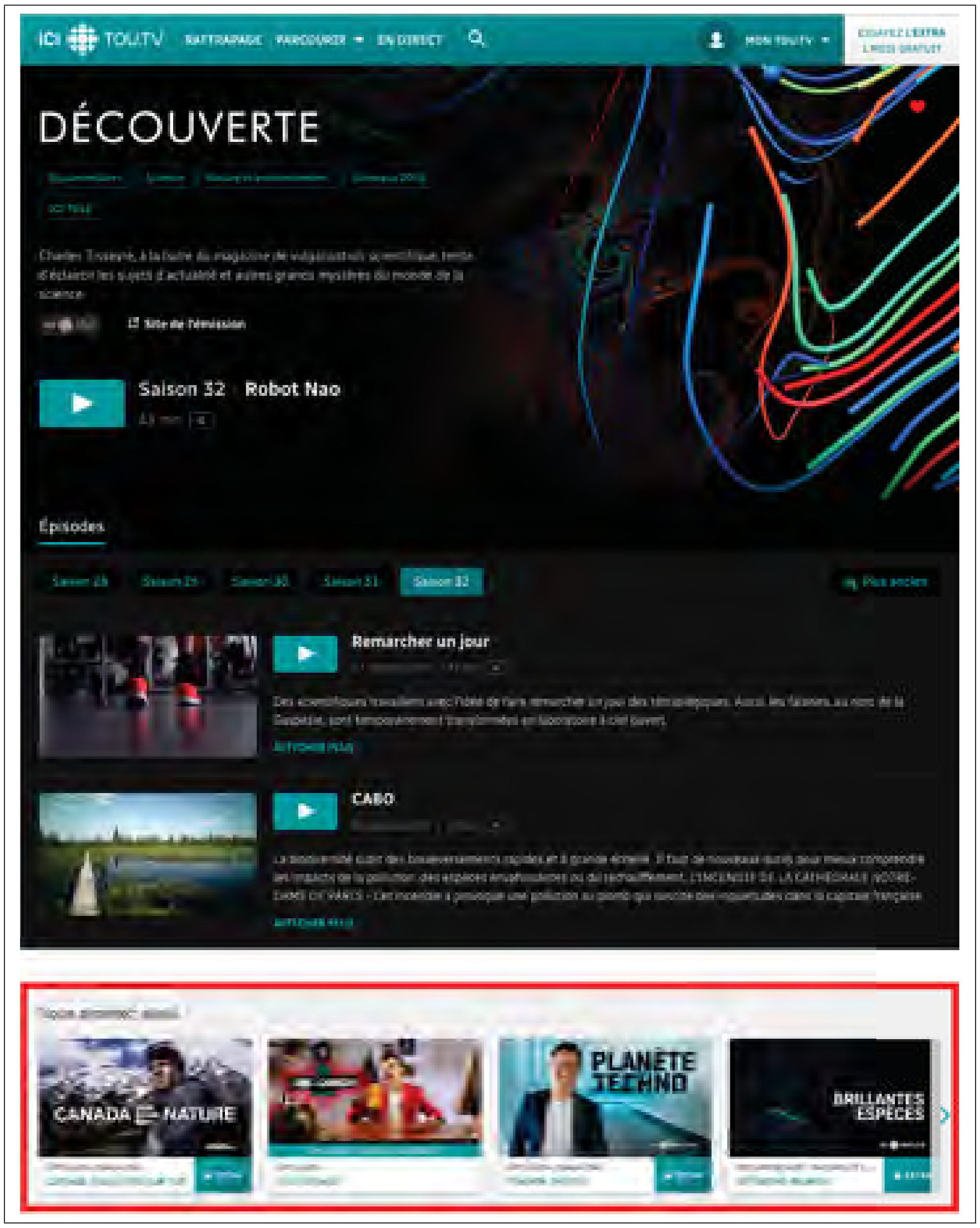


Figure 3.2 Homepage of a TV show. Content associated with this TV show is at the bottom. Recommendations are provided by a CBF-based model that uses handcrafted tags.

the week and the type of device used. The second is to suggest serendipitous suggestions and focus on exploration more than exploitation to increase the catalogue coverage. The third is to assist the user when looking for new content to watch. Finally, RC plans to use the RS on the home page, in a search results page and at the end of a video. On the home page, a new space with selected content from the RS will be displayed. The order of the content in the sections will be based on the likelihood given by the system. On a search results page, the content will be ordered by probability. Finally, when the content ends, the RS will choose the next one to play automatically.

Considering these quality attributes and requirements, the system should focus on quality rather than quantity and cover the majority of the catalogue. Hence, the goal of the model is to achieve a high precision while maintaining a good coverage. Precision is favoured over recall because the first considers the relevancy of a subset of recommended items while the second also considers the relevancy but over all items in the user history. Achieving a high coverage is easy and can be done by randomly recommending items, but this lead to a poor precision. A high precision is obtained by recommending the same content found in the user's history, but doesn't favour exploration, only exploitation. We measure the tradeoff using the ROC curves, precision-recall (PR) curves and the coverage (Equation 1.34).

3.1.2 MovieLens-20M

The model is trained on content from MovieLens (Harper & Konstan, 2015) to recommend movies to users. The requirements of this model are the same as presented in the last section. Namely, the system should focus on quality rather than quantity, should provide new and interesting recommendations and help the user find new content to watch. The metrics are also the same: the ROC curves and the PR curves are used as well as the coverage metric.

3.2 Dataset

The models presented in this thesis use implicit feedback. While the non-temporal model uses exclusively the user-item matrix, the temporal model can be enhanced with other features. For this reason, we have also gathered a substantial amount of textual information about the content and contextual information describing the interaction of the user. While the data might be sparse, i.e., most content is watched only by a subset of users, the extra information will be used to mitigate this issue. In this section, we present how the data was collected and transformed for both Tou.TV and MovieLens-20M.

3.2.1 Tou.TV

3.2.1.1 Data collection

The service was launched in 2010 but they only started to collect analytic data in 2016. We adopt the same vocabulary as RC: a “program” is comprised of one or more “media”. TV shows and movies are programs. The first have at least one associated media, while the latter has only one association.

Programs have general information shared with their related media. Like the title, description, tags and genres. Besides a title and a description, media has an availability date, rating, keywords, names of authors, comedians and characters. When we conducted this research, there were 4,080 programs, 1,133 are movies and 2,947 are TV shows. TV shows had on average 15 ± 118 media (i.e., episodes), one in particular had 5,961 media.

When users start watching content, a “signal” is captured. Signals are contextual and describes the device used by the user, his location, the media watched and the time spent watching the content. Since 2016, there are more than 73,000,000 signals available. These signals are implicit and needed to be filtered to remove “false” interactions that do not describe a positive interest, e.g., a user clicked on an item but went back immediately, a user did not watch the TV show long enough, etc. To remove these interactions, we’ve created a model that estimates the density of the relative time spent per content for all users using a univariate Gaussian kernel. The different modes indicate when the users usually stop watching content. Although most of the density was around a relative time spent equals to 1 (i.e., the content was completely watched), there was some minimas around 0 which signified a negative feedback. We found that most of the users who watched more than 6% of the content tended to finish it. Therefore, we removed all interactions with a relative time spent lower than ~ 0.0671%.

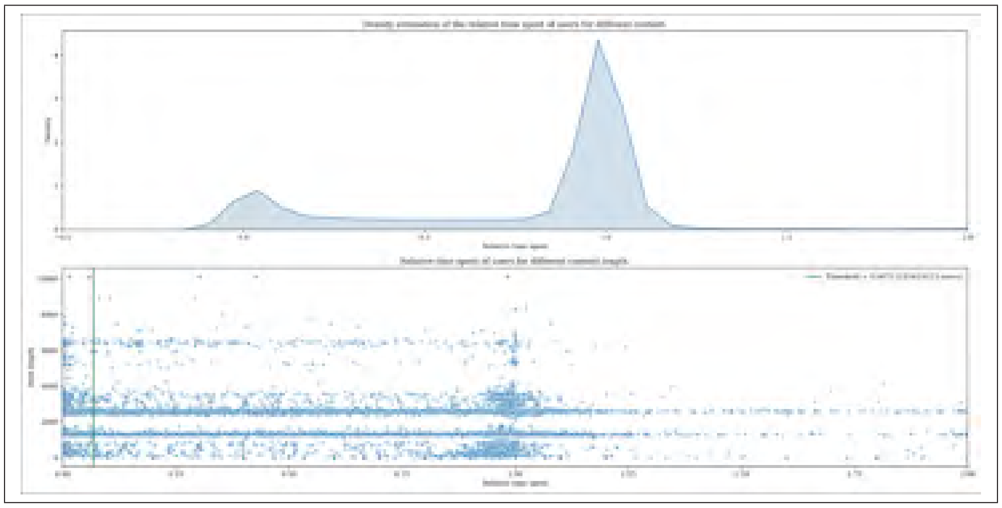


Figure 3.3 Density estimation of the relative time spent for 18,821 users on 4,077 content. The top figure shows the result of the univariate Gaussian kernel. Regions with high density corresponds to multiple users who share a similar time spent. Two high density regions are present: users who stop watching at the beginning and users who finish the content. The bottom figure shows the relative time spent of users (the blue dots) for a specific content length. The green line refers to a minimum found in the previous graph. This minimum is used as a threshold to keep only positive interactions.

For our needs, we've sampled 191,322 users having at least interacted with 50 pieces of content. We note that a user might've interacted with the same content more than once, but at a different period of time. The final dataset has 70,120,035 rows. Users have on average 366 ± 452 signals.

The extraction was done by querying RC's internal server which returned the latest signals. This method introduced a "most recent" bias. We consider this side effect desirable since the models will always be trained to use the newest content and user behaviours available.

3.2.1.2 Preprocessing

After collecting the signals, we've performed a minimal transformation to create the features used in the temporal model. We standardized the date and derived six features. The day of the week and hour of the day were transformed using a sin and cos function. Four different values representing the period of the day were derived from the hour and were again transformed with a sin and cos function. The type of devices used by users were also standardized to four different values: "mobile", "tablet", "computer" and "tv". A total of seven features are used to describe the context of the visit. An example of the results is provided in Table 3.1.

Table 3.1 Example of features after preprocessing.

user_id	**item_id**	**relative_time_spent**	**day_of_week_sin**	**day_of_week_cos**	**...**
1	24	0.25	0.8660	-0.5	
1	65	0.89	-0.8660	-0.5	
2	987	0.13	0	1	

In preparation of text modelling, we've performed basic transformations on the textual fields in the program dataset and the media dataset:

1. Convert all the text to lowercase.
2. Replace spaces in names (authors, comedians and characters) with hyphens.
3. Replace apostrophes with hyphens.
4. Remove all punctuation.
5. Remove special characters, like the accent from accented characters, and only preserve letters and digits.
6. Remove French stop words from titles and descriptions.
7. Perform stemming and lemmatization on titles and descriptions.

After performing basic transformations, program information such as the title and the description is concatenated to their associated media. We then build a TF-IDF model using these values.

Words with a document frequency lower than 4% and higher than 90% are removed from the vocabulary. We chose these thresholds in our experiment because frequent words are shared across the majority of the content and are not informative. Rare terms are a part of a very small group of content and are usually not meaningful.

Table 3.2 Size of the vocabulary of Tou.TV.

size	**mean**	**std**	**min**	**max**
300	24.17	27.95	2.00	167.00

With the following settings, the sparsity of the vocabulary is around 7.37%. We extract the 35 most likely terms using the TF-IDF model for each media. Because of the sparsity of the data, a PCA model was created using the most likely terms to reduce the number of dimensions to 23. We preserve 90% of the explained variance. We included the most likely terms as categorical features and the PCA values as numerical features.

The current RS on Tou.TV uses tags created by an editorial team. There are more than 116 tags available. These tags characterize the type of the content (i.e., series, documentary, shorts, ...), genre (i.e., animation, adventure, comedy, ...) and themes (i.e., art, economy and politics, nature and environment, ...). The majority of the tags have sub-tags that further describe the content. We selected the main tags for a total of 19 tags and used them as binary features.

Finally, most of the content had a similar length: episodes under 30 minutes, episodes between 30 minutes and an hour and movies. The length was encoded into three categories. We further normalized the time spent per content by dividing the total time spent per content by the length of the content. The majority of the users will have a value between 0 and 1, but some users might watch the content more than once. This results into a value superior to 1. The rating of a media was also included and standardized into four categories. The final feature vector has 84 dimensions.

The aggregated version of the dataset only uses implicit feedback. To provide more informative values, we use the relative time spent per content as values in the user-item matrix.

3.2.2 MovieLens-20M

3.2.2.1 Data collection

The dataset is publicly available and can be downloaded for free. It contains explicit ratings from a movie recommendation service called MovieLens. From January 1995 to March 2015, the service has collected more than 20,000,000 interactions between users and movies. More than 138,493 users have rated 27,278 movies.

Split between six files, the dataset contains various information about the movies and the interactions. For the purpose of our experiment, we only used the file containing the time and ratings of movies by users and the file containing the titles and genres of each movie.

We removed movies with low interactions and only kept engaged users. Due to the nature of the implicit system described in this thesis, we converted the explicit interactions into implicit interactions. Of the 20,000,000 ratings available, we chose to keep those higher than 3 because we define a rating of 3 and higher as a positive interaction. Furthermore, on the remaining 24,800 movies, more than 70% had fewer than 100 ratings. We kept the upper 30% which is around 7,461 movies. A similar analysis was done for the users and we only kept those with at least 100 ratings. The final dataset has around 11,799,484 interactions characterized by 41,787 users and 7,461 movies. The sparsity is around 3.78%.

3.2.2.2 Preprocessing

Minimal preprocessing was done. Each genre is encoded into a binary column and corresponds to the features used by the temporal model. Genres that are assigned to less than 1% of the movies are removed. The final feature vector used by the temporal model has 18 features.

3.3 Training

3.3.1 Tou.TV

Before training both models, we split the interactions into three disjoint sets. The training set contains 99% of the data, the validation set and test set have 0.5% of the remaining data. We chose these sizes because we still consider thousands of data in the validation and test set and the training step required a considerable amount of data due to the sparsity in the interactions. The validation set is used for early stopping when the model has reached a convergence. It's also used to find the best hyper-parameters. All the metrics reported in the following sections are performed on the same test set. The training set contains values from the past, while the valid and test sets have the most recent values. This was done to ensure that the experiments would reflect the normal usage in production. Finally, all the models used in this thesis needed to be primed with the history of a user before predicting the next content to watch. We used the first 80% of the test set for priming and the remaining 20% (i.e., the unseen data from the future) for evaluation.

We perform hyper-parameter searches for each model and present the results in the following sections. All experiments were performed using an NVIDIA 1080 Ti GPU.

3.3.1.1 Non-temporal model

Various batch sizes were tried, but none had a significant impact on the performance of the model, we selected 300 users per mini-batches for the remaining experiments.

Because we use a VAE architecture, the encoder and decoder layers use the same number of layers and neurons but in a different order. We tested models with 1, 2, 4, and 6 layers but found that two produced the best results. We tried a combination of 10, 30, 60, 100, 200, 600, 900 neurons per layer and linearly increased the size when there was more than one layer. We report the results in Table 3.3. Increasing the number of layers and neurons didn't always perform better. It also used more memory and took longer to train. We found that one hidden layer with 600 neurons and a code size of the same size produced the best results.

Table 3.3 Depth and layer capacity for non-temporal model trained on Tou.TV.

HiddenSize	**CodeSize**	**RMSE**	**Recall @10**	**Precision @10**	**F1 @10**	**Coverage @10**
10	10	0.2831	0.5729	0.4745	0.5191	0.0279
30	30	0.1347	0.8222	0.677	0.7426	0.4681
100	100	0.1095	0.9668	0.8032	0.8774	0.7175
200	200	0.0881	0.9956	0.8296	0.9051	0.8601
600	600	0.0561	**0.9998**	**0.8337**	**0.9092**	**0.9338**
900	900	**0.0531**	0.9996	0.8335	0.909	0.9109
30,60	30	0.1396	0.7844	0.6427	0.7065	0.4122
60,100	60	0.1294	0.8812	0.7268	0.7966	0.5522
100,200	100	0.1272	0.917	0.7577	0.8298	0.6285
200,600	200	0.1021	0.9877	0.8225	0.8975	0.799
600,900	600	0.0756	0.9978	0.8317	0.9072	0.888
60,100,200	60	0.139	0.8403	0.6914	0.7586	0.5165
100,200,600	100	0.1169	0.9654	0.8019	0.8761	0.7277
30,60,100,200	30	0.1502	0.6567	0.5383	0.5917	0.2087

We started every experiment with a single mixture, i.e., a standard VAE with Gaussian prior. All the other hyper-parameters were tuned before increasing the number of mixtures. We tried 1, 2, 4 and 8 components in the mixture of Gaussian. The impact of the number of components is reported in Table 3.4. We found the best combination to be 4 mixtures of Gaussian.

Table 3.4 Number of mixtures
for non-temporal model trained on Tou.TV.

π	**RMSE**	**Recall@10**	**Precision@10**	**F1@10**	**Coverage@10**
2	0.0732	**0.9949**	**0.829**	**0.9044**	0.8855
4	**0.0705**	0.9942	0.8188	0.898	**0.9086**
8	0.0809	0.9919	0.8264	0.9017	0.883

Dropout was used to regularize the network. As described in Chapter 2, we tried a MC approximation and transformed the GMM into standard Gaussian when computing the KL. We report the results in Table 3.5. During our experiments, we found that training the VAE was notoriously difficult because the KL would vanish. Annealing the KL solved this issue. Therefore, $\beta = 0$ was set during the first few iterations of all the experiments and slowly increased the value until $\beta = 0.4$. Finally, all models were optimized with SGD with a learning rate of 0.001 and a momentum of 0.9.

Table 3.5 KL approximations for non-temporal model trained on Tou.TV.

KL	**RMSE**	**Recall@10**	**Precision@10**	**F1@10**	**Coverage@10**
MC	0.0809	0.9767	0.8033	0.8815	0.8223
APPROX	**0.0705**	**0.9942**	**0.8188**	**0.898**	**0.9086**

We compare our model against several baselines. For all systems, positive outcomes are defined if the RS is able to recommend items that are present in the history of a user.

1. Most popular: a popular choice in a cold start scenario, the most popular items are recommended.
2. Existing system: the current RS implemented on Tou.TV.
3. VAE with Gaussian prior: our model but with a single component.
4. CF: predictions are made using the user and item latent matrices learned with ALS.
5. CBF: the sparse matrix is transformed using TF-IDF and cosine distance is used for recommendations.

3.3.1.2 Temporal model

Because the temporal model uses RNNs to discover long-term dependencies among the sequential history of users, this model takes much longer to train. Therefore, the scope of the hyper-parameter research was limited compared to the non-temporal model that was faster to train.

Due to memory and time constraints, we used at most 64 users with 30 time steps per mini-batches. With this setting, an epoch took $\sim$ 10,605 iterations to complete and 10 epochs is equivalent to fours days of computing time. The VAE and classifier networks were trained end-to-end: the VAE is optimized at the same time as the classifier layer. This choice was made because the decoded output from the VAE would theoretically always improve (before overfitting) and the classifier network trained on each iteration of the decoder would learn how to deal with a noisy input. Hence, we wished the classifier to be able to predict the right content even with noisy values.

We tested models with one layer and experimented with 64, 128 and 256 neurons. We found the best model only needed 256 units. Results on the depth and layer capacity are reported in Table 3.6.

Table 3.6 Depth and layer capacity for temporal model trained on Tou.TV.

HiddenSize	**CodeSize**	**RMSE VAE**	**Precision @10**	**NDCG @10**	**MRR @10**	**Coverage @10**
64	64	0.1707	0.8714	0.4569	0.4528	0.2372
128	128	**0.0928**	0.859	0.6981	0.6875	0.3099
256	256	0.1153	**0.9324**	**0.773**	**0.7612**	**0.3641**

In addition to the experiments described above, we tried three different number of components: 2, 4, 6 and found that 6 components yielded the best results. We also performed a similar experiment to the non-temporal model where we tried the two approximations of the KL term. Results are presented in Table 3.7. Like the non-temporal model, we use an annealing schedule on the KL term to mitigate the KL vanishing issue.

Table 3.7 KL approximations for temporal model trained on Tou.TV.

KL	**RMSE VAE**	**Precision @10**	**Accuracy @10**	**NDCG @10**	**MRR @10**	**Coverage @10**
MC	0.1048	**0.9312**	0.6479	0.7521	0.7402	**0.4661**
APPROX	**0.0928**	0.9289	**0.6632**	**0.7642**	**0.7524**	**0.4661**

The classification layer takes as input the output of the VAE. We tried an MLP with one or two layers with a capacity of 128, 256, 600 or 1,200 neurons. We found that one layer with 256 units produced the best results. Optimization of the two networks was done end-to-end with SGD with a learning rate of 0.001 and a momentum of 0.9.

The temporal model was compared to several baselines. In this setting, performance is measured at the end of each mini-batch. A valid outcome signifies that the items recommended by the model are in the current time steps of the mini-batch. The baselines are the four following algorithms:

1. Most popular: The same baseline used by the non-temporal model, the most popular items on Tou.TV are used as recommendations.
2. Existing system: The same baseline used by the non-temporal model, the current CBF-based system deployed on Tou.TV.
3. RNN-MLP: A single LSTM is used to construct the hidden vector used by an MLP for recommendations.
4. GRU4Rec (Hidasi & Karatzoglou, 2018): A popular model that uses web sessions to make predictions. It uses the user id, item id and time steps of the interaction.

3.3.2 MovieLens-20M

Before training the models, we split the dataset similarly to the Tou.TV dataset. 99% of the data were used in the training set, while the last 1% was split equally between the validation and the test set. The newest interactions were used in the validation and test set. Finding the best hyper-parameters was done using the validation set. We used the same priming technique as the previous dataset.

Both models use the same baselines defined by the non-temporal and temporal model in the Tou.TV section minus the current system baseline.

3.3.2.1 Non-temporal model

Because this dataset and the Tou.TV dataset have similar characteristics, we used the best model found with the latter dataset as a starting point to our experiment. Depth-wise, we experimented with 1, 2 and 3 layers with a capacity of 60, 200, 600 and 1,200 neurons. Results are reported in Table 3.8. The best configuration was similar to the non-temporal model trained on the Tou.TV dataset, two layers with 200 and 600 neurons and a code size of 200 performed the best.

Table 3.8 Depth and layer capacity for non-temporal model trained on MovieLens-20M.

HiddenSize	CodeSize	RMSE	Precision@10	Coverage@10
60	60	0.1586	0.9239	0.0323
200	200	0.1511	0.9689	0.0413
600	600	0.1433	0.9871	0.048
1200	1200	**0.1359**	**0.9901**	0.0523
60,200	60	0.1495	0.9268	0.0355
200,600	200	0.141	0.9756	**0.0865**
600,1200	600	0.1451	0.9833	0.047
200,600,1200	200	0.1506	0.9627	0.042

Next, we experimented with 2, 4 and 6 components in the GMM and found that 6 provided the best results. We then checked the impact of the two approximations of the variational term and compiled the results in Table 3.9. Finally, we used the same KL annealing technique and SGD configuration as the non-temporal model for Tou.TV.

Table 3.9 KL approximations for non-temporal model trained on MovieLens-20M.

KL	RMSE	Precision@10	Coverage@10
MC	0.141	0.9622	0.0681
APPROX	**0.1348**	**0.9775**	**0.0865**

3.3.2.2 Temporal model

This model was trained using similar parameters as the Tou.TV temporal model. However, because the dimensionality of this dataset is higher than the previous one by $\sim 28\%$, we've tried doubling the number of parameters. Hence, we experimented with one hidden layer for the VAE and tried 256 and 512 neurons. The size of the latent space is the same as the hidden layer. Results of this analysis are reported in Table 3.15.

Then, we tried different combinations of depth and capacity for the classification layer and found that 512 units produced the best results. We also experimented with different amounts of mixture components, ranging from 2 to 8 and found the optimal to be 6. The impact of the KL approximations presented in this thesis was also evaluated. Results are reported in Table 3.16.

3.4 Results and analysis

In this section, we present the results of the non-temporal and temporal model on both Tou.TV and MovieLens-20M datasets. Results presented for the non-temporal model include the Precision@k, Recall@k, F1@k and Coverage@k. While the temporal model includes the same metrics along with the NDCG@k and MRR@k. A value of $k = 10$ and $k = 20$ was chosen for both datasets because we originally focused on quality rather than quantity for Tou.TV and transposed the same requirements on the second dataset. We note that because all users in the MovieLens-20M dataset have a history of at least 100 pieces of content, the Precision@k and Recall@k are the same. We chose to show only one value for simplicity.

Table 3.10 Data distribution of the test set for the aggregated dataset used by the non-temporal model and the features-enhanced dataset for the temporal model.

Dataset	**# of users**	**# of items**	**sparseness**
Tou.TV	1,002	4,080	0.0702
MovieLens-20M	209	14,922	0.03

To illustrate this trade-off, we show the PR and ROC curves as well as the area under the ROC curve (AUC) for a subset of items defined by the value of k. The ROC probability curve measure the true positive rate (TPR) against the false positive rate (FPR), both are defined by:

$$\text{TPR} = \frac{TP}{TP + FN} \tag{3.1}$$

$$\text{FPR} = \frac{FP}{TN + FP} \tag{3.2}$$

Where TP corresponds to the true positive (i.e., the recommended item is in the user history), TN to the true negative (i.e., items that were not recommended and are not part of the history), FN to the false negative (i.e., items that should have been recommended), FP to the false positive (i.e., items that were recommended and are not part of the history). In a recommendation scenario, a perfect ROC curve would yield a TPR equal to 1 and an FPR equal to 0 until all items in the top-k list are retrieved (i.e., all items are relevant).

3.4.1 Tou.TV

3.4.1.1 Non-temporal model

Of the two techniques proposed in this thesis to compute the variational term, the approximation of the GMM into a single Gaussian distribution performed the best (Table 3.11). Although not far behind, the MC approximation produced comparable results in terms of recall and precision, but the coverage of the system wasn't on par with the best model. Both models were trained under the same settings (i.e., same number of components, hidden layers, neurons and regularization) but with a different KL approximation method. The convergence of the best model was faster than the other by about 72% (Figure 3.5), which is normal since MC techniques requires multiple samples to converge. We believe the difference in coverage is due to the approximation of the KL. The approximation aggregates all the items given by the components of the GMM and weights them based on the components probabilities instead of considering only the most probable GMM and discarding the information of other GMMs.

Table 3.11 Comparison of baselines on non-temporal model trained on Tou.TV.

Model	k	Recall@k	Precision@k	F1@k	Coverage@k
CF (ALS)	10	0.7373	0.5719	0.6441	0.5863
CF (ALS)	20	0.8612	0.4709	0.6088	0.7386
CBF (TFIDF)	10	0.5491	0.5009	0.5238	0.1954
CBF (TFIDF)	20	0.5837	0.3517	0.4389	0.6447
VAE	10	0.8739	0.7107	0.7839	0.5152
VAE	20	0.8593	0.4875	0.6221	0.6853
Current System	10	0.1904	0.1715	0.1804	0.2843
Current System	20	0.2251	0.1477	0.1783	0.3274
Most Popular	10	0.5789	0.4773	0.5232	0.0254
Most Popular	20	0.6227	0.3548	0.452	0.0508
Ours-MC	10	0.9767	0.8033	0.8815	0.8223
Ours-MC	20	0.9606	0.5622	0.7093	0.9645
Ours-SingleGaussian	10	**0.9942**	**0.8188**	**0.898**	0.9086
Ours-SingleGaussian	20	0.9807	0.578	0.7274	**0.9873**

We find that the two proposed methods outperform popular baselines. A basic VAE is able to recommend items with high confidence as shown in Figure 3.4. The simple improvements proposed in this thesis are able to increase the TPR and decrease the FPR of this popular architecture. When looking at the ROC curves, the closest popular RS to our models is the CF with ALS optimization. However, due to the sparseness of the dataset, these curves might present an optimistic view of the performance. Hence, by looking at the PR curves (Figure 3.4), the CF model doesn't perform as well as the proposed methods by nearly 39%. Table 3.11 describes the difference in performance of all methods, with the current system deployed on Tou.TV being the worst. This is because the recommendations corresponds to a very small subset of the catalogue. When presenting a small number of recommendations ($k = 10$), ~ 81% of our predictions are correct and covers a high proportion of the catalogue, compared to ~ 57% for collaborative systems, and ~ 47% for the popularity approach. The coverage drops substantially with other systems as they're more biased towards popular items.

We found that increasing the layer depth and capacity didn't always perform better (Table 3.3). We found that increasing the capacity of the VAE yielded consistent improvement until around 600 neurons. One layer was sufficient and produced the best results. More than two layers

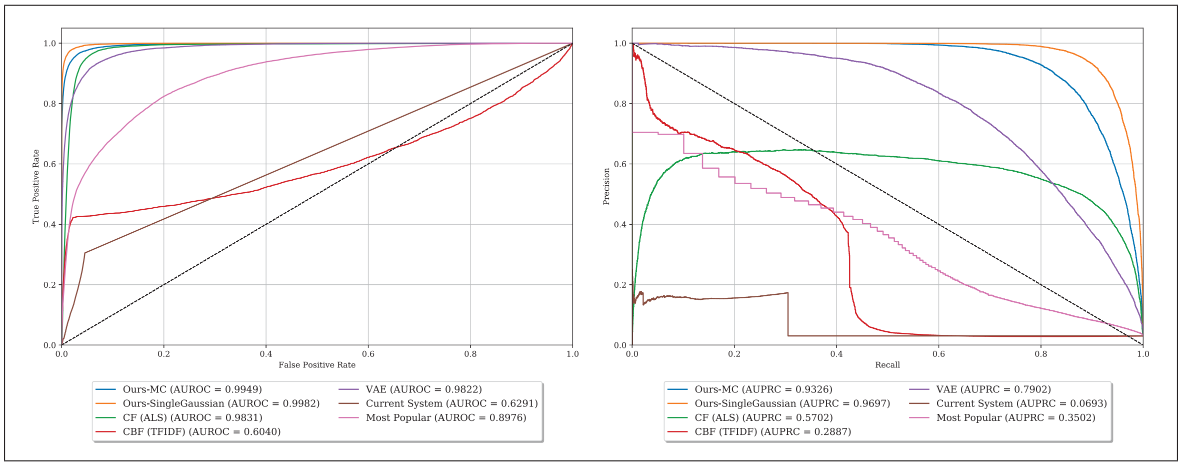


Figure 3.4 ROC and PR curves for the non-temporal model trained on Tou.TV.

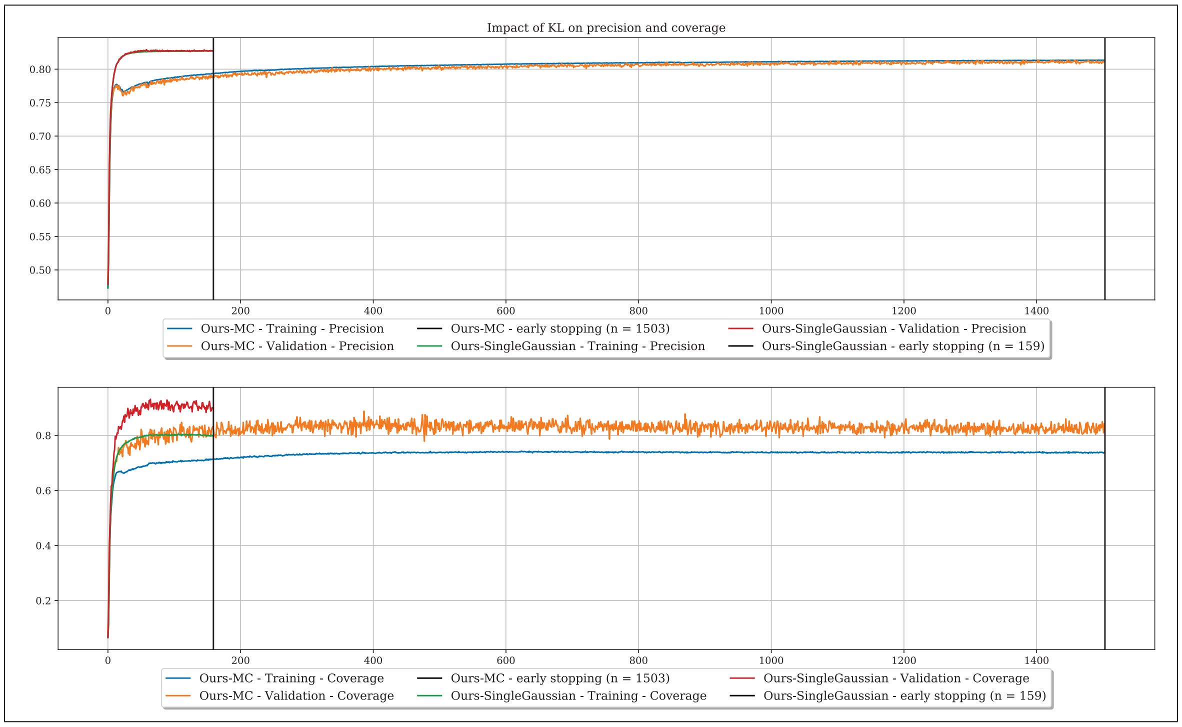


Figure 3.5 Precision and coverage curves for the non-temporal model trained on Tou.TV.

produced significantly worst results. As for the number of components in the GMM, 2 gave the best precision while 4 achieved the highest coverage. We note that the difference in precision is only by $\sim 1\%$ between 2 and 4 components. A higher number of mixtures achieved a similar precision but a worse coverage. Based on RC's success criteria, we deemed the best model to be a VAE with a GMM of 4 mixtures, one hidden layers of 600 units and a KL approximated to a single Gaussian distribution.

3.4.1.2 Temporal model

Unlike the non-temporal task, the MC approximation of the KL term produced better results when a small value of k is selected. At $k = 20$, the other approximation performs slightly better in terms of precision but not in coverage. We also note that this approximation of the KL term achieved better performances when ranking the recommendations by $\sim 1\%$. Finally, the RMSE reported in Table 3.7 indicates that the MC approximation method is not able to reconstruct the original feature matrix as well as the other method. In light of the other metrics, the models shouldn't be optimized only on the RMSE.

Table 3.12 Comparison of baselines on temporal model trained on Tou.TV.

Model	**k**	**Precision@k**	**NDCG@k**	**MRR@k**	**Coverage@k**
RNN+MLP	10	0.4729	0.2223	0.1993	0.1932
RNN+MLP	20	0.5599	0.25	0.2048	0.2159
Current System	10	0.1258	0.0362	0.0353	0.1949
Current System	20	0.2462	0.054	0.0431	0.2373
Most popular	10	0.5746	0.3743	0.3666	0.0282
Most popular	20	0.7021	0.4052	0.3756	0.0565
GRU4REC	10	0.5403	0.2751	0.2572	0.658
GRU4REC	20	0.6253	0.3073	0.2632	**0.6753**
Ours-MC	10	0.9312	0.7521	0.7402	0.4661
Ours-MC	10	0.9695	0.7725	0.7429	0.5169
Ours-SimpleGaussian	10	0.9289	0.7642	0.7524	0.4661
Ours-SimpleGaussian	20	**0.9698**	**0.7831**	**0.7553**	0.5113

The class imbalance is clearly seen when looking at the ROC and PR curves (Figure 3.6). Looking at the ROC curves, the models proposed in this thesis outperformed the baselines, but are followed closely by the most popular and "RNN+MLP" baselines. However, the PR curves clearly indicates that the baselines are suboptimal and do not perform better than random.

We explain these results by the fact that the temporal dataset contains minimal information at each time steps (i.e., only the content at t_{t-1} are known while the non-temporal model considers all content in its prediction). The "RNN+MLP" baseline was trained using the same features as our model. It's also important to note that the popular model "GRU4REC" was only trained on the content seen by a user at each time step while our model was enhanced with extra features.

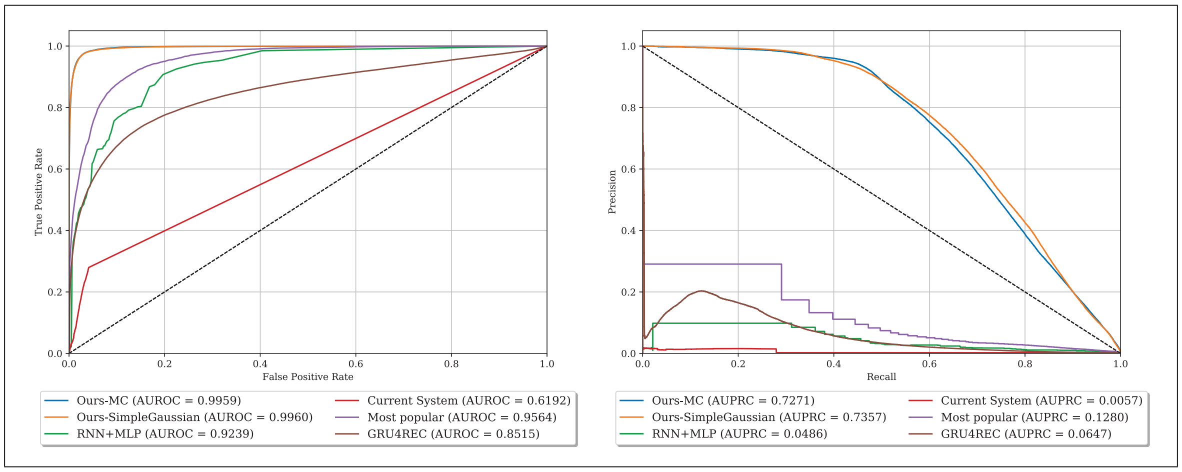


Figure 3.6 ROC and PR curves for the temporal model trained on Tou.TV.

As expected, when performing the optimal hyper-parameters search (Table 3.6), we found that increasing the layer capacity made the network perform better and didn't increase the time of computation by much. Furthermore, we found that a lower RMSE, which characterizes the network reconstruction capacity, isn't necessarily correlated with better performance. Similarity, a higher code size made the network better at ordering the prediction by $\sim 30\%$ as well as increasing the coverage of the system. Based on those results and RC's requirements, the best temporal network had one layer, 256 units, 6 mixtures and used the MC approximation.

3.4.2 MovieLens-20M

3.4.2.1 Non-temporal model

We compared our proposed models to the baselines in Table 3.13. Considering the dimensionality and sparsity of the dataset, optimizing for a high coverage was challenging. The CF method produced really good results regarding this metric. However, the precision indicates that this system wasn't as good as other baselines to recommend the right content. When demanding high confidence for recommendations, our proposed systems were able to outperform other RS in terms of precision by at least ~ 3% while achieving a high coverage compared to the best value. In this setting, the performance of the two approximations found in this thesis do not have a significant difference in terms of precision nor coverage. But like the previous dataset, the model using the approximation of the GMM into a single Gaussian distribution has generally better performance and converge faster (Figure 3.7).

Table 3.13 Comparison of baselines on non-temporal model trained on MovieLens-20M.

Model	**k**	**Precision@k**	**Coverage@k**
CF (ALS)	10	0.712	0.1242
CF (ALS)	20	0.7038	**0.1949**
CBF (TFIDF)	10	0.8928	0.0109
CBF (TFIDF)	20	0.8861	0.0199
VAE	10	0.9383	0.0358
VAE	20	0.9158	0.0587
Most Popular	10	0.689	0.0008
Most Popular	20	0.6742	0.0017
Ours-MC	10	0.9622	0.0681
Ours-MC	20	0.9498	0.1037
Ours-SimpleGaussian	10	**0.9756**	0.0865
Ours-SimpleGaussian	20	0.9584	0.1112

Looking at the ROC curves (Figure 3.8), the two models presented are able to achieve high TPR while keeping the FPR low. A standard VAE is also able to achieve high performance comparable to a CF system.

Similar to the previous dataset, the sparsity of this dataset is very high and is reflected in the difference of performance shown between the ROC and the PR curves (Figure 3.8). The standard VAE outperform the CF system by at least ~ 21%. But when enhancing this architecture the best approximation achieves ~ 3% more precision. We further conclude our evaluation that most of the baselines are not capable of producing good recommendations and sometimes perform no better than random.

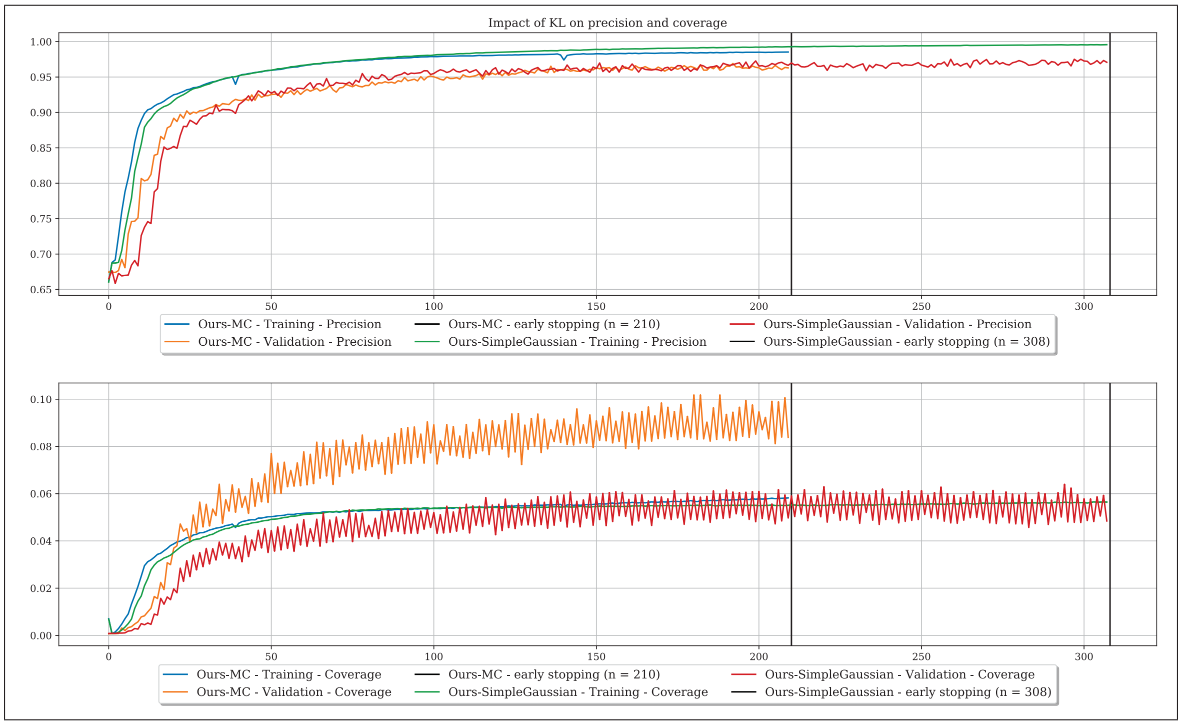


Figure 3.7 Precision and coverage curves for the non-temporal model trained on MovieLens-20M.

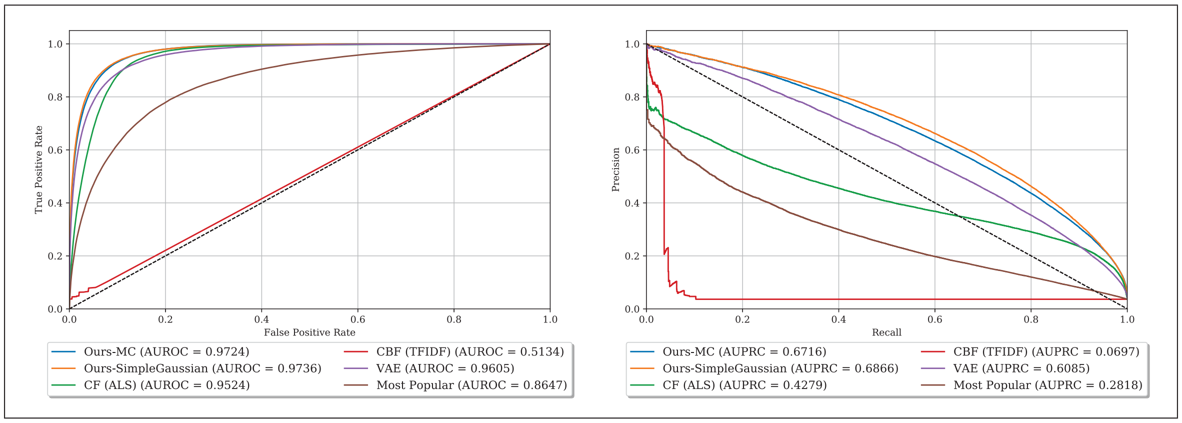


Figure 3.8 ROC and PR curves for the non-temporal model trained on MovieLens-20M.

Increasing the depth of the network and neuron capacity didn't always perform better. Adding more neurons provided better results until 1,200 units. Similarly, adding more layers achieved a higher coverage but at the cost of a longer training time. After 2 layers the performance of the network didn't increase. As for the approximation of the GMM, the approximation to a standard Gaussian distribution provided a small gain of precision but $\sim 1.2\%$ in coverage. Therefore, the architecture that is the most appropriate for the problem has 2 layers with 200 and 600 neurons, 6 components and uses the approximation of the GMM to a standard Gaussian.

3.4.2.2 Temporal model

Both approximation of the KL term produced similar results in terms of precision and coverage, as reported in Table 3.14. We experimented with different values of k , but the outcome was the same. Compared to other baselines, the models presented in this thesis clearly outperform the "most popular" and "RNN+MLP" baselines as well as the popular system "GRU4REC". Optimizing for a higher code size only increased the precision by less than a 1%, but increased the ranking capability of the network (see the NDCG and MRR metrics) by $\sim 2\%$.

Table 3.14 Comparison of baselines on temporal model trained on MovieLens-20M.

Model	k	Precision@k	NDCG@k	MRR@k	Coverage@k
RNN+MLP	10	0.0198	0.0057	0.0055	0.002
RNN+MLP	20	0.0376	0.0085	0.0067	0.0035
Most Popular	10	0.0248	0.0077	0.0075	0.0017
Most Popular	20	0.0461	0.0111	0.0089	0.0033
GRU4REC	10	0.0858	0.0308	0.0298	0.2004
GRU4REC	20	0.1413	0.0409	0.0335	0.2837
Ours-MC	10	0.6042	0.3761	0.3635	0.2155
Ours-MC	20	**0.6997**	**0.406**	0.3701	0.2894
Ours-SimpleGaussian	10	0.6014	0.3757	0.3637	0.2256
Ours-SimpleGaussian	20	0.6963	0.4054	**0.3702**	**0.2961**

The ROC and PR curves (Figure 3.9) indicate that the two models presented achieve high precision but low recall. They return very few results, but the majority of the predictions are in the user's history. In a top-k setting, this is a desirable outcome. Inspecting the precision-recall curves confirms the previous statement, the two approximations perform almost congruently. The influence of the class imbalance has a significant impact on the baselines. While the ROC curves of the most popular and "GRU4REC" approaches performs better than the "RNN+MLP", the most popular exceed the other baselines in the PR graph. Comparing the area under the curves, the best model is almost 1.5 times more accurate than the best baseline in the ROC graph, while the best model is almost 4.5 times better in the precision-recall space.

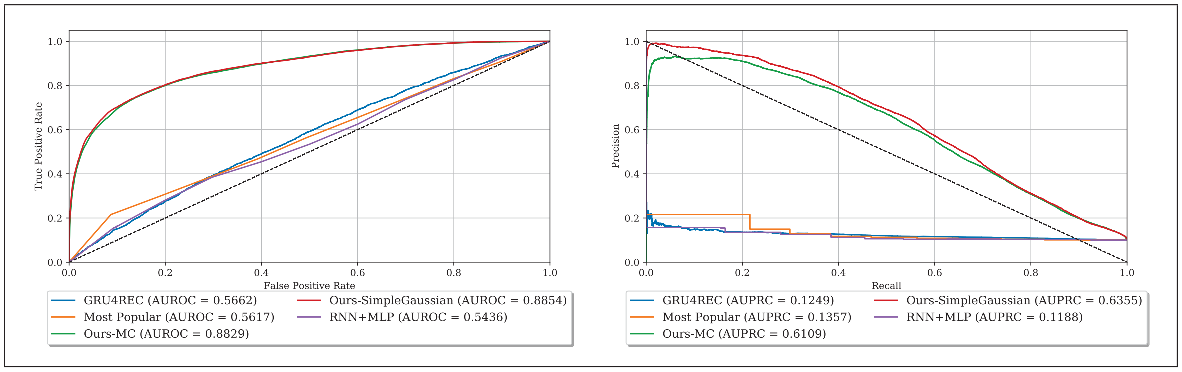


Figure 3.9 ROC and PR curves for the temporal model trained on MovieLens-20M.

Doubling the code size had a minor impact on the performance of the network (Table 3.15), but was still able to produce better results in terms of precision and ranking at the expense of the coverage by $\sim 1\%$. The two approximations of the KL term also didn't impact the performance of the network by much. In light of these results, the most appropriate architecture for this task has one hidden layer and a code size of 512 units, 6 Gaussian mixtures and the KL is approximated to a single Gaussian.

Table 3.15 Depth and layer capacity for temporal model trained on MovieLens-20M.

HiddenSize	**CodeSize**	**RMSE VAE**	**Precision @10**	**NDCG @10**	**MRR @10**	**Coverage @10**
256	256	0.0961	0.5989	0.3576	0.345	**0.2358**
512	512	**0.0891**	**0.6042**	**0.3761**	**0.3635**	0.2155

Table 3.16 KL approximations for temporal model trained on MovieLens-20M.

KL	**RMSE VAE**	**Precision @10**	**Accuracy @10**	**NDCG @10**	**MRR @10**	**Coverage @10**
APPROX	0.0961	0.6014	**0.2603**	0.3757	**0.3637**	**0.2256**
MC	**0.0891**	**0.6042**	0.2581	**0.3761**	0.3635	0.2155

In this chapter, we analyzed the performance of the two models that were proposed in this thesis. The non-temporal and the temporal model were compared against several baselines on two datasets. The first was provided by Radio-Canada and contained information about the visit of a user, such as the content seen, the duration and the device used. The second is a popular dataset called MovieLens-20M, a popular choice in the recommender system community when comparing algorithms. The next chapter concludes this thesis and is a summary of the methods presented in this document, their limitations and directions for future research.

CONCLUSION AND RECOMMENDATIONS

In this chapter, we summarize our contributions, discuss the limitations of the presented approach and provide direction towards future work.

4.1 Contributions

As more information becomes available, consumers might find themselves lost and not able to correctly find the best content for them. Recommendation systems are increasingly popular because they provide the right information at the right time to the users. There are many challenges related to developing the perfect system. Capturing the interaction between a user and content is key to produce the next recommendation. Implicit interactions are binary indicators describing the engagement of a user and to a content. They're becoming a popular choice of interactions due to their simplicity and ease of collection. However, they provide many challenges because they don't produce any information about the intent of the user nor the appreciation.

Radio-Canada is the official French broadcaster in Canada. We were asked to develop a novel recommender system for their streaming platform called "Tou.TV". This system should be contextual, personalized based on the history of each user and use their implicit data. In this thesis, we present a hybrid approach extending traditional variational autoencoder architectures with a Gaussian mixture prior to better describe the latent space with multiple Gaussians. We further expand our approach into two models, one for different scenarios. The first considers only the user-item matrix while second enhances it by adding temporal and contextual information. We show that our systems perform remarkably well compared to popular off-the-shelf solutions. We also demonstrate the efficacy of our models on the popular dataset MovieLens-20M.

4.2 Limitations

While our models are able to outperform most of the simple baselines, some problems remains unsolved. The models are able to achieve a high precision most of the time. However, the coverage of the recommender systems are not perfect. While the non-temporal model is able to achieve $\sim 90\%$ of coverage on the Tou.TV dataset, it rapidly falls on the MovieLens-20M dataset with only $\sim 8\%$.

The temporal architecture presented in this thesis is practical when current context should be taken in consideration when recommending content. This architecture was able to outperform baselines in terms of precision and ranking metrics. But compared to the popular "GRU4REC" model, the coverage is inferior by at least $\sim 19\%$.

4.3 Future work

Due to the nature of the problem presented in this thesis, we exclusively focused our work on implicit data. It would be interesting to try both models on explicit data because of the extra information provided by this information. The non-temporal model could be enhanced with ratings values while the temporal model could contain a rating column.

The results presented in this thesis are empirical. Deploying the systems in a real environment could provide more feedback and might reveal their weaknesses or strengths. Qualitative testing of our models might provide better insights.

Finally, applying VAE to recommender systems is relatively new and investigating other types of architecture changes, like we did with the prior might lead to interesting attributes and better results.

BIBLIOGRAPHY

Abbi, R., El-Darzi, E., Vasilakis, C. & Millard, P. (2008, Sep.). Analysis of stopping criteria for the EM algorithm in the context of patient grouping according to length of stay. *2008 4th International IEEE Conference Intelligent Systems*, 1, 3-9-3-14. doi: 10.1109/IS.2008.4670413.

Aggarwal, C. C. (2016). *Recommender Systems: The Textbook* (ed. 1st). Springer Publishing Company, Incorporated.

Ali, H. H. & Kadhum, L. E. (2017). K-Means Clustering Algorithm Applications in Data Mining and Pattern Recognition.

Amatriain, X. & Basilico, J. (2012). Netflix Recommendations: Beyond the 5 stars (Part 1). Consulted at https://medium.com/netflix-techblog/netflix-recommendations-beyond-the-5-stars-part-1-55838468f429.

Balabanović, M. & Shoham, Y. (1997). Fab: Content-based, Collaborative Recommendation. *Commun. ACM*, 40(3), 66–72. doi: 10.1145/245108.245124.

Bedi, P., Vashisth, P., Khurana, P. & Preeti. (2013). Modeling user preferences in a hybrid recommender system using type-2 fuzzy sets. *IEEE International Conference on Fuzzy Systems*. doi: 10.1109/FUZZ-IEEE.2013.6622471.

Bhatia, N. & Vandana. (2010). Survey of Nearest Neighbor Techniques. *ArXiv*, abs/1007.0085.

Billsus, D., Pazzani, M. J. & Chen, J. (2000). Learning agent for wireless news access. *International Conference on Intelligent User Interfaces, Proceedings IUI*.

Bishop, C. M. (2006). Pattern Recognition and Machine Learning. *Springer-Verlag*.

Bishop, C. M. & Nasrabadi, N. M. (2007). Pattern Recognition and Machine Learning. *J. Electronic Imaging*, 16, 049901.

Blei, D. M., Kucukelbir, A. & McAuliffe, J. D. (2017). Variational Inference: A Review for Statisticians. *ArXiv*, abs/1601.00670.

Burke, R. (2002). Hybrid Recommender Systems: Survey and Experiments. *User Modeling and User-Adapted Interaction*, 12. doi: 10.1023/A:1021240730564.

Çano, E. & Morisio, M. (2017). Hybrid recommender systems: A systematic literature review. *Intelligent Data Analysis*, 21(6), 1487–1524. doi: 10.3233/IDA-163209.

Cao, R. (2002). A short introduction to nonparametric curve estimation.

Chen, R., Hua, Q., Chang, Y. S., Wang, B., Zhang, L. & Kong, X. (2018). A survey of collaborative filtering-based recommender systems: from traditional methods to hybrid methods based on social networks. *IEEE Access*. doi: 10.1109/ACCESS.2018.2877208.

Chung, J., Kastner, K., Dinh, L., Goel, K., Courville, A. & Bengio, Y. (2015). A Recurrent Latent Variable Model for Sequential Data.

Davidson, J., Liebald, B., Liu, J., Nandy, P., Van Vleet, T., Gargi, U., Gupta, S., He, Y., Lambert, M., Livingston, B. & Sampath, D. (2010). The YouTube Video Recommendation System. *Proceedings of the Fourth ACM Conference on Recommender Systems*, (RecSys '10), 293–296. doi: 10.1145/1864708.1864770.

De Campos, L. M., Fernández-Luna, J. M., Huete, J. F. & Rueda-Morales, M. A. (2010). Combining content-based and collaborative recommendations: A hybrid approach based on Bayesian networks. *International Journal of Approximate Reasoning*. doi: 10.1016/j.ijar.2010.04.001.

Doersch, C. (2016). Tutorial on Variational Autoencoders. *ArXiv*, abs/1606.05908.

Durrieu, J. L., Thiran, J. P. & Kelly, F. (2012). Lower and upper bounds for approximation of the Kullback-Leibler divergence between Gaussian mixture models. *ICASSP, IEEE International Conference on Acoustics, Speech and Signal Processing - Proceedings*, (Mc), 4833–4836. doi: 10.1109/ICASSP.2012.6289001.

Gomez-Uribe, C. A. & Hunt, N. (2015). The Netflix Recommender System : Algorithms , Business Value ,. *ACM Transactions on Management Information Systems (TMIS)*, 6.

Goodfellow, I., Bengio, Y. & Courville, A. (2016). *Deep Learning*. The MIT Press.

Gopalan, P., Hofman, J. M. & Blei, D. M. (2015). Scalable Recommendation with Hierarchical Poisson Factorization. *Proceedings of the Thirty-First Conference on Uncertainty in Artificial Intelligence - UAI '15*, 326-335.

Graves, A. (2013). Generating Sequences With Recurrent Neural Networks. *ArXiv*, abs/1308.0850.

Guo, G., Wang, H., Bell, D., Bi, Y. & Greer, K. (2003). KNN model-based approach in classification. *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics).*

Harper, F. M. & Konstan, J. A. (2015). The MovieLens Datasets: History and Context. *ACM Trans. Interact. Intell. Syst.*, 5(4), 19:1–19:19. doi: 10.1145/2827872.

Hidasi, B. & Karatzoglou, A. (2018). Recurrent Neural Networks with Top-k Gains for Session-based Recommendations. *Proceedings of the 27th ACM International Conference on Information and Knowledge Management*, (CIKM '18), 843–852. doi: 10.1145/3269206.3271761.

Higgins, I., Matthey, L., Pal, A., Burgess, C., Glorot, X., Botvinick, M. M., Mohamed, S. & Lerchner, A. (2017). beta-VAE: Learning Basic Visual Concepts with a Constrained Variational Framework. *ICLR*.

Higgins, I., Amos, D., Pfau, D., Racanière, S., Matthey, L., Rezende, D. J. & Lerchner, A. (2018). Towards a Definition of Disentangled Representations. *ArXiv*, abs/1812.02230.

Horný, M. & Lin, M.-Y. (2014). Bayesian Networks.

Hu, Y., Koren, Y. & Volinsky, C. (2008, dec). Collaborative Filtering for Implicit Feedback Datasets. *2008 Eighth IEEE International Conference on Data Mining*, 148(4), 263–272. doi: 10.1109/ICDM.2008.22.

Jorge, A. M., Vinagre, J., Domingues, M., Gama, J., Soares, C., Matuszyk, P. & Spiliopoulou, M. (2017). Scalable Online Top-N Recommender Systems (vol. 2, pp. 3–20). doi: 10.1007/978-3-319-53676-7_1.

Kingma, D. P. & Welling, M. (2013a). Auto-Encoding Variational Bayes. *CoRR*, abs/1312.6114.

Kingma, D. P. & Welling, M. (2013b). Auto-Encoding Variational Bayes. 1-14.

Koren, Y. (2008). Factorization meets the neighborhood: A multifaceted collaborative filtering model. *Proceedings of the ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. doi: 10.1145/1401890.1401944.

Lampropoulos, A. S., Lampropoulou, P. S. & Tsihrintzis, G. A. (2012). A Cascade-Hybrid Music Recommender System for mobile services based on musical genre classification and personality diagnosis. *Multimedia Tools and Applications*. doi: 10.1007/s11042-011-0742-0.

Leonard, D. (2016). Spotify Is Perfecting the Art of the Playlist. Consulted at https://www.bloomberg.com/news/articles/2016-09-21/spotify-is-perfecting-the-art-of-the-playlist.

Liang, D., Krishnan, R. G., Hoffman, M. D. & Jebara, T. (2018a). Variational Autoencoders for Collaborative Filtering. *Proceedings of the 2018 World Wide Web Conference on World Wide Web - WWW '18*, pp. 689–698. doi: 10.1145/3178876.3186150.

Liang, D., Krishnan, R. G., Hoffman, M. D. & Jebara, T. (2018b). Variational Autoencoders for Collaborative Filtering [Audiovisual Material]. doi: 10.1145/3178876.3186150.

Linden, G., Smith, B. & York, J. (2003). Amazon.com recommendations: Item-to-item collaborative filtering. *IEEE Internet Computing*, 7, 76-80. doi: 10.1109/MIC.2003.1167344.

Ma, H., Liu, C., King, I. & Lyu, M. R. (2011). Probabilistic factor models for web site recommendation. 265. doi: 10.1145/2009916.2009955.

MacKenzie, I., Meyer, C. & Noble, S. (2013). How retailers can keep up with consumers. Consulted at https://www.mckinsey.com/industries/retail/our-insights/how-retailers-can-keep-up-with-consumers.

Martens J., S. I. (2011). Learning Recurrent Neural Networks with Hessian-free Optimization. *Proceedings of the 28th International Conference on International Conference on Machine Learning*, (ICML'11), 1033–1040. Consulted at http://dl.acm.org/citation.cfm?id=3104482.3104612.

Mooney, R. J. & Roy, L. (2000). Content-based book recommending using learning for text categorization. *Proceedings of the ACM International Conference on Digital Libraries*.

Moore, J. F., Chen, S., Thorsten, J. & Turnbull, D. (2013). Taste Over Time: The Temporal Dynamics of User Preferences. *Proc. International Society for Music Information Retrieval Conference (ISMIR)*, 401-406.

Oord, A. v. d., Dieleman, S. & Schrauwen, B. (2013). Deep content-based music recommendation. *Inst Electr Eng, Conf Publ*.

O'Bryant, J. (2017). A survey of music recommendation and possible improvements.

Paterek, A. (2007). Improving regularized singular value decomposition for collaborative filtering. *KDD Cup and Workshop*. doi: 10.1145/1557019.1557072.

Ramos, J. (2003). Using TF-IDF to Determine Word Relevance in Document Queries. *New Educational Review*. doi: 10.15804/tner.2015.42.4.03.

Rezende, D. J., Mohamed, S. & Wierstra, D. (2014). Stochastic Backpropagation and Approximate Inference in Deep Generative Models. *ICML*.

S., H. & J., S. (1997). Long Short-Term Memory. *Neural Comput.*, 9(8), 1735–1780. doi: 10.1162/neco.1997.9.8.1735.

Salakhutdinov, R., Mnih, A. & Hinton, G. (2007). Restricted Boltzmann machines for collaborative filtering [Audiovisual Material]. doi: 10.1145/1273496.1273596.

Schuster, M. & Paliwal, K. K. (1997). Bidirectional recurrent neural networks. *IEEE Transactions on Signal Processing*, 45(11), 2673–2681. doi: 10.1109/78.650093.

Sedhain, S., Menon, A. K., Sanner, S. & Xie, L. (2015). AutoRec : Autoencoders Meet Collaborative Filtering. 0-1.

Seyednezhad, S. M. M., Cozart, K. N., Bowllan, J. A. & Smith, A. O. (2018). A Review on Recommendation Systems: Context-aware to Social-based. Consulted at http://arxiv.org/abs/1811.11866.

Sorzano, C. O. S., Vargas, J. & Pascual-Montano, A. D. (2014). A survey of dimensionality reduction techniques. *ArXiv*, abs/1403.2877.

Srivastava, N., Hinton, G., Krizhevsky, A., Sutskever, I. & Salakhutdinov, R. (2014). Dropout: A Simple Way to Prevent Neural Networks from Overfitting. *Journal of Machine Learning Research*, 15, 1929-1958. Consulted at http://jmlr.org/papers/v15/srivastava14a.html.

Steinley, D. L. (2006). K-means clustering: a half-century synthesis. *The British journal of mathematical and statistical psychology*, 59 Pt 1, 1-34.

Sutskever & Ilya. (2013). Training Recurrent neural Networks. *PhD thesis*, 101.

Takács, G., Pilászy, I., Németh, B. & Tikk, D. (2008). Matrix Factorization and Neighbor Based Algorithms for the Netflix Prize Problem. *Proceedings of the 2008 ACM Conference on Recommender Systems*, (RecSys '08), 267–274. doi: 10.1145/1454008.1454049.

Takáes, G., Pilászy, I., Németh, B. & Tikk, D. (2009). Scalable collaborative filtering approaches for large reeommender systems. *Journal of Machine Learning Research*.

Tan, Y. K., Xu, X. & Liu, Y. (2016). Improved Recurrent Neural Networks for Session-based Recommendations. 0-5. doi: 10.1145/2988450.2988452.

Vincent, P., Larochelle, H., Bengio, Y. & Manzagol, P.-A. (2008). Extracting and Composing Robust Features with Denoising Autoencoders. *Proceedings of the 25th International Conference on Machine Learning*, (ICML '08), 1096–1103. doi: 10.1145/1390156.1390294.

Vincent, P., Larochelle, H., Lajoie, I., Bengio, Y. & Manzagol, P.-A. (2010). Stacked Denoising Autoencoders: Learning Useful Representations in a Deep Network with a Local Denoising Criterion. *Journal of Machine Learning Research*, 11, 3371-3408. doi: 10.1111/1467-8535.00290.

Vozalis, M. G. & Margaritis, K. G. (2007). A Recommender System using Principal Component Analysis. *11th Panhellenic Conference in Informatics References*, 271-283.

Wang, H., Wang, N. & Yeung, D.-Y. (2014). Collaborative Deep Learning for Recommender Systems.

Y., B., P., S. & P., F. (1994). Learning Long-term Dependencies with Gradient Descent is Difficult. *Trans. Neur. Netw.*, 5(2), 157–166. doi: 10.1109/72.279181.

Yoshii, K., Goto, M., Komatani, K., Ogata, T. & Okuno, H. G. (2008). An efficient hybrid music recommender system using an incrementally trainable probabilistic generative model. *IEEE Transactions on Audio, Speech and Language Processing*. doi: 10.1109/TASL.2007.911503.

Zhang, S., Yao, L., Sun, A. & Tay, Y. (2019). Deep learning based recommender system: A survey and new perspectives. *ACM Computing Surveys*, 52(1), 1–35. doi: 10.1145/3285029.

Zheng, Y., Tang, B., Ding, W. & Zhou, H. (2016). A Neural Autoregressive Approach to Collaborative Filtering. 48.