What Do you Like? - Multi-Network for Implicit Video Recommendations

Author:
Omer Nivron

Supervisors:
Prof. John Shaw-Taylor
Dr. Emine Yilmaz

A dissertation submitted in partial fulfillment
of the requirements for the degree of
MSc Computational Statistics and Machine Learning
of
University College London,
Department of Computer Science.

February 27, 2018

Abstract

To predict future choices, recommendation systems rely on user's past experiences. But while this is a reasonable method to use, it ignores one significant question: were you satisfied with all those choices that you had made? If so, how often do you wear the Christmas sweater that you bought last winter, because you found it funny? Or, how much do you actually like the Britney Spears song that your little sister convinced you to listen to last week? This thesis aims to help you avoid such unpleasant moments. To do so, we introduce the novel **Like** model: a multi-network combining an RNN and a Neural Network meeting at the output layer: making use not only of common user and video features as age and genre but also of the novel video subtitle feature. Thus we learn about user's likes and dislikes. Our model proves to be as competitive as top state of the art recommendation systems and much more interpertable than them.

• All code is readily available at: `https://github.com/omernivro/recommendation_systems`

Acknowledgements

I would like to thank all the people that has supported me in bringing this master thesis into life. Special thanks to my supervisors Professor John Shaw-Taylor and Dr. Emine Yilmaz and to my colleague Andrew Mpaplika that has spent countless hours with me and was a great supporter professionally and socially. Further thanks to hezi for everything.

Contents

List of Figures

List of Tables

This report is submitted as part requirement for the MSc/MRes Degree in Machine Learning/CSML/Data Science at University College London. It is substantially the result of my own work except where explicitly indicated in the text.

Chapter 1

Introduction

"Did you like this video?"
Finishing a video at Netflix, this is a question you probably have encountered many times. Yet, how many of those did you actually answer? Don't worry to say "none", you won't be alone. Implicit data, i.e. feedback retrieved indirectly from user's behaviour, account for majority of data companies, governments and even scientists have currently available. Yet, most of them recommend millions of products and services to their users and citizens every day by asking themselves one single question: What did the user watch, listen to, or buy in the past? Recommendation systems using implicit feedback data base their predictions upon a person's past experience. But while this may seem a reasonable method to use, left on its on, it might lead to an unpleasant user experience. Take the following example: Jane used her Sunday night to watch "Selma" together with her parents. She did not like it. But how many historical dramas does she now need to watch until something likable pops up again?

1.0.1 Problem

The gist of our problem is to recommend that particular video or series a person wants to watch next. More specifically, we want to know *What* to recommend?, *When* to recommend it? and *Where* to place our recommendation? - In the following, we shall call this problem the **Joint** recommendation problem (figure 1).

Already the first question, ***What*** to recommend, poses a challenge. Imagine Jane, a user of our video online portal. Despite using our website on a daily basis,

our knowledge about her is at best partial. We may know which videos she has watched, but little about how much she liked them - rating videos has never really been her thing. But to predict Jane's next view, it is exactly these likes and dislikes that we need to understand. Previous models have provided a good foundation [2][61][10][68]. Yet, there is ample room for improvement. While some acknowledge dislikes but tackle it either implicitly through broad assumptions [6] and in a non interpertable way [39], most of them ignore the relevance of dislikes to predict user's next view. But as our paper argues, dislikes matter for at least two reasons: First, we learn from psychologists and social scientists that a person's preferences and actions are influenced by both, positive and negative experiences [23]. Therefore, knowing what a person dislikes will provide us with a good negative advice. Second, a person may fall short to negativity biases [43][65]; Often, a negative experience can be far more influential despite it being equal in intensity to a positive one. Think, for example, do you appreciate every moment the road to work is clear, or are you more likely to only complain once you are stuck in the traffic jam?

Once we have learned what a person likes to watch, we need to understand ***When*** that person wants to watch it. Video recommendation systems relate to the temporal nature of human behaviour. It is likely that our preferences change over time, and/or our choices are made in a sequential order. [79] has contributed on that subject matter, by separating recommendation problems into two categories: those that are time dependent like RNNs[33] and TimeSVD++[46], also called sequential recommendation problems, and those that are time independent, the so-called spatial recommendation problems as the matrix completion methods[67] and Restricted Bolzman Machines[68]. In this light, extra attention shall be paid to new users and the problem of cold start [70]. If a user is new to a platform, no history exists that could help us recommend him or her when to watch which video. But does that mean we need to leave him or her alone in the jungle of videos and TV shows?

Lastly, it is not enough to know when to recommend what, but also ***Where*** to place that recommendation. Often for a service provider it makes sense to provide

more than one option. Yet, space is limited; a screen can only present to a user few recommendations. Moreover, recent studies in behavioural economics show that the amount of options and the variety of options provided, influence the decisions a user makes. For example, placing a comedy next to two tragedies may increase the chances of watching that comedy. Hence, for a user to find its way and have a positive experience, a service provider has to organize its products smartly [4]. Our paper aims to fill those research gaps and address the **Joint** recommendation problem by using data from the British TV platform Channel 4.

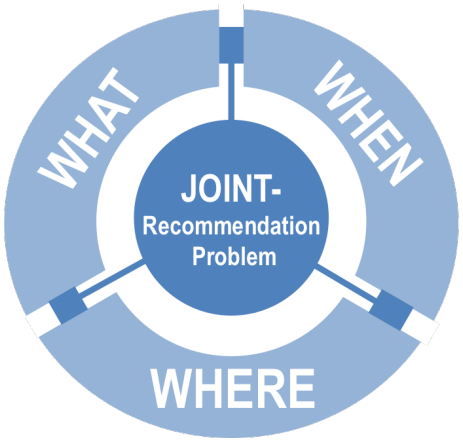

Figure 1.1: The **joint** recommendation problem diagram; Every recommendation problem is the joint of *What* to present ? *When* to present it ? and *Where* to place it?. The *What* further breaks down to the half moon of likes (bottom right)

1.0.2 Data

So often with problems grounded in reality, their complexity exceeds that of the initial general questions. In the field of video recommendation that additional layer of complexity is the availability of data. Currently, the majority of data collected is implicit. That means preferences can only be inferred indirectly from user behaviour. For example, if Jane watched the video “Sliding doors” we will find a “1” in our data, while otherwise we would see a “0” or a blank. While implicit data is much easier to collect, it is also more difficult to interpret. In other words, could we infer that Jane must have liked the video just because she watched it? In contrast stands explicit data, such as ratings via stars, thumbs and smileys. Explicit data may provide a more accurate picture of a user but is much harder to collect. Just

ask yourself, how often did you rate a video at Netflix or a product that you bought in Amazon? In addition, ratings may be biased; some users may only rate videos they like and others only those they dislike.

Currently most implicit recommendation systems are similar to the 'Sliding Doors' video; once you have picked a video, it is considered as a positive experience and your recommendation path is determined accordingly. Such are most content based techniques [2][61] that construct user profile according to any observed activity, collaborative filtering methods such as matrix factorization [67][10], Restricted Bolzman Machines [68] and even the newer hybrid methods as RNNs [6] and multi-networks [78].

The limits of such a "Sliding Door" approach become especially apparent if viewed through the lens of our **Joint** recommendation problem. Imagine the case, that Jane has just finished watching the comedy "Hangover" (after a struggle to finish it). Based on her implicit data track record, the system now recommends her to watch "Hangover 2". So, how will Jane react? Will she remain on the platform and invest the time necessary to search the right video for her, or will she simply stop watching? Addressing the limits of the 'Sliding Door' and so spare Jane from a negative experience in the first place is one of the main objectives of this paper.

1.0.3 Model

After examining our **Joint** recommendation problem and its complexities, we can specify the solution we are searching for:

• *What* interpretable: a solution that can argue for implicit feedback data sets why a certain video was watched, i.e. did the user like or dislike it?

• *When* temporal: our solution needs a) to predict videos in a sequential manner and b) use the sequential nature of human choice to provide predictions.

• *Where* ranked: our solution needs to optimize towards an evaluation metric, i.e. taking into account where a certain recommendation was presented to the user. Besides these three main features, we want our solution to be:

• *Comprehensive*: our solution needs to produce a prediction regardless whether an item or user is new (cold start) as well as it takes into account the contextual infor-

mation and the user's/item's explicit features.

• *Real-time*: our solution needs to be scalable and fast enough to produce recommendations in a real world platform such as Channel 4.

A solution that is interpretable, temporal, ranked, as well as comprehensive and practical in real-time, currently does not exist and requires a novel approach. We, therefore, propose our novel **Like** model. The **Like** model incorporates a vanilla RNN - temporal and practical (real-time) - into a novel architecture that a) introduces a Funnel Neural Network to learn about the video features a user dislikes, which is then combined with the RNN at the output layer b) introduces a new subtitle feature for comprehensive video representation, and c) uses an appropriate loss to rank videos. Accordingly, our hypotheses are: 1. Incorporating disliked videos to recommendation systems can lead to a better prediction accuracy. 2. Incorporating explicitly disliked video features can produce an interpertable prediction. i.e. one can see which videos are driven more by the dislike portion of the network.

1.0.4 Results

Our results suggest great potential for our **Like** model (which we had to stop prematurely): producing around 53 percent for top 15 accuracy over 7724 labels, which is competitive with state-of-the-art results[33]. Furthermore, to our knowledge, this is the first model allowing clear interpretability whether it is a liked video or a disliked video.

The results stand out as we compare our **Like** model with our implementation for other prominent solutions from the field of recommendation systems to check for robustness. Comparing it with content-based item-to-item, collaborative filtering item-to-item methods, and a package implementation for the renowned matrix factorization, we find an increase of 2-50 percent points to the best results achieved by these three models.

1.0.5 Paper structure

This paper comprises of four sections. The first section discusses the most prominent literature currently available in the recommendation systems field - including

the recent RNN model that also serves as our baseline and on whose foundations we build the new model. The second section presents our model design where we introduce our **Like** multi-network model that enables the interpretation of likes vs. dislikes. In the third section we test the robustness of our model and discuss the results of our findings. Finally, the last section will outline our conclusion and provide suggestions for future research.

Chapter 2

Background

2.1 Chapter structure

This chapter will first discuss our view of recommendation framework inspired by literature works. We then present a short surveying of the recommendation literature before delving in more depth to the most notable methods in the field; explaining the specific popular methods used, their strengths, pitfalls and takeaways for our solutions. We discuss collaborative filtering methods in more emphasis since they comprise of the biggest chunk in recommendation literature. Lastly, we discuss the idea of dislikes in literature and its scarcity in the recommendation arena, which further encouraged us to address the issue and is our main contribution to the field.

2.2 Recommendation solution framework

Based on comprehensive works by [2] [61], we can organize the solutions for recommendation systems, as presented in Figure 1. Solutions are most commonly divided to one of collaborative filtering (CF) [31], content based or hybrid methods [2]; Each one of them can then be applied using either a model based approach or a memory/heuristic based approach. The main difference is that a model based approach is using the database to learn a model which is then used for predictions[17]. Whereas memory based approaches use the full database in each turn.

We mention for completeness that in recent years there is also a separation between session based recommendation [33][76][58] and long term history based recommendation. Session based refers to short term behaviour of an individual that

is new to the system at every visit. While all solutions can be applied to session based, some of them will perform rather poorly due to their sparsity (e.g. matrix factorization techniques for completion). At this work we focus solely on long term history data sets.

The subsections below will explore each block, its advantages and deficiencies culminating in our proposed model.

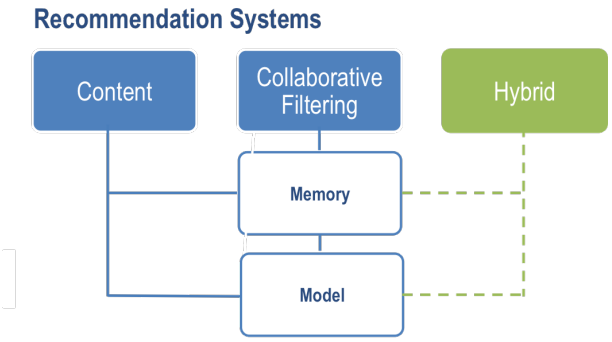

Figure 2.1: The recommendation solution framework. Solutions are either content based, collaborative filtering based or a hybrid combination. Each one of these can then be applied using a memory based method or a model based method

2.3 literature surveying

Public recommendations were part of people's life for many years. Any merchant faces the problem of what items to recommend from his merchandise. While this is an interesting problem, the scale of it erupted with the rise of the web. Any platform can offer millions of items without owning it in stock or pay for it. However, a screen can only show a handful of items. Hence the need to optimize the recommendations such that most benefit is extracted both for the user and the merchant.

The big boom in the field came about with the Netflix prize [14]. Where the wining paper and its extensions [9][47][10] presented a very comprehensive blend of models comprising of both simple models like K-NN [53] (applied either in Content based methods or Collaborative filtering), matrix factorization (Collaborative filtering) and more elaborated models such as boosting and Restricted Bolzman Machines[68] (Collaborative filtering or Hybrid) - with conclusions emphasizing

the importance of feature selection, matrix completion models and temporal behavior, which we will address here-under. Also, it was mentioned that neighborhood models still play an important role due to their interpertability and ease to solve problems such as cold start and offline computations. We further explain the K-NN methods, matrix factorization and RNN and also implement them to serve as benchmarks for comparison.

The recommendation community did not subside afterwards, with the boom affect to e-commerce businesses like Amazon and the appearance of music recommendation platforms like Spotify and Itunes.

At the same time deep learning has returned to fashion and presented solutions consisting of temporal models such as RNN and its variations like LSTM[36] (Collaborative filtering or Hybrid) which have shown great promise[78][79] and are serving as our baseline model and building block to our **Like** model. Another successful deep learning model is the Auto-encoder (Collaborative filtering) which was first mentioned at [66] and has been applied successfully [72] for filling missing values; It is the deep learning extension for matrix factorization with completion. This serves as an inspiration for our funnel Neural Network which is one part of our **Like** model. Further models which were not applied specifically to video recommendation for our knowledge but can be adapted with ease are bidirectional RNNs [71] which were used to predict missing values [15]. These are some of the most notable literature review in the field.

Many of the methods above are either probabilistic by their nature, or they gained variations that can explain them in a probabilistic way such as PMF [68], RNNs with softmax, which we use in some of our models and PMCF [80]. The closest to our novel architecture draws its core from AutoRec [72] and the vanilla RNN combined[36].

In the subsequent subsections we delve into one renowned model under each of the main branches presented in the solution framework (See Figure 2).

2.3.1 Utility matrix

The utility matrix is the main starting point for any recommendation problem, whether explicitly or implicitly; once we are interested in user behavior we are essentially interested in the utility matrix. It is a representation of the data set in a way that if a user has interacted fully with an item we mark the intersection of user-item in the utility matrix as 1 or by the rating the user has provided. Otherwise we leave it empty or mark it as 0. If we refer to Table 2.1, we see that While Jane has watched only the video 'Sliding doors', John has watched both 'Sliding doors' and 'The Dreamers'.

The data inside the utility matrix is obtained by either asking the user - which is often referred as an explicit data set, or by inferring the behavior of the user - which is often referred as an implicit data set, just as in our case.

Asking the user to rate items involves the patience of the user, her inability to rate unknown items[80] and the bias associated with the 'not missing at random problem'[7]. This means that whether a rating is missing depends on the video. In video ratings this can mean that a user might not rate disliked videos or videos that he is embarrassed sharing.

The main hurdle with the utility matrix or more generally with users' behaviour is usually the relative low interaction with the system which leaves the matrix very empty (sparse).

The fundamental role of the utility matrix as explained above appears in most methods and its concept will continue with us through all explanations.

2.3.2 Content based recommendations

The simple idea behind content based methods says: if a user has watched a video about crime[61], recommend content related to crime. In order to do so, content based methods construct an item profile; containing any relevant features characterizing the item e.g. words that characterizes item topic. The latter example will play a major role in our novel architectures - where we construct compressed representation of video subtitles. Once a profile representing an item is constructed, it requires only to measure similarity between item' properties, and then recommending those

Utility matrix			
User/Item	Sliding doors	Dreamers	Mommy
Jane	1	-	-
John	1	1	-

Table 2.1: Utility matrix - we place 1 if the video was watched by user and blank otherwise.

that are most similar. To be more concrete, we calculate a matrix A where a_{ij} is the entry to i-th row and j-th column representing similarity between item i and item j. Given user U rated items $\{i, j, k\}$, go to rows i, j and k, order by descending value and pick top-k to recommend. Popular similarity measures in information retrieval are cosine measure and Jaccard similarity [29]. Both similarities are intuitive, with cosine defining similarity by geometric meaning - closeness of angle between two vectors. While Jaccard is measuring similarity by proportion of overlapping items. In a more formal way:

1. $cosine_\theta = \frac{A \cdot B}{\|A\| \cdot \|B\|}$

2. $Jaccard_{A,B} = \frac{|A \cap B|}{|A| + |B| - |A \cap B|}$

Whereas this is the simplest way to construct item similarity, more robust ways exist which take into account user to item interaction[61]. The method above did not take into account the user entity. Interaction between users and items will provide more information on how a user reacts to an item. This idea of interaction brings back the use of the utility matrix.

Once calculated an item profile as above, a user profile is constructed to repre-

sent how much a user interacts with items with certain properties. So how would a user profile look like? we would illustrate it by a simple example: Suppose we are in a universe with only two genres $\mathfrak{S} = \{Drama, Comedy\}$. If Jane watched one out of two possible Drama videos her Drama score would be $\frac{1}{2}$ while she has not watched the only Comedy video available so her score would be $\frac{0}{1} = 0$. So now we can calculate the cosine similarity between the Drama 'Sliding doors' represented by the vector $[1,0]$ and Jane's profile being $[0.5,0]$.

The strength of this method lies in its simplicity and interpertability. Apart from the fact that similarity is feature selection dependent and thus needs to be hand engineered, some of the main arguments rising against the use of content based methods are the inability to distinguish items with same features (same vector representation will result in exact same similarity), the inability to offer to users content different from what they have already watched (similarity to Indonesian cookbook is hard to establish if I have never read cookbooks before), the issue of recommending something which is too similar to previous items (If I just bought sports shoes, I might not want to buy immediately another similar pair) and the problem of the new user (cold start)[2].

It is worth to dwell on the new user problem (cold start) [79] since it is something that most solutions still find hard to deal with. Although content based methods can easily deal with a new item since user attitude towards this item already exist through the user profile, a new user must rate/buy/use few items before she can get similarity to other items and reasonable recommendations. An alternative is to include user features which will be discussed under the hybrid models and which we incorporate in our network.

Although taking the individual users into account by creating a user profile, the interaction between users is ignored. However, our friends' recommendation is definitely sending some of us to watch a video. The idea of recommending a video to a user based on what his 'friends' have watched, is the one directing us to collaborative filtering methods.

Our method will try to take the ideas of item similarity such as items' features.

We also take the inspiration from methods such as TF.IDF (See model design) to represent a document (subtitles) in a compressed representation. The difficulty of hand picking features still stay with us.

2.3.3 Collaborative filtering

Collaborative filtering[2] is the umbrella name for all methods that use similarity among users' activity or items' ratings to offer recommendations. The main idea at its core is the construction of the utility matrix. The aim is to fill each partially missing matrix row (representing a user) either in full or with significant K values such that recommendations will be possible.

Following the solution part of the recommendation framework, we can see the collaborative filtering is branched to model based and memory based approaches. The simplest and what was quite popular way within memory based methods [17] is to find N users similar to user U, then we can average their ratings for item I. Similarity can be measured by the cosine or Jaccard measurements [69] mentioned above. This naive way suffers from many issues such as long computation times due to the need to recalculate similarities once a user has made a choice, limited offerings for users with unusual preferences (similar to no one) and the cold start problem - no recommendations for new users and poor ones for users with low activity.

The slow computation times renders this user-to-user method impractical in real world problems; when Jill joins the platform for the first time, no ratings or activity is available. Once the first feedback is provided, we would need to calculate Jill's values against other users, and in worst case against all users.

The impracticality of the latter method brought upon the item-to-item method [53]. The change is to look at items in the utility matrix instead of looking at users. i.e we can compute cosine similarity between pairs of columns in the utility matrix. All these calculations can be made offline so it can be very quick to recommend an item in the online setting; we do not need to know how close Jill is to John, since we have already calculated what items are closest to what Jill has selected. Here the problem of the cold start is reformulated by the item cold start issue. The

new item does not have any ratings associated with it, so can not be recommended. Another issue is when an item is liked by all; universally liked items are not useful in capturing similarity; If I know that both Jane and John like only 'Gone with the Wind' I can not differentiate between them. But If new distinct viewing activity is observed then we can start detaching between Jane and John.

The other group under the collaborative filtering umbrella is the model based approaches. One of the most renowned of these methods for many years is the dimensionality reduction and in particular Matrix Factorization in its various forms [67][46] [59][11][48]. The simplest form is derived from the Principal component analysis (PCA) method [38].

PCA finds a lower dimensional representation of the data, that can then be reconstructed back to its original form. The idea can be thought in the same way every number can be deconstructed to its prime numbers, so does matrices [40]. We want to find the dimensions that are communicating the most amount of variation in the data. Think about a data set that contains both the weight in grams and in Kg. One of the two is redundant since they communicate exactly the same information and hence one of which can be discarded. With PCA trying to find these so called redundancies.

Formally, we are trying to find a shared space R^f such that the original matrix $X \in R^{MXN}$ can be deconstructed to two matrices $\{U_i, V_i\} \in R^f$ giving the equation:

$$X = UV^T \tag{2.1}$$

with the goal to minimize the reconstruction error:

$$\sum_{i,j}(Um_{i,j} - (UV)_{i,j})^2 \tag{2.2}$$

It turns out that the solution can be obtained through the SVD of X, where the first K eigen-vectors represent the lower dimension basis.

In the video industry, an example can be made by thinking about the shared space of videos and items as finding the genres shared by the different videos. The resulting matrices would be one with the number of videos as rows and the number of genres as columns. The second matrix would be the number of genres as rows and the number of users as columns. Thus, the multiplication of the two would represent how similar a user preference of certain genres is to the genres representing a video.

The deconstruction is applied to fully filled matrices. But what do we do with a sparse matrix such as the utility matrix? i.e. we want to know based on previous actions of a user what will be the other videos from the optional video space this user will watch as well.

As explained in [7] the reconstruction is very similar to the PCA idea. Assuming we have a Utility matrix U_m with dimensions kXn for k users and n items, we can then choose two matrices U,V representing a lower dimensional user matrix and video matrix respectively. So in order to get the reconstruction $U_m = UV^T$ we need to give U dimensions kXm and V mXn, with m being the lower dimension basis which is often called in literature the joint space of users and items. Using the Mean squared reconstruction error:

$$\sum_{i,j} \mathbb{1}_{i,j}(Um_{i,j} - (UV)_{i,j})^2 \tag{2.3}$$

we would only measure the loss in positions that values do exist - this is enforced through the indicator function. Finding U and V is done by first consider random values an then either by applying alternating least squares or stochastic gradient descent methods.

Evolution of the model is concerned with introducing the bias of the user/video with respect to ratings which are independent of the joint factor state of both [48]. Such is the case if John is more harsh in rating than the average user represented as B_u or if 'Sliding doors' is more acclaimed than the average video represented as B_i. Further, we can also add temporal dynamics to the biases and to user preferences P_u to account for changing preferences over time or to a user becoming more critical than before. Formally we can write:

$$\hat{R_{ui}} = \mathscr{M} + B_u(t) + B_i(t) + V_i U_i(t) \tag{2.4}$$

Where $\mathscr{M}$ is mean rating over all users and $U_i(t)$ accounts for the changing user preferences over time.

However the story is a bit different for implicit data sets since even if we can reconstruct the perfect matching of the utility matrix it might still be useless in practice - did we predict a 1 representing what the user likes or is it a 1 represents user's boredom. [39] Suggests to solve the equation:

$$min(U,V)\sum_{i,j} C_{i,j}(Um_{i,j} - (UV)_{i,j})^2 + \lambda(\sum_i ||U_i||^2 + \sum_j ||V_j||^2) \tag{2.5}$$

Where $C_{i,j} = 1 + \alpha R_{ij}$, with R_{ij} being the confidence (How many percentage we have watched from the video) of user i to video j. E.g. With 100 percent watched $R_{ij} = 1$. α is a hand picked factor. Differently from explicit matrix factorization, we go over all pairs (i, j) and not only over observations. In order to optimize in a large scale environment we can use the alternating least squares method.

The popularity of collaborative filtering matrix factorization is attributed to its great results, its independence from domain knowledge, fewer data points are required to preform the model and the discovery of latent factors modeled by user behavior.

On the flip side, matrix factorization techniques are mostly time independent, while humans' preferences are time dependent [45]. Even the notable Netflix prize winner TimeSVD++, which does take time into account needs to account for time in a manual manner - time window is hand picked. Another problem that arises in the context of matrix completion in all its forms is how to choose the number of features f that will constitute the lower dimension representation. Collaborative filtering

does not include explicit features of users and items which might add predictive power. Importantly we notice that a solution to the matrix reconstruction problem is an interpolation technique - meaning we are trying to construct the space of views for Jane rather than predicting what will be her next view.

In the recent years with the rise of Neural Networks and its family, many other methods were presented for matrix completion among them Restricted Bolzman Machines (RBM) [68][67, 34], BRNN [71] for filling missing values and auto-encoders [72]. With the latter being the closest to the PCA completion method and has shown the most promising results regarding video recommendation using collaborative filtering[72]. However, currently was only applied on explicit videos' rating data sets. A special care needs to be taken for the input of these Auto-encoders models. If the missing values are set to zero, we are biasing the model towards zero values which we do not want to do, especially not with implicit data set where zeros have a meaningful interpretation as non active. Few initial values could be considered, such as the mean value of items, or the output of a matrix completion procedure. These values would still not be considered in the loss, and will not be back propagated during training, but would affect the interaction of the other weights and hence should be reasonable.

From the collaborative filtering method we would like to take user's activity and its relation to other users (when deriving the squared reconstruction loss it can be easily shown that each video in matrix V is affected by the different users; and so does users in U). These attributes are intuitive when thinking how do we find out about new recommendations. In addition to these features, we would like to add explicit features as well and take into account the time dimension. The former is complementary to latent factors found in the data while the latter is accounting for the nature of people's choice and predictions. All of the above leads us to the search for a hybrid method.

2.3.4 Hybrid methods

Hybrid methods are exactly the ones that combine characteristics both from content based methods and collaborative filtering methods. The hybrid combination helps

us circumvent issues[5][8][22][70] presented either in content based or in collaborative filtering approaches or in both mentioned above. In our solution we choose to employ a hybrid method, since this, as mentioned in the introduction, answers the requirements of the requested solution - the joint of *What*? *Where*? *When*? under the limitation of implicit data. Most importantly, it enables us to model the negative feedback (dislikes) and thus argue for its affect on implicit data set predictions.

Until this point we have mostly considered models and memory based techniques that are neither time dependent nor feature rich. For example we have not considered Jhon's age to affect his viewing behavior. But it is quite intuitive that a six year old John is less likely to watch the TV show 'Transparent' than his older counterpart. If our technique takes this feature and others into account, then it will presumably be much easier to group the different possible videos to recommend next to users.

Surveying the hybrid methods present in the literature, few methods seem more promising than others. Such are again ones from the dimension reduction sphere such as inductive matrix completion. The latter extends the idea that we can decompose the original matrix to two matrices to that of decomposing them to multiple matrices which contain also features like age, gender and postcode for the user, and features like genres and actors for the video [41][74]. Another method in the same sphere but with more general properties is Tensor decomposition. This method is promising since unlike matrix decomposition methods, the initial matrix (Tensor) can be in high dimensional space and thus much richer in features and even include temporal dynamics [3][25]. However, to date, tensor decomposition is only applicable to a very specific space of problems and does not yet have much research background.

Although Tensor decomposition can be considered for modeling temporal dynamics, it is more natural to think of it as a sequential model. RNNs are a natural choice since they account for varying sequence lengths.

RNN is a very popular method in recent years with promising results in generative modelling in general and in recommendations in particular [79][33][76].

Although [33] was applied in a session based framework, its architecture is readily applicable to long term history based data sets like ours.

2.3.4.1 Baseline model vanilla RNN

The RNN model is a natural choice for our baseline model since it is comprehensive in the way we can include both user and video features. It is practical as a real time recommendation system since the weights are fixed for one forward prediction. Additionally, it can answer the *Where* question by choosing to optimize towards one of the BPR/cross-entropy losses which we use or any other differentiable ranking loss. The temporal property of RNNs helps us to answer the *When* question - not only we predict in a temporal manner, but also take into account the temporal behavior of users and videos as inherent to prediction. Our prediction is the *What*, but the inability to interpret what we like and what we dislike has driven us to search for the missing portion (see **Like** model below).

RNN is a generative model that allows handling data sequentially; taking at every time step the new input and old hidden state. So it is especially useful for natural sequences appearing in audio, video and text [21] [75]. Such problems can be seen for example in Natural Language Processing (NLP) where a word in English has different length than a word in Chinese. In our case, we input at each time a new video that John has watched, memorizing through the hidden states the videos that he watched beforehand.

The standard Neural Networks lack the ability to handle such sequential data. Moreover, RNNs allow us flexibility of input and output size. e.g. when we want to input different sizes of words and get varying length words in different languages as an output. In our case, each user has watched different amount of videos. So the input and output lengths would be user dependent. But, even without obvious sequences, we can process the fixed size input or output sequentially.

The starting networks of RNN were built upon a simple equation, however its formulation have the problem of vanishing and exploding gradients [12][35][37]; hence we use the popular LSTM [36] model which tries to avoid the gradient stability and thus enables to learn long term dependencies.

The idea of LSTM is maybe more intuitive than the original RNN since we construct several simple control gates, namely F to forget, I to input and O for output, to decide if information goes through or not. Note that other variations like Gated Recurrent Units (GRU) [21] can as easily be applied to our baseline model.

The LSTM model serves as our baseline and the building block for our **Like** model. The architecture used here is not novel, but the synthesis and specifically the choice of features from the user domain and item domain as explained in subsection features below is. For completeness we provide its equations (see figure 2.2 for an LSTM cell visualization) but refer the reader to [28] for further details and to the Appendix to gradient calculations.

$$\begin{aligned}
\mathbf{I}_t &= \sigma(\mathbf{W}^i * [h_{t-1}, x_t] + \mathbf{b}^i) \\
\mathbf{f}_t &= \sigma(\mathbf{W}^f * [h_{t-1}, x_t] + \mathbf{b}^f) \\
\mathbf{O}_t &= \sigma(\mathbf{W}^o * [h_{t-1}, x_t] + \mathbf{b}^o) \\
\tilde{\mathcal{C}}_t &= \tanh(\mathbf{W}^c * [h_{t-1}, x_t] + \mathbf{b}^c) \\
\mathbf{C}_t &= \mathbf{f}_t * C_{t-1} + I_t * \tilde{C}_t \\
\mathbf{h}_t &= O_t * \tanh(\mathbf{C}_t)
\end{aligned} \tag{2.6}$$

The LSTM is comprised from flow control gates:
C_t is the cell state which is comprised from: the things we want to forget with the F gate about earlier states represented by C_{t-1} and the new things we want to memorize from the new candidates $\tilde{C}_t$ gated by gate I. The resulting cell C_t is then filtered through the output gate O.

Upon the LSTM we can apply for example more LSTM layers and/or an output using a fully connected linear layer. In our baseline model we apply the latter which gives a score for each of the video labels. Formally,

$$Output = K * LSTM(x_l) + b \tag{2.7}$$

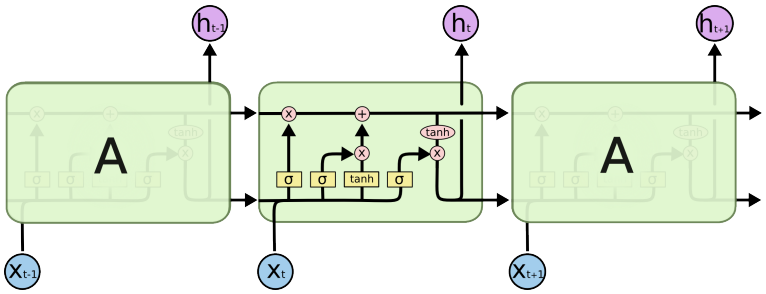

Figure 2.2: LSTM cell visualization. Image was taken from [56]

2.3.5 Negative feedback - dislikes

Using dislikes in recommendation literature is scarce. The idea of taking into account negative feedback, which we term dislikes (video watched less than 25 percent of its total duration time) has not, to our knowledge, been incorporated explicitly in the implicit video recommendation field. i.e. Finding characterization of disliked features and weight them during prediction to get overall score.

The idea appeared as well in different forms in other fields. However it is argued by [26] that it is hard to get negative feedback in an implicit way, since users mainly interact with content they are interested with, and hence it is more rare in literature. [54] Gives as an example the way users listen to music; avoid genres they dislike. A wide study made by [42] showed that only as few as 4 out of 27 papers could be used with negative feedback. Even the use that has been done was mostly not with regard to implicit data set.

The closest to our video recommendation idea of decomposing the *What* to likes and dislikes is [39]: which offers a matrix factorization technique (see collaborative filtering above). Their idea is to incorporate positive and negative feedback by the confidence it has in that video. So that, if a video was watched three times it can receive three times the weight to it, whereas it can be weighted by 0.25 if it was watched 25 percent of its total duration. In this way it accounts for user's dislikes.

The latter method is trying to learn the hidden space of videos with lower and higher confidence combined and predict 1 where it is observed. In contrast, we would like to learn an explicit feature representation indicating the disliked video characteristics of a user separated from those liked by her. The search for a good

explanation to an implicit recommendation is very important [31] and although the latter method suggests a linear model to explain how the score of one prediction is comprised of other actions, it is still unable to say if it was liked or disliked.

Such is the case when a similar video contributed most of the prediction to another, both have been watched but both disliked. i.e. it can be inferred that one video is the driver of another but whether they are liked or not is vague - can only be implicitly inferred when driving videos are with confidence lower than 1. Also, it requires from us to have a data set which includes many repetitions of viewings per user to establish a strong confidence; Only if Jill has watched the same video more than once, can we get a confidence bigger than 1. [30] is an extension to the latter model which adds a popularity weighting to the confidence attributed to negative feedback to better differentiate it from unknown data. We try to simplify what was offered by [39] by simply looking which portion of our **Like** multi-network contributed more to the general score of the item. Further, we will consider explicit features and treat the problem in a sequential manner.

[33] introduces an RNN model which both treat the the problem in sequential manner and is applied to implicit data set - which we use as our baseline model. It does note the importance of dislikes, but only incorporates it through the loss, where it assumes that if a video is popular and was not watched, then a dislike can be established. While this assumption might produce good results, it is reasonable to doubt its generalization.

More recently, [24] shows how explicit data set can be better evaluated using a tensor decomposition, treating ratings, users, and items as triplets such that we have a function from $userXratingsXitem \longrightarrow score$. By adding the rating to the usual matrix of userXitem, we can account for new users which provided low score for the first item appropriately - recommend different things from that initial rated video. The method of tensor decomposition and its later advancements [3] can be looked as a higher dimensional, general form of matrix factorization.

This method, like ours suggests the importance of considering negative feedback towards prediction, but it is motivated by the cold start problem, and as a by

product improves accuracy using negative feedback. This work concerns an explicit data set and also does not offer a solution to the prediction problem but more to interpolation problem similar to matrix decomposition techniques.

In [52] there is an attempt to combine negative feedback in the job search field. With a small controlled experiment which is checked manually, the results are encouraging - negative preferences can increase recommendation quality. Just like [52] we introduce our own definition for what is a negative feedback and do not infer it from user behavior like in [60]. However, the latter which is done in the e-commerce field using content based methods is unable to establish an individual's taste and concedes to an aggregate over all users to identify negative preferences. Some methods like [19] explain the importance of including negative preferences, but are directed to solve the problem of group concessions. They use a radio that plays only songs which are not disliked by all group participants, rather than to figure out individual's preferences.

Chapter 3

Model design

3.1 Chapter structure

In this chapter we will present first our novel **Like** model: a multi-network with one netowrk for views represented by an RNN and one network for disliked video features represented by a funnel Neural Network, both meeting at the output layer. We then examine its interesting and/or unique components - mainly the novel subtitle feature input (which we hypothesize is a driver for our results) and the loss. We also discuss shortly the other parts of the network such as optimization technique for completeness.

3.2 The Like model

The Vanilla RNN model explained above helps us answer the general questions of *What?*, *Where?* and *When?* rising in any video recommendation system. But, since we are dealing with an implicit feedback data set, our answers to these questions still do not help us practically to recommend to users; we do not know if the video was liked or not. This issue affects also our answer to the *Where* question. Since even if we have predicted correctly everything the user will watch next and ranked it accordingly, we still might present a disliked video in this list.

For that reason we construct a multi-network, that from one hand predicts views and from the other hand accounts for how well these views are liked. This is achieved by our architecture of two networks: an RNN and a funnel Neural Network meeting at the output prediction layer. - see figure 3.1 for the **Like** model; The

Neural Network is constructed to learn a pseudo lower dimensional feature representation for disliked videos, while the RNN's aim is to predict the next video given previous fully watched videos.

Our funnel Neural Network part is comprised of two layers shrinking in size. Formally, we have:

$$h_1 = RELU(Wx_d + b_{hidden}) \tag{3.1}$$

$$h_2 = RELU(Uh_1 + b_{hidden2}) \tag{3.2}$$

Where $RELU(x) = max(0,x)$, W, U and b are the Neural Network parameters and x_d is the input of disliked video features for a particular user U (disliked = watched less than 25 percent of total video duration).

The RNN part is constructed the same way as in Vanilla RNN explained in the background section. The output layer of the **Like** model is then comprised of:

$$dislike = Vh_2 + bias_{dislike} \tag{3.3}$$

$$like = K * LSTM(x_l) + bias_{like} \tag{3.4}$$

$$Output = like + dislike \tag{3.5}$$

With x_l is the input of video features, user features and contextual features for liked videos by user U (liked = watched fully).

The extra layer of either softmax for the cross entropy loss or the sigmoid for the BPR-max loss is added during loss calculations for stable implementation. We further add a L_2 regularization term to the loss to avoid over-fitting.

• We provide gradient calculations for the **Like** model in the Appendix.

Since we have modeled the dislike portion in a non temporal manner, its output is 1 layer deep and the same layer is added to the "like" portion upon all its

layers. The like model could be thought of as a cube (Figure 3.2), each layer of it consisting of predictions for 1 time stamp for all users in the batch. Implementation wise, the number of layers is determined by the user with the maximum amount of views among batch users. Users with fewer views are then padded with 0's and the corresponding zeroed layers are not contributing to the loss.

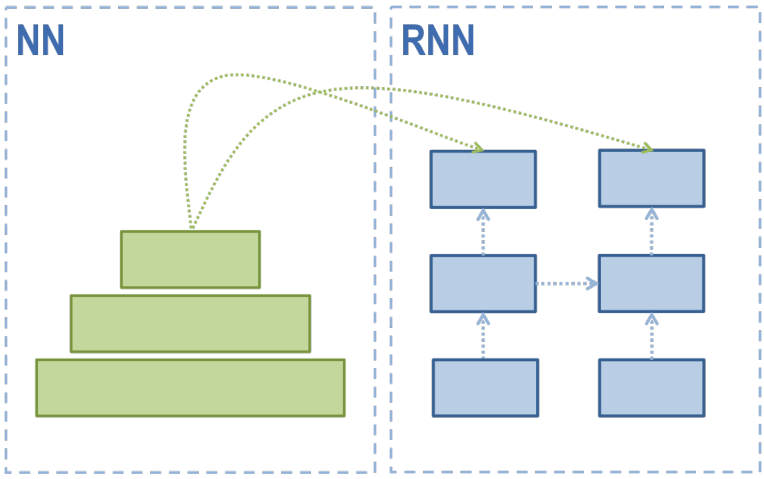

Figure 3.1: The **Like** model: the left network is a funnel Neural Network learning to represent hidden features of disliked videos. The right LSTM network meets the funnel network at the output layer to produce a combined prediction for the next video the user will watch.

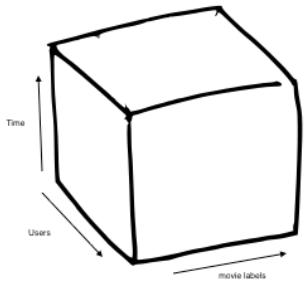

Figure 3.2: Output of the like portion of the model: we predict for each time-stamp (height) and user (length) the video labels scores (width)

The most similar to our **Like** model architecture is [78] which proposes a Neural Network composed of RNN and Neural Network meeting at the output. However, it is trying to solve the heterogeneous recommendation problem of implicit and explicit data sets combined and is unable to say if a video was liked or disliked.

Though different, our inspiration comes from the paper by [79] where two

RNN networks meet at each output unit in the sequence. In the **Like** model we replace one of the RNN networks with a simple Neural Network; where the inspiration comes from [72] and generally from the ideas introduced by Auto-encoders [66] and lower dimensional feature representation such as the one in Word2Vec and its variants.

Relating to our problem, with enough examples we would like to learn a low dimensional representation of features characterizing user's dislikes. In the same way as Auto-encoders can deconstruct to lower feature space that holds enough information to characterize the higher dimensional input space, we show by experiment that our network is able to learn a predicatively meaningful dislike feature space.

We choose to employ a multi-learning, meaning we construct the two networks and learn them together - in this specific case with a combined loss (This can be viewed loosely as some sort of multi-task network). Since the two networks are learned together, we can not clearly say during training that one network is solely representing dislikes and the other solely representing likes, since the gradient with respect to the weights of the dislike network will be affected from the loss generated also from the "like" LSTM network. During testing though, after all weights are fixed, we can still see where the bigger contribution comes from.

Aside from the concept of likes versus dislikes and the model architecture, the input and loss to the **Like** model is tailored.

3.3 Model components

The input, loss function and optimization procedure are as important to the model as the architecture of the network and/or the chosen algorithm.

Our **Like** model consists of our novel feature such as the Doc2Vec video subtitle representation with more common features like video genre and user's age. Moreover, we construct two loss functions as the cross entropy and BPR-max to provide appropriate ranking.

3.3.1 Input features

As mentioned in the content based part above, an important part of a recommender system is to find items' similarity - videos in our case. Some of the common features to use are genres[2], which we use as well. Genres are assumed to be very helpful for recommendation, since if John is watching only romantic comedies, then it will be quite simple to construct the table of possible videos he is going to watch next. Since the number of genres is modest, we can use one hot encoding to represent them. i.e. construct a list of genres, and place a 1 if genre is active in this specific video and 0 otherwise. example: With genres $\mathscr{G} = \{Comedy, Horror, Thriller\}$ if John watched 'When Harry met Sally' the corresponding vector would be $\{1, 0, 0\}$.

Another feature, which is less common for video recommendation is to use the script/subtitles of the video as a characteristic of it. To our knowledge, this is a novel feature for video recommendation. The use case of it being: suppose a stand-up video is tagged as a comedy and stand-up genre. However, if the video's content is about drug abuse, and your next view is of 'Trainspotting' (video about drug abuse), the subtitles would be more telling than the genre tags. Although this idea is not very common in the video recommendation industry, the idea is prevalent in written documents classification to topics[2]. i.e. as one of the features, use the words that charcterize the document topics.

We first discuss our novel feature and its Doc2Vec implementation. In the appendix, we explain shortly other methods (TF.IDF and LDA) that are common in written document classification and can be used instead Doc2Vec.

In order to find distributed representation of the subtitles in the video, we choose the Word2Vec [13] method from the Natural language processing field or more appropriately its Doc2Vec extension [50]. We would like our lower distributed representation to capture ideas from the video; such can be for example the location of scenes, the underlying concepts or just the repetition of certain words. The Doc2Vec approach was chosen since it is a natural part of the Neural Network methods that are used in our architecture. Moreover, according to [55] it has promising results and is better scaled than LDA.

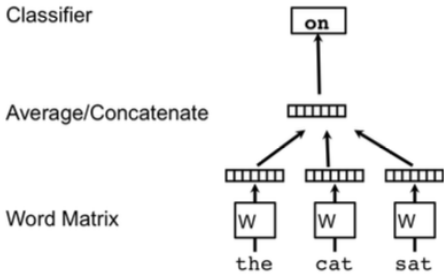

Figure 3.3: The Word2Vec model illustrated through a simple example; given previous words ”the”, ”cat” and ”sat” predict the next word ”on” - firstly each word is transferred to a continuous vector representation, then all words are concatenated and go through a Neural Network to classify ”on”. The image was taken from [55]

What is Word2Vec? given a document (script) and words $\{w_1, w_2, ..., w_n\}$ we would like to maximize the average log probability:
$\frac{1}{K}\sum_{l=k} logP(w_l|w_{l-k}, ..., w_{l+k})$.

With K being the window around each word - see figure 3.3. In simple words, we would like to find the most likely word given the previous K words. In figure 3.3, given the words ”the”, ”cat” and ”sat”, w would like to predict the word ”on”. The Word2Vec idea is derived from the assumption that words with similar meaning are mentioned frequently in similar contexts and thus sentences containing these words will have similar semantics.

A clear example of the idea is mentioned by [13]:
We would like the sentence *The cat is walking in the bedroom* which is seen in training to give high probability (generalize) to the sentence:
A dog was running in a room
Since the meaning of cat/dog, walking/running and room/bedroom can interchange and is expected to appear in similar semantic sentences.

In the model, word vectors are randomly initialized and a matrix $\mathscr{C}$ is then constructed; The randomized vector for word i is represented by $\mathscr{C}_i$ and positioned in row i. This model then applies a Neural Network over C with the goal to maximize the probability of seeing the next word in the window given previous words in that window - usually with a fixed window size (Referring once more to figure

3.3). Meaning, we have an X vector which is the concatenation of the words in our fixed window. This is formally then calculated by:

$$y = bias + UX \tag{3.6}$$

$$P(w_l|w_{l-k},...,w_{l+k}) = softmax(y) \tag{3.7}$$

Where C are free feature vector parameters and U, bias are the Neural Network parameters. Since for example the word cat and the word dog can interchange and expected to appear in overlapping contexts, their feature vectors will be close in Euclidean space. The feature vectors are learned simultaneously to the probability of a given word conditioned on previous words. all learning can be done with stochastic gradient descent via backpropogation.

The idea is an evolution of the n-gram models, which have the same goal but executed differently; by creating a conditional table and concatenating overlapping words in sentences. This accounts for the fact that similar words appear closely with the same other meaning words frequently. n-grams suffer from both scalability to account for many overlapping words, settling commonly on a 3-gram model. A major hurdle that Word2Vec is tackling is to account for the semantic meaning of similar words such as 'massive' and 'huge', which would not be taken into account in an n-gram model.

Word2Vec results are surprisingly good and lead to word representations that capture semantics. In known examples[13], the vector arithmetic of 'France' - 'Paris' would be very close to the vector of 'Italy' - 'Rome'. In the same manner the Word2Vec model captures similar words we would like to capture similarity of scripts. i.e. if one video is discussing nature, we would like it to be captured in our compressed representation.

Doc2Vec extends the idea of Word2Vec to multiple words with any length

needed. with addition of another Matrix $\mathscr{D}$ representing paragraphs or documents. Unlike the word matrix $\mathscr{C}$, $\mathscr{D}$ is a unique paragraph identification - see Figure 3.4. The same procedure of concatenating words to form input X in Word2Vec is done on word level vectors and paragraph level vectors combined. The goal of the model can be viewed as predicting the next word, but this time keeping paragraph context. This allows to keep from partially labeled data. At test time Doc2Vec still needs to infer the paragraph identification, setting the words as fixed.

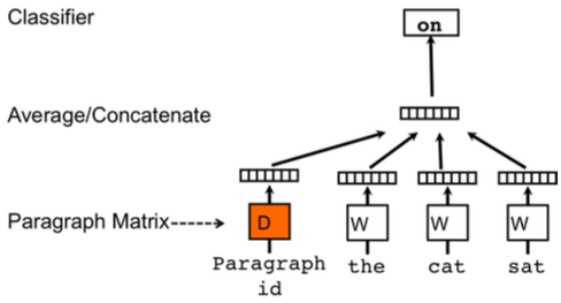

Figure 3.4: The Doc2Vec model illustrated through a simple example; given previous words "the", "cat" and "sat" predict the next word "on" given it is connected to paragraph D. Firstly each word is transferred to a continuous vector representation, then all words are concatenated together with a continuous paragraph vector. Then all go through a Neural Network to classify "on". The image was taken from [55]

Using Doc2Vec [50] provides powerful results on IMDB sentiment data and information retrieval query tasks. These results provide evidence of the meaningful representation of the paragraph and/or document. This can be thought of as a summary of the data, though not in a way we can interpret in human language just yet. But same as with words, distances and positions in space are still taking into account document "semantics".

As genres are general and broad, giving us a circle space around the user, the hope is that the Doc2vec gives us a specific direction in that space.

Several other techniques are available with the widely used TF.IDF from information retrieval and Latent Dirchlet Allocation (LDA) [16] from unsupervised learning using variational inference. What is common to the two methods with Doc2Vec is that all can be used to find similarity in space of words and documents to one another. For completion we provide in the appendix a short explanation

regarding TF.IDF and LDA .

In our models we want to synthesize information of all entities: the video, the user and context. The video features are comprised of concatenating genres and subtitle Doc2Vec features. For the user we add information such as the user's age and gender. A feature which we attribute to each user is the average of the Doc2Vec of the first three videos watched, to account for the user starting taste. We then always start training and testing from video number three onwards. Finally we add also contextual info [1]: which device the user has used at what time of the day and at which day of the week; Perhaps Jill likes to watch different videos over the weekend through the computer to the videos she watches on her mobile during weekdays tube travels.

While selecting/processing features to serve as input, a note should be made; when choosing hand engineered features, we are limiting our solution to data sets which can extract the same features.

3.3.2 Loss

The loss function helps us to answer the *Where* question. By choosing the appropriate loss we can optimize towards our goal: ranking video predictions such that they are ordered; prevent disliked videos to appear on the user's screen while showing more desired videos in a higher position. This is aside from the overall need in all machine learning applications to choose an appropriate loss so that weights will learn to find an optimum and provide meaningful results.

Assuming we know *What* videos people would like to watch and *When* they would like to watch them, we still need to organize at each time-stamp *Where* these recommendations would appear. Imagine Jane opens her favorite video platform - the number of videos appearing on the screen would be more than just the next video predicted she would watch. Thus, it is important to direct the model not to place unwanted videos along with the correct prediction. This direction is done through the loss.

The first loss we are using is the common cross entropy loss - which is the default for many machine learning classification problems . However, it is important

to understand how this loss relates to our ranking problem and what are its problems that drive us to look for an alternative - such is the Bayesian Personalized Ranking [63] (BPR) and its BPR-max [33] variant loss which we apply, and the Mean Reciprocal Rank and [20] (ERR) loss which we use as a metric and suggest future implementation of it as a loss to improve our work based on (MRR) [73].

3.3.2.1 Cross-entropy

Here, we explain first the fit of cross entropy to classification problems in general going through some concepts such as the entropy.

If we look at the video field, we can consider we have a budget to spend for making three types of videos: Comedy, Horror or a Thriller. The basic idea of entropy says that if the comedy genre comprises of 95 percent of the market, the optimal thing to do is to invest 95 percent of our budget in producing these type of videos.

Imagine we can only communicate the genres to our director by zeros and ones, and we pay for each number we send - then, our goal would be to send a message with the shortest length to pay as little as possible. The cost for each such message is $\frac{1}{2^{message-length}}$. Since we have only two options $0,1$ for each letter, if we send a message starting with 1 we can not use 1 any more as a starting letter and half of our options are gone. Otherwise, we would not know the meaning of the message 110; is it 11 and then 0 or is it 1 and 10.

Writing a lengthy message, we will have to multiply the cost times the probability that this message will be read - we want to pay as little as possible while communicating the message. The message length is the reciprocal of the cost: $\log_2 \frac{1}{cost}$. If the optimal spending is p(x), then the average message length using the optimal spending is $H(p) = \sum_x p(x) \log_2 \frac{1}{p(x)}$ which is the definition beyond the entropy term.

Why is p(x) the optimal spending? Consider we spend 55 percent of our budget on a message that 5 percent read and 45 percent on a message that is widely read. If our investor sees this behavior, he will ask why are we not spending more on a widely read message? If we then decide to spend all on the widely read mes-

sage then we will lose 5 percent of the budget although it was not required. The mathematical proof follows the same lines of thought.

Since p(x) is the probability that the message will be read, if we plug into the entropy term a probability, the more concentrated the probability the shorter my messages can be to convey the same idea.

Continuing with cross entropy, my close colleagues might know that I won the lottery when I say 'we won'. If I have colleagues from many sectors then I would need to explain who won? where? and when?. Now if I do not know the distribution of colleagues and still try to write 'we won', this can potentially be very confusing to my colleagues.

This confusion is the idea of cross entropy. How far are we from the entropy, which is the optimal spending for our real distribution. Formally, $H_p(q) = \sum_x q(x) \log(\frac{1}{p(x)})$ is the cross entropy of P with respect to Q.

In classification problems, we exchange P with the predicted probability and Q with the real label. i.e. Q will be a probability concentrated at one X_i value with probability 1. So the cross entropy measures how confused we are about predicting the true video label - how much we have spent on the true video in compare to its distribution.

How does it fit our ranking problem? As explained in [18] and [32] Cross entropy with a softmax layer (which is the type of loss we use) is essentially a list-wise loss. The softmax enforces a constellation in which if higher ranked videos are placed higher the assigned probability of top k would be higher. list-wise in general tries to enforce the ordering and ranking of the video that comes second after the desired one, the video that comes third and so forth.

However, problems arise when dealing with large scale tasks; Since the cross entropy is usually applied with a softmax layer which computes the probability for all possible labels. When the number of labels increases we can encounter instability rising from underflow, overflow and heavy computation times.

Solutions for that are based upon looking at only part of the labels, consider them as undesired examples and compare to the desired one (next video). Such

methods are the Bayesian personalized ranking [63] loss and the mean reciprocal rank [20, 73] approaches. The former tries to maximize the desired one, while the latter tries to enforce the correct ordering of the first K videos. We will explain here under the intuitive use of BPR-max since this is what we use and specify advantages and pitfalls.

3.3.2.2 BPR and BPR-max

BPR-max [33]Top-k Gains for Session-based based on Bayesian Personalized Ranking (BPR) [63] is a loss that tries to circumvent the problem that is introduced when there are too many labels to classify. It is considered a pairwise cost as it operates on pairs of labels at a time.

As mentioned above, using a softmax as an output layer to calculate the probability of all labels such that they sum to one can be a computational hurdle. Think about a case with one million labels. As we start our prediction, we assign probabilities in more or less random manner such that it can be thought as distributing the different probabilities uniformly. In that case each label gets a probability of $\frac{1}{1,000,000}$ which can lead to underflow. The further we continue our training, some probabilities will increase, while others decrease to preserve the sum to one probability rule. i.e. a situation of underflow is easily reached.

A solution that comes into mind is not to use the softmax to calculate the full output. Instead the BPR suggests to use the sigmoid function and apply it over two labels at a time - forcing one to be greater than the other. Furthermore, as it is too expensive to use all labels, the BPR samples only a fraction of labels. We note that we compromise here stability and scalability suggested by BPR with exact ordering suggested by cross entropy for example.

Formally, we define the loss function to be:

$$BPR = -Mean(\sum_{j} \log \sigma(y_i - y_j)) \tag{3.8}$$

where $j \in negativesamples$: negative samples are considered as videos watched

later than current prediction or not watched at all. i is the ground truth position of the label and

$$\sigma(y_i - y_j) = \frac{1}{1 + \exp^{-(y_i - y_j)}} \tag{3.9}$$

In order to understand the loss term, we consider few values of the sigmoid function:
when $(y_i - y_j) \longrightarrow \infty$:
the closer the sigmoid expression is to 1. Applyig log on 1 yields zero. If on the other hand, the term:
$(y_i - y_j) \longrightarrow -\infty$
then the sigmoid term is going to 0 which means the log argument will yield a big negative number. Our goal is to minimize the loss and hence what we ask is the constellation in which the predicted value at the ground truth label position is bigger than all the negative sampled predictions y_j.

From our experiments using the BPR with random sampling yields poor results. And thus, the question to rise is how to choose the samples smartly? should we pick the ones coming subsequently after the the current time stamp?
An interesting approach is to choose according to how well they have preformed at prediction.

The BPR-max method does just that: adding a weighting factor. We want to give to each loss term a weight corresponding to its importance. This can be viewed as some sort of a discount factor; discount the importance of a negative example if it does not contribute much to the gradient update.

3.3.3 Optimization

Optimization both of BPR-max and Softmax can be done simply through backpropogation as explained for example by [51]. This can be done by using the common gradient descent approach - see Appendix for gradient calculations. We chose specifically to use Adam optimizer[44] for this case. The optimization is done in batches since we are dealing with very large data.

ADAM optimizer shows significant improvement in pace of convergence com-

pared to stochastic gradient descent and other optimization algorithms; an intuitive reason is that unlike many first order algorithms it takes into account the trend shown in past time-stamps. By taking the exponential moving average of the gradient (representing the mean) and the exponential moving average of the squared gradient (representing variance). The update is preformed by reducing from the parameters some rate of the approximated mean-to-variance ratio of the gradient. In simple words, the certainty I have in the direction of the true gradient.

Chapter 4

Experiment

In this section we will examine the results achieved by our novel **Like** model and compare it with the Vanilla RNN model and with the older industry standards of content based and collaborative filtering. With a content based item-to-item model from the former, and Amazon's item-item from the latter. We then observe the loss, different metrics, the computational complexity and interpertability of the features and architecture; did we predict a like or a dislike?

4.1 infrastructure

For our experiments we use British Channel 4 data set which consists of around two million users activity over approximately 10 years duration in their online platform. As this is heavy in terms of size and memory consumption, we need to choose a small subset of the data to run our experiments on. In order to get a very informative space we choose the top 2000 users from the last 2 years that watched at most 500 videos. We get 7724 unique videos across all users. We note that it is assumed that all the 7724 videos will be available to the users during the entire two year period. However, this might not be the case in practice because old videos might become inactive or not available to users on the Channel 4 platform.

From this subset we then divide the data to 25 percent testing and 75 percent training; testing with the RNN baseline model and **Like** model on new users only. Since the other models suffer from the cold start problem (item and/or user), we re-divide the data into 25/75 ratio such that all users are part of both training and

testing.

Choosing 2000 users as a subset is clearly limited, and can be considered as a prediction network for frequent users. We note that the same construction might produce different results if we introduce less frequent users. However, in order to introduce preliminary results for our hypotheses, 2000 users are adequate. Even with such a subset the training procedure lasted around a week for the RNN baseline model and for the **Like** model.

4.2 Baseline models

In order to evaluate how well our model is performing, we compare it with some baselines:

• Popularity - our own implementation which recommends to users the most popular items - see popularity section below for why did we check this model.

• Content based item-to-item - this model is a nearest neighbour model which was quite popular in the past. We provide here our own implementation based on the algorithm proposed in [2]

• Collaborative filtering item-to-item - this model is also a nearest neighbour model where similarity is measured through similarity of users/items in the utility matrix. Although it is already old, up until today it is a very popular model, especially in the e-commerce community. We implement our own version of it based on the algorithm proposed by Amazon at [53]

• Hybrid RNN - one of the more recent state of the art models. This model allows us to use all our features and hence the best comparison for our novel model - we implement a similar model to [6] but we include our novel Doc2Vec subtitle representation which is not part of their model.

We use all these baseline models in a temporal manner, so that they are comparable to the temporal manner in which our **Like** model makes predictions. For example, for content based item-to-item, during testing:

1. pick a user U
2. for t in ordered videos watched by U:

3. observe video t, update U's profile
by a weighted average of training user profile
and t's profile.
4. calculate user's profile similarity to videos
and produce a recommendation.

4.3 Metrics

Since we are interested in a ranking which will answer *Where* to place videos on the platform, we resort to two common metrics:

• Top K - this metric evaluates whether the positive video is in one of K top predicted positions, if so we give our model 1 otherwise 0. We check for each model for $K \in \{5, 10, 15\}$. In video recommendation, the justification for using top K is that any platform suggests more than one video to watch in one screen. This metric is sometimes referred as $recall@k$.

• Mean Reciprocal Rank (MRR) - This metric is position specific and allows us to evaluate how well we ranked our positive video. If a the predicted video fall on position 15, we get the reciprocal $\frac{1}{15}$ as our MRR score [6].

4.4 Results

Results indicate that our **Like** model is competitive with the state of the art methods like the Vanilla RNN with bpr-max loss: with accuracy of 52, 48 and 43 for top 15, 10 and 5 respectively over 7724 video labels. All accuracy results for testing and training are presented in tables 4.2-4.3. The results for the loss functions during testing are found in table 4.1.

Furthermore, we conduct a paired T-test (figures 4.1-4.5) for The Mean Reciprocal Rank of 0.36 in compare to 0.37 for the RNN with bpr-max loss. Results show that we can not say that their MRR are different.

4.4.1 Paired T-test

We provide in figures 4.1-4.5 pair T-test (see [64] for details) upon our MRR results per user; To validate that our **Like** model is significantly better not only on average.

Intuitively, if the paired difference standard deviation is too large relative to the mean difference, we would not be able to conclude that one model is better than the other. Results are provided in the figures below and indicate that our **Like** model can not be distinguished from the state of the art RNN model. All other figures indicate a significant difference between both the baseline RNN model with BPR-max and our **Like** model to the collaborative filtering model and RNN using cross-entropy. All models' MRR distributions are presented in figure 4.6.

```
t = 0.93201, df = 363, p-value = 0.3519
alternative hypothesis: true difference in means is not equal to 0
95 percent confidence interval:
 -0.01068293  0.02993191
sample estimates:
mean of the differences
             0.00962449
```

Figure 4.1: Paired T-test between the Vanilla RNN with bpr-max loss to our **Like** model. H_0 : MRR of the RNN is equal to the MRR of the like model. Results indicate that we can not reject H_0

```
t = 32.804, df = 363, p-value < 2.2e-16
alternative hypothesis: true difference in means is greater than 0
95 percent confidence interval:
 0.2475477       Inf
sample estimates:
mean of the differences
             0.2606508
```

Figure 4.2: Paired T-test between the Vanilla RNN with cross-entropy loss to Vanilla RNN with (bpr-max). H_0 : MRR of the RNN (xent) is bigger than MRR of the RNN (bpr-max) model. Results indicate that we can not reject H_0

```
t = 7.6284, df = 363, p-value = 1.055e-13
alternative hypothesis: true difference in means is greater than 0
95 percent confidence interval:
 0.05322536        Inf
sample estimates:
mean of the differences
             0.06790468
```

Figure 4.3: Paired T-test between the Vanilla RNN with bpr-max loss to collaborative filtering item-to-item. H_0 : MRR of the CF is bigger than MRR of the RNN (bpr-max) method. Results indicate that we can reject H_0

```
t = -6.3033, df = 363, p-value = 1
alternative hypothesis: true difference in means is greater than 0
95 percent confidence interval:
 -0.07352734         Inf
sample estimates:
mean of the differences
            -0.05828019
```

Figure 4.4: Paired T-test between the Vanilla RNN with bpr-max loss to collaborative filtering item-to-item. H_0 : MRR of the RNN (bpr-max) is bigger than MRR of the CF method. Results indicate that we can not reject H_0

```
t = -28.679, df = 363, p-value = 1
alternative hypothesis: true difference in means is greater than 0
95 percent confidence interval:
 -0.2654607         Inf
sample estimates:
mean of the differences
             -0.2510263
```

Figure 4.5: Paired T-test between the Vanilla RNN with cross entropy loss to our **Like** model. H_0 : MRR of the **Like** model is bigger than MRR of the RNN with cross-entropy. Results indicate that we can not reject H_0

Loss - testing		
Model	Xent	BPR-max
Vanilla RNN	0.033	0.203
Like model	-	0.278

Table 4.1: Loss measurements during testing for Vanilla RNN and the **Like** models

4.5 Interpertability

4.5.1 Likes vs. Dislikes

In this subsection we capture the influence of the dislike portion (Neural Network) of the network on the prediction. We present the interpretability of the results through three questions: does negative prediction contribute to prediction? do correct predictions can also be tagged as a dislike? and does the dislike network learn to contribute to the original disliked videos?

- Does negative feedback help prediction?

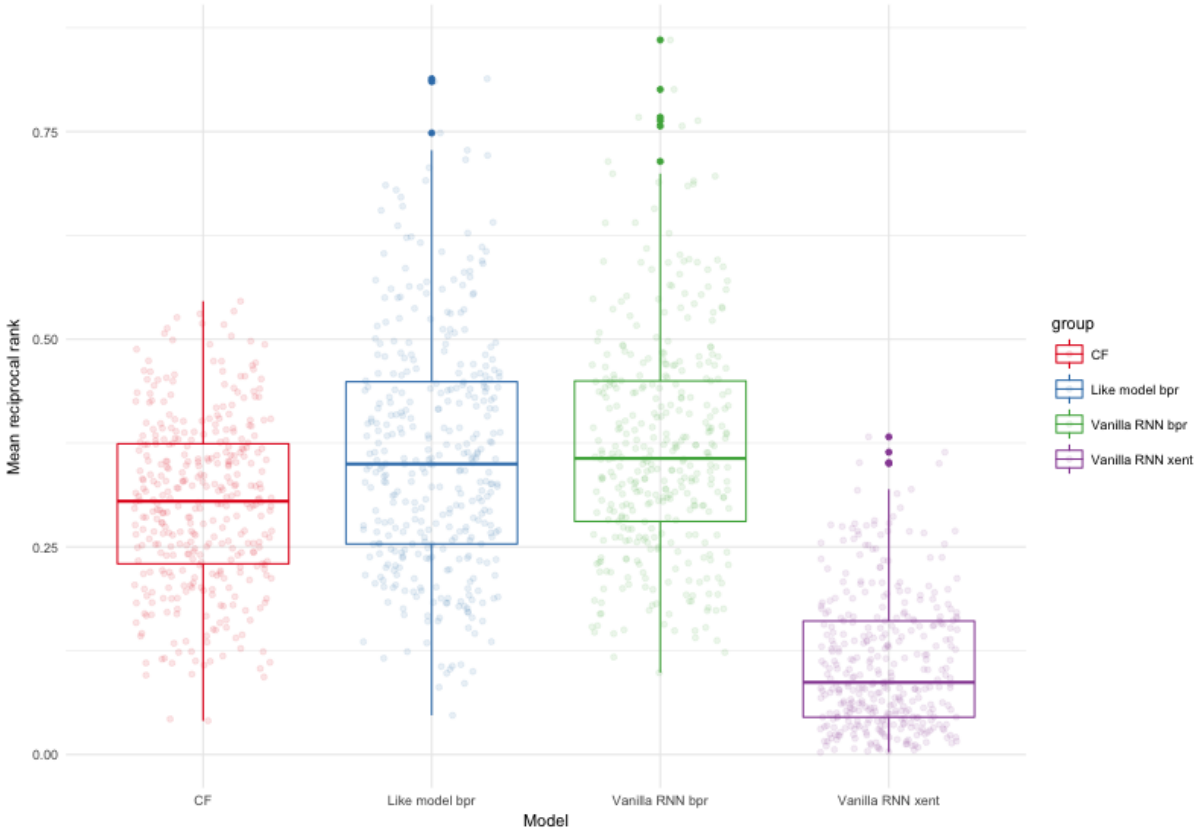

Figure 4.6: Box plot for the experimented models. From left to right: CF, The **Like** model, Vanillla RNN with bpr-max and Vanilla RNN with cross-entropy(xent) loss

Metric - training			
Model	Accuracy 15	Accuracy 10	Accuracy 5
baseline RNN (xent)	0.239	0.213	0.172
baseline RNN	0.761	0.710	0.620
Like model (BPR-max)	0.721	0.667	0.578

Table 4.2: Table presenting training accuracy of model predictions. From left to right: share of videos appearing in top 15 predictions, top 10, top 5 and the average reciprocal rank of a video.

Metric - testing				
Model	Accuracy 15	Accuracy 10	Accuracy 5	MRR
Popularity	0.015	0.010	0.005	0
Content based	0.001	0.001	0.0005	0
Collaborative filter	0.510	0.482	0.414	0.300
Vanilla RNN (xent)	0.212	0.188	0.151	0.110
Vanilla RNN (bpr-max)	0.538	0.506	0.452	0.371
Like model (BPR-max)	0.523	0.488	0.43	0.361

Table 4.3: Table presenting testing accuracy of model predictions. From left to right: share of videos appearing in top 15 predictions, top 10, top 5 and the average reciprocal rank of a video.

From the accuracy results we see that the **Like** model can make quite good prediction. If we further look at figures 4.7-4.8, we can see that the dislike portion pushes some of the scores towards its direction. We can say with that which videos are disliked and which correct predictions result from a dislike.

• Do the **Like** network correct predictions are driven also by dislikes?

we note that 12.5 percent from total predictions were driven more by dislikes than likes. Furthermore, 2.4 percent of correct predictions which are 2101 videos, were driven by dislikes rather than likes. We have to remember that we had many more positive examples than negative examples with a 50 to 1 ratio. We believe that with

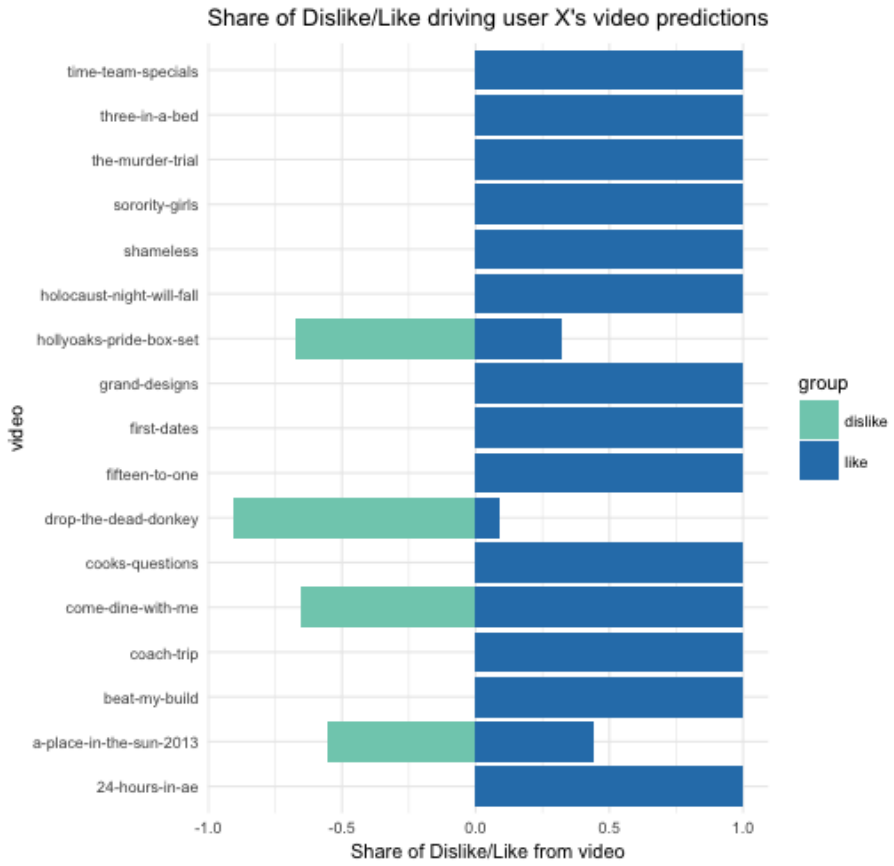

Figure 4.7: Horizontal bar chart comparing the contribution share of the like portion versus the dislike portion for randomly selected user x

more negative examples, the percent of predictions driven from dislike will increase as well. Also it is reasonable to think that most videos that were fully watched are still the big majority of the cake.

- Do we learn a dislike score for the true disliked videos?

By this we are trying to check if the input disliked videos affected their corresponding label in making a prediction. We report that the number of disliked videos is fixed to 10 and on average the network learns a contribution of 3 of these labels as having a disliked portion, with the maximum being 5. In figure 4.9 we show an example of the network learning five out of ten initial disliked videos as having a dislike contribution.

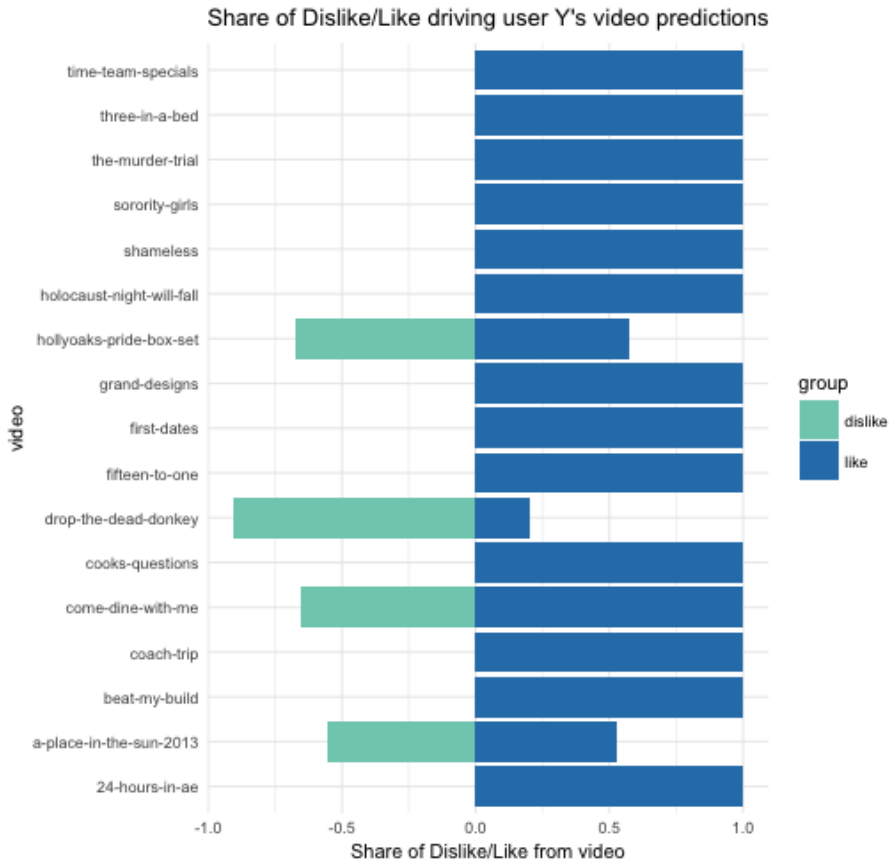

Figure 4.8: Horizontal bar chart comparing the contribution share of the like portion versus the dislike portion for randomly selected user y

4.5.2 Popularity

Since we are picking the top users, and after results are given. We can see that item-to-item collaborative filtering is doing very well (51 ,48 and 41 percent accuracy for top 15, 10 and 5 respectively). This implies that similar users watched similar items. We can ask whether these watched videos were very popular and hence viewed by most users? firstly we refer to figure 8 to show the portion which popular videos comprise from the total - around 13 percent of videos contribute around 50 percent of all views. A simple model to check this hypothesis would be to just recommend the most popular videos. We construct a simple model which uses this idea. Popularity is simply the sum of views by the users. Results for this model are poor compared to the other models but are much better than random: around 1.5, 1 and 0.5 percent accuracy for top 15, 10 and 5 respectively - see table 4.3.

dislike_score	like_score	video
0.000000	0.179691	homes-by-the-sea
0.624724	-0.648184	the-undateables
0.000000	0.473738	posh-pawnbrokers
0.368117	0.315545	car-sos
0.556636	0.201900	the-murder-detectives
0.313797	-0.335353	grand-designs-revisited
0.549838	0.012353	helluva-tour
0.000000	0.574325	grand-designs-revisited
0.000000	0.678743	married-at-first-sight-usa
0.000000	1.069999	the-restoration-man

Figure 4.9: Example for the scores from the like and dislike portion of the network for labels that are known to be disliked

4.5.3 Item similarity - qualitative analysis

Item similarity measures allow us to easily say which videos are similar to others. We examine this by picking few videos and checking what would be the top recommendation for a user who has watched them. We present it for similarity among users' activity in collaborative filtering item-to-item methods, and leave out content based similarity due to its poor results .

Extracting any insight from figures 4.12-4.13 is tricky and might imply that watching "black mirror" is a good predictor for watching "my mad fat diary" and "four in bed" and the other way around as well; this might suggest that people watch a wide variety of genres with emphasis on popular shows. In figure 4.11 it is more clear that a CF recommendation to a user who has watched "come dine with me" is either another episode of the same show or another Lifestyle genre show.

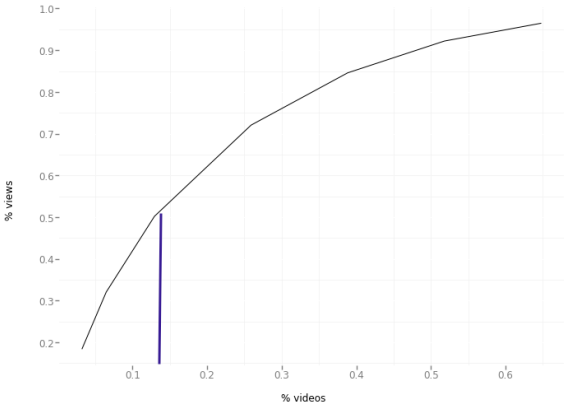

Figure 4.10: Popularity graph : here we examine the phenomenon that small number of videos are in charge for a big portion of total views. The purple vertical line shows that around 13 percent of videos constitute around 50 percent of total views.

brand	genre
come-dine-with-me	Lifestyle
jamies-money-saving-meals	Lifestyle
come-dine-with-me	Lifestyle
jewish-mum-of-the-year	Entertainment
the-romanians-are-coming	Factual
inspector-borowski	Drama
come-dine-with-me	Lifestyle
kotv-boxing-weekly	Sports
come-dine-with-me	Lifestyle
gadget-man	Lifestyle
virtually-famous	Entertainment
come-dine-with-me	Lifestyle
countdown	Entertainment
come-dine-with-me	Lifestyle
motor-sport	Sports

Figure 4.11: CF similarity to ”come dine with me”

brand	genre
my-mad-fat-diary	Drama
four-in-a-bed	Lifestyle
black-mirror	Drama
black-mirror	Drama
the-adam-and-joe-show	Comedy
kotv-boxing-weekly	Sports
come-dine-with-me	Lifestyle
10	Drama
married-at-first-sight	Factual
come-dine-with-me	Lifestyle
celebs-go-dating	Factual
room-for-improvement	Lifestyle
come-dine-with-me	Lifestyle
four-in-a-bed	Lifestyle
come-dine-with-me	Lifestyle

Figure 4.12: CF similarity to ”my mad fat diary”

brand	genre
black-mirror	Drama
black-mirror	Drama
my-mad-fat-diary	Drama
four-in-a-bed	Lifestyle
kotv-boxing-weekly	Sports
come-dine-with-me	Lifestyle
no-limit	Drama
food-unwrapped	Factual
four-in-a-bed	Lifestyle
stand-up-to-cancer	Entertainment
black-books	Comedy
double-your-house-for-half-the-money	Lifestyle
drifters	Comedy
come-dine-with-me	Lifestyle
unreported-world	Factual

Figure 4.13: CF similarity to ”black mirror”

4.5.4 Metric suggestion

Another interesting thing to check is if our network already knows things about later predictions within earlier timestamp. I.e. how quickly do the features we have weighted by learnt parameters, already capture the space of videos that is watched by the individual user. We can capture this phenomenon by checking at each prediction step if our top K predictions include later views for that user. We call this

measurement quick-frame-k (QF_k) and mean quick-frame-k (MQF_k). Formally, this is defined by: $V = \{v_1, v_2, .., v_t\}$ $top_{kit} = \{P_{1it}, P_{2it}, .., P_{kit}\}$ Where P is the indices of predictions for each user sorted by highest to lowest value.

$MQF_{ki} = \frac{1}{t} \sum_t \frac{(top_{kit} \subseteq V_{t+1})}{\sum \mathbb{1} V_{t+1}}$

Where $V_{t+1} = \{v_{t+1}, v_{t+2}, ..., v_{t+n}\}$

$QF_{kit} = top_{kit} \subseteq V_{t+1}$ We note that due to time constraints we do not include its implementation and results.

Chapter 5

Conclusions

This paper has focused on the problem of whether a person likes or dislikes a video and how this can help in better recommendation and interpretation. This was seen from the perspective of the general recommendation problem of *What* to recommend? *When* to recommend it? and *Where* to place our recommendation? which we termed as the **Joint** recommendation problem.

Accordingly, in order to solve this problem, we have constructed the novel **Like** model: a multi-network comprised of an RNN and a funnel Neural Network meeting at the output layer. Though done in a small scale (and was stopped prematurely), this solution is competitive with recent state-of-the-art models and can enable to interpret if a video is liked or disliked (What) - since the Neural Network learns a pseudo lower dimensional representation of the disliked features. It learns *When* should we recommend a video - achieved through the temporal nature of the RNN and *Where* to place it given by the chosen ranking loss.

The **Like** model has been measured against our own implementation for recent state of the art model as RNN[33] and also against notable methods as collaborative filtering item-to-item, content based item-to-item and matrix factorization model (package implementation). It was shown that the **Like** model outperforms these models or at least matches the state of the art results. Significance test was conducted to ensure the model robustness.

With this solution, not only we can better predict videos, since it is argued that both negative and positive experience can direct actions, but also we can interpret if

a video is going to be liked or not and thus avoid from recommending it.

This model holds few limitations which we can categorize under the **Joint** recommendation problem (figure 1) and map exactly to future research suggestions.

Firstly, regarding *What* to recommend, our **Like** model can not say in a clear cut that the portion attributed through the disliked network is a real dislike, since the loss is learned jointly to the like portion. Secondly, we limit our dislike portion by considering it as constant measure added to all time-stamps. However, we might benefit from a temporal dislike model (*When*) - as it is reasonable to consider preferences changing through time in both the negative and positive dimensions. Lastly, incorporating a more specific ranking loss could provide with a better order of *Where* to place each recommendation. These limitations lead us directly for future research opportunities.

5.1 Future development

What - Our **Like** model could be considered as a sort of multi-task network. So, although a better accuracy is expected when using multi-task learning - since separating the tasks is a clear way to accumulate loss, it would be of interest to implement the network in a way such that the dislike portion is learned by an Auto-encoder prior to combining it with the RNN. A-priori dislike learning using an Auto-encoder guarantees a clear cut separation of dislikes and likes. Moreover, we can expect a richer representation of the dislike feature construction than in the **Like** model since the deconstruct-reconstruct procedure forces meaningful features

When - Considering dislikes as changing over time, further works might benefit from two RNN networks. - architecturally close to [79]. Where one RNN network can represent the ”likes”, while the dislikes can be the represented by a second RNN-auto-encoder network similar to [21].

Where - we have emphasized during this work the importance of ranking videos and presenting to the user positive experiences. This can be done with more attention if the loss to which we optimize is a specialized ranking loss. One such loss is the mean reciprocal rank [20, 73] that is a list-wise loss that enforces ordering

of rank, which we have used only as a metric.

Many other such losses exist[49], with many being non-differentiable. One optional solution is to use Reinforcement Learning methods. For example, The REINFORCE algorithm [77] does not need to be differentiable with respect to rewards [62]. This means that it can be applied to many problems which optimize a specific loss which is non-differentiable such as BLEU [57]. Thus, the great possibility to choose any metric that we would like, even if non differentiable and optimize accordingly; the non-differentiable part is replaced by the reward signal.

We have started to experiment and implemented a GAN [27] recommendation model using the REINFORCE algorithm [77] , but left it out due to time constraints.

Appendix A

Subtitle feature alternatives

Apart from our representation of video subtitles using the Doc2Vec method, we present here-under two additional popular methods that could be used as well to characterize the video subtitles; the one being the hitorical TF.IDF and the other the more recent successful LDA.

TF.IDF can be explained easily as a counting method. Where we count the number of times a term occurs in a document and we weight this counter by the frequency across all documents in the set. If the same word appears frequently in all documents, there is not much we can learn about the specific topic. The simplest example would be the so-called stop words like {the, on, a} which are widely used with no informative meaning behind. Once a score vector is obtained, we can pick the top K words as document characterizations. Counting however does not necessarily imply on the underlying characteristics of the text.

LDA[16] is an unsupervised model. Meaning we do not have topics characterizing documents before learning, but we construct them from scratch. The approach is used to classify words in a document to topics. Whilst the idea is simple, its mathematical formulation is more evolved, and will not be stated here. The idea is that once we have a collection of documents, each of these documents has a different distribution over words. We then want to assign a word in a document to a topic without forgetting the probability of this word under that topic. To clarify, It might be that a document consists of two topics with one topic appearing 99 percent of the time and the other 1 percent of the time. However, the probability of the word

'qualia' is zero under the topic that appears in 99 percent chance. In that case, it would still make sense to choose the topic with the one percent chance.

Appendix B

Gradients

In order to learn parameters it is essential that we can differentiate the loss back to the appropriate weight. Hence is also the importance in choosing a loss that will enable learning. We present here-under the loss of the **Like** model.

B.0.0.1 Cross-entropy

We have the loss being

$$Xent = \sum_i y_i \log \hat{y}_i \tag{B.1}$$

Where y_i is the true label and

$$\hat{y}_i(z) = \frac{\exp z_i}{\sum_j \exp z_j} \tag{B.2}$$

with

$$z_i = like + dislike \tag{B.3}$$

.

According to equations 8-12, we have:

$$Like = W_i * LSTM(X) + b_i \tag{B.4}$$

$$Dislike = Vh_2 + bias_d islike \tag{B.5}$$

Hence,

$$\frac{\partial Xent}{\partial \hat{y}_i} = y_i * \frac{1}{\hat{y}_i} * \frac{\partial \hat{y}_i}{\partial z_i} \tag{B.6}$$

$$\frac{\partial \hat{y}_i}{\partial z_i} = \frac{\exp z_i * \sum_j \exp z_j + \exp {z_i}^2}{\sum_j \exp {z_j}^2} \tag{B.7}$$

$$= \hat{y}_i - (\hat{y}_i)^2 = \hat{y}_i\big(1 - \hat{y}_i\big) \tag{B.8}$$

Where to further find the ”Like” portion and its LSTM parameters, we would need to use the parameter tying technique. This basically means that even though we use the same parameters across the network, we can first treat them as different, use a regular back-propogation and only in the end tie them up by renaming all to the same name. We will present here-under the derivation for the weights with respect to W_inp which are the parameters of the input gate. The other weights are calculated in a similar manner.

$$\begin{aligned}
\frac{\partial \hat{z}_i}{\partial h_t} &= W_t \\
\frac{\partial \hat{h}_t}{\partial C_t} &= O_t \cdot (1 - \tanh(C_t)^2) * C_t^{'} \\
\frac{\partial \hat{C}_t}{\partial I_t} &= \tilde{C}_{t-1} \\
\frac{\partial \hat{I}_t}{\partial W_i} &= I_t * (1 - I_t) * [h_{t-1}, x_t]
\end{aligned} \tag{B.9}$$

Depending on the desired parameters, we multiply the appropriate derivatives presented above to make the update.

For the ”Dislike” portion of the network,

$$\begin{aligned}\frac{\partial \hat{z}_i}{\partial V} &= \hat{h}_2 \\ \frac{\partial \hat{h_2}}{\partial \hat{h}_1} &= \mathbb{1}(\hat{h}_2 > 0) * U \\ \frac{\partial \hat{h}_1}{\partial W} &= \mathbb{1}(\hat{h}_1 > 0) * x_d\end{aligned} \tag{B.10}$$

B.1 BPR-max

Here we have the loss

$$BPR_{max} = -1/N_s \log \sum_j \sigma(\hat{y_i} - \hat{y_j}) * \hat{s_j} \tag{B.11}$$

With N_s being the number of negative samples and,

$$\hat{s_j} = softmax(y_j), j = 1, .., N_s \tag{B.12}$$

So we have the derivative with respect to the desired label y_i to be:

$$\frac{\partial \hat{L}_{bpr-max}}{\partial y_i} = \frac{-1}{N_s} \frac{\sum_j \hat{s_j} \sigma(\hat{y_i} - \hat{y_j}) * (1 - \sigma(\hat{y_i} - \hat{y_j}))}{\sum_j \hat{s_j} \sigma(\hat{y_i} - \hat{y_j})} \frac{\partial \hat{L}_{bpr-max}}{\partial y_j} \quad = \hat{s}_j - \frac{\hat{s}_j \sigma^2(\hat{y_i} - \hat{y_j})}{\sum_j \hat{s_j} \sigma(\hat{y_i} - \hat{y_j})} \tag{B.13}$$

Where $y_i = Like_i + dislike_i$ and the other derivatives are similar to the ones calculated for the cross-entropy section above.

(example) This document was set in the Times Roman typeface using LaTeX and BibTeX, composed with a text editor.

B.1.1 Subtitle feature alternatives

Apart from our representation of video subtitles using the Doc2Vec method, we present here-under two additional popular methods that could be used as well to characterize the video subtitles; the one being the hitorical TF.IDF and the other the more recent successful LDA.

TF.IDF can be explained easily as a counting method. Where we count the number of times a term occurs in a document and we weight this counter by the frequency across all documents in the set. If the same word appears frequently in all documents, there is not much we can learn about the specific topic. The simplest example would be the so-called stop words like {the, on, a} which are widely used with no informative meaning behind. Once a score vector is obtained, we can pick the top K words as document characterizations. Counting however does not necessarily imply on the underlying characteristics of the text.

LDA[16] is an unsupervised model. Meaning we do not have topics characterizing documents before learning, but we construct them from scratch. The approach is used to classify words in a document to topics. Whilst the idea is simple, its mathematical formulation is more evolved, and will not be stated here. The idea is that once we have a collection of documents, each of these documents has a different distribution over words. We then want to assign a word in a document to a topic without forgetting the probability of this word under that topic. To clarify, It might be that a document consists of two topics with one topic appearing 99 percent of the time and the other 1 percent of the time. However, the probability of the word *'qualia'* is zero under the topic that appears in 99 percent chance. In that case, it would still make sense to choose the topic with the one percent chance.

B.1.2 Gradients

In order to learn parameters it is essential that we can differentiate the loss back to the appropriate weight. Hence is also the importance in choosing a loss that will enable learning. We present here-under the loss of the **Like** model.

B.1.2.1 Cross-entropy

We have the loss being

$$Xent = \sum_i y_i \log \hat{y}_i \tag{B.14}$$

Where y_i is the true label and

$$\hat{y}_i(z) = \frac{\exp z_i}{\sum_j \exp z_j} \tag{B.15}$$

with

$$z_i = like + dislike \tag{B.16}$$

.

According to equations 8-12, we have:

$$Like = W_i * LSTM(X) + b_i \tag{B.17}$$

$$Dislike = Vh_2 + bias_dislike \tag{B.18}$$

Hence,

$$\frac{\partial Xent}{\partial \hat{y}_i} = y_i * \frac{1}{\hat{y}_i} * \frac{\partial \hat{y}_i}{\partial z_i} \tag{B.19}$$

$$\frac{\partial \hat{y}_i}{\partial z_i} = \frac{\exp z_i * \sum_j \exp z_j + \exp {z_i}^2}{\sum_j \exp {z_j}^2} \tag{B.20}$$

$$= \hat{y}_i - (\hat{y}_i)^2 = \hat{y}_i\left(1 - \hat{y}_i\right) \tag{B.21}$$

Where to further find the ”Like” portion and its LSTM parameters, we would

need to use the parameter tying technique. This basically means that even though we use the same parameters across the network, we can first treat them as different, use a regular back-propogation and only in the end tie them up by renaming all to the same name. We will present here-under the derivation for the weights with respect to W_inp which are the parameters of the input gate. The other weights are calculated in a similar manner.

$$\begin{aligned}
\frac{\partial \hat{z}_i}{\partial h_t} &= W_t \\
\frac{\partial \hat{h}_t}{\partial C_t} &= O_t \cdot (1 - \tanh(C_t)^2) * C_t^{'} \\
\frac{\partial \hat{C}_t}{\partial I_t} &= \tilde{C}_{t-1} \\
\frac{\partial \hat{I}_t}{\partial W_i} &= I_t * (1 - I_t) * [h_{t-1}, x_t]
\end{aligned} \tag{B.22}$$

Depending on the desired parameters, we multiply the appropriate derivatives presented above to make the update.

For the ”Dislike” portion of the network,

$$\begin{aligned}
\frac{\partial \hat{z}_i}{\partial V} &= \hat{h}_2 \\
\frac{\partial \hat{h}_2}{\partial \hat{h}_1} &= \mathbb{1}(\hat{h}_2 > 0) * U \\
\frac{\partial \hat{h}_1}{\partial W} &= \mathbb{1}(\hat{h}_1 > 0) * x_d
\end{aligned} \tag{B.23}$$

B.1.2.2 BPR-max

Here we have the loss

$$BPR_{max} = -1/N_s \log \sum_j \sigma(\hat{y}_i - \hat{y}_j) * \hat{s}_j \tag{B.24}$$

With N_s being the number of negative samples and,

$$\hat{s_j} = softmax(y_j), j = 1,..,N_s \tag{B.25}$$

So we have the derivative with respect to the desired label y_i to be:

$$\frac{\partial \hat{L}_{bpr-max}}{\partial y_i} = \frac{-1}{N_s} \frac{\sum_j \hat{s_j} \sigma(\hat{y_i} - \hat{y_j}) * (1 - \sigma(\hat{y_i} - \hat{y_j}))}{\sum_j \hat{s_j} \sigma(\hat{y_i} - \hat{y_j})} \frac{\partial \hat{L}_{bpr-max}}{\partial y_j} \quad = \hat{s}_j - \frac{\hat{s_j} \sigma^2(\hat{y_i} - \hat{y_j})}{\sum_j \hat{s_j} \sigma(\hat{y_i} - \hat{y_j})} \tag{B.26}$$

Where $y_i = Like_i + dislike_i$ and the other derivatives are similar to the ones calculated for the cross-entropy section above.

Bibliography

[1] G. Adomavicius and A. Tuzhilin. “Context-aware recommender systems”. In: *Recommender Systems Handbook, Second Edition*. 2015, pp. 191–226. ISBN: 9781489976376. DOI: 10.1007/978-1-4899-7637-6_6. arXiv: 3.

[2] Gediminas Adomavicius and Alexander Tuzhilin. *Toward the next generation of recommender systems: A survey of the state-of-the-art and possible extensions*. 2005. DOI: 10.1109/TKDE.2005.99. arXiv: 3.

[3] Anima Anandkumar et al. “Tensor Decompositions for Learning Latent Variable Models”. In: *Journal of Machine Learning Research* 9355 (2014), pp. 19–38. ISSN: 16113349. DOI: 10.1007/978-3-319-24486-0_2. arXiv: 1210.7559.

[4] Chris Anderson. “The Long Tail”. In: *WIRED magazine* 12.10 (2004), pp. 170–177. ISSN: 1580979X. DOI: 10.3359/oz0912041. URL: http://www.geeknewscentral.com/2010/08/13/the-long-tail/.

[5] Marko Balabanović and Yoav Shoham. “Fab: content-based, collaborative recommendation”. In: *Communications of the ACM* 40.3 (1997), pp. 66–72. ISSN: 00010782. DOI: 10.1145/245108.245124. arXiv: 10.

[6] Alexandros Karatzoglou Balazs Hidasi. “Session-based Recommendation with Recurrent Neural Networks”. In: *ICLR* (2016), pp. 1–10. arXiv: arXiv:1511.06939v2.

[7] David Barber. “Bayesian Reasoning and Machine Learning”. In: *Machine Learning* (2011), p. 646. ISSN: 9780521518147. DOI: 10.1017/CBO9780511804779. arXiv: arXiv:1011.1669v3. URL: http://eprints.pascal-network.org/archive/00007920/%7B%5C%%7D5Cnhttp://scholar.google.com/scholar?hl=en%7B%5C&%7DbtnG=Search%7B%5C&%7Dq=intitle:Bayesian+Reasoning+and+Machine+Learning%7B%5C#%7D0.

[8] Chumki Basu, Haym Hirsh, and William Cohen. “Recommendation as Classification : Using Social and Content-Based Information in Recommendation”. In: *Proceedings of the Fifteenth National Conference on Artificial Intelligence* (1998), pp. 714–720. DOI: 10.1.1.36.4620. arXiv: 13. URL: http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.36.4620.

[9] R Bell, Y Koren, and C Volinsky. “The BellKor 2008 Solution to the Netflix Prize”. In: *Netflix prize documentation* 12 (2009), pp. 1–21. ISSN: 1768-3114. DOI: 10.1016/j.patbio.2009.09.005. URL: http://www.ncbi.nlm.nih.gov/pubmed/19898912.

[10] Robert M Bell, Yehuda Koren, and Chris Volinsky. “The BellKor solution to the Netflix Prize A factorization model”. In: *KorBell Teams Report to Netflix* 2 (2007). URL: http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.142.9009%7B%5C&%7Drep=rep1%7B%5C&%7Dtype=pdf.

[11] Robert Bell, Yehuda Koren, and Chris Volinsky. “Modeling relationships at multiple scales to improve accuracy of large recommender systems”. In: *Proceedings of the 13th ACM SIGKDD international conference on Knowledge discovery and data mining, KDD’07* 5.1 (2007), pp. 95–104. ISSN: 1567133X. DOI: 10.1145/1281192.1281206. URL: http://portal.acm.org/citation.cfm?doid=1281192.1281206.

[12] Yoshua Bengio, Patrice Simard, and Paolo Frasconi. "Learning Long-Term Dependencies with Gradient Descent is Difficult". In: *IEEE Transactions on Neural Networks* 5.2 (1994), pp. 157–166. ISSN: 19410093. DOI: 10.1109/72.279181. arXiv: arXiv:1211.5063v2.

[13] Yoshua Bengio et al. "A Neural Probabilistic Language Model". In: *The Journal of Machine Learning Research* 3 (2003), pp. 1137–1155. ISSN: 15324435. DOI: 10.1162/153244303322533223. arXiv: arXiv:1301.3781v3.

[14] James Bennett and Stan Lanning. "The Netflix Prize". In: *KDD Cup and Workshop* (2007), pp. 3–6. ISSN: 1554351X. DOI: 10.1145/1562764.1562769.

[15] Mathias Berglund et al. "Bidirectional Recurrent Neural Networks as Generative Models". In: *Advances in Neural Information Processing Systems 28* (2015), pp. 856–864. arXiv: 1504.01575. URL: http://papers.nips.cc/paper/5651-bidirectional-recurrent-neural-networks-as-generative-models.pdf.

[16] David M Blei et al. "Latent Dirichlet Allocation". In: *Journal of Machine Learning Research* 3 (2003), pp. 993–1022. ISSN: 15324435. DOI: 10.1162/jmlr.2003.3.4-5.993. arXiv: 1111.6189v1.

[17] John S. Breese, David Heckerman, and Carl Kadie. "Empirical analysis of predictive algorithms for collaborative filtering". In: *Proceedings of the 14th Annual Conference on Uncertainty in Artificial Intelligence* (1998), pp. 43–52. ISSN: 15532712. DOI: 10.1111/j.1553-2712.2011.01172.x. arXiv: 1301.7363.

[18] Zhe Cao et al. "Learning to Rank : From Pairwise Approach to Listwise Approach". In: *Proceedings of the 24th international conference on Machine learning* (2007), pp. 129–136. ISSN: 1595937935. DOI: 10.1145/1273496.1273513. URL: http://research.microsoft.com/en-us/people/tyliu/listnet.pdf.

[19] Dennis L. Chao, Justin Balthrop, and Stephanie Forrest. “Adaptive radio: achieving consensus using negative preferences”. In: *Proceedings of the 2005 international ACM SIGGROUP conference on Supporting group work - GROUP ’05* (2005), p. 120. DOI: 10.1145/1099203.1099224. URL: http://dl.acm.org/citation.cfm?id=1099203.1099224.

[20] Olivier Chapelle and Donald Metlzer. “Expected reciprocal rank for graded relevance”. In: *Proceedings of the 18th ...* (2009), pp. 621–630. ISSN: 9781605585123. DOI: 10.1145/1645953.1646033. URL: http://portal.acm.org/citation.cfm?id=1645953.1646033%7B%5C%%7D5Cnhttp://portal.acm.org/citation.cfm?doid=1645953.1646033%7B%5C%%7D5Cnhttp://dl.acm.org/citation.cfm?id=1646033.

[21] Kyunghyun Cho et al. “Learning Phrase Representations using RNN Encoder-Decoder for Statistical Machine Translation”. In: *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)* (2014), pp. 1724–1734. ISSN: 09205691. DOI: 10.3115/v1/D14-1179. arXiv: 1406.1078. URL: http://arxiv.org/abs/1406.1078.

[22] Mark Claypool et al. “Combining content-based and collaborative filters in an online newspaper”. In: *Proceedings of Recommender Systems Workshop at ACM SIGIR* (1999), pp. 40–48. DOI: 10.1.1.46.3659. URL: http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.45.5230%7B%5C&%7Drep=rep1%7B%5C&%7Dtype=pdf.

[23] J Elster. *Nuts and bolts for the social sciences*. 1989, pp. viii, 184. ISBN: 0521374553. URL: http://www.loc.gov/catdir/description/cam023/89031449.html.

[24] Evgeny Frolov and Ivan Oseledets. “Fifty Shades of Ratings: How to Benefit from a Negative Feedback in Top-N Recommendations Tasks”. In: *Proceedings of the 10th ACM Conference on Recommender Systems - RecSys ’16*.

2016, p. 8. ISBN: 9781450340359. DOI: 10.1145/2959100.2959170. arXiv: 1607.04228. URL: http://arxiv.org/abs/1607.04228.

[25] Evgeny Frolov and Ivan Oseledets. *Tensor methods and recommender systems*. 2017. DOI: 10.1002/widm.1201. arXiv: 1603.06038.

[26] Susan Gauch et al. "User Profiles for Personalized Information Access". In: *The Adaptive Web* 4321 (2007), pp. 54–89. ISSN: 10849521. DOI: 10.1007/978-3-540-72079-9_2. URL: http://www.springerlink.com/index/10.1007/978-3-540-72079-9%7B%5C_%7D2.

[27] Ian Goodfellow. "NIPS 2016 Tutorial: Generative Adversarial Networks". In: *NIPS*. 2016, p. 57. ISBN: 1581138285. DOI: 10.1001/jamainternmed.2016.8245. arXiv: 1701.00160. URL: http://arxiv.org/abs/1701.00160.

[28] Alex Graves. "Generating sequences with recurrent neural networks. preprint". In: *arXiv:1308.0850* (2013). ISSN: 18792782. DOI: 10.1145/2661829.2661935. arXiv: arXiv:1308.0850v5. URL: http://arxiv.org/abs/1308.0850.

[29] Lieve Hamers et al. "Similarity measures in scientometric research: The Jaccard index versus Salton's cosine formula". In: *Information Processing and Management* 25.3 (1989), pp. 315–318. ISSN: 03064573. DOI: 10.1016/0306-4573(89)90048-4.

[30] Xiangnan He et al. "Fast Matrix Factorization for Online Recommendation with Implicit Feedback". In: *Proceedings of the 39th International ACM SIGIR conference on Research and Development in Information Retrieval - SIGIR '16* (2016), pp. 549–558. DOI: 10.1145/2911451.2911489. arXiv: 1708.05024. URL: http://dl.acm.org/citation.cfm?doid=2911451.2911489.

[31] Jonathan L Herlocker, Joseph a Konstan, and John Riedl. "Explaining collaborative filtering recommendations". In: *Proceedings of the 2000 ACM conference on Computer supported cooperative work* (2000), pp. 241–250. ISSN: 00318655. DOI: 10.1145/358916.358995. arXiv: 48. URL: http://dl.acm.org/citation.cfm?id=358995.

[32] Balázs Hidasi and Alexandros Karatzoglou. "Recurrent Neural Networks with Top-k Gains for Session-based Recommendations." In: *CoRR* abs/1706.03847 (2017). URL: http://dblp.uni-trier.de/db/journals/corr/corr1706.html#HidasiK17.

[33] Balázs Hidasi et al. "Session-based Recommendations with Recurrent Neural Networks". In: *Iclr* (2015), pp. 1–9. ISSN: 10797114. DOI: 10.1103/PhysRevLett.116.151105. arXiv: 1511.06939. URL: http://arxiv.org/abs/1511.06939.

[34] Geoffrey E Hinton, Simon Osindero, and Yee-Whye Teh. "A Fast Learning Algorithm for Deep Belief Nets". In: *Neural Computation* 18.7 (2006), pp. 1527–1554. ISSN: 0899-7667. DOI: 10.1162/neco.2006.18.7.1527. arXiv: 1111.6189v1. URL: http://www.ncbi.nlm.nih.gov/pubmed/16764513%7B%5C%%7D5Cnhttp://www.mitpressjournals.org/doi/abs/10.1162/neco.2006.18.7.1527.

[35] Josef Hochreiter. "Untersuchungen zu dynamischen neuronalen Netzen". PhD thesis. 1991, pp. 1–71. URL: http://scholar.google.com/scholar?hl=en%7B%5C&%7DbtnG=Search%7B%5C&%7Dq=intitle:Untersuchungen+zu+dynamischen+neuronalen+Netzen%7B%5C#%7D0.

[36] S Hochreiter and J Schmidhuber. "Long Short-Term Memory". In: *Neural Computation* 9.8 (1997), pp. 1735–1780. ISSN: 0899-7667. DOI: 10.1162/neco.1997.9.8.1735. arXiv: 1206.2944. URL: http://ieeexplore.ieee.org/xpl/freeabs%7B%5C_%7Dall.

jsp?arnumber=6795963%7B%5C&%7DabstractAccess=no%7B%5C&%7DuserType=inst.

[37] Sepp Hochreiter et al. "Gradient flow in recurrent nets: the difficulty of learning long-term dependencies". In: *A Field Guide to Dynamical Recurrent Networks* (2001), pp. 237–243. ISSN: 1098-6596. DOI: 10.1109/9780470544037.ch14. arXiv: arXiv:1011.1669v3. URL: http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.24.7321%7B%5C&%7Drep=rep1%7B%5C&%7Dtype=pdf.

[38] Harold Hotelling. "Analysis of a complex of statistical variables into principal components." In: *Journal of Educational Psychology* 24.6 (1933), pp. 417–441. ISSN: 0022-0663. DOI: 10.1037/h0071325.

[39] Yifan Hu, Chris Volinsky, and Yehuda Koren. "Collaborative filtering for implicit feedback datasets". In: *Proceedings - IEEE International Conference on Data Mining, ICDM*. 2008, pp. 263–272. ISBN: 9780769535029. DOI: 10.1109/ICDM.2008.22.

[40] Aaron Courville Ian Goodfellow, Yoshua Bengio. *Deep Learning*. 2016, pp. 330–373. ISBN: 3540620583, 9783540620587. DOI: 10.1016/B978-0-12-391420-0.09987-X. arXiv: arXiv:1011.1669v3.

[41] Prateek Jain and IS Dhillon. "Provable inductive matrix completion". In: *arXiv* (2013), pp. 1–22. arXiv: 1306.0626. URL: http://arxiv.org/abs/1306.0626.

[42] Diane Kelly and Jaime Teevan. "Implicit feedback for inferring user preference: A Bibliography". In: *ACM SIGIR Forum* 37.2 (2003), pp. 18–28. ISSN: 01635840. DOI: 10.1145/959258.959260. arXiv: 57.

[43] Elizabeth A Kensinger. "Remembering the details: Effects of emotion." In: *Emotion Review* 1.2 (2009), pp. 99–113. ISSN: 1754-0739. DOI: 10.1177/1754073908100432. URL: http://www.ncbi.nlm.nih.gov/pubmed/19421427%7B%5C%%7D5Cnhttp://www.ncbi.nlm.nih.gov/pubmed/19421427.

[44] Diederik P. Kingma and Jimmy Lei Ba. "Adam: a Method for Stochastic Optimization". In: *International Conference on Learning Representations 2015* (2015), pp. 1–15. ISSN: 09252312. DOI: http://doi.acm.org.ezproxy.lib.ucf.edu/10.1145/1830483.1830503. arXiv: 1412.6980.

[45] J.Z. Kolter and M.a. Maloof. "Dynamic weighted majority: a new ensemble method for tracking concept drift". In: *Third IEEE International Conference on Data Mining* (2003), pp. 123–130. ISSN: 15504786. DOI: 10.1109/ICDM.2003.1250911. URL: http://ieeexplore.ieee.org/xpls/abs%7B%5C_%7Dall.jsp?arnumber=1250911.

[46] Yehuda Koren. "Collaborative filtering with temporal dynamics". In: *Proc. of KDD '09* (2009), pp. 447–456. ISSN: 00010782. DOI: 10.1145/1557019.1557072. URL: http://dx.doi.org/10.1145/1557019.1557072.

[47] Yehuda Koren. "The bellkor solution to the netflix grand prize". In: *Netflix prize documentation* August (2009), pp. 1–10. ISSN: 08953309. DOI: 10.1.1.162.2118. URL: http://www.stat.osu.edu/%7B~%7Ddmsl/GrandPrize2009%7B%5C_%7DBPC%7B%5C_%7DBellKor.pdf.

[48] Yehuda Koren et al. "Factorization meets the neighborhood: a multifaceted collaborative filtering model". In: *Proceedings of the 14th ACM SIGKDD international conference on Knowledge discovery and data mining* (2008), pp. 426–434. ISSN: 1605581933. DOI: 10.1145/1401890.1401944. arXiv: 62. URL: ACM.

[49] Quoc Le and Alexander Smola. "Direct Optimization of Ranking Measures". In: *arXiv preprint arXiv:0704.3359* 1.2999 (2007), pp. 1–29. arXiv: 0704.3359. URL: http://arxiv.org/abs/0704.3359.

[50] Qv Le and Tomas Mikolov. "Distributed Representations of Sentences and Documents". In: *International Conference on Machine Learning - ICML 2014* 32 (2014), pp. 1188–1196. ISSN: 10495258. DOI: 10.1145/

2740908.2742760. arXiv: 1405.4053. URL: http://arxiv.org/abs/1405.4053.

[51] Y LeCun. *A theoretical framework for Back-Propagation*. 1988. DOI: 10.1007/978-3-642-35289-8. arXiv: arXiv:1011.1669v3.

[52] Danielle Lee and Peter Brusilovsky. "Reinforcing Recommendation Using Implicit Negative Feedback". In: *User Modeling Adaptation and Personalization* 5535.1 (2009), pp. 422–427. ISSN: 10985549. DOI: 10.1007/978-3-642-02247-0. URL: http://www.springerlink.com/content/v12186r1715k5601.

[53] Greg Linden, Brent Smith, and Jeremy York. "Amazon.com recommendations: Item-to-item collaborative filtering". In: *IEEE Internet Computing* 7.1 (2003), pp. 76–80. ISSN: 10897801. DOI: 10.1109/MIC.2003.1167344. arXiv: 69.

[54] Benjamin M Marlin et al. "Collaborative Filtering and the Missing at Random Assumption". In: *Proceedings of the Twenty-Third Conference on Uncertainty in Artificial Intelligence* (2007), p. 9. DOI: 10.1.1.80.6934. arXiv: 1206.5267. URL: http://arxiv.org/abs/1206.5267.

[55] Tomas Mikolov et al. "Efficient Estimation of Word Representations in Vector Space". In: *Proceedings of the International Conference on Learning Representations (ICLR 2013)* (2013), pp. 1–12. ISSN: 15324435. DOI: 10.1162/153244303322533223. arXiv: arXiv:1301.3781v3. URL: http://arxiv.org/pdf/1301.3781v3.pdf.

[56] Christopher Olah. *Understanding LSTM Networks*. 2015. URL: http://colah.github.io/posts/2015-08-Understanding-LSTMs/.

[57] Kishore Papineni et al. "BLEU: a method for automatic evaluation of machine translation". In: ... *of the 40Th Annual Meeting on* ... July (2002), pp. 311–318. ISSN: 00134686. DOI: 10.3115/1073083.1073135. arXiv: 1702.00764. URL: http://dl.acm.org/citation.cfm?id=1073135.

[58] Sung Eun Park, Sangkeun Lee, and Sang Goo Lee. "Session-based Collaborative Filtering for predicting the next song". In: *Proceedings - 1st ACIS/JNU International Conference on Computers, Networks, Systems, and Industrial Engineering, CNSI 2011*. 2011, pp. 353–358. ISBN: 9780769544175. DOI: 10.1109/CNSI.2011.72.

[59] Arkadiusz Paterek. "Improving regularized singular value decomposition for collaborative filtering". In: *KDD Cup and Workshop* (2007), pp. 2–5. ISSN: 00121606. DOI: 10.1145/1557019.1557072. arXiv: 07/0008 [978-1-59593-834-3].

[60] Ladislav Peska and Peter Vojtas. "Negative Implicit Feedback in E-commerce Recommender Systems". In: *Proceedings of the 3rd International Conference on Web Intelligence, Mining and Semantics* (2013). ISSN: 1066-8888. DOI: 10.1145/2479787.2479800.

[61] Anand Rajaraman and Jeffrey D Ullman. "Mining of Massive Datasets". In: *Lecture Notes for Stanford CS345A Web Mining* 67 (2011), p. 328. ISSN: 01420615. DOI: 10.1017/CBO9781139058452. arXiv: arXiv:1011.1669v3. URL: http://ebooks.cambridge.org/ref/id/CBO9781139058452.

[62] Marc'Aurelio Ranzato et al. "Sequence Level Training with Recurrent Neural Networks". In: *Iclr* (2016), pp. 1–15. ISSN: 15537358. DOI: 10.1371/journal.pcbi.1005055. arXiv: 1511.06732. URL: http://arxiv.org/abs/1511.06732.

[63] Steffen Rendle et al. "BPR: Bayesian Personalized Ranking from Implicit Feedback". In: *Proceedings of the Twenty-Fifth Conference on Uncertainty in Artificial Intelligence* (2009), pp. 452–461. ISSN: 1469493X. DOI: 10.1145/1772690.1772773. arXiv: 1205.2618. URL: http://dl.acm.org/citation.cfm?id=1795114.1795167.

[64] John a Rice. *Mathematical Statistics and Data Analysis*. Vol. 72. 462. 1995, p. 330. ISBN: 0534399428. DOI: 10.2307/3619963. URL: http://www.jstor.org/stable/3619963?origin=crossref.

[65] Paul Rozin and Edward B. Royzman. "Negativity Bias, Negativity Dominance, and Contagion Paul". In: *Personality and Social Psychology Review* 5.4 (2001), pp. 296–320. ISSN: 1088-8683. DOI: 10.1207/S15327957PSPR0504.

[66] D. E. Rumelhart, G. E. Hinton, and R. J. Williams. "Learning Internal Representations by Error Propagation". In: *Readings in Cognitive Science: A Perspective from Psychology and Artificial Intelligence*. 2013, pp. 399–421. ISBN: 1558600132. DOI: 10.1016/B978-1-4832-1446-7.50035-2. arXiv: arXiv:1011.1669v3.

[67] R Salakhutdinov and A Mnih. "Probabilistic Matrix Factorization." In: *Proc. Advances in Neural Information Processing Systems 20 (NIPS 07)* (2007), pp. 1257–1264. ISSN: 1049-5258. DOI: 10.1145/1390156.1390267. arXiv: 1705.05355. URL: http://discovery.ucl.ac.uk/63248/.

[68] Ruslan Salakhutdinov, Andriy Mnih, and Geoffrey Hinton. "Restricted Boltzmann machines for collaborative filtering". In: *Proceedings of the 24th international conference on Machine learning - ICML '07* (2007), pp. 791–798. DOI: 10.1145/1273496.1273596. arXiv: 1606.07129. URL: http://portal.acm.org/citation.cfm?doid=1273496.1273596.

[69] Badrul Sarwar et al. "Analysis of recommendation algorithms for e-commerce". In: *Organization* 5.1/2 (2000), pp. 158–167. ISSN: 1581132727. DOI: 10.1145/352871.352887. arXiv: 117. URL: http://portal.acm.org/citation.cfm?doid=352871.352887.

[70] Andrew I Schein et al. "Methods and metrics for cold-start recommendations". In: *Proceedings of the 25th annual international ACM SIGIR confer-*

ence on Research and development in information retrieval SIGIR 02 46.Sigir (2002), pp. 253–260. ISSN: 01635840. DOI: 10.1145/564376.564421. URL: http://portal.acm.org/citation.cfm?doid=564376.564421.

[71] M. Schuster and K. K Paliwal. "Bidirectional recurrent neural networks". In: *IEEE Transactions on Signal Processing* 45.11 (1997), pp. 2673–2681. ISSN: 1053-587X. DOI: 10.1109/78.650093. arXiv: arXiv:1011.1669v3. URL: http://ieeexplore.ieee.org/xpls/abs%7B%5C_%7Dall.jsp?arnumber=650093.

[72] Suvash Sedhain et al. "AutoRec : Autoencoders Meet Collaborative Filtering". In: *WWW 2015 Companion: Proceedings of the 24th International Conference on World Wide Web* (2015), pp. 111–112. DOI: 10.1145/2740908.2742726.

[73] Yue Shi et al. "CLiMF: Learning to Maximize Reciprocal Rank with Collaborative Less-is-more Filtering". In: *Proceedings of the Sixth ACM Conference on Recommender Systems* (2012), pp. 139–146. DOI: 10.1145/2365952.2365981. URL: http://doi.acm.org/10.1145/2365952.2365981.

[74] Donghyuk Shin, Suleyman Cetintas, and Kuang Chih Lee. "Recommending tumblr blogs to follow with inductive matrix completion". In: *CEUR Workshop Proceedings*. Vol. 1247. 2014.

[75] Ilya Sutskever, Oriol Vinyals, and Quoc V Le. "Sequence to sequence learning with neural networks". In: *Advances in Neural Information Processing Systems (NIPS)* (2014), pp. 3104–3112. ISSN: 09205691. DOI: 10.1007/s10107-014-0839-0. arXiv: 1409.3215. URL: http://papers.nips.cc/paper/5346-sequence-to-sequence-learning-with-neural.

[76] Yong Kiam Tan, Xinxing Xu, and Yong Liu. "Improved Recurrent Neural Networks for Session-based Recommendations". In: *arXiv* (2016), pp. 0–

5. DOI: 10.1145/2988450.2988452. arXiv: 1606.08117. URL: http://arxiv.org/abs/1606.08117.

[77] Ronald J. Willia. "Simple Statistical Gradient-Following Algorithms for Connectionist Reinforcement Learning". In: *Machine Learning* 8.3 (1992), pp. 229–256. ISSN: 15730565. DOI: 10.1023/A:1022672621406.

[78] Caihua Wu et al. "Recurrent neural network based recommendation for time heterogeneous feedback". In: *Knowledge-Based Systems* 109 (2016), pp. 90–103. ISSN: 09507051. DOI: 10.1016/j.knosys.2016.06.028.

[79] Chao-yuan Wu et al. "Recurrent Recommender Networks". In: *Proceedings of the Tenth ACM International Conference on Web Search and Data Mining - WSDM '17*. 2017, pp. 495–503. ISBN: 9781450346757. DOI: 10.1145/3018661.3018689. URL: http://dl.acm.org/citation.cfm?doid=3018661.3018689.

[80] Kai Yu et al. "Probabilistic Memory-Based Collaborative Filtering". In: *IEEE Transactions on Knowledge and Data Engineering* 16.1 (2004), pp. 56–69. ISSN: 10414347. DOI: 10.1109/TKDE.2004.1264822.