UiO : Department of Mathematics
University of Oslo

Øystein Skauli

Modelling Short Term Changes in User Interest for Online Marketplaces

DS5960 — Master's Thesis

Supervisor: Ida Scheel, Stefan Wender, Simen Eide

2021

Contents

Abstract

This work analyzes the use of hidden Markov model, HMM, based recommendation systems to predict the evolution of user interests. The HMM is combined with ideas from matrix factorization to fit more sparse data and further exploit potential correlation between user interests. The main focus of this work is on analyzing whether the model gives a reasonable modeling of user interests. The model turns out to give surprisingly intuitive results for such a simplistic modeling of user interests. While not the main focus, this system was also tested as a recommendation system. The model was put into production at FINN giving recommendations on the site in real time. The model performed at roughly 75% of the click rate of the best model FINN currently have implemented. Several potential avenues for improvements are discussed, but due to time and software issues have not been tested live on the FINN site.

1 Motivation

With the large amount of products currently available to a customer through online stores and streaming services its no longer possible for a average user to always know of their desired products, or look though all products available to find them. It would be much more desirable if the user was presented with only ads they might not know exists but are interested in. This is the problem recommendation systems try to solve. The goal is to learn a users iterest such that the user can be shown only relevant ads.

A popular online marketplace in Norway is FINN (Schibsted 1996). This site allows user to place ads for properties, cars, jobs and travel as well as for smaller items like furniture, clothes and appliances. With users being able to post ads, not only is the number of ads available large but what ads are available is also changes a lot over time. To get some sense of the scale around 1 million ads can be placed within a month. This situation underlines the need for a recommendation system, no user can be expected to know what ads are available at any given time.

The FINN website gives users recommendations in two main ways. The user can select a category they are interested inn and ads within that category will be shown, ranked according to how much a recommendation system believes the user would like said ad. The user is also given recommendations on any ad for any other ads a recommendation system predicts a user to be interested, given a users previous actions up to and including the ad the user is currently viewing.

Such online marketplaces with a huge amount of possible selections for a user is a relatively new problem. When such marketplaces first arrived the tools available to find ads were relatively simplistic and required some work by the users. These are tools like a search functionality and being able to sort within a category based on specific criteria, like popularity. As the user bases for such sites grew and our ability to collect and process large amount of data bettered it became plausible to create models to predict which ads a user is possibly interested in. However using past data to train such a model presents a problem, user interests are not static. This means older data could be irrelevant to a users current interests, and only serves to skew the resulting recommendations

away from more relevant ads. This is a quite common experience for users where the recommended ads seem stuck on a past interests even when the user feels this should be obviously incorrect based on their recent actions.

The goal with this work is to attempt to model such change in iterest events and generate recommendations using a system that allows for such changes in iterest. A signal for a change in iterest event could also be used in other recommendation systems to remove the influence of older potentially irrelevant ads.

As part of this work a significant amount of code has been written, implementing a recommendation model. The recommendation model has been implemented to run training on the GPU to make it possible to train a model on the large amounts of data available. Furthermore the presented model has be implemented to give recommendations live on the FINN website giving reasonable results.

This report will first give some background information on recommendation systems and the difficulties in creating them. Then some of the current solutions used at FINN are presented along with potential shortcomings. Then the statistical definition of iterest used for our system is presented along with its relevance to hidden Markov model. A previous hidden Markov chain recommender is then presented along with simulation results which show that this model could not effectively be used on FINN. We then present some improvements on their recommender and show results both from offline testing and from running the recommender live on FINN. Finally results in prediction of change in iterest events are given.

2 Background

2.1 Recommendation Systems

Recommendations given by a recommendation system consist of a small number of ads such that its reasonable to expect a user to see and consider all of them. Some systems take into account that only a few ads are show and try to optimize this subset for example by trying to keep the recommendations varied(Zhang and Hurley 2008, Adomavicius and Kwon 2012). It is however much more common to have the goal of a recommendation model to be giving a score to each item. Then usually pick the highest scoring ads as the recommendation This simplifies the problem as each ad can be considered independently at prediction. The recommendation models presented in this work focus on other areas of improvement for recommendation systems, so we will use this simplification and consider the goal of a recommendation system to be generation a ranking or giving a score to all ads.

The goal of a recommendation system is to generate recommendations that promote some desired user behaviour. To generate recommendations a recommender needs to take the past actions of the user recommendations are generated for, and potentially past actions of other users, to to predict a score or ranking over all ads. Therefore this score should be optimized such that the highest scoring ads for a given user are the ads that are most likely to make the user perform the desired action. This could be clicking an ad, purchasing an advertised item, giving a rating and so on.

A common assumption to use data from other users to generate recommendations for a specific user is, if two users have shown interest in the same ads then ads only one user has shown interest in might be of interest to the other user. Models that try to find user interests through such an assumption are called collaborative filters(Goldberg et al. 1992). This assumption is vague and can be included in a model in many, potentially more strict, ways.

The recommendation methods considered in this work are trained offline. Offline training means previous user actions are stored, and models are trained to predict a users interest based on this history. This is in contrast to recommenders who treat recommendation as a bandit problem, dynamically updating recommendations based on user feedback to its current recommendations (Li et al. 2010).

Data used for offline training is usually of ads users have clicked on and/or purchased. Data can also include more detailed information for example whether the user purchased the advertised item or provided any rating of the ad. The goal of the recommender in offline training then to maximize the likelihood or other score function over user actions given the past.

If data for the desired user behaviour is readily available then the making a training scheme can be quite simple. The goal is for the model to give a high score for the ads where the desired user behaviour was observed. The model should be sufficiently restrictive such that maximising the score for ads where the desired behaviour is observed also results in a high score where the desired behaviour was not observed in training data but is likely to happen if the user was presented with the specific ad in the future. For example, if the model can recognize that two ads are similar, if the desired behaviour is only observed on one ad, we would still expect the same user to be interested in the second ad, and both ads should be given a high score. The opposite can also be the case where a user was observed interacting with an ad different to all other ads a user had interacted with. This could indicate that even if the interaction was observed the ad should be given a low score.

2.2 Previous Work

The problem of giving personolized recommendations is an active area of research and new models are presented regularly. Even so, simpler models, even just giving recommendations based purely on popularity, can perform surprisingly similar or even better than much more complicated models (Ludewig and Jannach 2018). This could however change depending on how much information is available on users and ads.

This paper will combine previous work around hidden Markov model, HMM, recommendation model and matrix factorization. The HMM recommendation model which will be presented in chapter 6. The model presented in Sahoo, Singh and Mukhopadhyay 2012 will be used as a reference for such models. Hidden Markov models do however require the probability of a users actions to be calculated. Some further analysis of the problems and potential solutions around fitting such probabilities are given in chapter 5.

Matrix factorization, which be presented in chapter 4.2, became a popular recommendation tool after its performance in the Netflix prize (Funk 2006). These methods are widely explored with many variants and ways of fitting,

regularizing, including data and allowing for behaviour changes over time (Y. Koren, Bell and Volinsky 2009a, Yehuda Koren 2009). How these methods for recommendation relate to the presented recommendation system is covered in chapter 8.

A method for using neural networks to generate recommendations is also covered in chapter 4.1. This is used as example of models FINN currently have implemented that can fit changing user interest. Specifically the model is that presented in Hidasi et al. 2016. This model is one of the better performing neural recommendation methods currently available (Ludewig and Jannach 2018).

2.3 Explicit vs. Implicit Feedback

Recommendation systems usually divide data into two main categories , explicit and implicit feedback. Explicit feedback is data where the user is stating that they in fact are interested in an ad. This could be through for example a 5 star rating system, a thumbs up/down rating or an only positive thumbs up system. If we get such feed back we can know it means the user was/wasn't interested in a specific ad.

Implicit feedback is data where a user interacts with an ad in a way which may indicate interest but not necessarily. This can include clicking on an ad and further interaction with an item listing. Such actions are an indication of interest but not necessarily and could be a result of anything form the user finding the listing funny while having no interests of a purchase to a user clicking it by mistake. Weaker and implicit feedback usually provide much denser data than stronger explicit feedback as it takes less effort from the user to generate it. How useful implicit feedback is and how simply it can be used may depend on the goal of the recommender. A recommendation system just focusing on maximising a click though rate can use data of what users clicked directly maximising the probability of past clicks though some model definition. For a recommendation system that wants to maximize the number of purchases it is no longer necessarily correct to maximize click probability.

2.4 Issues with Creating Recommendation Systems

Due to the fact that recommenders deal with real world, usually massive amounts of data, concerning real people, there are some issues to consider when implementing a recommender.

Firstly different data sources can present quite different problems. Different marketplaces can experience different user behaviour, have a different number of available ads, different ways of including recommendations or different types of data available. Having a high number of ads to choose between can lead to very sparse datasets, this problem is further exaggerated due to the fact that there can be very skewed popularity, meaning there is a small sample of ads that get a majority of all the traffic while a majority of ads will have barely no user interaction at all. There could be different ammount and type of correlation between and within user data due to different user behaviour, and different input data to fit a model.

The goal of a recommendation model might even be different, online marketplaces would usually want to maximize purchases, but other goals may include maximizing clicks or a rating. For these reasons this paper will be limited to specifically optimize a model and test it to a specific use case and not as a general recommender. The fact that different recommendation data can present different challenges can be seen from the fact that the relative performance of recommenders can be quite different between data sets (Ludewig and Jannach 2018).

Secondly many problems encountered when creation a recommendation model are more closely related to psychology than data science. This is a problem as, while a good solution to maximize some goal can be found through data based methods, it to some extent can seem like a problem where it is inappropriate to choose the optimal solution without considering the underlying reason why users respond to a specific solution. It is not at all guaranteed that a solution which maximizes sales is also good for users happiness or satisfaction. It might even be beneficial to appeal to a users negative emotions to encourage a sale. This is not only a problem from a moral standpoint but also for the problem itself as the models are often trained over shorter periods of new data. Over shorter periods the potential benefits of creating a happy satisfied user base might not be seen, while appealing to a users anxieties/insecurities might be a much better solution in the short term.

Making some assumptions on user behaviour is still necessary to create a statistical model so in this paper some assumptions around user behaviour will be used and the strengths and weaknesses of these assumptions will be discussed. Determining whether these assumptions and the model itself is good for users outside of the optimization goal is considered out of scope for this paper but we mention that this is an important consideration for such models.

The problem of privacy also contributes to the complexity of creation a recommendation model. It is clear that the more information one has about users the greater a models possible ability to explain and predict user interests gets. However there are types of data that would not be considered appropriate or illegal to use and collect. The solution is of course to only use data that is appropriate to use, but there are some issues related to this that should still be considered. Firstly from a modeling perspective leaving out such information leads to a lot of unknowns which can make model definition more difficult and having to make a lot of assumptions and approximations. An example of this would be if one wanted data of which ads a user has been shown to a user. The naive solution is that if an ad was loaded on a users page then a user has been seen this ad. This is however not guaranteed as a user might not have looked at the ad. The additional data one would want is eye/cursor tracking to see if the ad was looked at, but many people would consider this too invasive. Without the additional information one is then left with the choice of either assuming that if an ad was show it was seen or allow for the possibility that ads shown were not seen, increasing model complexity.

It also needs to be considered that there is possibly correlation between such sensitive data and user interests, which is exactly the motivation to use it in the first place. The model could therefor indirectly predict sensitive information about the user to generate recommendations. In generating personalized recommendations it is unavoidable for the results not to give some information on the users interests and further the users personality and life situation. There

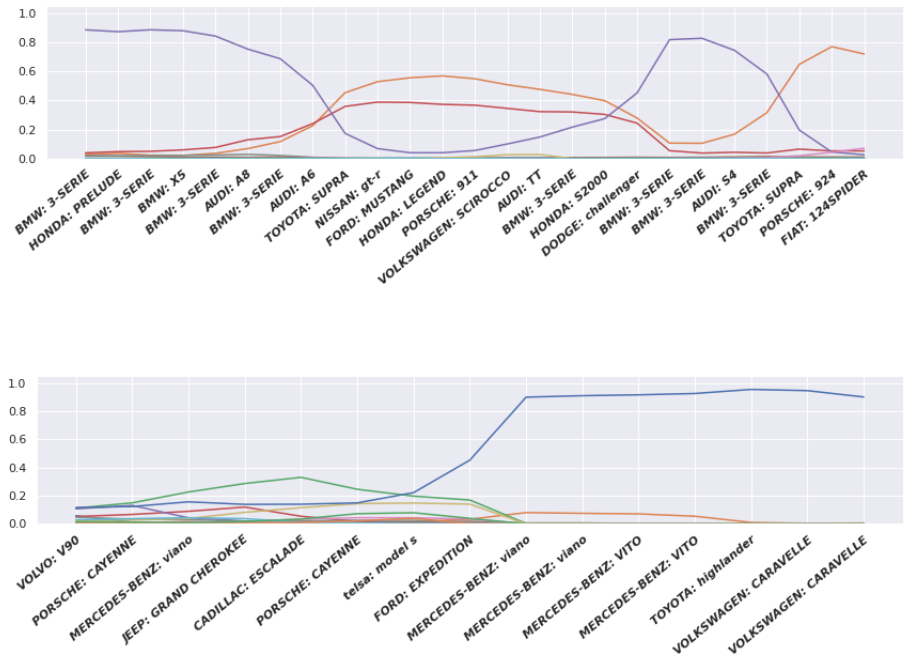


Figure 1: Plots showing the final interest prediction by the model. Each line represents a different interest the model observed in the dataset. The y-axis essentially represents the predicted probability that the user belongs to the specific interest.

are however some areas, when it comes to people, where making decision based on maximizing clicks or purchases would be considered inappropriate.

2.5 Problem Formulation

This work will, again, be mainly focused on the problem of user interests evolving over time. Especially short term interest changes are the main focus in this work. Figure 1 shows some examples of what his work aims to achieve. The model should recognize when the user is within the same interest over time, when a change in interest occurs and if the user has any specific interest at all.

With marketplaces like FINN where data is continually collected, its often only recent data that is used for training. This is especially the case for FINN as new ads are published and old ads are removed continuously. Therefore slow changes in interest are less of a concern as there is no very old data being used for training that could bias the resulting recommendations.

Such short term iterest changes can be caused by several different factors. Some of these factors are unobserved, for example the user could have lost or broken an item they had and now need a replacement. The event that caused a

Observations for torget	
Unique Users	1 997 254
Unique ads	2 232 315
Clicks	32 586 368

Table 1: Number of unique users and ads as well as the number of observations for different event types and clicks observed in a sampled week. Observations are measured across "torget" for one week. Torget is where ads for smaller items that do not belong in any of the larger categories are placed.

Observations for Cars	
Unique Users	1 301 120
Unique ads	189 403
Clicks	18 465 247

Table 2: Number of unique users and ads as well as the number of observations for different event types and clicks. Observations are measured across ads for cars for one week.

change in interest could alternatively be observed if a user looking at an ad is what caused it. Outside events that affect user interest could also be correlated with the ads a user has interacted with in the past. We will treat all of these possibilities the same as no matter the cause the model will have to learn the correlation between interaction with an ad and future iterest.

To start solving the problem of modelling a users interests and its changes one could initially want to define what the user interests are and/or what constitutes as a change in iterest. Defining what is a change of interest is however subjective, especially with smaller changes. One user might be exploring products and another might be looking for specific ads with the same click history these two users could disagree on whether a change of interest occurred at all. This makes creating a dataset where changes in interest have been labeled very difficult as these are several possible solutions with probably varying probabilities. A person could still look at a history over items a user have puchased or looked at and make a good guess as to what they are interested in and where that interest changed.

The ads a user did interact with is however known and can easily be used as data. If one then assume that a good recommendation system also has a good understanding of user interest, which is reasonable as knowing what to recommend a user and knowing what a user is interested in is essentially the same problem, the problems with directly modelling user interest can be bypassed. The problem then becomes defining a model such that the models

understanding of a users interest can be analyzed as a process though time.

3 Data

3.1 User-Ad Interaction Data

FINN stores, among others, events of ads being clicked, publishers of a ad being contacted and ad purchases. For each such event a user id and an ad id for the relevant user and ad is stored in addition to when it occurred, in what context the ad was clicked (whether ad was found in a search result, as a recommendation etc.), ad category, in addition to other information that will not be used.

This data is continuously updated as data is collected form the FINN website, and as ads are purchased or otherwise removed older events become less relevant as the ad referenced in the event is no longer available. This does however not mean that these events are entirely irrelevant. Interactions with an ad that is no longer available could still help recognize correlations between events for ads that are available. Older data is still expected to be less relevant, as the events with available ads become less frequent.

Since the amount of data available is very large, and the decreasing relevance of older data, all models used by FINN are trained on only recent data. How far in the past the cutoff is set depends on the model. Usually a couple of weeks to a few moths of data is used.

The length of user sequences is also given a limit. This is mostly for practical reasons as long sequences can cause problems for batching. Long sequences are also require a large amount of memory for the models presented. Limiting sequence lengths to a few hundred observations is not expected to be a large issue when short term interest changes are the main focus.

Different categories and events have different levels of data sparsity. We can see in Table 1 and Table 3. There is a large difference in average clicks per ad for different categories. Torget is a more difficult situation as there are less observations per ad so the data set is more sparse.

For most of the models presented, it is natural to split the data into separate sets of observations for each user. This set can be ordered as a time series per user and will often be referred to as the user history. So when referring to the next ad in this user history, this refers to the user event that happened after the current event according to the event timestamp. This is convenient as the goal of a recommender is to predict future actions of the user.

The user history can be further divided into sessions. The definition of a session can be fuzzy. In this work a session will be defined as the items a user clicked in a single sitting. once the user leaves their computer/website the session is over. This can be use full as one can make the assumption that a users interest is constant within a session. Some experimentation was done with splitting user history into sessions when more than a set amount of time had elapsed between observations. The FINN data was however not split into sessions when fitting to real data for reason discussed later.

This paper will focus on car data. This data set has fewer ads which is very helpful if one wants to fit models that are slower and has a larger memory usage. This is because with fewer ads, less data is needed to get enough data per ad

to a reasonable fit. Since we want to explore how a users interests changes over time it would also not necessarily work well to only collect data on a subset of ads. Then events which led a user form one ad to another will be lost and it could become more difficult to find correlation between ads as a time series. It therefore did not seem appropriate to exclude a large portion of ads.

There are however users an ads that are included in none to only a handfull of events. Giving recommendations to such ads and users are known as the cold start problem (Schein et al. 2002). With collaborative filters giving recommendations to such users and especially of such ads is difficult. When none or very little data is available on an ad or user there are few samples to compare and ad or users similarity to other ads or users. Excluding ads with few observations is expected to have less of an impact on the models ability to predict future events. By only excluding ads with low popularity less observations are removed from the data set.

FINN has implemented data based recommendation models as well as a search functionality which do show ads to users, independent of previous clicks to some extent. Further improvement of such systems is considered a separate problem and will mostly not be considered in this work. Ads and users with few observations are therefore ignored.

3.2 Ad Data

Data for each ad can also be accessed. Information such as category, title, postal code where ad was placed, description, and other ad specific information. For cars one has manufacturer, production year, miles, model and fuel type. Some of this information is filled by users, for example car model, and therefore requires further processing to be useful, this goes for ad title and description and images as well. Such information is not used here. However with the advent of neural methods, models for processing text and images are readily available, especially if gradients are available from the training method and loss/score function used.

Some users fill information such as car manufacturer manually which leads to some strange entries. However as long as all text is set to use the same case information in the vast majority of ads can easily be recognized. All data, even numerical, such as production year is split into categorical variables. This is because, with users being able to input such numbers manually some users may input only the correct decade some the exact year and some the wrong decade or even century all together.

The data included for car ads in the production year, split into decades between 1900s to the 2020s, the make, fuel type and body type. We do not discuss the effect of including such data too much as the effects are very limited. This is a common result for collaborative filtering models trained at FINN.

To load data for both events and ads one has to choose how far in the past one wants to gather data from. Data for events is sorted after when the event occurred and data for ads is sorted after when the ad was published. Due to restrictions in both computation time and memory not all ads can be loaded. This can lead to and ad referenced in an event not being loaded in the data-set for ads. These events are however often involve a user looking at their own older published ads and are therefore less important. The events referencing

Observations for Cars	
Unique Users	1 101 917
Unique ads	129 852
Clicks	64 338 371

Table 3: Total number of observations ads and user sequences in the offline training set.

ads not available in the loaded part of the ad data-set were therefore ignored when training.

3.3 Offline data

All models except those mentioned in chapter 7 and 9.1 are trained on the same offline data set. The data set what gathered from 35 days of click data. Only items with over 100 observations and users over 10 were included. User data was limited to 200 observations and users with longer sequences of observations had their data split into several sequences. The sequences split form the same user and a sequence from a different user were treated the same. This can be somewhat problematic as there likely is some correlation between sequences from the same user. However the vast majority of sequences were below 200 observations so these potential effects were ignored.

A random 20% of the sequences in the full data set were selected at random to be used as a test set to give results completely independent of the parameter estimation process. Another 10% of the sequences were used to create a validation set. The validation set was used for hyperparameter optimization.

4 Current Models Running at FINN

FINN uses several model to generate recommendations, this includes both image matching algorithms and neural networks. This section will discuss two of the methods used to generate user recommendations at FINN. This first method is a recurrent neural network, RNN, recommendation system which shows another way of handling changing iterest. The second method is a matrix factorization method, which is quite popular in collaborative filtering. In the following sections these methods are briefly discussed.

4.1 RNN Recomender

The RNN recommender FINN uses is based on the model described in Hidasi et al. 2016. This model uses an RNN to predict the next ad in a session based on the previous ads. While a standard RNN usually struggles with the problem of vanishing gradients this recommeder uses a gated recurrent unit (Cho et al. 2014) to mitigate this problem. It introduces an update gate, z_t, and a reset gate, r_t, to the basic RNN. The equations for updating the RNN then are defined as,

$$z_t = \sigma(W_z x_t + U_z h_{t-1})$$

$$r_t = \sigma(W_r x_t + U_r h_{t-1})$$
$$\hat{h}_t = \tanh(W x_t + U(r_t \odot h_{t-1}))$$
$$h_i = (1 - z_t)h_{t-1} + z_t \hat{h}_t.$$

Here h_t is the hidden state, $\hat{h}_t$ is the proposed hidden state, and x_t is the input for the t'th ad clicked in a session. the initial state, h_0 is set to zeros. The matrices, W_z, W_r, W, U_z, U_r, U are parameters to be optimised. We write the sigmoid function as σ and element-wise product as $\odot$. Since h_{t-1} is the previous hidden state we see that by introducing z_t and r_t the network can select which part of the previous and proposed state to keep in the current state. This can help the model store information from further back in the session thereby making more informed decisions.

The input x_t is an embedding of t'th ad clicked. With N ads and and embedding dimension d, we can note this as a matrix X of size $d \times N$ that transforms one-hot encoded vectors, y_i, of size N to x_t, $x_t = Xy_t$. Essentially we are trying to learn a dense representation of each ad that carries some information about what the ad is so that similar ads have similar embeddings. The matrix X can be pre-trained using a word2vec model (Mikolov et al. 2013) though this has little impact on performance according to FINN.

The output is generated by a feed forward neural network that takes the current state, h_t, as input and transforms it to a vector of size N. This final vector represents the score for each ad given the sequence of ads up to t. The ads with the highest score are recommended to the user, though FINN limits the recommendations to ads from the last 1-2 weeks and excludes any ads the user has seen before.

For parameter optimization the authors of the recommender introduce a ranking loss, where negative examples are sampled according to popularity. Though other loss functions can also be used. This reduces computational cost as negative samples can be sampled from other batches. They also argue that the user is more likely to know about the popular ads and therefore we can have higher confidence that a user does not currently want those ads. With a sample of scores of other ads, $(\hat{r}_1, \ldots, \hat{r}_{N_s})$, and score for selected ad $\hat{r}_s$, their ranking loss is defined as,

$$L_s = \frac{1}{N_s} \sum_{t=1}^{N_s} \sigma(\hat{r}_t - \hat{r}_s) + \sigma(\hat{r}_t^2)$$

The RNN clearly allows for fitting changing interests. Especially with the forget and update gate, the model can decide what will influence future predictions based on the current input. However interpreting neural networks is notoriously hard, therefore figuring out when iterest changes occur or even understanding to what extent the model is actually recognising a change in interest can be difficult. This makes the RNN recommender less relevant to our problem

4.2 Matrix Factorization

The matrix factorization method used by FINN (Spark 2021) is static and does not adapt to changing user interests. There are heuristics to mitigate this problem (Yehuda Koren 2009, Y. Koren, Bell and Volinsky 2009a) but they

are not considered here. Instead of using the data as a time series this matrix factorization method uses all past events to generate a score for each ad for every user. What score is given for each type of ad interaction has to be set before training. Typically a user purchasing an ad is given a higher score than a user just clicking on an ad and so on. For such implicit feedback data the score should essentially represent both our confidence that the user likes the ad and how much the user likes the ad which can make the score a bit difficult to interpret.

These scores the fill a, typically sparse, matrix where a lot of the entries are 0 as the user has not interacted with the ad. If there are N ads and K users, these scores fill a $K \times N$, Y. Entry $y_{k,i}$, k'th row and i'th column of Y represents the score of ad i for user k. If user k clicked on ad i this would be a lower score than if user i purchased ad i and if the user did not interact with ad k ad all $y_{k,i}$ would be 0. This Y matrix is used as input for the matrix factorization model.

Matrix factorization works by approximating the input matrix with a lower rank matrix. This lower rank matrix W can be represented as the product of a matrix U of dimension $K \times m$ and a matrix O of dimension $N \times m$.

$$W = U \times O^T$$

The value m is will be termed the embedding dimension. The matrices U and O are found by minimizing the distance between W and Y under some measure. The method used at fin minimizes the residual sum of squares so, writing u_k and o_i as the rows of S and B respectively,

$$\sum_k^K, \sum_i^N (y_{k,i} - w_{k,i})^2 = \sum_k^K, \sum_i^N (y_{k,i} - u_k \cdot o_i)^2$$

is subject to minimization.

The method used at FINN uses alternating least squares (Y. Koren, Bell and Volinsky 2009b; Y. Koren, Bell and Volinsky 2009a) to solve this problem. This essentially involves solving for the optimal value of one of U and O keeping the other matrix static. This process alternates between which matrix is solved and which is kept static until convergence.

One helpful interpretation of such a model is that we create an embedding, or latent factors, for each ad $\boldsymbol{o}_i$ and user, $\boldsymbol{u}_k$ and the interest score is represented by a function of these embeddings. In the simple case presented this is simply a dot product.

5 Definition of User Interest

5.1 Overview

To define models exploring the evolution of interest over time a definition of user interest is needed. The definition used for our model will be the same as in Sahoo, Singh and Mukhopadhyay 2012. However a discussion around the assumptions this definition implicitly makes is also given. In addition why the assumptions are probably incorrect but also useful simplifications of reality. To give a cleaner definition of interest it is assumed that users have been separated

into groups that have the same interests and we will consider modelling the interest of one of these groups.

The goal for this paper is to model user interests and not to maximize any specific action. The densest available data to indicate user interests was therefore chosen, namely clicks on ads. While such data can be supplemented with other inputs like purchases and so on, one would then need to define the relation between a users interests and all of the inputs taken in addition to how the different inputs relate to each other.

Many recommendation systems, like the one mentioned in chapter 4.2, create this relation by giving each interaction a score based on a weighted sum of all the inputs. Again, the score should essentially represent both our confidence that the user likes the ad and how much the user likes the ad. While this could improve recommendations, the goal of this paper is to model and recognize user interests. Since interpreting a model on top of such a score can quickly become more difficult, the presented recommender only uses click data.

As defined in Sahoo, Singh and Mukhopadhyay 2012 there is assumed to be a discrete amount of interests that a user can have. An interest consists of a probability distribution over all ads for which a user will click on next. Initially we just define a unique selection probability per ad. Writing ads clicked by the user as the series $(c_1, c_2, \ldots, c_t)$ and user interests as the series $(u_1, u_2, \ldots, u_t)$, the k'th interest over n ads is defined as the probability vector $(v_{1,k}, v_{2,k}, \ldots, v_{n,k})$. This interest defines the probability,

$$P(c_t = i | u_t = k) = v_{i,k}$$

So given the user belongs to interest k, the user has probability $v_{i,k}$ of selecting ad i.

5.2 Assumptions

This definition firstly assumes that given a user has the same interests the probabilities that a user clicks an ad remain constant. This is a reasonable assumption alone, but we also assume there to be a discrete set of interests. A somewhat low number of interests will be used for the model presented due to practical limitations. With such a discrete set of interests it becomes less reasonable to expect there to be no variation between users who are modeled to belong to the same iterest. An easy example to see where such an assumption is incorrect is that the probability that a user clicks an ad remains the same after a user has clicked on it, if the users general interest remains the same. It is more reasonable to expect this probability to ether significantly decrease as the user realizes they are not interested in that specific ad or increase as the user sees it as a potential purchase.

Therefore it is potentially more correct to view these discrete interests as the marginal distribution over many/a continuous distribution of similar interest. How a users interest evolves through theses similar interests will then not be captured by a model using discrete interests. This could potentially be captured by a model using a more complex interest definition. However limiting the model to a finite set of interest can also be a useful restriction exactly because it requires creating these marginal distributions, essentially generating a clustering

of what is potentially a much more complex space. Interpretation of the model and analysis for the evolution of interests can then become much simpler.

Such a definition also implicitly assumes that the user is presented with, and is selecting from, all ads. When users are only presented with a subset, the selection probabilities do not sum to one and user behaviour is undefined. A simple assumption to make to solve this issue is that user ad selection is independent what ads are being shown to the user. If the user click, sampled from interest probability vector, is not in the set of ads shown to the user, then the user wont click anything. In this case no observation is made. This is equivalent to if the shown ads sum to v_s then the user has a probability $1 - v_s$ of selecting nothing at all. And again these events are not observed as we only consider click data.

It should also be noted that it is somewhat naive to assume users probability of selecting an ad is independent of the fact that it is shown. An alternative is that when a user is presented with a subset of ads they click according to a transform of the click probabilities such that they sum to one over the shown ads. For example users click according to normalizing shown ad click probabilities.

How exactly a user behaviour is affected by what is recommended could be considered more of a a psychology question than statistics and is considered out of scope here. True user behaviour is most likely a mixture of the two possibilities. When only a subset is shown the user will have a higher probability of clicking nothing at all while also having a higher probability of clicking the shown items.

5.3 Observed Click Probability vs. Interest

The probability of an ad appearing in the subset of ads for the user to select at time t will be noted as $s_{i,t}$, these do not sum to one over ads as the number of presented ads is greater than one. Given the assumption of independence mentioned above the observed selection probabilities are in fact,

$$P(c_t = i | u_t = k) = \frac{v_{i,k} * s_{i,t}}{\sum_{j=1}^{n} v_{j,k} * s_{j,t}} \tag{1}$$

Since a previous recommender placed ads in the subset a user selects from (ignoring use of search and other similar functionality), the probabilities that an ad would be in the set of ads a user is presented with is neither constant over time or ads. This because previous recommenders have been retrained during data collection or adapt to user data. Furthermore these probabilities are likely also highly correlated with ad selection probabilities as previous recommenders will likely recognize similar user interests. If the previous recommenders are unavailable or are not easily interpretable finding $s_{i,t}$ can be difficult.

There are methods to minimize the dependence on the previous recommender numerically, using the data over all ads presented to the user, for example the previously mentioned ranking loss. However methods that involve processing the ads not clicked by the user as well are expected to be slower, since the methods presented here are quite slow to begin with using recommended ads to estimate the true ad selection probabilities is not considered here.

However, while using data for all recommended ads, not just clicked ads, during training is expected to be too slow, one can estimate the true selection

probabilities after training. As mentioned above if we train on click data directly, the model estimates the probabilities given in (1). If we assume $s_{i,t}$ to be constant over time, which again is not expected to be entirely correct, then these probabilities can be directly estimated from data. One just needs to calculate how often an ad appears in a users recommendations. The true selection probabilities can then also easily be estimated,

$$v_{i,k} \propto \frac{P(c_t = i | u_t = k)}{s_{i,t}} \tag{2}$$

This estimator will have a high variance for ads rarely show to a user. So some care has to be taken when estimating the true selection probabilities. For example using the lower bound of a confidence interval.

Estimating the true selection probabilities using this method will however treat each iterest equally. It is therefore less important to better our predictions of user interest. It would be more relevant for giving better recommendations or recommending less popular items. In our case it underlines what probabilities the model actually fits. The probabilities fit by the model is not necessarily representative of the actual click rate of an interest but can be skewed by previous recommendations.

6 HMM Recommender

6.1 Introduction

Hidden Markov models, HMM, model a sequence of observations whose distribution depend on some unobserved state of the system. The focus here will be on a discrete HMM with a finite number of states. This means the system has discrete steps between each observation. For each observation the system belongs to a specific state which can change from one step to another. The distribution of the observations depend on this unobserved state.

6.2 HMM

The observations used for this work are sequences over time. We note these sequences as $(X_1, X_2, \ldots, X_{s_u})$. Here X_t is a random variable for which item(s) a user clicked at time t, and s_u is the sequence length for user u. The hidden state at time t is noted as Z_t and form a parallel sequence to the observations. We again assume there to be a finite number of states. The variable K will be used to represent the number of states. If the state at time t is k we write $z_t = k$.

Hidden Markov models follow the dependence structure show in figure 2. The hidden states are assumed to have the Markov property meaning, given Z_{t-1}, the state at Z_t is independent of all previous states. Furthermore given Z_t the observation X_t is independent of all previous observations, and states.

Transitions between hidden states are defined by the probabilities,

$$P(Z_{t+1} = k | Z_t = l), \quad k, l \in [1, \ldots, K].$$

For finite sequences an initial state distribution also needs to be defined, $P(Z_0 = k)$. The distributions of the observed variable given the hidden state is referred

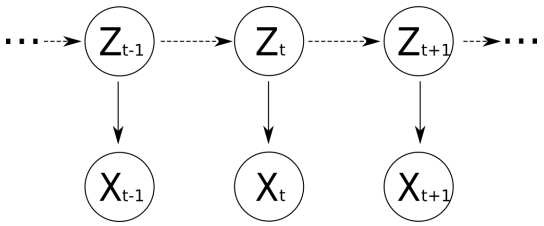


Figure 2: Dependence structure in a Hidden Markov Model. The unobserved hidden states are labeled Z and the observed events are labeled X. Dashed arrows represent dependence according to a Markov property. Solid arrows represent dependence trough the defined emission distribution

to as the emission distribution. The emission distribution can be continuous but will be assumed to be a discrete distribution that can take a finite number of values. This emission distribution then defines $P(X_t = i|Z_t = k), i \in [1, \ldots, N]$, where the emission distribution can take N possible values.

6.3 Recommendation Model

From the definition of a HMM, we can see that the interest definition given in chapter 5 fits quite well as an emission distribution. This definition describes the user behaviour given that we know a user to belong to a specific interest. The HMM allows the use of such a definition by modeling the unobserved states, which in this case is which interest a user currently has.

By using the definition of interest presented in chapter 5 in an HMM one gets a similar model to that presented in Sahoo, Singh and Mukhopadhyay 2012. User interests are represented by K probability vectors which contain the observed selection probabilities (1) for each state. Each user belongs to one state, the state of a user determines which of the K probability vectors the observations from the user currently follow. The hidden Markov chain describes how the users move/transition between these states.

Again, in a hidden Markov model the hidden states are assumed to have the Markov property. So transitions to the next hidden state only depend on the current hidden state. This assumption is restrictive. One could easily describe some reasonable user behaviour where the Markov property is not valid. Though user interest is influenced by a lot more outside factors, like items getting lost or broken, than just previous interest. So while this assumption is restrictive it might also help with not over fitting such outside influences to the many possible permutations of previous states.

In this simple HMM recommendation model, unique probabilities are fit to each of the, initial state probabilities, $P(Z_1 = k) = \pi_k$, transition probabilities, $P(Z_{t+1} = k|P(Z_t = l) = A_{k,l}$, and emission probabilities, $P(X_t = i|Z_t = k) = v_{k,i}$.

The matrix A is often referred to as a transition matrix giving probabilities of moving between states. The vector π is the distribution of starting states. With N possible ads this gives model parameters as shown in table 4.

It should also be mentioned that when predicting recommendations these states need to be estimated from the observed data. Therefore probabilities

Prameter	Size	Description
$\boldsymbol{\pi}$	$1 \times K$	Starting probabilities
$\boldsymbol{A}$	$K \times K$	Transition probabilities
$\boldsymbol{V}$	$K \times N$	Selection probabilities for each state

Table 4: Parameters in simple HMM model

for the next observation can depend on many previous observations selections, as the probabilities of the current hidden states can depend on many previous observations. This is only to say that while the Markov assumption would too strict on the item level, moving this assumption to a hidden state relaxes it, while we still are able to observe what state the model believes a user belongs to.

Of course there is the extreme case where each hidden state describes the selection of one item, having probability zero for all other items. This means there is no longer any uncertainty in the hidden states and we have the Markov property on the item level as well. This situation should be avoided through regularization as we only want low probabilities of selecting an item if this is significant from the data.

The model presented in Sahoo, Singh and Mukhopadhyay 2012 worked on a less granular timescale considering each observation to be the selections made by a user over a month. This means each observation potentially consists of several click events. From a modelling perspective having multiple or single click events in an observation is quite similar. Both cases represent draws from a multinomial distribution with probabilities given by $\boldsymbol{V}$, just with a different number of trails.

While the data used here allows for a single click event per observation, model definitions and equations will be given to allow for multiple click events per observation. In Sahoo, Singh and Mukhopadhyay 2012 the number of click events was also modeled, however here the number of clicks will be assumed to be independent of user state and therefor ignored.

6.4 Optimization

Optimization is done through the EM algorithm (Dempster, Laird and Rubin 1977). This is a method in which a local maximum of the likelihood is reached by iteratively updating the parameters. The parameter update comes from maximising the function,

$$Q(\theta|\theta^{(m)}) = E(l(\theta)|\theta^{(m)}),$$

which is the expected log-likelihood given previous parameter estimates, $\theta^{(m)} = (\boldsymbol{\pi}^{(m)}, \boldsymbol{A}^{(m)}, \boldsymbol{V}^{(m)})$, and observed data over the hidden states.

The likelihood for this model is additive over users. To avoid cluttered indexes we first derive $Q_u(\theta|\theta^{(m)})$ for each user where,

$$Q(\theta|\theta^{(m)}) = \sum_{u=1}^{U} Q_u(\theta|\theta^{(m)}),$$

for U users. To find this function we start with the complete log-likelihood for one user, u, with s_u sessions,

$$\begin{aligned} l(\theta)_u =& \log(P(Z_1 = z_1)) + \sum_{i=1}^{N} x_{1,i} \log(P(X_1 = i | Z_1 = z_1)) \\ &+ \sum_{t=2}^{s_u} \log(P(Z_t = z_t | Z_{t-1} = z_{t-1})) + \sum_{i=1}^{N} x_{t,i} \log(P(X_t = i | Z_t = z_t)). \end{aligned} \tag{3}$$

Note each user has unique observations $x_{t,i}$ the index is ignored for clarity. Here, X_t is a random variable for selecting items at each session, t. The number of items i observed in session t is $x_{t,i}$. This will usually be 1 or 0. Lastly, z_t is the hidden state for session t. We can introduce indicator variables to (3) indicating which hidden state each observation comes from,

$$\begin{aligned} l(\theta)_u =& \sum_{k=1}^{K} \mathbb{1}(z_1 = k)[\log(P(Z_1 = k)) + \sum_{i=1}^{N} x_{1,i} \log(P(X_1 = i | Z_1 = k))] \\ &+ \sum_{t=2}^{s_u} \sum_{k=1}^{K} \sum_{l=1}^{K} \mathbb{1}(z_t = k, z_{t-1} = l) \Big[\log(P(Z_t = k | Z_{t-1} = l)) \\ &+ \sum_{i=1}^{N} x_{t,i} \log(P(X_t = i | Z_t = k)) \Big]. \end{aligned}$$

We recognize, $P(Z_1 = k) = \pi_k$, as the initial state probabilities, $P(Z_t = k | Z_{t-1} = l) = A_{k,l}$, as the transition probabilities and, $P(X_t = i | Z_t = k)) = v_{k,x_t}$, as the multinomial selection probability from the model definition. The likelihood is then,

$$\begin{aligned} l(\theta)_u =& \sum_{k=1}^{K} \mathbb{1}(z_1 = k)[\log(\pi_k) + \sum_{i=1}^{N} x_{1,i} \log(v_{k,i})] \\ &+ \sum_{t=2}^{s_u} \sum_{k=1}^{K} \sum_{l=1}^{K} \mathbb{1}(z_t = k, z_{t-1} = l)[\log(A_{k,l}) + \sum_{i=1}^{N} x_{t,i} \log(v_{k,i})]. \end{aligned} \tag{4}$$

Taking expectation of (4) gives,

$$\begin{aligned} Q_u(\theta | \theta^{(m)}) =& \sum_{k=1}^{K} Pr(Z_1 = k | \boldsymbol{x}, \theta^{(m)})[\log(\pi_k) + \sum_{i=1}^{N} x_{1,i} \log(v_{k,i})] \\ &+ \sum_{t=2}^{s_u} \sum_{k=1}^{K} \sum_{l=1}^{K} Pr(Z_t = k, Z_{t-1} = l | \boldsymbol{x}, \theta^{(m)})[\log(A_{k,l}) + \sum_{i=1}^{N} x_{t,i} \log(v_{k,i})]. \end{aligned} \tag{5}$$

For update equations we need the probabilities $Pr(Z_t = k | \boldsymbol{x}, \theta^{(m)})$ and $Pr(Z_t = k, Z_{t-1} = j | \boldsymbol{x}, \theta^{(m)})$, which can be found by a forward and backward

pass trough the data. For ease of notation and reading we write,

$$q_{t|j}(k) = Pr(Z_t = k|\boldsymbol{x}_{1:j} = (\boldsymbol{x}_1, \ldots, \boldsymbol{x}_j), \theta^{(m)})$$
$$f_k(\boldsymbol{x}_t; \theta) = Pr(\boldsymbol{x}_t|Z_t = k, \theta^{(m)}) = \frac{(\sum_{i=1}^N x_{t,i})!}{\prod_{i=1}^N x_{t,i}!} \prod_{i=1}^N (v_{k,i}^{(m)})^{x_{t,i}}$$

Note the normalizing factor of the multinomial distribution never needs to be calculated as it cancels out when normalizing the $q_{t|j}(k)$ values. Since we take the previous parameter estimates, $\theta^{(m)}$ as given, we can find $q_{1|1}(k)$ with the initial state probabilities, $\pi_k^{(m)}$,

$$q_{1|1}(k) = \frac{\pi_k^{(m)} f_k(\boldsymbol{x}_1; \theta)}{\sum_l^K \pi_l^{(m)} f_l(\boldsymbol{x}_1; \theta)} \tag{6}$$

The previous transition probability estimates are also given, $A_{k,l}^{(m)}$, therefore we can also find $q_{2|1}(k)$ trough,

$$\begin{aligned} q_{2|1}(k) &= \sum_{l=1}^K Pr(Z_2 = k|Z_1 = l, \boldsymbol{x}_1) Pr(Z_1 = l|\boldsymbol{x}_1) \\ &= \sum_{l=1}^K A_{k,l}^{(m)} q_{1|1}(l), \end{aligned} \tag{7}$$

and $q_{2|2}(k)$,

$$q_{2|2}(k) = \frac{Pr(Z_2 = k|\boldsymbol{x}_1) P(x_2|Z_2 = k)}{P(\boldsymbol{x}_2|\boldsymbol{x}_1)} \propto q_{2|1}(k) f_k(\boldsymbol{x}_1; \theta), \tag{8}$$

of course $\sum_k q_{t|t}(k) = 1$. In fact we see that (7, 8) can be used to update any $q_{t|t}(k)$ to $q_{t+1|t+1}(k)$. So with $q_{1|1}(k)$ we can find $q_{t|t}(k)$ for $t \in (1, \ldots, s_u)$.

What we wanted was $q_{t|s_u}(k)$, i.e. the probability of $Z_t = k$ given the whole session, not just up to t. With (7, 8) we can find $q_{s_u|s_u}(k)$ so we only need backward updates to find, $q_{t|s_u}(k)$ from $q_{t+1|s_u}(k)$,

$$\begin{aligned} q_{t|s_u}(k) &= P(Z_t = k|\boldsymbol{x}_{1:s_u}) \\ &= \sum_{l=1}^K P(Z_t = k|Z_{t+1} = l, \boldsymbol{x}_{1:t}) P(Z_{t+1} = l|\boldsymbol{x}_{1:s_u}) \\ &= \sum_{l=1}^K \frac{P(Z_{t+1} = l|Z_t = k, x_{1:t}) P(Z_t = k|x_{1:t})}{P(Z_{t+1} = l|x_{1:s_u})} q_{t+1|s_u}(l) \\ &= \sum_{l=1}^K \frac{A_{l,k}^{(m)} q_{t|t}(k)}{q_{t+1|t}(l)} q_{t+1|s_u}(l). \end{aligned}$$

We also need $Pr(Z_t = k, Z_{t-1} = l|\boldsymbol{x}_{1:s_u}, \theta^{(m)})$,

$$\begin{aligned}
Pr(Z_t = k, Z_{t-1} = l|\boldsymbol{x}_{1:s_u}, \theta^{(m)}) &= Pr(Z_{t-1} = l|Z_t = k, \boldsymbol{x}_{1:t-1}, \theta^{(m)}) q_{t|s_u}(k) \\
&= \frac{Pr(Z_t = k|Z_{t-1} = l, \boldsymbol{x}_{1:t-1}) Pr(Z_{t-1} = l|\boldsymbol{x}_{1:t-1})}{Pr(Z_t = k|\boldsymbol{x}_{1:t-1})} q_{t|s_u}(k) \\
&= \frac{q_{t-1|t-1}(l) A^{(m)}_{k,l}}{q_{t|t-1}(k)} q_{t|s_u}(k)
\end{aligned}$$

The steps required to find $Q_u(\theta|\theta^{(m)})$ becomes,

$$q_{1|1}(k) = \frac{\pi^{(m)}_k f_k(x_1; \theta)}{\sum_l \pi^{(m)}_l f_l(x_1; \theta)} \tag{9}$$

$$q_{t+1|t}(k) = \sum_{l=1}^{K} A^{(m)}_{k,l} q_{t|t}(l) \tag{10}$$

$$q_{t+1|t+1}(k) \propto q_{t+1|t}(k) f_k(x_t; \theta) \tag{11}$$

$$q_{t|s_u}(k) = \sum_{l=1}^{K} \frac{A^{(m)}_{l,k} q_{t|t}(k)}{q_{t+1|t}(l)} q_{t+1|s_u}(l) \tag{12}$$

$$\hat{\pi}^u_k = q_{1|s_u}(k) \tag{13}$$

$$\hat{A}^u_{l,k} = Pr(Z_t = k, Z_{t-1} = l|x_{1:s_u}, \theta^{(m)}) = \frac{q_{t-1|t-1}(l) A^{(m)}_{k,l}}{q_{t|t-1}(k)} q_{t|s_u}(k) \tag{14}$$

The process is initialized by (9) and then iterates through the session with (10) and (11) to obtain $q_{s_u|s_u}(k)$. With $q_{s_u|s_u}(k)$ we can start iterating backward through the session with (12) to obtain $q_{t|s_u}(k) = Pr(Z_t = k|x_{1:s_u}, \theta^{(m)})$. We can find the expected initial state probabilities and transition probabilities for a user session with (13) and (14). We assume user sessions to be independent so this can be done independently for each user session. The complete $Q(\theta|\theta^{(m)})$ for all data is then,

$$Q(\theta|\theta^{(m)}) = \sum_{u=1}^{U} Q_u(\theta|\theta^{(m)})$$

With $Q(\theta|\theta^{(m)})$ we want to maximize for the model parameters, however Sahoo, Singh and Mukhopadhyay 2012 found that maximum likelihood estimators (MLE) was not appropriate in this situation. We can for example easily assign 0 probability for a transition or selection of an item if the transition or selection is not observed in the data. We want the model to be more restrictive so instead use maximum a posteriori (MAP), assigning a dirichlet prior to model parameters,

$$\boldsymbol{\pi}, A_{k,*} \sim \text{Dir}(\boldsymbol{x}; \boldsymbol{\alpha}_k), \boldsymbol{\alpha}_k = \{\alpha_{k,l}\}_{1:K}$$

$$v_{k,*} \sim \text{Dir}(\boldsymbol{x}; \boldsymbol{\beta}_k), \boldsymbol{\beta}_k = \{\beta_{k,i}\}_{1:N}$$

where $A_{k,*}$ is the k'th row of $\boldsymbol{A}$ and $v_{k,*} = (v_{k,1}, \ldots, v_{k,N})$ is the k'th row of $\boldsymbol{V}$. The MAP estimates for $\boldsymbol{\pi}$ and $\boldsymbol{A}$ are given as normal with HMM with the

added dirichlet prior,

$$\pi_k^{(m+1)} = \frac{\big[\sum_u (q_{1|s_u}(k))\big] + \alpha_{k,0} - 1}{\big[\sum_u \sum_k (q_{1|s_u}(k))\big] + (\sum_k \alpha_{k,0}) - K} \tag{15}$$

$$A_{k,l}^{(m+1)} = \frac{\big[\sum_u \sum_t Pr(Z_t = k, Z_{t-1} = l | x_{1:s_u}, \theta^{(m)})\big] + \alpha_{k,l} - 1}{\big[\sum_u \sum_t \sum_l Pr(Z_t = k, Z_{t-1} = l | x_{1:s_u}, \theta^{(m)})\big] + (\sum_k \alpha_{k,l}) - K} \tag{16}$$

To find estimators for $v_{k,i}$ we maximise $Q(\theta|\theta^{(m)})$. We have that $\sum_i^N v_{k,i} = 1$ since these are probabilities for the multinomial distribution, so a lagrange multiplier is added to maximize within this constraint.

$$\begin{aligned} Q(\theta|\theta^{(m)}) = &\sum_{u=1}^{U} \Bigg[\sum_{k=1}^{K} \hat{\pi}_k^u [\log(\pi_k) + \sum_{i=1}^{N} x_{1,i} \log(v_{k,i})] \\ &+ \sum_{t=2}^{s_u} \sum_{k=1}^{K} \Big[\sum_{l=1}^{K} \hat{A}_{l,k}^u [\log(A_{k,l}) + \sum_{i=1}^{N} x_{t,i} \log(v_{k,i})] \Big] \Bigg] \\ &- \lambda \sum_{k=1}^{K} \big(\sum_{i=1}^{N} v_{k,i} - 1 \big) + \sum_{k=1}^{K} \sum_{i=1}^{N} (\alpha_{k,i} - 1) \log(v_{k,i}). \end{aligned}$$

Taking derivative w.r.t $v_{k,i}$,

$$\begin{aligned} \frac{\delta Q(\theta|\theta^{(m)})}{\delta v_{k,i}} = &\sum_{u=1}^{U} \Bigg[x_{1,i} \hat{\pi}_k^u \frac{1}{v_{k,i}} + \sum_{t=2}^{s_u} x_{t,i} \frac{1}{v_{k,i}} \Big[\sum_{l=1}^{K} \hat{A}_{l,k}^u \Big] \Bigg] \\ &- \lambda + (\alpha_{k,i} - 1) \frac{1}{v_{k,i}}. \end{aligned}$$

Since,

$$\sum_{l=1}^{K} \hat{A}_{l,k}^u = \sum_{l=1}^{K} Pr(Z_t = k, Z_{t-1} = l | x_{1:s_u}, \theta^{(m)}) = Pr(Z_t = k | x_{1:s_u}, \theta^{(m)}),$$

this simplifies to,

$$\begin{aligned} \frac{\delta Q(\theta|\theta^{(m)})}{\delta v_{k,i}} &= \sum_{u=1}^{U} \Bigg[x_{1,i} q_{1|s_u}(k) \frac{1}{v_{k,i}} + \sum_{t=2}^{s_u} x_{t,i} q_{t|s_u}(k) \frac{1}{v_{k,i}} \Bigg] - \lambda + (\alpha_{k,i} - 1) \frac{1}{v_{k,i}} \\ &= \frac{1}{v_{k,i}} \sum_{u=1}^{U} \Bigg[\sum_{t=1}^{s_u} x_{t,i} q_{t|s_u}(k) \Bigg] - \lambda + (\alpha_{k,i} - 1) \frac{1}{v_{k,i}}. \end{aligned}$$

Taking derivative w.r.t λ gives,

$$\frac{\delta Q(\theta|\theta^{(m)})}{\delta \lambda} = \sum_{k=1}^{K} \big(\sum_{j=1}^{N} v_{k,j} - 1 \big)$$

Solve for 0,

$$\sum_{j=1}^{N} v_{k,j} - 1 = 0 \tag{17}$$

$$\frac{1}{v_{k,i}} \sum_{u=1}^{U} \left[\sum_{t=1}^{s_u} x_{t,i} q_{t|s_u}(k) \right] - \lambda + (\alpha_{k,i} - 1) \frac{1}{v_{k,i}} = 0 \tag{18}$$

Rewriting (18),

$$\frac{1}{\lambda} \sum_{u=1}^{U} \left[\sum_{t=1}^{s_u} x_{t,i} q_{t|s_u}(k) \right] + \frac{1}{\lambda}(\alpha_{k,i} - 1) = v_{k,i}$$

Using this $v_{k,i}$ in 17 gives,

$$\sum_{j=1}^{N} \left[\frac{1}{\lambda} \sum_{u=1}^{U} \left[\sum_{t=1}^{s_u} x_{t,j} q_{t|s_u}(k) \right] + \frac{1}{\lambda}(\alpha_{k,i} - 1) \right] = 1$$

$$\sum_{u=1}^{U} \left[\sum_{t=1}^{s_u} \sum_{j=1}^{N} x_{t,j} q_{t|s_u}(k) \right] + ((\sum_i \alpha_{k,i}) - N) = \lambda$$

$$\sum_{u=1}^{U} \left[\sum_{t=1}^{s_u} q_{t|s_u}(k) \sum_{j=1}^{N} x_{t,j} \right] + ((\sum_i \alpha_{k,i}) - N) = \lambda$$

With the value of λ we get the next estimate for $v_{k,i}$,

$$v_{k,i}^{(m+1)} = \frac{[\sum_{u=1}^{U} \sum_{t=1}^{s_u} x_{t,i} q_{t|s_u}(k)] + \alpha_{k,i} - 1}{[\sum_{u=1}^{U} \sum_{t=1}^{s_u} q_{t|s_u}(k) \sum_{j=1}^{N} x_{t,j}] + (\sum_i \alpha_{k,i}) - N} \tag{19}$$

With (15, 16, 19) we have all parameter updates. The EM algorithm guarantees that the likelihood for estimates $\theta^{(m+1)}$ will be higher than $\theta^{(m)}$, therefore parameters need to be updated until convergence. It should be noted that it is not guaranteed to reach the maximum likelihood, only a local maximum.

7 Simulation Testing

There are some pitfalls with such a model which is easiest to show though simulated data. The fitted model can then be compared against the true distribution. To simulate data we need to generate the distribution parameters for this true model, Markov chain parameters and multinomial selection probabilities.

By looking at a sample of user sessions it is clear that users are mostly looking for specific items or item types. If it is the case that users are only looking for a subset of items then using a uniform prior to generate the true distribution, does not necessarily reflect this behaviour. A large amount of selection probabilities for each state are expected to be close to zero.

To get simulated data closer to this behaviour we zero expand $\boldsymbol{V}$, meaning only a subset of item selections are possible. The fact that there are items

users find not interesting makes it easier to distinguish and recognize states, as selection probabilities become very different.

In real data there is probably some correlation between states of what items are not interesting. For example there could be several states describing different interest in different clothing all of which may have no interest in boats. This correlation is exploited in chapter 8, but is ignored for now.

There are also some distributions that are not fitted in our model but are needed for simulation. This includes the distributions for session length, and number of session per user. Data indicates we can approximate them with a geometric distribution. This is also convenient as we then have a distribution equivalent to that of a HMM with smaller sessions but higher diagonal transition probabilities.

Such simulation testing might seem somewhat unnecessary in a situation where we have an unending ammount of data. By the time a model has finished training a test set can be created from new observations. However from a statistical point of view we want to figure out what problem the model can solve. What is the distribution we are actually fitting when training the model?

The parameters of the true distribution will be noted as $\theta = (\boldsymbol{\pi}, \boldsymbol{v}, \boldsymbol{A})$ and for the estimated model as $\hat{\theta} = ((\hat{\boldsymbol{\pi}}, \hat{\boldsymbol{v}}, \hat{\boldsymbol{A}}))$. After generating some true parameters we can simulate data and fit the model. Comparing likelihood of the true and fitted model will mostly be used to evaluate the fitted parameters. Comparing the true and fitted model directly is however difficult as discussed in chapter 7.1.

7.1 Label Switching

A state k is defined by, $(\pi_k, \boldsymbol{v}_k, \boldsymbol{A}_{k,*})$, where π_k is the probability of starting in the state, $_k$ is the selection probabilities within the state and $\boldsymbol{A}_{k,*}$, k'th row of A, is the transition probabilities out from the state. The model behaviour is independent of which k one assigns the state to. Note that if one reassigns the order of states, both rows and columns of $\boldsymbol{A}$ need to be reordered the same to keep the same transition probabilities.

With a rearrangement $r(k)$, such that all $r(k)$ contain the same set of values as all k the following rearrangement, performed for all k, results in the same model,

$$\begin{aligned}
\pi'_k &= \pi_{r(k)} \\
\boldsymbol{v}'_k &= \boldsymbol{v}_{r(k)} \\
\boldsymbol{A}'_{k,*} &= \boldsymbol{A}_{r(k),*} \\
\boldsymbol{A}''_{*,k} &= \boldsymbol{A}'_{*,r(k)}
\end{aligned}$$

As in the model defined by, $(\pi, \boldsymbol{v}, \boldsymbol{A})$ is the same as the model defined by $(\pi', \boldsymbol{v}', \boldsymbol{A}'')$.

This independence of ordering can be seen in the likelihood. In (4) the result is the same if we used indexes $r(k)$ for k and $r(l)$ for l as a new order is just permuting a sum. This means the first hidden state in the true parameters may not be fitted to the first hidden state in the estimated model, depending on the random initialization. Therefore it is not appropriate to compare the i'th state from the true distribution to the i'th state from the model directly.

This issue is referred to as label switching. The label in this case is the index we assign to the states. A reasonable loss function between the true model parameters and the fitted model then needs to be label invariant. Label invariant loss functions (Jasra, Holmes and Stephens 2005) are loss functions that give the same result no matter what order labels are in. The loss function,

$$L = \sum_i l(\hat{\boldsymbol{v}}, \boldsymbol{v}_i).$$

for example, comparing the i'th state from the distribution and model is not label invariant. With this loss switching the i'th state of the model with another would lead to different loss, which is strange as the model before and after switching is equivalent. The estimated model and the true model may even have a different number of states in which case this loss could not be implemented at all. Comparing the likelihood of two models is label invariant.

Any further comparisons between the model should also be label invariant. One way of achieving this is by relabeling the fitted model. This relabeling process should be such that the states in the fitted model are given the same label as the closest state in the fitted model. However one then needs to define closeness in relation to states.

Due to the fact that state selection probabilities need to sum to one the model is motivated to separate the data as much as possible. The more the model manages to separate different states the higher the selection probabilities for the items within the state will be. To separate the states the model need to create state definitions such that the user only has a high probability of existing in the state over specific observations, Figure 1 shows this trend.

States are therefore, if the model is able, defined such that they only have a high probability for a certain subset of observations in the training data. A similar state would then be expected to have a high probability for approximately the same observations in the training set. To measure the similarity between states we can therefore calculate the correlation between the probability of users belonging in the a state for every state in the true and fitted model.

The fitted model can then be relabeled by first calculating the correlation between $q_{t|s_u}(k)$ calculated using true parameters and $q_{t|s_u}(l)$ calculated using fitted parameters for all combinations of k and l.

Comparing each true state to the closest estimated state is label invariant as the closest estimated state is invariant to a reordering of the estimated states. Note though that this reordering does not necessarily result in the same model, some states may be excluded. We can also get duplicate states if one state of the fitted model is the closest to several states. Though this is not really an issue for what this process will be used for. So we have the following method of

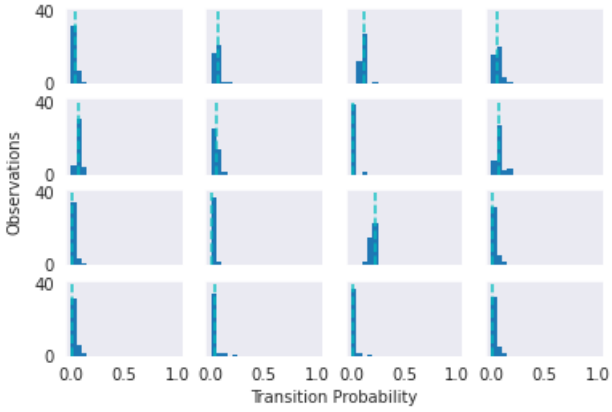


Figure 3: Plot showing a histogram over transition probabilities for a subsection of the fitted transition matrix. Each model was fitted to a different data set sampled from the true distribution. The true transition probabilities are shown in the dashed line.

relabeling,

Algorithm 1: Relabel Estimated Model

```
Initialize index list r for all true states j do
    i = fitted state with highest correlation to j;
    r[j] = i
end
Reorder π with r
Reorder rows of v with r
Reorder rows of A with r
Reorder columns of A with r
(Row normalize A and v)
```

Figure 3 shows that the relabeled models seem to have distributions centered around the true transition probabilities which shows that we mange to recognize similar states.

This relabeling will be needed in chapter 9.4 to perform hyperparameter tuning in a reasonable amount of time and without using too much computational resources. Essentially this allows a previous model to be used a an initialization by collapsing states which are most similar. The assumptions that are made for this to be reasonable are mentioned there.

7.2 Simulation Performance

To evaluate the performance of the model, the fitted model is compared to the true model and a naive popularity recommender. In the context of our hidden

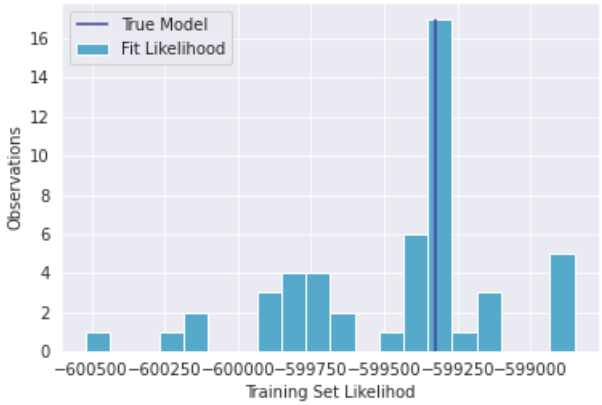


Figure 4: Plot showing a histogram of likelihoods of models trained on the same dataset with random initializations. The likelihood of the true model is given in the vertical line.

Markov model, this is a model with one state with selection probabilities,

$$v_i = \frac{\alpha_n - 1 + \sum_u^U \sum_t^{s_u} x_{t,i}}{\alpha - N + \sum_u^U \sum_t^{s_u} \sum_i^N x_{t,i}}$$

This will serve as an indication of whether the fitted model has found some useful clustering of the data. If the fitted model performs better than this naive model on a test set this indicates we have found some generalizeable information.

Firstly as the fitting algorithm only finds a local maxima of the likelihood we need to check to what extent the model manages to find a solution close to the global maxima. We then expect the likelihood on the training set to be as large or greater that the true likelihood if we have converged to the maximum likelihood estimators.

Figure 4 shows that the final model training set likelihood varies with different initializations. If one had the maximum likelihood estimators all solutions would arrive at the same likelihood, but in this case we do get struck in local maxima. In general this means that, if possible, its good to start with some different random initialization as there are cases where we do not converge to a good solution.

While getting a better likelihood than the true model is a good indicator for convergence. It is a bad indicator for model performance, in fact it can indicate bad model performance as we have over fitted to the training data. This means the model fit is too close to the observed numerical distribution having also fitted some of random variance in the training set. That could be and over/under representation of an ad or state transition. While this leads to higher likelihood for the training set it leads to a worse likelihood to an independent test set. To evaluate how well the model managed to fit the true distribution we evaluate it on a independent test set.

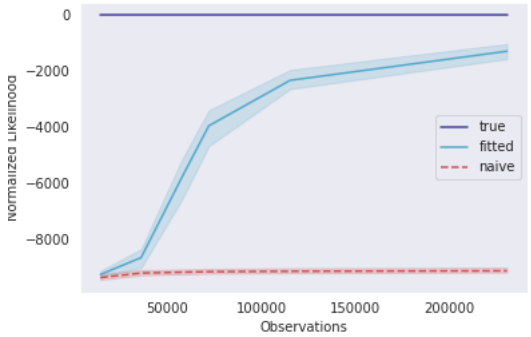


Figure 5: Test performance on a simulated test and training set of increasing size for the presented model and a popularity recommender. Since datasizes are changing the log likelihood observations are subtracted by the true log likelihood. We see that as data increases model performance converges towards the true model.

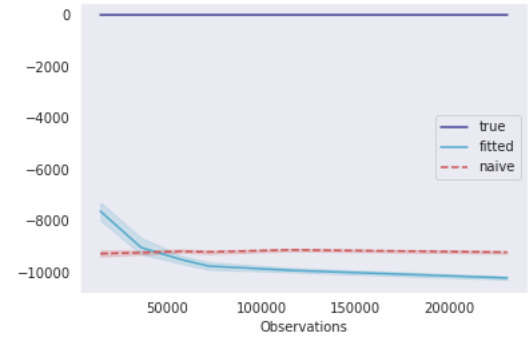


Figure 6: Test performance on a simulated test and training set with increasing size with constant item-observation ratio for the presented model and a popularity recommender. Since datasizes are changing the log likelihood observations are subtracted by the true log likelihood. As both data and number of possible items increase we see a decrease in model performance.

Here the each user follows the observed distribution from FINN data over 5 days splitting sessions at 30 minute breaks. This leads an expected sessions per user of 14.7 and expected items per session of 5. The simulated users could select between 200 items. and could move between 50 sates. As the amount of training data increases we see the expected increase in test performance as the observed distribution approaches the true distribution.

In the real data around 100 clicks per item is observed on average. An important check is then to see if this recommendation model performs well with this observation to ad ratio. Therefore performance as the model is trained on an increasing number of observations and ads keeping the observation to ad ratio approximately constant is tested.

Figure 6 we see the opposite trend than what we want. As data and number of ads increase model performance decreases. This test was done using 50 states and 20 observations per item. The trend worsens when increasing states. With such a large number of items and users one would expect a lot more states to be reasonable.

Additionally, the FINN data also contains a huge amount of variation in popularity between items. Depending on how this variation manifests itself within states this could be very detrimental to model performance compared to the simple recommender. If, for example all state selection probabilities are scaled by popularity, this is a correlation this recommender does not take into account. For the current recomender all states are fitted independently which means we need enough observations within each state to fit this popularity scaling.

Both the number of items and difference in popularity of items in the true distribution indicate that we need a more restrictive recommender.

7.3 Issues with the Pure HMM Recommender

This model can not generalize between items. Every item is seen as a completely separate entity. This can quickly lead to issues like overfitting and mostly giving popular items as recommendations. Not generalizing between items also leads to a inefficient use of data. We expect there to be a correlation in interest between similar items. This current model does not exploit any such correlation. Distance from some item embeddings could be used as a difference measure. Smoothing selection probabilities according to this distance measure. Comparing observed frequency and obervations within a state to see if the difference in selection likelihood between similar items is in fact significant.

One could include different measures of "interaction strength". Just a click is a weak interaction and browsing an items images is a stronger interaction. However this would change the model interpretation. An observation is then no longer a click on an item but a score. Furtermore v is then not an array of selection probabilities but a matrix with rows of normalized scores. This make statistical interpretation more difficult.

8 HMM with Matrix Factorization

8.1 Introduction

The model presented in Sahoo, Singh and Mukhopadhyay 2012 was not sufficiently restrictive to get a good fit with the number of items observed in the FINN dataset. A more restrictive model can be achieved without changing the model definition, for example by using fewer states or using restrictive priors.

Using fewer states will limit the number of different recommendations users can be given, and reduce the extent of niche recommendations. With more restrictive priors, some prior information to define them is required. This essentially moves the problem of getting good state definitions to the prior definitions.

Due to the amount of items in these datasets, most of the parameters are the selection probabilities, V, Table 4. Other collaborative filtering methods

are based on a user-item matrix similar to V. Matrix factorization presented in chapter 4.2 is one of these methods. In V each entry represents a user in state interests in a particular item. In matrix factorization data, referred to as Y in chapter 4.2, is represented as a user-item matrix, so each row is the score for each item given the user data. Under the assumption that users belong to specific interests used by the recommendation models presented in this work, the user in state interests and user interests matrices should be equivalent. Assuming of course data is collected over a short period, the user interest matrix only contain duplicate rows for users in the same state. In matrix factorization however matrix entries are a arbitrary score. The matrix V is different in that entries are instead a selection probability, meaning they must sum to 1.

8.2 Probability Matrix Factorization

The selection probability matrix, V, is a $K \times N$ matrix. Since the matrix V is a probability matrix where rows sum to 1. To restrict this matrix through matrix factorization it is reasonable to use a link function between the matrix multiplication of parameters S and B and the selection probabilities V. The link function used here is the softmax. This is mostly for practical reasons as in can be computed in log space and is quite efficient. The entries of V are defined as,

$$S \times B^T = H$$
$$v_{k,i} = \frac{e^{H_{k,i}}}{\sum_j^N e^{H_{k,j}}}. \tag{20}$$

The linear combinations in H are then transformed to a $[0, 1]$ range where each row sums to one.

The model presented in Sahoo, Singh and Mukhopadhyay 2012 needs K, the number of states, new parameters for each new item in the dataset. In a matrix factorization model we only need as many as the embedding dimension. With such a model we assume that there exists a vector representation of states and items of a specific size such that a funcion of these vector representation gives the item selection probability.

While one could draw some similarities between our model and logistic regression, interpretation of the model is much closer to that of a matrix factorization model as the goal is to allow the model generalize between states. From this point on it is assumed that V is of a higher rank that the embedding dimension. This is not guaranteed, but still reasonable since V results from a random initialization.

8.3 Optimization

To fit the S and B parameter matrices the main choices are alternating least squares (ALS) and stochastic gradient ascent (Y. Koren, Bell and Volinsky 2009a). While both algorithms have the undesirable quality that the time/iterations needed to converge is unknown and it is not exact. Gradient ascent does allow complete freedom in defining restrictions so long as gradients can be calculated. With alternating least squares one matrix is fixed, without the link function, the problem then becomes quadratic and an exact solution to the

other matrix can be found. The link function causes some problems for the ALS approach as the optimization equations to be solved for S and B become quite similar to those of logistic regression. The highly non-linear link function means there is no simple solution and the optimum has to be found by some form of gradient ascent. ALS is therefore expected to be substantially slower in this case and stochastic gradient ascent is used to fit S and B.

The gradients for the parameters can be calculated by inserting the definitions of $v_{k,i}$ for the factorized model, (20), into (5). Calculating the Q function by iterating though data would be too slow to do gradient ascent. We can however reorder this function such that we only need to iterate though a $K \times N$ matrix. In very sparse datasets there could be more entries in this matrix than data points, however in practice performing computation on matrices is much faster than performing computation on lists of varying lengths. Furthermore many libraries exist to easily perform gradient acent/decent with matrices on the GPU.

The Q function,

$$Q(\theta|\theta^{(m)}) = \sum_{u=1}^{U} \left[\sum_{k=1}^{K} \hat{\pi}_k^u [\log(\pi_k) + \sum_{i=1}^{N} x_{1,i} \log(v_{k,i})] + \sum_{t=2}^{s_u} \sum_{k=1}^{K} \Big[\sum_{l=1}^{K} \hat{A}_{l,k}^u [\log(A_{k,l}) + \sum_{i=1}^{N} x_{t,i} \log(v_{k,i})] \Big] \right]$$

with respect to $v_{k,i}$ is proportional to,

$$\begin{aligned}
&\sum_{u=1}^{U} \left[\sum_{k=1}^{K} \hat{\pi}_k^u \sum_{i=1}^{N} x_{1,i} \log(v_{k,i}) + \sum_{t=2}^{s_u} \sum_{k=1}^{K} q_{t|s_u}(k) \sum_{i=1}^{N} x_{t,i} \log(v_{k,i}) \right] \\
&= \sum_{u=1}^{U} \sum_{t=1}^{s_u} \sum_{k=1}^{K} q_{t|s_u}(k) \sum_{i=1}^{N} x_{t,i} \log(v_{k,i}) \\
&= \sum_{k=1}^{K} \sum_{i=1}^{N} \log(v_{k,i}) \sum_{u=1}^{U} \sum_{t=1}^{s_u} x_{t,i} q_{t|s_u}(k) \\
&= \sum_{k=1}^{K} \sum_{i=1}^{N} \log(v_{k,i}) \tilde{V}_{k,i} \\
&\tilde{V}_{k,i} = \sum_{t=1}^{s_u} x_{t,i} q_{t|s_u}(k)
\end{aligned} \tag{21}$$

So maximizing Q is equivalent to maximizing the sum of element-wise multiplication between the matrices $\log(V)$ and $\tilde{V}$. Since $\tilde{V}$ is independent of V, we do not need to go through the data for each iteration.

Some additional degrees of freedom are limited, both for interpretation and identifiability. Each column of B are centered. Since the rows of V are normalized a constant offset to a row of H would be normalized away,

$$v_{k,i} = \frac{e^{H_{k,i}+C}}{\sum_j^N e^{H_{k,j}+C}} = \frac{e^{H_{k,i}}}{\sum_j^N e^{H_{k,j}}}.$$

This leaves the model unidentifiable if rows of S and columns of B are not centered.

Furthermore both rows of S and columns of B can adjust the variance of the rows of V, so one needs to be limited to a constant variance, we chose B. We can see this issue as,

$$\begin{bmatrix} s_{1,1}c_1 & s_{1,2}c_2 & \cdots & s_{1,m}c_m \\ s_{2,1}c_1 & s_{2,2}c_2 & \cdots & s_{2,m}c_m \\ \vdots & \vdots & & \vdots \\ s_{K,1}c_1 & s_{K,2}c_2 & \cdots & s_{K,m}c_m \end{bmatrix} \times \begin{bmatrix} b_{1,1}/c_1 & b_{1,2}/c_1 & \cdots & b_{1,N}/c_1 \\ b_{2,1}/c_2 & b_{2,2}/c_2 & \cdots & b_{2,N}/c_2 \\ \vdots & \vdots & & \vdots \\ b_{m,1}/c_m & b_{m,2}/c_m & \cdots & b_{M,N}/c_m \end{bmatrix}$$
$$= \begin{bmatrix} \boldsymbol{b}_1 \cdot \boldsymbol{s}_1 & \boldsymbol{b}_2 \cdot \boldsymbol{s}_1 & \cdots & \boldsymbol{b}_N \cdot \boldsymbol{s}_1 \\ \boldsymbol{b}_1 \cdot \boldsymbol{s}_2 & \boldsymbol{b}_2 \cdot \boldsymbol{s}_2 & \cdots & \boldsymbol{b}_N \cdot \boldsymbol{s}_2 \\ \vdots & \vdots & & \vdots \\ \boldsymbol{b}_1 \cdot \boldsymbol{s}_K & \boldsymbol{b}_2 \cdot \boldsymbol{s}_K & \cdots & \boldsymbol{b}_N \cdot \boldsymbol{s}_K \end{bmatrix} = H$$

where $(c_1, \ldots, c_m)$ is some constant.

We can also naturally get a popularity parameter by centering columns of S, but giving each item an intercept by adding a column to S with all entries set to one. This intercept would then represent how much more or less prevalent this item is than what its data/embedding would fit. This is desirable as the data we use to train the model is the result of another recommender which may have a bias towards some items. Therefore one might want to give recommendations that are less affected by popularity. At prediction, the popularity sensitivity of item selection probabilities can be adjusted by multiplying the intercept by a constant between 0 and 1. This intercept is similar to the bias term introduced in many collaborative filtering methods Y. Koren, Bell and Volinsky 2009a; Yehuda Koren 2009. However centering the columns of S in not mentioned in the referenced papers. Without centering the factorization can still implicitly fit an item bias as shown below.

$$\begin{bmatrix} s'_{1,1} + c_1 & s'_{1,2} + c_2 & \cdots & s'_{1,m} + c_m \\ s'_{2,1} + c_1 & s'_{2,2} + c_2 & \cdots & s'_{2,m} + c_m \\ \vdots & \vdots & & \vdots \\ s'_{K,1} + c_1 & s'_{K,2} + c_2 & \cdots & s'_{K,m} + c_m \end{bmatrix} \times \begin{bmatrix} b_{1,1} & b_{1,2} & \cdots & b_{1,N} \\ b_{2,1} & b_{2,2} & \cdots & b_{2,N} \\ \vdots & \vdots & & \vdots \\ b_{m,1} & b_{m,2} & \cdots & b_{M,N} \end{bmatrix}$$
$$= \begin{bmatrix} \boldsymbol{b}_1 \cdot \boldsymbol{s}'_1 + \boldsymbol{c} \cdot \boldsymbol{b}_1 & \boldsymbol{b}_2 \cdot \boldsymbol{s}'_1 + \boldsymbol{c} \cdot \boldsymbol{b}_2 & \cdots & \boldsymbol{b}_N \cdot \boldsymbol{s}'_1 + \boldsymbol{c} \cdot \boldsymbol{b}_N \\ \boldsymbol{b}_1 \cdot \boldsymbol{s}'_2 + \boldsymbol{c} \cdot \boldsymbol{b}_1 & \boldsymbol{b}_2 \cdot \boldsymbol{s}'_2 + \boldsymbol{c} \cdot \boldsymbol{b}_2 & \cdots & \boldsymbol{b}_N \cdot \boldsymbol{s}'_2 + \boldsymbol{c} \cdot \boldsymbol{b}_N \\ \vdots & \vdots & & \vdots \\ \boldsymbol{b}_1 \cdot \boldsymbol{s}'_K + \boldsymbol{c} \cdot \boldsymbol{b}_1 & \boldsymbol{b}_2 \cdot \boldsymbol{s}'_K + \boldsymbol{c} \cdot \boldsymbol{b}_2 & \cdots & \boldsymbol{b}_N \cdot \boldsymbol{s}'_K + \boldsymbol{c} \cdot \boldsymbol{b}_N \end{bmatrix}$$

Here $(s'_{1,i}, \ldots, s'_{K,i}) = (s_{1,i} - c_i, \ldots, s'_{K,i} - c_i)$, and $c_i = \sum_k^K s_{k,i}$. The intercept/item bias implicitly fit for item i when a restriction is not applied is, $\boldsymbol{c} \cdot \boldsymbol{b}_i$. These methods also implement a sum of squares regularization to both the S and B matrix but not on the bias, so at convergence the S matrix will be centered over columns, and the correct bias/intercept will be achieved to minimize the regularization loss.

For our model we have an iterative method to fit the matrix factorization within an iterative method to fit the Markov chain. It is the reasonable, to

expect that at least for the initial initializations running the matrix factorization optimisation to convergence to in a waste of computation time. Therefore we implement the intercept with this limit on S such that even with a early stop the intercept is correctly defined.

This presents a problem in theory as the EM algorithm only guarantees an increase in the likelihood if the true maximum of the Q function is found. It does however not seem to be a problem in practice. The likelihood always increases unless the model is close to convergence where numerical inaccuracy also plays a role.

8.4 Including Ad Data

Matrix factorization methods also allow for the inclusion of data Y. Koren, Bell and Volinsky 2009a. While the states are generated by the model and there is no natural data to include for them some data is available for the items. This could include price, postcode, item category and some item information specific to its category. This item data can given in matrix form B_d similar to linear regression methods and corresponding coefficients can be fit for each item.

The full H matrix, including item data is then,

$$S \times B^T + S_d \times B_d^T + \boldsymbol{b} = H$$

Where $\boldsymbol{b} \in \mathbb{R}^N$ is the item bias/intercept. Which through the link function gives the item selection probabilities.

8.5 Prediction

Theory

When predicting a users current state we do not have future observations, meaning we can only predict, $\boldsymbol{q}_{t+1|t}$, the next state given previous observations, and not given all data. As the user gives click events for the next observation, one can choose to update the prediction as the user clicks further ads to be included in the observation. This would give the following prediction algorithm.

When a new user arrives, at $t = 0$ the current state is initialized by the estimated initial state probabilities, $\hat{\boldsymbol{\pi}} = \boldsymbol{q}_{0|0}$. When a new observation starts we have their current estimated state probabilities $\boldsymbol{q}_{t|t}$ and can predict the next user state for this new observation with 10 and get $\boldsymbol{q}_{t+1|t}$. As the user starts clicking ads for this observation, if we choose to update the current state based on this partial observation we can use 11 to get $\boldsymbol{q}^*_{t+1|t+1}$. As the user adds further events to the observation $\boldsymbol{q}^*_{t+1|t+1}$ can be recalculated.

With the current predicted state we need to generate some recommendations. While one could just select the state with maximum probability according to the predicted state. Then generate recommendations based on this states selection probabilities. This would not take in to account the certainty of the predicted state. To take into account the certainty of the predicted state we can instead generate recommendations based on a linear combination of the selection probability vectors, $\hat{\boldsymbol{v}}_k$, proportional to the predicted current state

probabilities,

$$v^* = \sum_{k=1}^{K} q^*_{t+1|t+1}(k)\hat{\boldsymbol{v}}_k$$

With the predicted current state probabilities, v^*, we can generate recommendations either by sampling or by selecting the most probable items.

8.6 Implementation

If the given data has a somewhat small amount of users or items the models can be implemented directly however with a larger number of items and observations some numerical issues will arise.

Firstly, with a large amount of items, calculating the multinomial probability, $f_k(\boldsymbol{x}_t; \theta)$, will require handling numbers very close to zero before normalization. This problem arises from calculating,

$$\prod_{i=1}^{N} (v_{k,i}^{(m)})^{x_{t,i}},$$

for each session. With a large number of items the relevant $v_{k,i}^{(m)}$ will be very small, at least at initialization. The result could then easily become too small to be represented directly on a computer. We therefore perform this calculation in log space,

$$\log(f_k(\boldsymbol{x}_t; \theta)) = C + \sum_{i=1}^{N} (v_{k,i}^{(m)}) x_{t,i}.$$

These log values eventually need to be normalized. To normalize we first subtract the largest value, C_{max}, from all log values. so that the larges value is 0. This means taking the exponential will give values we can represent.

Secondly, with a large number of items, the observation matrices are very sparse, for each observation most of the items were not clicked. This leads to wasted memory usage and compute power with a direct implementation. Converting this model to take a sparse data structure input, only non-zero values, is trivial. However it is difficult to achieve similar performance (in computation time) for less sparse situations due to lack of vectorization.

If the model only uses a single click event per observation implementation can be much more efficient. Users history can the easily be batched together with user histories of equivalent length. Preforming operations on batches is much more efficient than performing the computation on each user individually. With such matrix operations efficient libraries can be used, larger batches results in less calls to the Python API which is generally slow.

To further accelerate fitting user data sequences had up to 5 elements removed to batch as many user sessions together as possible. With these larger batches it became very beneficial to use the GPU to also calculate HMM parameter equations given in chapter 6.4, in addition to the matrix factorization.

Online Prediction

It would be somewhat troublesome for FINN the add additional data to be stored with users. This means the current user state probabilities $\boldsymbol{q}_{t|t}$ are not

stored. To get the current state probabilities one then needs to iterate through the whole user history. To give recommendations in an online setting however predictions need to be fast. Furthermore prediction runs on a single core of a CPU so there is quite a bit less compute power than when training.

To make predictions fast enough to be able to run an online test the length of the user data included had to be limited to 25-10 ads. Furthermore when performing the weighted sum over the selection probabilities only the top 10 states with the highest probabilities were used. Finally while iterating though the user history user states, state prababilities are not fully normalized. The max value is subtracted in log space to avoid numerical issues, however after moving back from log-space state probabilities are not normalized as the solution is the same only multiplied by a constant. Full normalization can be performed as a final step.

Click Events per Observation

The model assumes the user can only belong to one state within an observation. If the recommendation model includes several click events per observation as the recomendation model presented above allows, It seems less reasonable for this assumption to be true. For this assumption to be true it means that the click events have been separated into observations at the exact points where changes in interest occurred. With the data considered in this work, where click events are split into observations is more or less arbitrary. While it might be more reasonable to expect a change in interest to occur when there is a larger amount of time between click events, there is no reason why changes of interest can't occur between shorter time intervals.

When an observation does include clicks from several different interests there is no way for the model to separate these events. The model would have to assign all of the click events to the probable states as seen in equation 19. This means the model would not have the ability to completely separate interests. One could perhaps relax the assumption that users only belong to one state, allowing the model to separate the more messy user sessions. However a simpler solution that works withing the presented model framework is to only use one click event per observation.

The useful restriction combining click events close in time into one observation enforces is that interests do not change very rapidly. Say the model has defined two states where users within either tend to with high probability move between them. A situation like this will result in the transition probability of staying within a state is similar to the probability of transitioning to the other state. It would be reasonable in this case that the two states actually describe the same interest as the user has similar probability for clicking an ad in their current state as the other.This can be seen in Figure 1 as the model predicts the user returning to a previous interest several times. While the change in interest might make some sense in when only considering the local observations the fact that the user moves back ant fourth between interests does make it more difficult to conclude that there was in fact a change in interest.

However the belief users belong to a state for an extended period of time can also be rephrased as a prior belief. In which case it would be correct to include this belief in the priors for the transition matrix. If the belief is that

users belong to the same state for an extended period of time this would mean a higher prior should be included in the diagonal of the transition matrix prior.

Using a more informative prior for the transition matrix while using only one event per observation allows the model to predict changes in interest between any click event while still allowing the model to be restricted in how often a change in interest should be predicted. However calculating the forward and backward equations 7, 8 for every event can be computationally slow.

In this work model trained on real data only used single click events per observation. This is because evaluating how a users interests evolves a a time series is more straightforward when observations have the same size and one does not need to handle the possibility that a user might have been in multiple states within a observation. If the goal was purely recommendation however grouping click events in observations might be beneficial for the decreased computation time.

9 Results and Discussion

9.1 Online Testing

As discussed in the introduction, evaluating whether predicted changes in interest are good is difficult to do directly. It is easier to test if the model has the ability to gives good recommendations since, to give good recommendations the model needs to understand a users current interests. Due to the prediction function slowing down for some unknown reason after software updates the model using optimal parameters, chapter 9.4, could not be tested. The model tested had 200 hidden states, an embedding dimension of 50, no penalty and uninformative priors.

Testing model performance was done in an online test. This means the model was put in production at FINN and users were given recommendations predicted by the model. The performance of the model is determined by the click rate of users. The click rate is determined by how often users click an ad given in a recommendation. Say a user was given 10 recommended ads and clicks one of them, then the user was given 10 recommended ads and clicks none, then the click rate is at 0.5 as one of the two recommendations were interacted with.

The recommendations were given to a user in a category search. The user selects some overarching categories they would like recommendations within. Then the items within those categories with the highest click probabilities were shown to the user.

The model was tested against the current recommendation model used by FINN for this purpose. This model is a pure matrix factorization model using the Alternating least squares, ALS, implementation in pyspark (Spark 2021). This model trains on 60 days of data with an embedding dimension of 100. It uses the default restriction parameters and no item data.

The ALS model is retrained every 24 hours so it still does adapt to user interests. This is also in contrast to our model which was retrained every 48 hours. The ALS model recommendations was shown to 80% of users while the HMM model was shown to 20% of users. The HMM model was first trained for 160 iterations. This required quite a large amount of computation so later the

Model	Click Rate	Relative to Max
ALS	37.5%	1.0
HMM 160-iter	29.0%	0.77
ALS	39.0%	1.0
HMM 90-iter	31.2%	0.80

Table 5: Results of an online test comparing our HMM implementation and pySpark ALS. The HMM tests for different number of iterations were done at different times so ALS results during over the same time period are given for both tests. Otherwise the HMM model used 200 hidden states with an embedding dimension of 50.

model was adjusted to only use 90 iterations to train. These tests were not run at the same time which means different ads and users may be active over that time period adding some bias to the results. Therefore ALS result is given for both tests.

Table 9.1 shows the resulting action rates from the online test. The ALS method clearly performs better here. However the results shows that recommendations are reasonable and can to some extent compare to the best recommendation model at FINN for this purpose. So while the goal in this work was not to give good recommendations but rather to detect interest changes, this result does show that the ads the model ranks as likely for a user to be interested are often relevant to the users current interests.

9.2 User Interest Prediction

To show how the model predicts the evolution of a users interests and furthermore when a change in interest occurs both $\boldsymbol{q}_{t|t}$ and $\boldsymbol{q}_{t|s_u}$ are plotted against the items clicked by the user. The probabilities $\boldsymbol{q}_{t|s_u}$ are calculated backward from the last item shown in the plot. This way both what the model believed the user interests to be at the time of the observation and given future observations can be seen.

To show what the user clicked, the make, model and production year of the car in the ad clicked is given. The class of car is also given to give some easy indication of what type of car the specific make and model is. The images from the ad could not be used as the rights are held by the user who published the ad.

The user sessions shown were chosen at random, though some are excluded as they show a similar trend to other user sessions included. To select the sessions shown, the test set was given a random order, then sessions were considered in the order assigned in this test set. The indexes included to be shown were $[0, 1, 2, 4, 8, 9, 10, 12, 16]$. The plots also only include state probabilities from states that had the highest or second highest probability at any point. Including all states leads to a lot of distracting noise between 0 and 0.1.

For each of the sessions included, state probabilities were calculated both using the optimal model found in chapter 9.4 and the same model just using 150 states, half the number of states. This is to show that while the likelihood would indicate that more states would lead to better recommendations this does

introduce some issues when predicting user interests. These plots are given in appendix .1. The different examples are labeled session 1 through session 8.

The plot of state probabilities given past data only, $\boldsymbol{q}_{t|t}$, shows at what point in the session the model predicted a specific interest. The plot of these probabilities do not show what the model believed the evolution of user interests to be at any point. To show this one would need to calculate $\boldsymbol{q}_{t|s_u}$ backward from each observation. The plot of $\boldsymbol{q}_{t|t}$ should therefore be taken as point estimates rather than a series. This plot is however drawn as lines to more easily see which states are the same between observations.

The user state probabilities given all shown observations, $\boldsymbol{q}_{t|s_u}$, however is how the model believes the specific users interest evolved given all of the shown observations. This plot therefore represents how the model, models users interests. This plot is obviously the same as $\boldsymbol{q}_{t|t}$ at the final observation as this is $\boldsymbol{q}_{s_u|s_u}$.

Simple Interest Changes

Firstly the model manages to recognize a changes in interest when the change is clear. Session 1 and 7 are good examples of such simple cases. Here the user is clearly looking for a specific model of car and then switch to look for a different model of car. This change was recognized as an interest change quite quickly, see $\boldsymbol{q}_{t|t}$ plot, usually only taking two observations of a different model to recognize a change, i.e. predict a higher probability for a different state.

The speed of this change is of course dependent on how likely the different ads are to be clicked for the different interest states. If the different ads for cars have very similar probability of being clicked then the change would be slow. A very slow or fast predicted change in interest is not a problem, so long as it is sensible from data. Ideally a slow change, meaning similar selection probabilities means the ads are very similar and a user clicking one or the other indicates very similar interest.

When comparing the state probabilities given past and given all data for session 1 and 7 we see that while the model is a bit delayed in recognizing the change in interest this is corrected in the backward step. The switch in interest is then placed more correctly. Though in session 1 the 300 state model seemingly predicted the change in state early, granted with very low probability. This is caused by the difference in transition probabilities between the two states and the quite slow change in interest the model seems to predict in this instance.

The high values in the diagonal of the transition matrix and similar selection probabilities lead to these long tails, slow interest changes. Though this behaviour can seem a bit strange under the model assumptions, that the probability of the user having changed state increases even as the types of ads the user clicks remains the same. It makes more sense if one sees the process as a little less discrete, even though this is not the process the model assumes. Then one could see the user as becoming more and more inclined to switch state until they finally end up clicking different types of ads.

Ultimately this shape of the modeled state probabilities is a consequence of that interest states can not be completely separated leading to items potentially having similar selection probabilities between states as seen in figure 14. If the

interest transitions seem unreasonably slow this could then be improved by increasing the number of states or using a different hidden model.

Exploratory Behaviour

The model seems to also be able to give quite reasonable results in cases where the user interest appear to be non specific and more exploratory. Take session 2 where initially it is difficult to understand what the user is specifically looking for. High and low price, station wagons, vans and electric cars are clicked in quick succession. The model then seems to have a low confidence in any interest state which is reasonable, more reasonable than being very confident in a specific interest.

The model in some cases seems to fit a less noisy result with exploratory behaviour as seen in session 3. Here the user also explores less different cars, but we observe the 150 state model has a somewhat constant prediction with two states with somewhat high probability. The 300 state model is however quite confident in a single state, and given that the user is not clicking too different ads neither predicted behaviour seem unreasonable.

In session 2 the 150 state model seems to have fit a quite high selection probability to a specific state for the Tesla ad clicked by the user. At this point the model becomes very confident that the user has switched state. When conditioning on future data this confidence is lowered but still apparent. This could be a case of overfitting by the model, having too low selection probability for the Tesla in other states ans the user seems uninterested in Tesla ads considering the future clicks. It could also be the observation which is the outlier and one actually would, with a high probability, expect users to continue exploring Teslas after one click. The fact that we do not see this behaviour in the 300 state model points towards this behaviour not being very apparent from data as the higher state model has not fit the same amount of confidence in a Tesla interest state.

This behaviour of quite sudden confidence when a user clicks a Tesla is observed for the 300 state model towards the end of session 6. Here the model shows predicts that the previous interest state has a probability close to 0 after one click on a Tesla event though the last click by the user seems to follow this interest. It could be that user behaviour around Teslas consists of users either clicking many Teslas in a row creating states with very high concentration around Teslas, while there also exists quite a number of users who click a Tesla ad once. If the model does not form two separate states for this behaviour for all Tesla ads it can lead to such model predictions. We see that in this case it is the 150 state model which has fit seemingly more reasonable user behaviour.

In session 4 we observe what we would prefer in situations where the user explores a somewhat wide range of ads. After the user stops clicking very specific ads, the model starts out having low confidence for most states but eventually one state wins out. This means not significant interest changes are predicted over the period where the user explores different options. This is important as all ads this user clicked after the Hyundai Kona do seem to be related to the same interest so all items could help give some indication of the users current interests. If this model was then used to eliminate irrelevant clicks for a separate recommendation model, none of these clicks would be removed.

Session 9 is a bad example when it comes seemingly spurious interest changes. All the clicks in this session could easily be from the same interest. The user may be undecided if they want a normal sedan or station wagon or something more robust and the user is exploring options withing these categories. In such situations one would very much want the model to not predict any interest changes. Predicting changes in interest where there potentially is none will lead to the conclusion that clicked ads that in reality relate to the users current interest are irrelevant.

The sudden interest changes with the Tesla ads in session 2 and 7 while having predicted quite slow changes in session 1 and 8 underlines the need for a more complex interest model. With the current number of states, to help the issues is session 2 and 7 one would either need to restrict the model to higher transition probabilities along the diagonal. This can be done through higher priors or grouping clicks into a single observation. This will also inevitably lead to slower transitions between states. On the other hand one could lower the transition probabilities along the diagonal through priors. This will speed up transitions but also lead to more spurious interest changes.

In theory increasing the number of states can solve these problems giving the model the ability to fit more specialized states and transition probabilities. In practice this is difficult for a number of reasons. Firstly, as previously mentioned, training time is already an issue and transitions, forward and backward, are an $O(n^2)$ operation so computation time increases quite quickly. Increasing the number of states likely requires more data because, while figure 9 shows quite good correlation, there is still a good amount of noise especially for transition probabilities. Since there is no form of restriction on the transition probabilities it is expected that to increase the number of states significantly one would also want to use more data. Beyond just increasing training time this would include data from further back in time. This means the time dependence of state definitions is even more prevalent and many of the new states available to fit some interest might not even be useful as they relate to older items.

The model can then give quite reasonable predictions of the evolution of a users interest. This is especially the case for the extreme cases of the user either considering very specific items or exploring very different items. Both of these behaviours are quite clear from what the model predicts.

Cases where a user is exploring somewhat different ads, especially if the user is interested in a few different types of cars as seen in session 9, the model performance is less consistent. Here the model can predict spurious interest changes which is very undesirable if one wants to use such a model to determine what observations a relevant to the users current interests.

9.3 Convergence

The model presented is slow to train. This is highly dependent on the number of states fitted, however for the highest number fitted for this work, 300 hidden states, running 90 iterations took 46 hours on a NVIDIA TESLA P100 GPU. This training time is too high to compete with the models used at FINN. All of these models train withing 24 hours. This is especially disappointing as this model handles older data quite naturally. We expect to see almost only benefits

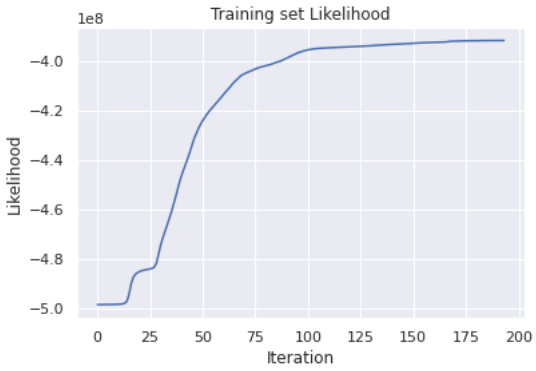


Figure 7: Evolution of training set likelihood over training iterations. Model was trained with 100 hidden states and a embedding dimension of 30.

to using older data as the state of a user is allowed to change over time at any point.

Training the model for 90 iterations does not result in convergence. However as discussed in chapter 9.1, further iterations does not seem to have a large, if any, impact on model performance. This is likely due to further iterations making smaller adjustments to probabilities, however the order of especially higher probability items could still be constant. It would then look like the model was still improving when considering likelihood, while the recommendations stay the same.

Even so, 46 hours to run 90 iterations is too much. Figure 7 shows the likelihood over training. This shows that the initial steps of the model are very slow. This can indicate a bad initialization. The reason is likely that sampling initial parameters from the prior results in parameters with very similar values. Since this means the different user states are very similar, it then makes sense that the training is initially slow as the different user states have little to separate them. The users then have a similar probability to belong to all of the states which means most items will be included with some significant probability in all states, as seen in equation 19.

Some simulation testing was done to see in using clustering to initialize parameters could be helpful. The states were then initialized with small random values for most items except for a small portion of the items that were often seen together, which were given a high probability. While this did to some extent help models converge faster it also increased the variance in the likelihood of the resulting models for different random initializations.

Increasing the likelihood of landing in a bad local maximum is very undesirable in this case as one cannot train several models due to the time required. When only one model is trained there is nothing to compare to. So without being able to train several models one cant know if the model trained got stuck in a bad local maximum. Instead the previous model parameters were used as an initialization when necessary as discussed in chapter 9.4.

9.4 Hyperparameter Optimization

With the amount of time needed to fit a model, hyper parameter tuning becomes problematic. Optimally one would be able to train several models on a grid of hyperparamenters. Since the model only converges to local optima it would also be useful to train several models with random initializations for each point of the grid of hyperparameters.

Due to the time constraints of this work, low expectation of significant gains in performance and also cost as servers used for training is lent by FINN full grid search was not preformed. Hyperparameter tuning was therefore done linearly for individual parameters with some consideration for possible correlations between parameters. The search was further limited to the parameters considered most important, which includes number of hidden states and embedding dimension. Priors, when less strict, were observed to have little effect on model performance which is reasonable given the amount of observations.

For hyperparameter training models were also trained from a previously fitted model. In general this should be avoided for models that optimize through gradient ascent/descent. If the model used to initialize the training process is in a local maximum then the model trained may not have the opportunity of converging out of this maximum. Then every model trained will get stuck in the same maximum, and there is no possibility to train several samples to explore the full likelihood function. However as mentioned, training several samples is not a possibility with this model.

If it is firstly assumed that all models using the same model as an initialization converges to the same local minimum. This means there is a continuous evolution of the likelihood the model converges to as hyper parameters are changed, if the model jumped to a new maximum we would also observe a jump in the likelihood. If it is also assumed that the change in likelihood due to different hyper parameters is the same among all maximum in the likelihood function. This second assumption is likely incorrect especially as the models become more different and it is only used as several samples cannot be calculated. If these assumptions are valid then initializing from a pretrained model means no noise in results and the results are representative of the model in general.

We then need to define how to use a previously trained model to initialize a model with potentially different number of states, embedding dimension and penalty. Fitting a different embedding dimension and penalty is simple. First the observed click probabilities, equation 21, are found using the original model then the matrix factorization can be fitted with the new parameters to this observed matrix.

To fit a different number of states, only reducing the number of states was considered as the preferred time to fit a model for online testing was around 48 hours, so the max number of hidden states had to be around 300 or below to fit this time window. To reduce the number of states, the correlation between states were used as described in chapter 7.1. States with high correlation are combined weighted by their respective popularities.

There is expected to be a large amount of correlation in the effect of penalty and embedding dimension used in the matrix factorization. Increasing the dimension of the matrix factorization however increases training time. Therefor the model penalty was first increased to a point where it made a sizable

Figure 8: Plot showing the test set likelihood of models with 150, 200, 250 and 300 states. The other model parameters were optimized for a 300 state model.

improvement on the test set likelihood then the optimal embedding dimension was found with this penalty.

There is expected to be some correlation between number of hidden states and embedding dimension/penalty as well. It is expected that for lower number of states the restriction provided by the matrix factorization is less necessary and therefore a higher number of dimensions or lower penalty leads to better results. It was chosen to optimize the embedding dimension with a model with a high number of states as from preliminary results these models seemed to perform better. So this may result in models with fewer number of states having a somewhat lower likelihood than optimal.

The process of hyper parameter tuning then consisted of first fitting a model to be used as an initialization. This model was fitted with the max number of states one could reasonably expect to fit which was 300. An embedding dimension of 50 was chosen for this model with low penalty. Using this initialization, first a penalty was selected which gives some improvement, but is still less than optimal so that the embedding dimension can also be reduced to improve training time.

Then the optimal embedding dimension was found, and finally the optimal number of states at or below 300. This gave the model parameters of 300 states with an embedding dimension of 30. Plotting the likelihoods over different number of states shows that there might be more to gain from using a higher number of states, as seen in figure 8. While the other hyper parameters were optimized for a 300 state model, this plot does not seem to show a maxima at 300 states.

9.5 Transition matrix

To get some idea of the correctness of the fitted transition probabilities, fitted transition probabilities are plotted against the observed transitions in the test set as seen in figure 9. The observed transitions are calculated from equation 16 using the fitted model parameters. The observed transitions are therefore calculated given the model parameter estimates. The current estimate of the

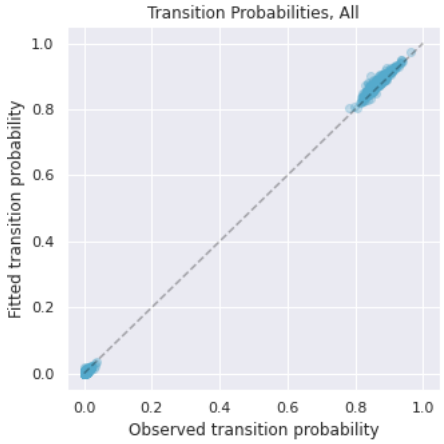


Figure 9: Fitted transition probabilities plotted against observed transitions in a test set.

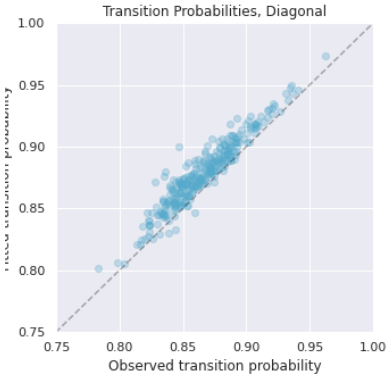


Figure 10: Sub plot of figure transition probabilities plotted against observed transitions in a test set with diagonal elements only.

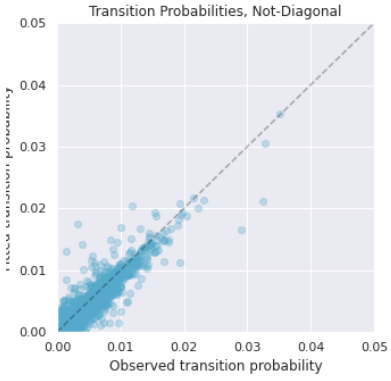


Figure 11: Fitted transition probabilities plotted against observed transitions in a test set with non-diagonal elements only

transition matrix then influence the observed transition probabilities. However we need a HMM model to calculate what transitions are observed and the model parameters are calculated completely independently of the test set. Furthermore it will be seen later the transitions seem very much determined by the observations.

This plot will show if the transitions described by the fitted model occur in the test set. Note the deviation between fitted and observed transitions is both caused by parameters overfitting to the training set and the limited size of the test. limited test set causes observed transitions probabilities in the test set to also be different from the true distribution. This plot should therefore not be seen as indication variation of transition probabilities. It rather indicates how well the fitted transition probabilities matched that observed in the test set. This is similar to variance but should not be seen as a direct relation.

The most distinct feature in this plot is that we clearly have two separate distributions. It turns out that the cluster of larger probabilities are the diagonal elements of the transition matrix show in figure 10. The cluster in the lower left are transition probabilities between different states shown in figure 11. This same behaviour is observed when using uniform priors for the transition matrix. This means that for this data there is no need to enforce the assumption that a users interest last over several observations. With the model assumptions this property if iterest is very much observed in the data.

Interestingly the value of these transitional probabilities seem to be quite consistent between models. The model shown in Figure 9, with 300 states, had an average diagonal transition probability of 0.879 while the model trained with 150 states had an average diagonal transition probability of 0.890. This means in the 300 state model users on average spent 8.2 clicks in the same interest before a change. In the 150 state model users spent on average 9.1 clicks before a change in interest. This shows that the increase in likelihood when increasing the number of states to a larger extent comes from more accurate state definitions, and does not result in much more rapid changes in interest. This is also a good indication that the changes in interest are often quite clear and not continuous. We also see this in the user session examples. The models mostly agree on how many interest changes occurred. It should be added that these models were trained completely independently, using random initialization.

The high values in the diagonal of the transition matrix indicate that the model will always predict the user to be in a similar state and changes in state only occur when it is indicated from an observation. This does not mean the transition matrix is uninformative. There is a large variance in the probabilities for transitioning between different states. Some transitions have a probability very close to zero. This means that even if an observation is most likely observed in one state the transition from the users current state to this state may be very unlikely resulting in a different predicted state transition.

This plot also shows a quite high correlation between the fitted and observed probabilities. This shows that there is quite a large agreement between the transitions predicted by the transition matrix and the test data. The only slightly worrying result is that there seems to be quite a lot of transitions with a fitted probability near zero which have a larger probability of appearing in the test set. This low transition probability will result in the model being slower to recognize the state change. This can be an indication that some further

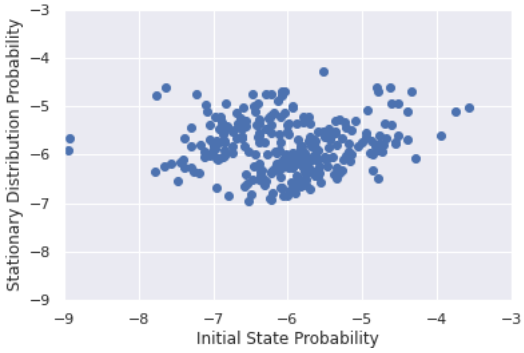


Figure 12: Plot with the initial distribution plotted against the stationary distribution of the fitted Markov chain. We do not see the expected correlation in this plot.

restriction to the transition matrix can be useful. The transition probabilities with a high fitted probability but low observed probability are of less concern as the test set is smaller than the training set, so some transitions being unobserved is expected.

An interesting observation is also that the diagonal elements of the transition matrix have a strong tendency to have a larger fitted probability than observed probability. A contributing factor to this is likely that the selection probabilities fit the training set better than the test set. This means that the certainty with which a user is predicted to belong to a state is on average lower on the test set than the training set. If the probability that the user belongs to a specific state in is lower then the probability of any specific transition is dragged towards the average.

Another sanity check one can make on the hidden Markov chain is to study its stationary distribution and its relation to the initial distribution. The stationary distribution of a Markov chain is the distribution the chain converges to as the number of transition go towards infinity. This stationary distribution is just a property of the Markov chain and is unrelated to the observations.

By our model assumptions user interests follow this Markov chain. The data we have collected is however not necessarily at the beginning of a users data history, and definitely not at the point when the user started to form and change interests. This is only the case for users who just started using FINN during the month data was collected from (and were just born). We would therefore expect the initial distribution to be somewhat close to the stationary distribution, as for older users their interests at the beginning of this dataset is the result of all the past transitions/changes in interest.

If figure 13 the stat probability in the stationary distribution and initial distribution is plotted against each other. However we do not see the expected correlation between the stationary and initial distributions. This result is quite strange as it indicates that according to the model parameters the expected state for a user to belong to changes over the series of observation. This could be caused by mostly new users or some correlation over time for which items

a user is recommended or the change in available items means that states are created for items available at different times.

The fact that the current items change over time and likely also the general interest and behaviour of the user base is problematic for the Markov chain model. This means that the users current interest, and how users change interests, depend on more that just the previous state but the current time as well. The user state is then not only dependent on the previous state.

It would be undesirable for state definitions to be dependent on when an item is available. However this is difficult to avoid. Take a group of items one would consider to belong to the same interest and one would expect them to be clicked together with items from a particular state. If the state definition is fitted to items that were available earlier then this new group of items would not be recognized as the same interest. Since state definitions are created from observing ads being clicked close together the changing availability of ads can end up being an important, yet undesirable, factor in the state definitions.

State definitions should relate to an interest not to what items are available at any given time. There are therefore further reasons, than just maintaining the Markov property, to reduce the potential of states being defined around what ads are currently available. An interesting restriction to the model could be to restrict the initial distribution to the stationary distribution of the transition matrix. Further restriction to the model could require a constant population of users in each state over each period of time. This would potentially limit the extent to which states are dependant on when an item is available in the store. Finding explicit solutions to initial distribution and transition probabilities however difficult with such a restriction. Optimizing the Markov chain parameters would then likely also need to be optimized numerically.

9.6 Importance of Older Observations

The presented recommendation model is based on using previous observations from a user to generate recommendations. One can the question how much of this information does the model use to generate its recommendations. Since users do change their interests over time we do expect the most recent items to be the most important for predicting future actions. Furthermore the model is based on the Markov assumption and has somewhat few states which also pushes the model to mostly use the most recent observations.

To get some idea of the extent to which past data is important on can calculate the likelihood with varying number of preceding observations given as input. To calculate the likelihood over a consistent amount of data when varying the number of preceding observations only the likelihood of the last observation is calculated given the preceding observations.

This plot shows that only very recent observations have an impact of model predictions. From this plot it seem like using more than 10 observations has very little impact on predictions. While one could argue that only the last 10 observations are usefull, form the online results the model is lacking in recommendation performance. The fact that information in observations beyond the last 10 is essentially discarded could be a contributing factor to this.

When considering the transition probabilities given in Figure 9 the average time a user spends within an interest a little under 10 observations. So This

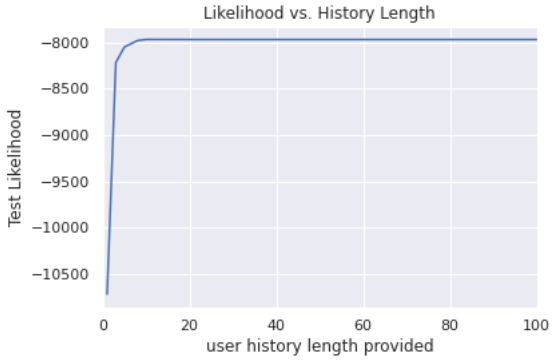


Figure 13: Plot of the likelihood of the last observation in all test-set user histories. This likelihood is calculated given a varying number of preceding observations as input. This was calculated for a history length of 200 as well with no improvement.

result indicates that the transition matrix actually does not provide that much information on predictions. Observations beyond the average time spent in an interest are unimportant.

9.7 Recommendation Analysis

There are some clear weaknesses with the model which underline the difference between detection of users interests and giving good recommendations. This is also expected from previous offline tests, usually neural networks or some version of matrix factorization work best (Ludewig and Jannach 2018). The problems with this recommender can be related to the lack of variation in the recommended ads.

Firstly the recommendations are very popularity dependent. Since some ads have a much higher popularity than others this also means they will be fitted with a much higher probability of being selected. Some very popular ads will often be fitted with a high probability for several interest groups. This results in recommendations for users being less varied. Often the top items can tend to remain the same even when the users state probabilities seem to show a change in state.

One then needs to discuss how recommendation performance relates to predicting interest changes. This is because this model behaviour is not necessarily wrong. A popular item might not provide a good indication of which state a user belong in. In fact popular items may usually be worse predictors of interests. The fact that an ad is popular means that many different people find it interesting which points to popular items being less informative of what interests a user currently has. A good example form the data is a six-wheeled custom monstrosity of a car which often popped up as having a large clickrate for several states. This isn't surprising as most people would want to look at more images of such a car. However this also means that if someone did click this car it is not very informative of what their current interests are. The model

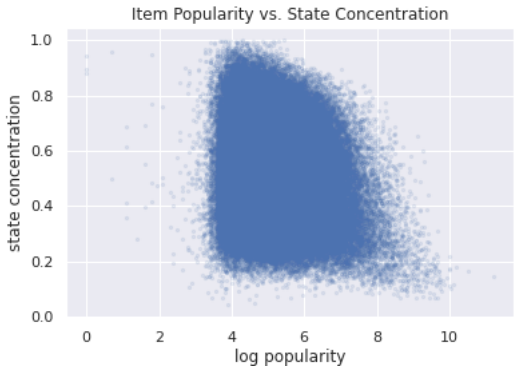


Figure 14: Plot showing the correlation between item popularity and the items concentration in a single state. A high concentration means the items selection probability is mostly concentrated in a single state. The plot shows a trend of items with higher popularity being less concentrated in a single state.

is correct in not jumping to the conclusion that the user now only wants 6 wheeled cars.

To show this trend a measure of state concentration for items is defined. This measure is achieved by taking the selection probabilities for an item at each state, i.e. columns of V. These selection probabilities are normalized to sum to one. The largest value is then defined as the state concentration. This measure will be 1 if the item is only selected within a single state and has a selection probability of zero for all other states. The minimum value of this measure is $\frac{1}{K}$, where K is the number of states. This is achieved if the item has a equal selection probability for all states.

In figure 14 this concentration measure is plotted against item popularity. We see the expected correlation between item popularity and state concentration. Higher popularity items to a lesser extent belong to a single state.

The sudden cutoff is caused by only items with more than 100 clicks were included in the offline data set. There are some items with seemingly 0 or very few observations in the training set which would seem very unlikely with random sampling. However there are quite a few user sessions whom mostly have clicks for a single item. If such sessions are not included in the training set then some items can be observed to have a very low popularity.

While this correlation looks significant and fits the proposed theory around popularity, there are no uncertainty estimates for these points. This is important as popularity, as in number of observations, very much determines the variance of the observations and an increasing amount of variance can also explain the shape of this plot. When an item has fewer observations it would have a larger random chance of only being observed around a very separable set of observations. As the number of observations increase the ad is to a larger extend observed in the different contexts the item may be observed. This means more popular items have a smaller chance of getting a high concentration due to random chance. This is the reason no attempt at regression to determine the significance on the correlation was made. It would be pointless without some

sort of estimate of the posterior distribution of these concentration estimates.

There does however seem to be some correlation. No popular items are observed with high concentration. Saying this correlation is purely due variance would then indicate that almost all items have its true concentration below 0.3. This seems unreasonable, that there is not a significant number of items that mostly belong to a single state

For less popular items, state concentration is very much item dependent and there are explanations for this observation beyond higher variance. This is not necessarily because the item doesn't appeal to a specific interest but could also be caused by the model fitting too few interest states.

The reason for an ad being popular is also skewed by the fact that the training data are clicks from users being given recommendations by a previous recommendation model. This leaves for the possibility of a feedback loop where an item is popular because it is recommended a lot and therefore would appear a lot in the training data so the next model will recommend the same item a lot as well and so on. This is a result of a lot of recommendation models implicitly or explicitly making the assumption mentioned in chapter 5, that the user has considered all other items when making their selection.

This assumption simplifies the model/loss-function definition as it means the click indicates the clicked ad is better than all other ads for the specific user at the specific time. Giving recommendations according to the true selection probabilities, equation 2, would potentially solve this issue however this test would have to be over a longer period of time to see how training form data from previous such models affect recommendations over time. Time constraints and problems with time consumption of the prediction function made completing such a test difficult. This test is also more relevant to recommendation performance, and not interest detection, which is the main focus of this work.

The fact that there is only 200 interest states fitted in the online testing model also means that there is only around 200 different recommendations the model can give. This is because, as seen later, the user state probabilities usually have one state with a much higher probability than the others, so states rarely combine to give a different ranking. This means the model cant give very personalized recommendations. However as mentioned in chapter 9.4 the limitation to below 300 states is related to training time. From model results is seems that adding more states will improve the model and further improve this issue. This is not an issue with the ALS model as this model fits a unique score for each ad for each user not just for each state.

9.8 State Definitions

It is difficult to get any indication of how good the state definitions are by just considering the ranking of items within them. The problem discussed in chapter 9.7 with popular items having selection probabilities more spread out contributes to this. The most popular items and therefore the items with the highest ranking within in a state will usually repeat themselves between states. The top items therefor do not give the best indication of what interest a state actually describes.

Looking deeper into a state however it seems that the model manages to create quite reasonable interest groups consisting of either a specific brand or type of car, excluding some outliers. This is however purely based on intuition on what users with similar interests would be interested in. To get a better idea of how the defined stated affect the predicted user interests the evolution of a users state probabilities, presented in chapter 9.2, are likely better indicators of how good the state selection probabilities are.

It could be assumed that these state definitions seem natural as the model is given item specific data, like car make and model, so it can easily separate these states. From some preliminary testing it however seems that the state definitions are similarly reasonable both with and without item data.

A contribution factor to why data does not seem to be that use full, which seems to often be the case for other model trained at FINN, could be the low state concentration of popular items. When maximizing the likelihood it is most important to give the most popular items accurate probability estimates. However these popular items can have probabilities that vary quite little. This means that for the model to be defined such that data has a large impact on the selection probabilities. The item and state embeddings need to be defined such that the effect of data is negated for popular items. This could make these embeddings much less useful for less popular items and leading to a worse model over all. If this is the cause of data not being exploited well, it could potentially be useful to add an additional parameter per item which describes how much the item data should affect the state probabilities.

9.9 Interpretability

While recomendation performance is lacking, this model, as opposed to the neural methods or even matrix factorization to some extent provides a good insight into the main categories in the dataset. This is a result of the limited number of states the model has available to fit user interest. Ordering a state based on selection probability, ignoring some of the top items, Gives items in the dataset which very much seem to belong together. The model achieves this even without data. This can be useful to recognize which categories users should be able to choose between. The model can generate a state for ads that thematically belong together, but do not have any special category. In this case it could be useful to create a category for those ads such that users with this interest can more easily fond satisfactory ads.

What would be difficult is assigning ads to a specific category based on these states. Looking at the top items, it is easy the see what, in general, the state has fit to. However at which selection probability to set the cutoff for when an item no longer "belongs" to the specific state is more difficult. These selection probabilities are also very much affected by popularity which makes setting a cutoff likely incorrect especially for very popular and unpopular ads. It may be more useful to consider how the selection probability within the state deviates from the items selection probability in general.

Another useful feature of states making somewhat intuitive sense is that predictions can quite easily be explained. Considering the users most recent actions it is easy to see why a specific state is chosen as most likely. This is helped by the fact that the transition matrix does not impact results too much,

which can be disappointing, but means interpretation is easier as one does not have to consider too many past observations. When the most likely states are known the model predictions are also easily explained usually only consisting of a linear combination of two states at most.

The fact that predictions usually only involve linear combinations of a few states it is easy to check what items might be predicted. This is useful when a website has user generated content. Checking what items can be predicted can be checked by looking though the top n items in each state. Checking why an item was predicted can be seen by checking in which states it has a high probability. These states will then show which ads users often clicked around this ad.

This allows easy enforcement of potentially privacy or fairness issues as the states are discrete, as opposed to neural networks, and there are quite few of them as opposed to matrix factorization. It is therefore feasible to manually perform checks on the model if this is necessary.

10 Future work

10.1 Variance Estimates

There is little literature on calculating uncertainty estimates for recommendation models in general. This is likely due to it being difficult due to these model usually having very high dimensionality. However in recommendation models variance estimates are only useful as a far as it can help improve prediction. There is not really any skewness in which items it would be bad to incorrectly recommend or nay other penalty. The goal is purely to maximize the number of times a recommended item was clicked.

There are the many different avenues to improve predictions other than calculation variance estimates. Improvements which are expected to be more computationally efficient, easier to implement, easier to derive and give a better improvement on predictions.

10.2 Computation Time

There are several improvements one can make to improve computation time. Firstly the python API is slow so while a lot of the computation is done with accelerated libraries there is still quite a bit of slowdown due to it being a python implementation. As opposed to normal neural networks which PyTorch is designed for. This model consist of very many relatively small matrix computations. Between each such computation calls to the python API is made which are a potential for improvement. An implementation in a more low level language like C is therefore expected to give substantially faster results.

10.3 Time Dependence in State Definitions

The fact that the model can create different states that describes the same interest but at different periods of time is a quite large limitation. This especially limits the models usefulness as a model to predict changes in users interest. This is because the model will predict can, or is even motivated to through

maximizing the likelihood, to predict a change in interest when a user clicks items that are rarely observed together just due to when the different items were available.

To mitigate this problem one would have to restrict state definitions such that the state is observed consistently often over all time periods. This does mean that there is a risk for the state definition do describe one interest an one time period and another at a different time period. How effective such a restriction is therefore needs to be explored further.

10.4 State Transitions

The model by which users transition between states in this model is simplistic but also of quite high computational complexity. the complexity of calculating state transitions are $O(n^2)$ which greatly hinders the number of states which can be used. This transition model also has difficulty remembering any long term user tendencies. The model also fits unique transition probabilities to each state while it would be reasonable that some especially similar states also have correlated transition probabilities.

One could instead desire a transition distribution which could exploit potential correlation in state transitions similar to what matrix factorization achieves for selection probabilities. A method by which some longer term information about the user is remembered such that state transitions and potentially also selection probabilities could depend on some more general user information. Allowing for some user specific variation could help the state definitions become more separate as some of the variance included in these could be moved to the user specific information.

Just introducing a more complex transition function would keep most of the interpretability benefits of the model. User interest plots and states could still be interpreted similarly. It could however become less obvious why two items were included in the same state or why a specific transition occurred.

11 Conclusion

This work has introduced a recommendation system with the goal being creating a model where the evolution of user interests is interpretable, and changes in interest can be identified. Since the data available is not directly related to user interest, the model was trained as a recommendation model, where latent variables had to be fitted, and where these could be interpreted as user interests.

The model is based on a hidden Markov model, but combined with ideas from matrix factorization to better handle sparse data and take advantage of correlations between interests. General user interests within the dataset should be reflected in the states of the HMM, which would allow for the prediction of user interests.

Even with a simple model for user interest, a Markov chain, the model gives a quite sensible modeling of how user interests evolve. This simple model can furthermore give some potential insight into user interests, for example that more popular items are popular due to being relevant for several different interests.

The model manages to find sensible states representing user interests at a given moment. Inspection of the fitted states suggests that items with high click probability appear to qualitatively related to each other. The fact that click probability is very much dependent on ad popularity does however mean that there can be some ads that seem out of place when ranking ads based on a states click probability. This means that while the states can give some useful insight into the dataset, and the general interests users have, they are less suited for making any predictions on the category of specific items.

The model predicts a plausible evolution of user interest in the extreme cases where the user either is very consistent or is exploring a varied selection of ads. The method occasionally falters, predicting spurious interest changes, when its less clear whether the user is exploring or trying to find something specific. This could be helped by more informative priors for high probabilities along the diagonal transition probabilities although this is a tradeoff against making the model less responsive to change.

There are however signs that the transition function of this model is too simplistic. The model has a very short memory, and a mismatch between the initial and the stationary distributions. This indicates that the transition function has a quite low impact on predictions and also that the model assumptions could be incorrect.

Since the model was trained as a recommendation system, it was also evaluated as such, after significant optimization of code efficiency. A version of the model was run live on the FINN site over a few weeks alongside other recommendation models developed by FINN generating 20% of recommendations in the category search of second hand cars. During this time recommendations from the model were shown around 100000 times leading to 32000 clicks in total. Roughly speaking the presented model achieved a click rate of 30% while the more mature FINN model achieved around 40%. The FINN model has been in production for a while beating out several other models as well. This online test was however performed with a suboptimal model due to time constraints and software issues. Testing a model using optimal hyper parameters and estimating the true selection probabilities (chapter 5) would potentially achieve better results. Potential avenues of improvement have been outlined.

References

[AK12] Gediminas Adomavicius and YoungOk Kwon. 'Improving Aggregate Recommendation Diversity Using Ranking-Based Techniques'. In: *IEEE Transactions on Knowledge and Data Engineering* 24.5 (2012), pp. 896–911. DOI: 10.1109/TKDE.2011.15.

[Cho+14] Kyunghyun Cho et al. 'Learning Phrase Representations using RNN Encoder-Decoder for Statistical Machine Translation'. In: *CoRR* abs/1406.1078 (2014). arXiv: 1406.1078. URL: http://arxiv.org/abs/1406.1078.

[DLR77] A. P. Dempster, N. M. Laird and D. B. Rubin. 'Maximum Likelihood from Incomplete Data via the EM Algorithm'. In: *Journal of the Royal Statistical Society. Series B (Methodological)* 39.1 (1977), pp. 1–38. ISSN: 00359246. URL: http://www.jstor.org/stable/2984875.

[Fun06] Simon Funk. Dec. 2006. URL: https://sifter.org/~simon/journal/20061211.html.

[Gol+92] David Goldberg et al. 'Using Collaborative Filtering to Weave an Information Tapestry'. In: *Commun. ACM* 35.12 (Dec. 1992), pp. 61–70. ISSN: 0001-0782. DOI: 10.1145/138859.138867. URL: https://doi-org.ezproxy.uio.no/10.1145/138859.138867.

[Hid+16] Balázs Hidasi et al. 'Session-based Recommendations with Recurrent Neural Networks'. In: (2016). arXiv: 1511.06939 [cs.LG].

[JHS05] A. Jasra, C. C. Holmes and D. A. Stephens. 'Markov Chain Monte Carlo Methods and the Label Switching Problem in Bayesian Mixture Modeling'. In: *Statistical Science* 20.1 (2005), pp. 50–67. ISSN: 08834237. URL: http://www.jstor.org/stable/20061160.

[KBV09a] Y. Koren, R. Bell and C. Volinsky. 'Matrix Factorization Techniques for Recommender Systems'. In: *Computer* 42.8 (2009), pp. 30–37. DOI: 10.1109/MC.2009.263.

[KBV09b] Y. Koren, R. Bell and C. Volinsky. 'Matrix Factorization Techniques for Recommender Systems'. In: *Computer* 42.8 (2009), pp. 30–37. DOI: 10.1109/MC.2009.263.

[Kor09] Yehuda Koren. 'Collaborative filtering with temporal dynamics'. eng. In: *Proceedings of the 15th ACM SIGKDD international conference on knowledge discovery and data mining*. KDD '09. ACM, 2009, pp. 447–456. ISBN: 1605584959.

[Li+10] Lihong Li et al. 'A Contextual-Bandit Approach to Personalized News Article Recommendation'. In: *Proceedings of the 19th International Conference on World Wide Web*. WWW '10. Raleigh, North Carolina, USA: Association for Computing Machinery, 2010, pp. 661–670. ISBN: 9781605587998. DOI: 10.1145/1772690.1772758. URL: https://doi.org/10.1145/1772690.1772758.

[LJ18] Malte Ludewig and Dietmar Jannach. 'Evaluation of session-based recommendation algorithms'. In: *User Modeling and User-Adapted Interaction* 28.4-5 (Oct. 2018), pp. 331–390. ISSN: 1573-1391. DOI: 10.1007/s11257-018-9209-6. URL: http://dx.doi.org/10.1007/s11257-018-9209-6.

[Mik+13] Tomas Mikolov et al. *Efficient Estimation of Word Representations in Vector Space*. 2013. arXiv: 1301.3781 [cs.CL].

[Sch+02] Andrew I. Schein et al. 'Methods and Metrics for Cold-Start Recommendations'. In: *Proceedings of the 25th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*. SIGIR '02. Tampere, Finland: Association for Computing Machinery, 2002, pp. 253–260. ISBN: 1581135610. DOI: 10.1145/564376.564421. URL: https://doi.org/10.1145/564376.564421.

[Sch96] Polaris Media Schibsted. *FINN*. 1996. URL: https://www.finn.no/ (visited on 09/02/2021).

[Spa] Apache Spark. *Collaborative Filtering*. URL: https://spark.apache.org/docs/2.2.0/ml-collaborative-filtering.html (visited on 14/04/2021).

[SSM12] Nachiketa Sahoo, Param Vir Singh and Tridas Mukhopadhyay. 'A Hidden Markov Model for Collaborative Filtering'. In: *MIS Quarterly* 36.4 (2012), pp. 1329–1356. ISSN: 02767783. URL: http://www.jstor.org/stable/41703509.

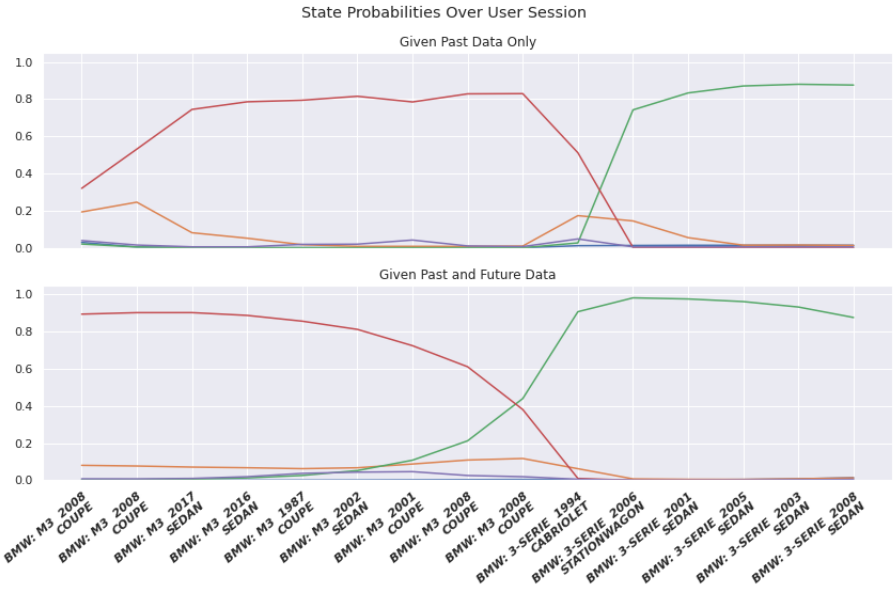
[ZH08] Mi Zhang and Neil Hurley. 'Avoiding Monotony: Improving the Diversity of Recommendation Lists'. In: *Proceedings of the 2008 ACM Conference on Recommender Systems*. RecSys '08. Lausanne, Switzerland: Association for Computing Machinery, 2008, pp. 123–130. ISBN: 9781605580937. DOI: 10.1145/1454008.1454030. URL: https://doi.org/10.1145/1454008.1454030.

Appendices

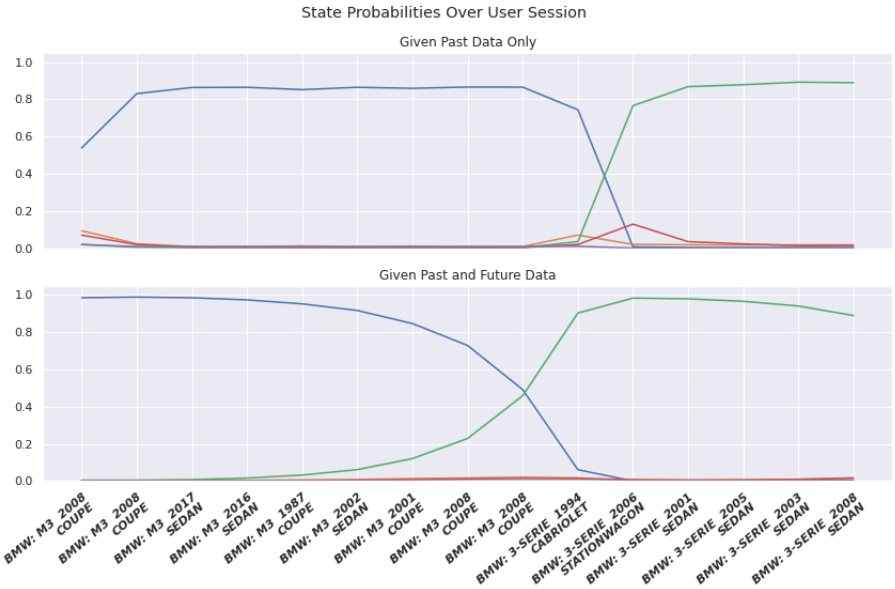
.1 User Session Examples

Session 1

300 States

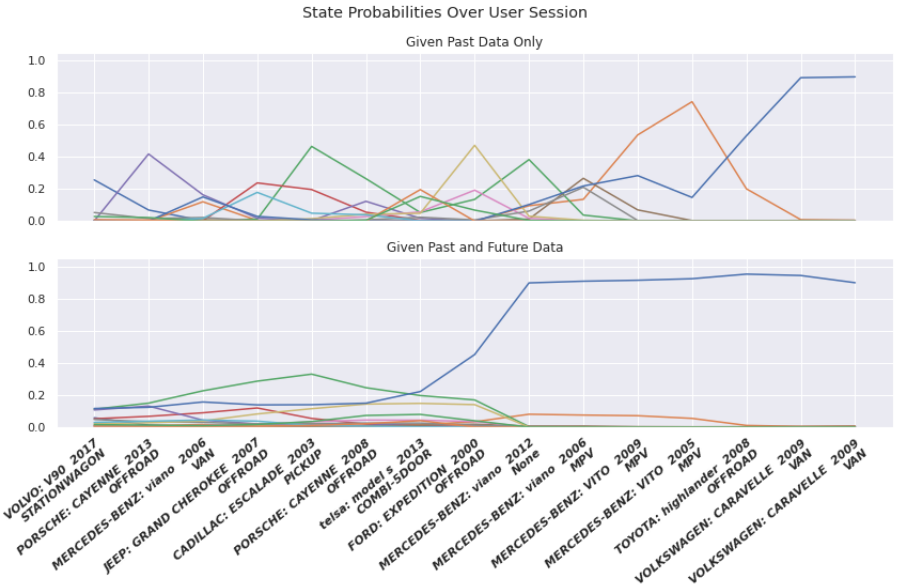


150 States

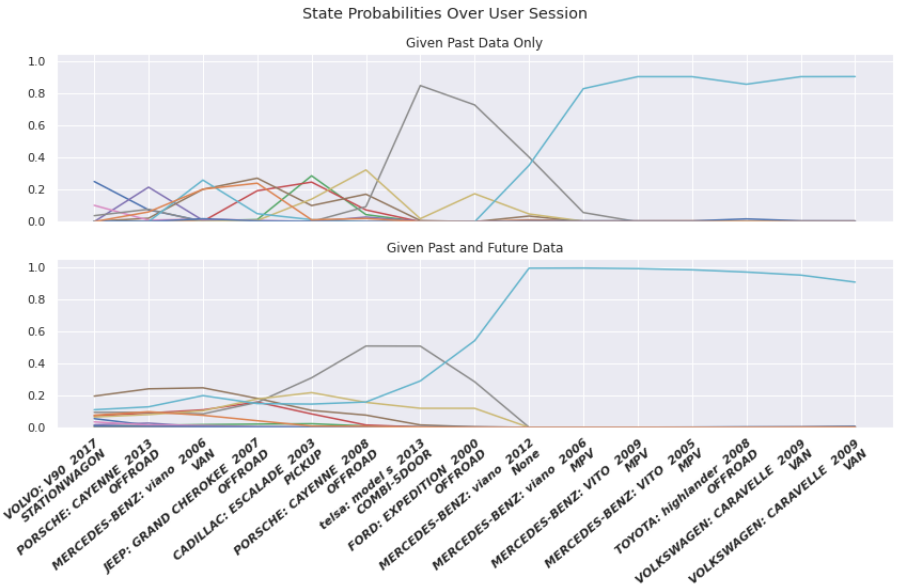


Session 2

300 States

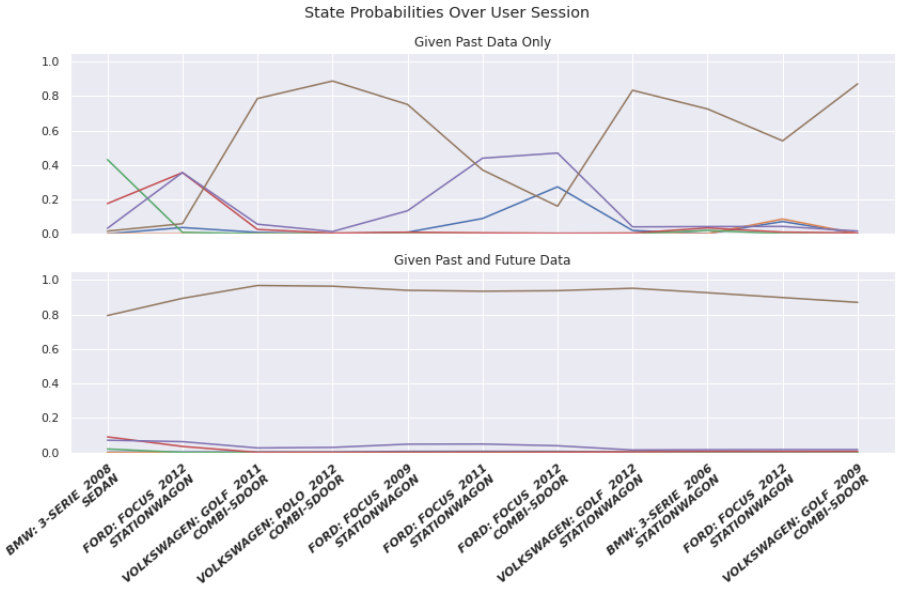


150 States

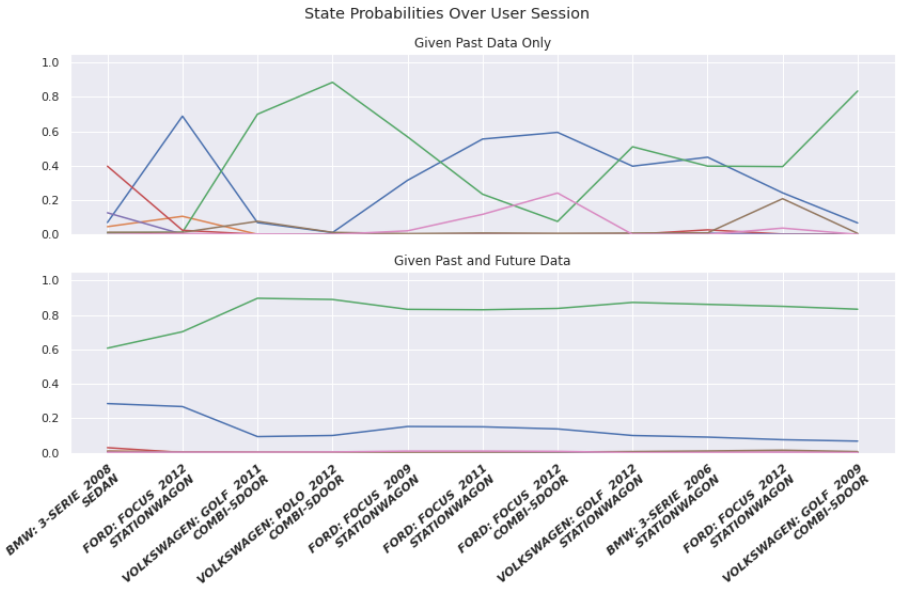


Session 3

300 States

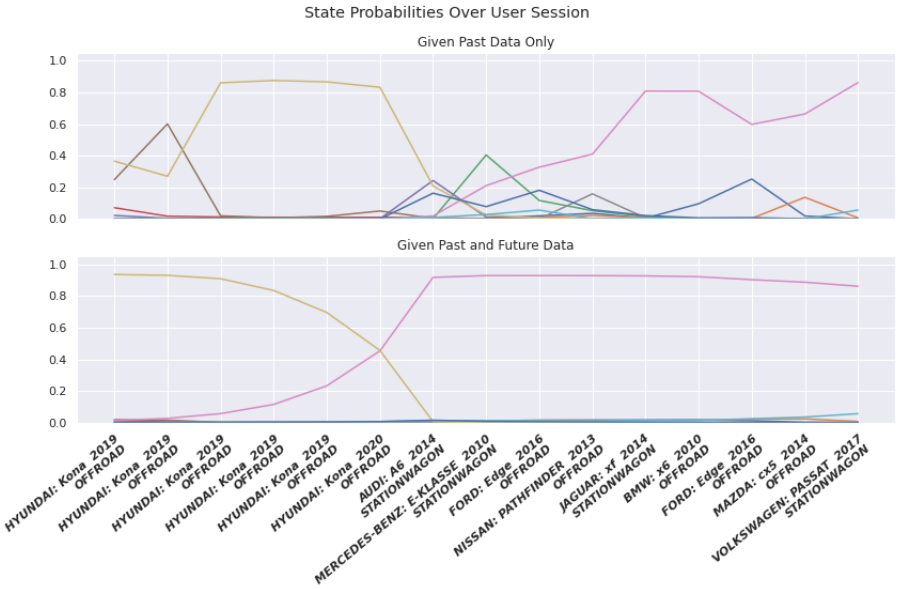


150 States

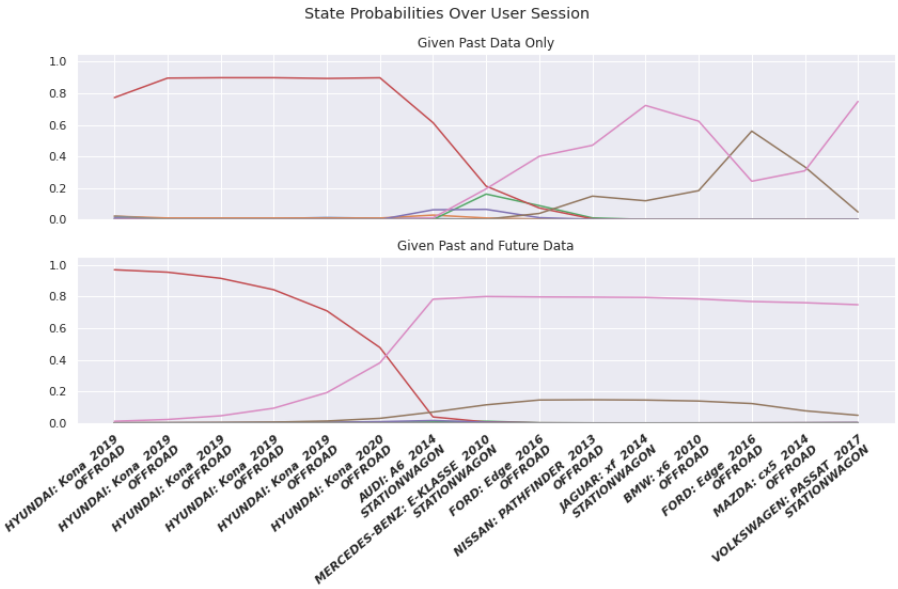


Session 4

300 States

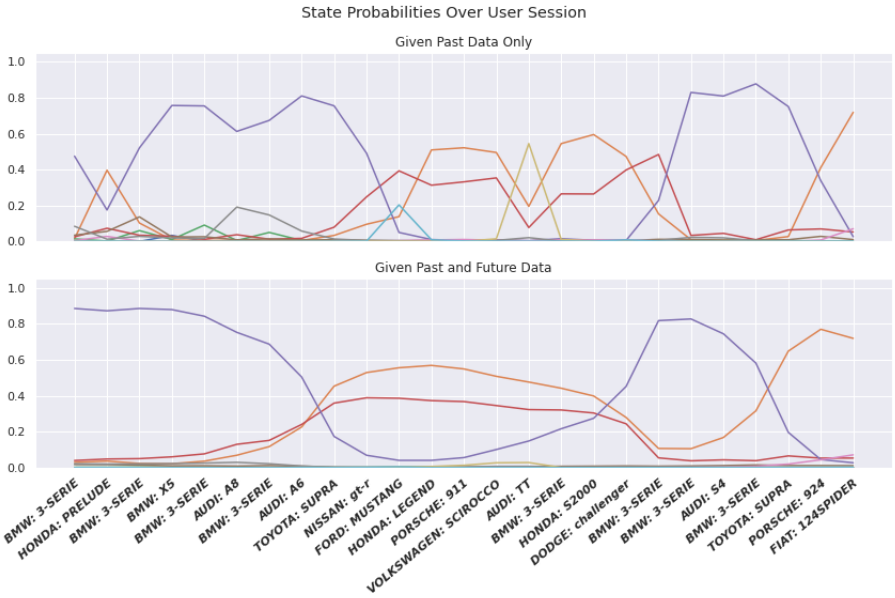


150 States

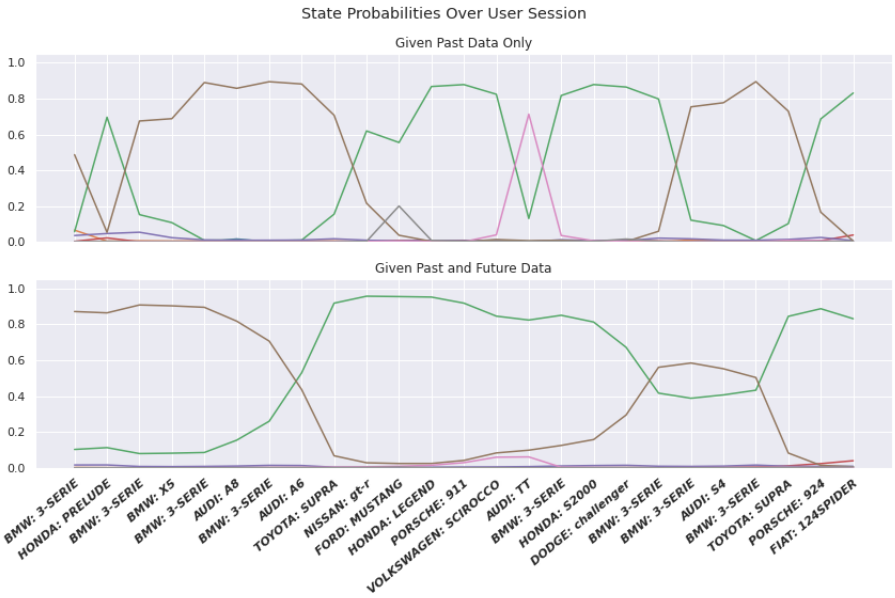


Session 5

300 States

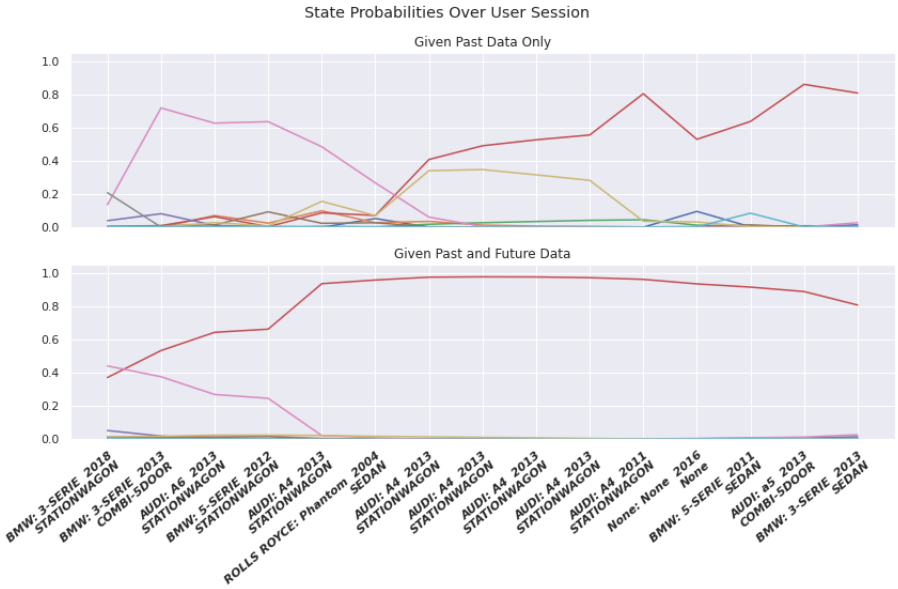


150 States

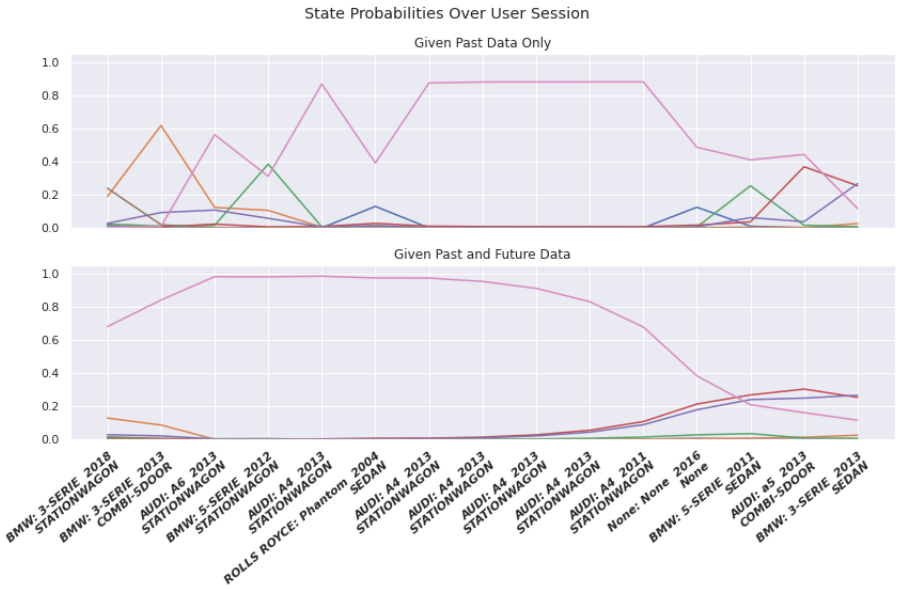


Session 6

300 States

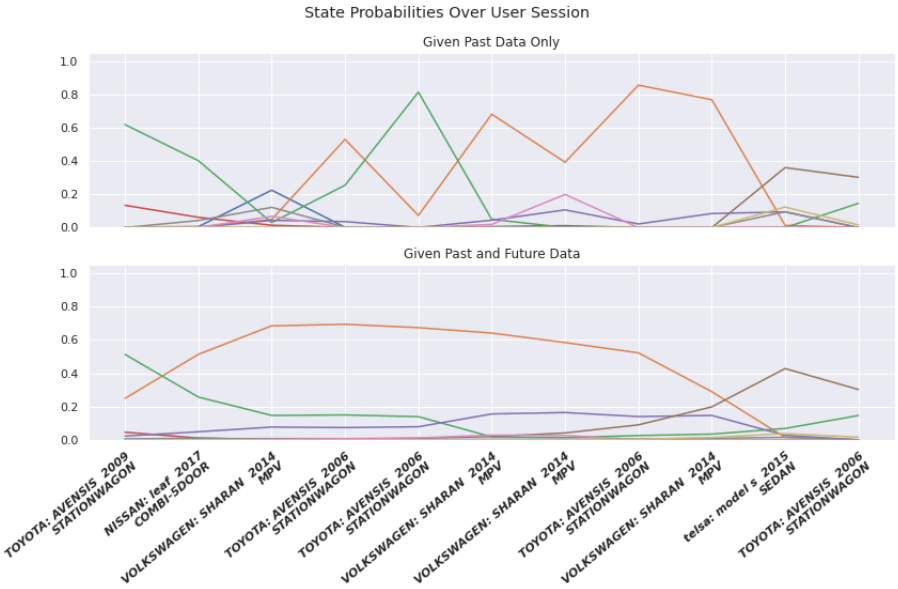


150 States

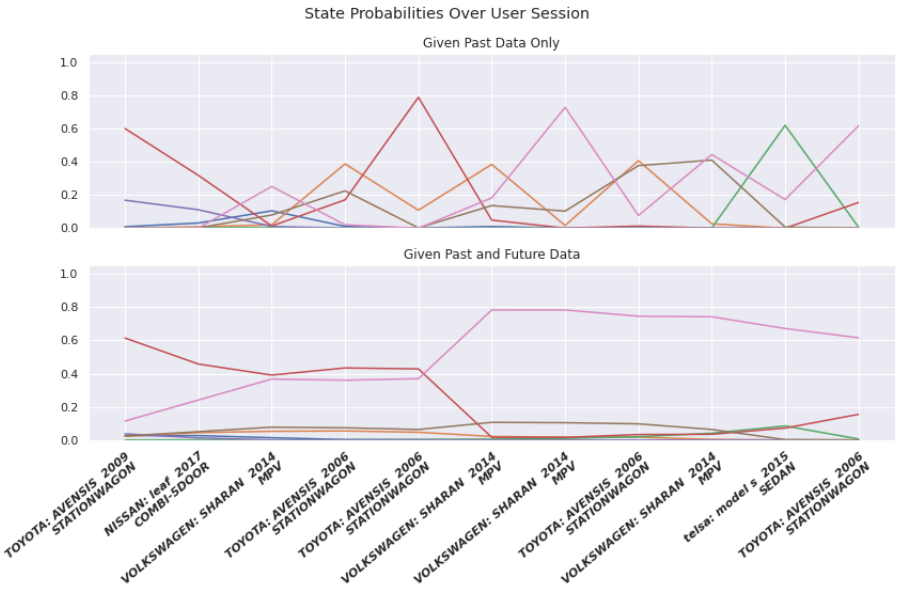


Session 7

300 States

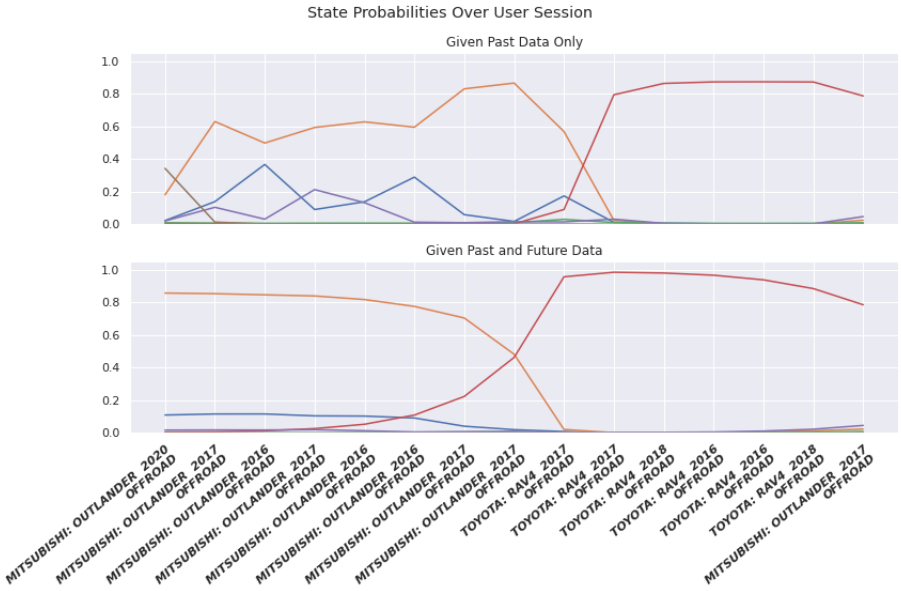


150 States

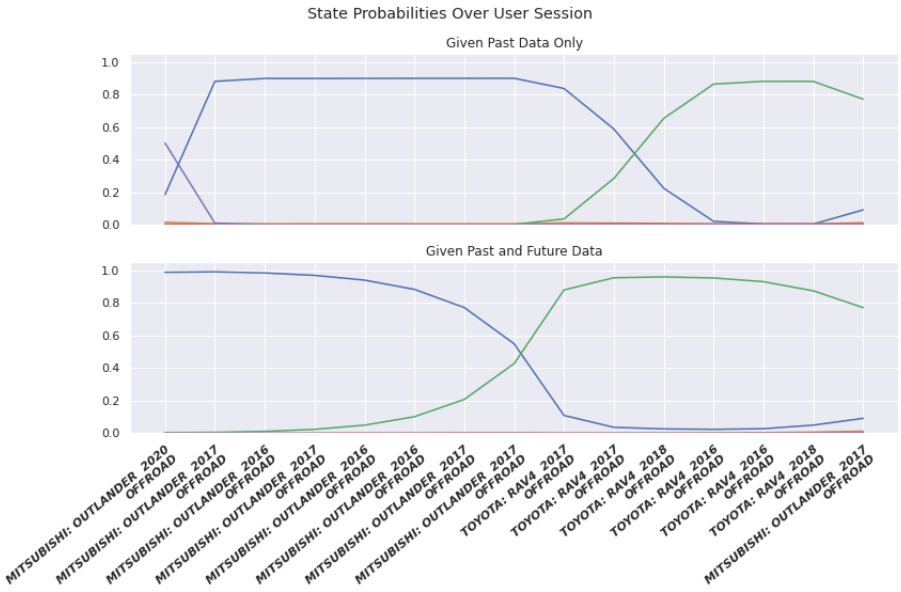


Session 8

300 States

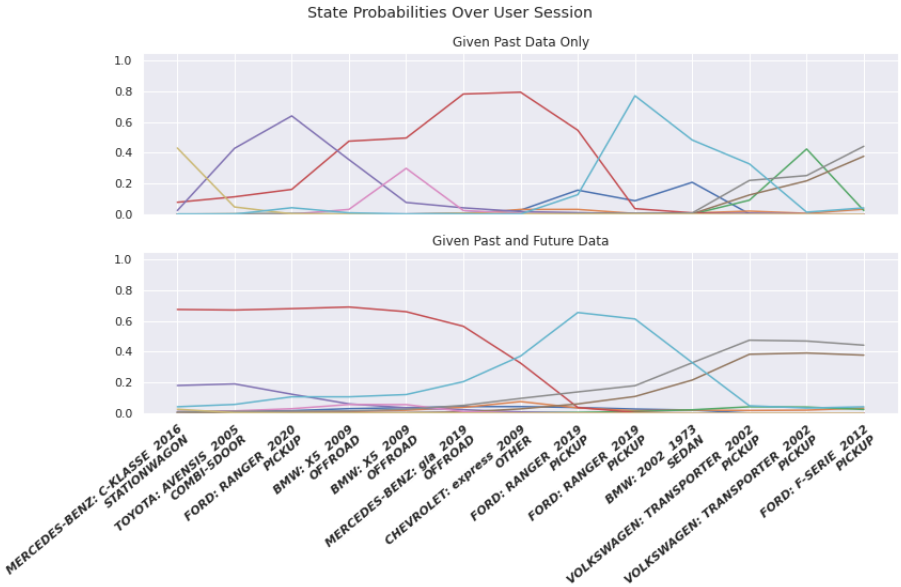


150 States

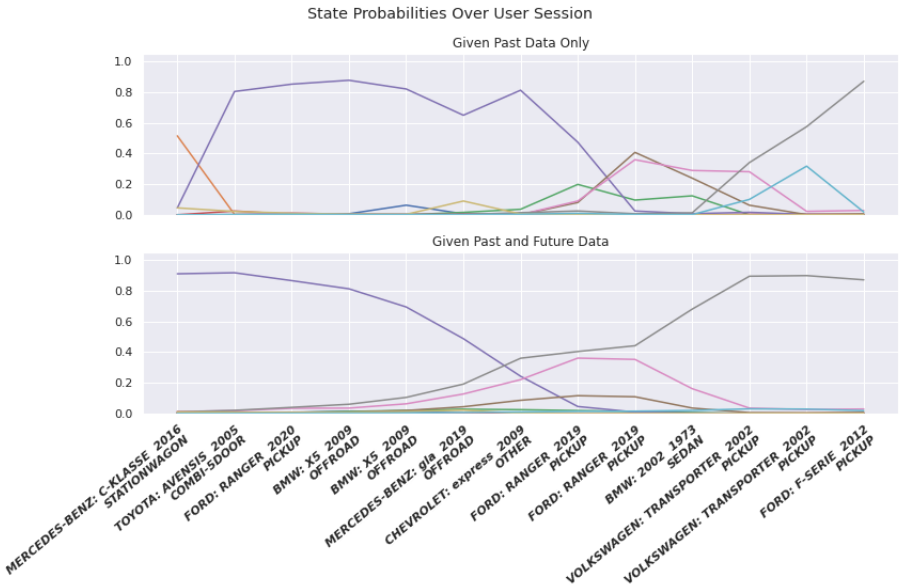


Session 9

300 States



150 States



.2 Implementation

Code written to fit the model and run simulation studies, as well as the various analysis of the model can be found on GitHub. This code does not include FINN specific functions to import data. Some functions do however assume data to be in the same format as that given by FINN. To fit this model to a new dataset one would therefore need to make some manual changes.

`https://github.com/oyboy/Hidden_Markov_Model_Recommendation_System`