ThisThat's user behavioural clustering for optimising distribution of public polls

MAXIMILIAN GIRKINS

4th Year Project Report
Computer Science

SCHOOL OF INFORMATICS

UNIVERSITY OF EDINBURGH

2019

Abstract

In this project we set out to build a fast and reliable recommender system to be integrated into the backend of ThisThat, an IOS polling application under development in Edinburgh, in order to improve user's poll recommendations and therefore increase user engagement on the platform. We face problems due to how users interact with the platform and certain business requirements which mean we can't rely entirely on existing solutions. We build a shallow autoencoder which is able to learn a latent model of user preferences from implicit feedback exploiting systematic confusion in the model to predict polls and employing a novel loss function that is able to take into account the confidence of implicit ratings. In testing our model is able to significantly outperform the current system used by ThisThat and achieve 67% recall in top-50 ratings, compared to 69% using the popular SVD++ approach. The final model is able to predict items with 50% accuracy to new users after as few as 10 interactions with the application.

Acknowledgements

Many thanks to my supervisors Pavlos Andreadis and Max Osborne for their advice and support throughout this project and for motivating me through the hard times. I would also like to thank my flatmates and friends for providing encouragement when bug hunting was lasting for days at a time.

4TH YEAR PROJECT REPORT

COMPUTER SCIENCE
SCHOOL OF INFORMATICS
UNIVERSITY OF EDINBURGH

CONTENTS

1

INTRODUCTION

In this project we set out to build a recommendation system for the ThisThat application in order to improve the user experience on the application, encourage users to spend more time on the app, answer more questions and therefore increase ThisThat's value to users.

Recommendation systems are designed to answer the question "What will a user like?" in order to provide users with a better experience and therefore increase time spent on applications such that the application providers earn more revenue from each user. Mainstream use of such systems can be seen in E-commerce sites such as Amazon,[1] social networks i.e. Facebook's news feed algorithm[2] and music streaming services such as Pandora.[3] Recommender systems have evolved from information retrieval (IR) systems[4] and approximation theory[5] to deal with the problem of best inferring utility of an item to a user. The recommendation problem can most simply be described as extrapolating to fill an incomplete $User \times Utility$ matrix (henceforth referred to as the $User \times Poll\_Utility$ matrix to suit our problem). These systems are very similar to IR systems in that they essentially answer an empty IR query for each user and try to provide the best results. This can of course be achieved in numerous ways, current approaches can broadly be split into 3 categories:[6] Content-based (recommending items that are similar to those a user has previously found useful), collaborative (recommending items that similar users have found useful) and hybrid methods (Amalgamating utility ratings from both content-based and collaborative methods either during utility inference or afterwards). These methods typically deal with explicit item ratings by users which are then used as the basis for recommendations, however ThisThat does not collect explicit ratings for how much a user enjoyed viewing a poll so we must explore methods to use implicit feedback in some way as an alternative for explicit ratings. These methods can be broadly useful for any application that collects implicit user feedback about items such as online communities like Reddit[7] where users can comment (implicit) and vote (explicit) on items or Instagram[8] which again collects both implicit and explicit feedback. We believe that these ap-

[1] Linden *et al.*, 'Amazon.com recommendations: item-to-item collaborative filtering', 2003.

[2] Kabiljo & Ilic, *Recommending items to more than a billion people*, 2018.

[3] Howe, 'Pandora's Music Recommender', 2009.

[4] Belkin & Croft, 'Information Filtering and Information Retrieval: Two Sides of the Same Coin?', 1992.

[5] Powell, *Approximation theory and methods*, 1981.

[6] Adomavicius & Tuzhilin, 'Toward the next generation of recommender systems: a survey of the state-of-the-art and possible extensions', 2005.

[7] Reddit.com

[8] Instagram.com

plications can benefit from adding implicit feedback as a source of information for recommendations in order to provide more detailed item ratings which will aid in making recommendations more accurate.

1.1 *ThisThat*

ThisThat is an IOS polling application currently being developed in Edinburgh which seeks to provide a platform for users to ask and answer binary questions - either 'this' or 'that' - in order to gather feedback on ideas and promote discussion around topics using a commenting mechanism. The importance of a good recommendation system for ThisThat is twofold, firstly by maximising the utility of the polls which a user is shown it is hoped that user engagement will increase as they enjoy time spent on the app more, this is a clear financial incentive for the firm. Secondly, if recommendations provided to users are accurate then users should mostly interact with polls they have some knowledge about which should lead to better discussions on a topic.

The ThisThat application presents users with one binary question at a time in the form of an onscreen card, including images, which the user can swipe left or right to indicate their answer. The user has a few other ways to interact with these questions, namely commenting, tracking[9], skipping, or thinking[10]. All of these engagement mechanisms provide implicit feedback about a question which we can use to infer utility to a user and therefore fill in part of the $User \times Poll\_Utility$ matrix to improve extrapolation over the rest of the matrix.

Figure 1.1: ThisThat poll flow.

[9] By tracking a question a user is able to follow the responses to that question and see the binary split of answers as a percentage either way.

[10] The "thinking" option allows a user to dismiss a question until a later date where it will reappear in their feed.

1.2 *Problem Description*

The focus of our project is to build a system that is able to learn from implicit feedback to predict polls to users in some way. We must explore methods by which we can best integrate implicit information to learn a user's enjoyment of a poll such that the interaction encodings are most useful to the system. Knowing that a user has interacted with an item does not necessarily imply that a user likes an item therefore we must find a way to learn whether a user likes an item from their interactions. We hypothesise that certain interactions with polls will provide more evidence than others to suggest a user enjoys a poll and that if multiple similar users are interacting with the same poll then the probability that each similar user enjoys the poll is higher than if only that user has interacted with the poll. We must therefore try to learn the probability that a user will enjoy a poll given the combined interactions over all other polls on the system.

1.3 Requirements

The motivation behind the project is to build a system that meets the business requirements of ThisThat. Due to the many different approaches available in the task of building a recommendation system it is important to narrow down what the main aims for the system are. There are trade-offs between accuracy, speed and scalability that must be balanced in order to best fit the requirements. The main requirements of the project are laid out below, which we use in the Related Works section (2.2) in order to guide our literature search and decision making processes. During testing of our model in the Experiments section (4.2) we devise experiments which test our model against these requirements to ensure that they are met.

- **Provide relevant recommendations to users.** Users should be shown polls which they will want to engage with and that will encourage discussion. Engagement here refers to positive interactions with a poll, ie. voting, commenting or tracking, but not skipping. The concrete measure of engagement that we will use is elaborated further in the Solution chapter (3). Recommendations should be 'good enough'[11], Due to the low friction nature of skipping questions on the app, it is not too important that every recommendation be perfect as it is so easy to dismiss a recommendation. The aim is to provide engaging recommendations without sacrificing on speed and efficiency.

[11] A high proportion of a user's suggestions should prompt interaction but we are not too concerned about false positives

- **Poll recommendations should not lead to biased sampling of opinion.**

  By learning which users are likely to engage with which polls it is important to try and avoid biasing the recommendations to favour a certain group of users, for example if a group of users all share an interest in Mexican food and a poll asks users to decide whether Mexican food is better than Italian food then despite knowing that we have a group of users who are likely to engage with a poll about Mexican food we should avoid solely showing this poll to them but must instead try to balance out which users see the poll such that the recommender system does not change the outcome of a poll[12].

[12] We want our system to send more users to relevant polls so the total number of responses increases, but the balance of votes should remain largely unchanged compared to if users were shown a chronological set of poll recommendations

- **Ensure that new polls are answered quickly and that users are able to boost their polls for greater engagement if they wish.**

  It is important for users posting polls to quickly receive feedback on their questions. We must therefore ensure that new polls are favoured by the recommender system. Boosting a poll is an option in the ThisThat app that allows users to pay a small amount of cryptocurrency[13] to have their polls shown to more users than a standard poll, the recommender system has to be able to incorporate this functionality such that boosted polls are not just shown to

[13] In this case Kin

a larger number of random users but to a wider audience of users who may be interested in the poll.

- **Allow users to control the degree by which they are affected by algorithmic curation.** Filter bubbles have become an increasing worry in today's society due to the widespread use of algorithmic curation and recommendation. It has become mainstream to worry about the polarizing effects of being stuck in such a filter bubble, articles such as Rader & Gray[14] point out awareness and attitudes to the filter bubble problem, with some users going so far as to actively engage with platforms in a random 'noisy' way in order to try and break out of their own bubbles. The ThisThat team believes in transparency and allowing users to control the degree to which algorithmic recommendations control which polls they will see. Therefore it is important to allow users to exert control over the degree to which recommendations are tailored to them personally. It is also hoped that this will foster more varied and engaged debates on the platform which is a stated goal of ThisThat.

[14] Rader & Gray, 'Understanding User Beliefs About Algorithmic Curation in the Facebook News Feed', 2015.

- **Ensure the system is able to adapt over time to changing user preferences and new polls.** A user's opinion is not a static thing,[15] it may change over time due to circumstance and experiences. It is important that the recommendation system be able to handle changing user preferences over time in order that recommendation quality does not degrade the longer a user is on the platform as their opinions shift.

[15] Friedkin & Johnsen, 'Social influence and opinions', 1990.

- **Allow ThisThat to cluster users based on behaviour for further business purposes.** The ThisThat team should be able to take any model we create regarding user preferences and easily infer similarity between users for potential advertising purposes if they wish to do so.

- **Build a system that can scale to allow thousands/hundreds of thousands of users.** As the user base of ThisThat grows it is important that the underlying platform is able to scale to accommodate these extra users without a degradation in the service provided. We must therefore ensure that our system is fast[16] and reliable for some large number of users in order to ensure the continued positive impact of the project.

[16] By this we mean not notably causing a bottleneck at inference time.

1.4 Project Summary

Given the requirements laid out above we first examine the data set provided to us for training by ThisThat in the Data section (2.1). Next we explore current and emerging approaches to recommendation systems which may be relevant or adaptable to our specific task in the Related Works section (2.2) while critiquing the various pros

and cons of each approach in relation to our requirements. Having completed this we begin to elaborate on our chosen solution in the Solution chapter (3) whilst providing more specific information about our model and how we process the data available to us to best provide recommendations. Next, in the Testing chapter (4) we run tests on our models against ThisThat's current recommendation approach and a popular matrix factorisation based technique (SVD++[17]) in order to verify that it does indeed meet the requirements set for us and to look for any shortcomings that we may need to address. Finally we evaluate the success of the project as a whole and discuss challenges that were faced in the project that might lead to further work in the future.

[17] Kumar *et al.,* 'Social popularity based SVD++ recommender system', 2014.

Our final solution employs a shallow autoencoder architecture using User-User collaborative filtering which takes implicit input and learns a latent model of user similarity in order to provide recommendations. The model learns to recommend polls by exploiting systematic reconstruction errors in the autoencoder output caused by coercing the data into a lower dimensional latent space. In testing we are able to achieve 67% recommendation recall on test data which outperforms ThisThat's current recommendation approach's 22% recall on the same data, and is sufficiently close to the 69% accuracy result achieved by SVD++ to be considered successful in this measure. The model we build falls under the category of collaborative filtering although as we discuss in the Conclusion chapter (6) we could hybridise the system in order to take into account poll content as well and perhaps improve recommendation recall further. We adapt methods from Strub *et al.*[18] to our data and providing a methodology for applying their work to User-User collaborative filtering on implicit data using a novel loss function and some post-processing steps rather than Item-Item collaborative filtering using explicit ratings.

[18] Strub *et al.,* 'Hybrid Collaborative Filtering with Neural Networks', 2016.

2

CONTEXT

In this chapter we perform a brief investigation into our data set, followed by a survey of some current approaches broken down by Adomavicius & Tuzhilin's[1] classification into content-based, collaborative and hybrid methods. We also look into evaluation metrics used for recommender systems, and discuss the differences between implicit data and explicit data.

[1] Adomavicius & Tuzhilin, 'Toward the next generation of recommender systems: a survey of the state-of-the-art and possible extensions', 2005.

2.1 Data

The data provided by ThisThat includes information about 771 Users, 570 polls, user interactions with polls and information on which users follow each other. We have information about which users follow each other which could be used to build a social network based recommender system. We also have information about polls including poll questions, poll answers and the pictures that go with the polls, this lends itself to content-based methods. We could encode information from the questions and answers in order to infer poll similarity and learn what user preferences are however on inspection of the data we learn that the mean length of the questions is 5 words and the mean answer length is 3 words giving us a mean total information of 11 words from the question and two answers. Images from each poll may be available on request but running image recognition algorithms for tagging may be beyond the scope of this project and is left for future work. We also have information on user interactions with polls, namely comments, votes, skips, ownership and tracking. In the training data we have been provided we can see that the average user has interacted with 50 polls out of a possible 570, This shows promise for collaborative filtering methods based on the implicit feedback gained.

2.2 *Related Works*

Research in the field of Recommendation systems is broadly split into three categories:[2] Content-based, Collaborative filtering and hybrid methods. In our search we shall explore both commonly used, mature approaches as well as more novel and perhaps niche approaches to the problem in order that we might cherry pick strategies from different areas that may be adapted to suit our specific problem domain. We shall present a brief introduction to the various approaches in this section while also critiquing them in relation to our requirements.

[2] Zhang *et al.*, 'Deep learning based recommender system: A survey and new perspectives', 2017.

There is an important distinction between implicit and explicit user ratings which must be accounted for. Explicit ratings of items involve a user providing feedback on items such a star ratings in the case of amazon product reviews. Implicit ratings do not necessarily define whether a user likes an item, for example time spent on a website, we cannot infer enjoyment of an item just because a user has interacted with it as - in the case of time spent on a site - for example the user may have just left their browser open, or may be researching a project without actually enjoying the content of the page[3]! Hu *et al.*[4] propose using implicit feedback as a means to extract a probability that a user enjoys an item, broadly speaking the more positive implicit feedback gained the greater the probability that a user enjoys an item. We think adapting this approach will be useful for our problem given the varying types of interaction that we hypothesised will tell us more or less about a user's preferences.

[3] Here we expect time spent on the page to correlate with usefulness not enjoyment

[4] Hu *et al.*, 'Collaborative filtering for implicit feedback datasets', 2008.

In the process of recommending items to users, there are two common problems which appear: the new user problem, and the new item problem. The new user problem is simply explained by the question "How do you recommend items to a user when you have no information about them?", Clearly it is impossible to effectively recommend items to users before they have interacted with or provided feedback on any other items for you to base the recommendations on. In our case, how do we extrapolate to fill a row of the $User \times Poll\_Utility$ matrix given an empty row to start with? This problem can be skirted in a few ways: Recommending the user random items in the beginning until enough information is available to improve recommendations, recommending items that are popular among all existing users[5] in the hope that the new user will also enjoy them until enough information is available to provide curated recommendations, or choosing a representative sample of all items in order to most quickly ascertain which type of items a user naturally gravitates towards.[6] The new item problem affects only collaborative filtering recommendation methods. The new item problem can be seen as "How do we recommend an item to a user without knowing anything about the item?" This problem does not affect content based methods as content based methods are based on some encoding of

[5] Geurts & Frasincar, 'Addressing the cold user problem for model-based recommender systems', 2017.

[6] Aleksandrova *et al.*, 'Identifying representative users in matrix factorization-based recommender systems: application to solving the content-less new item cold-start problem', 2017.

the item content and we can't have an empty item. Collaborative filtering methods struggle with the new item problem though as it is impossible to know what type of user will like an item when no other users have yet rated the new item. Strategies for dealing with the new item problem revolve around how to most quickly gather useful feedback about a new item in order to model its utility to different users.

It is difficult to test recommender systems that work using implicit information as we are unable to test whether users enjoy the items that they are presented without using proxies for enjoyment such as watch time for movie recommendation algorithms. Herlocker *et al.*[7] give a detailed comparison of different evaluation measures including Mean absolute error (MAE), Precision and Recall. In their evaluation they categorise different measures based on the prediction task at hand. Our task can be categorised accordingly as a 'find good items' task in which it is important to recommend good items to a user but not necessarily in a particular order. The 'find good items' task is one in which novel items are desirable and as such it is difficult to test in an offline manner when a users rating is unknown for these novel suggestions. Herlocker *et al.* suggest that using leave-n out testing with classification based metrics is favourable for this task as users either enjoy an item or not but are not so concerned with relative ordering such as in a task when a user must pick one or a small number of items from top suggestions such as a google search query. Using leave-n out testing involves hiding some previously rated items from the testing set and then evaluating how many of these hidden items appear in the systems recommendations using measures such as Precision and Recall. For top-k recommendations recall is often used so we shall do the same when evaluating our own models.

[7] Herlocker *et al.*, 'Evaluating collaborative filtering recommender systems', 2004.

2.2.1 *Content-based methods*

Content based methods rely on knowledge about the items you are trying to recommend. These methods can essentially be thought of as trying to place items accurately in an n-dimensional space, locating users in the same space and then finding some sort of distance between a user and all items from which you then recommend the closest items to the user. Or framed in a different way, they perform a function over the User's row in the $User \times Poll\_Utility$ matrix to fill in the empty numbers. These methods thus face three main challenges: how to position items in the space, how to position users in the space, and how to measure distance between users and items.

Content based approaches rely on some form of item encoding, be that latent factors of a song[8], tf-idf vector representations of web pages[9] or linguistic phrase based representations[10] to achieve the same task. Once a suitable item encoding is found we can em-

[8] van den Oord *et al.*, 'Deep content-based music recommendation', 2013

[9] M. Pazzani & Billsus, 'Learning and Revising User Profiles: The Identification of Interesting Web Sites', 1997

[10] Furnkranz *et al.*, 'A case study in using linguistic phrases for text categorization on the WWW', 2014

ploy methods such as Bayesian filtering[11], cluster analysis[12], decision trees[13], and neural networks[14] to effectively recommend those items to users by user-item comparison. These approaches do not suffer from the new item problem which is a useful feature to have however we focus our research on collaborative filtering approaches first[15] for the following reasons:

- The data provided by ThisThat shows that the average poll contains just 11 words in total. From Heap's law,[16] which states that the number of unique words in a corpus is roughly equal to the square root of the total words we suspect that this may be too few words to produce an effective content based system. Phrased in a different way, the probability that a word is not yet in a corpus reduces as the corpus becomes larger. As we have only 11 words for each poll therefore our corpus of words is small given the size of our entire data set, thus the likelihood that a new poll contains novel words which are useless[17] to a content based recommender is high.
- The second reason is that due to the nature of the polls which have binary options we suspect that it will be difficult to fulfill the bias requirement laid out in 1.3 as for example a poll asking users if they prefer pizza or pasta would recommend that poll to users who have shown interest in polls about pizza in the past but would not necessarily be able to recommend it to users who enjoy pasta if no other polls had involved pasta. Even if there were polls including pasta, the system may have different weightings for the importance of pizza and pasta so would not be able to recommend the poll comparing the two to a representative sample of the user base.

[11] de Gemmis *et al.,* 'Integrating Tags in a Semantic Content-based Recommender', 2008

[12] Iacobucci *et al.,* 'Recommendation agents on the internet', 2000

[13] Cho *et al.,* 'A personalized recommender system based on web usage mining and decision tree induction', 2002

[14] Zheng *et al.,* 'Joint Deep Modeling of Users and Items Using Reviews for Recommendation', 2017

[15] We later discuss how we might hybridise our model to gain the advantages from both approaches

[16] Egghe, 'Untangling Herdan's law and Heaps' Law: Mathematical and informetric arguments', 2007.

[17] If the system has not yet seen the word then it won't be able to encode the word in any way that is comparable to the rest of the corpus

2.2.2 *Collaborative filtering approaches*

Collaborative filtering (CF) uses the idea that similar users like similar things to recommend items. CF seeks to find out how similar users are and then in some way recommend the items to a user that the most similar users have already enjoyed. Collaborative filtering can be broken down into memory-based and model-based approaches, and further still into User-User and item-item approaches. All these methods take the $User \times Poll\_Utility$ matrix and use information gained about a column of polls from a group users to fill in the expected value of that poll for users who have not yet rated that poll.

One of the earliest CF recommender systems was Grouplens[18] which was used to filter Usenet[19] news. This system used a table of ratings from different users for different articles, and a table of correlations[20]. In order to provide recommendations the system then simply performs a weighted sum over all relevant users[21] This is

[18] Konstan *et al.,* 'GroupLens: Applying Collaborative Filtering to Usenet News', 1997

[19] Wikipedia contributors, *Usenet — Wikipedia, The Free Encyclopedia,* 2019

[20] In this case Pearson correlation coefficient (with a few tweaks to avoid spurious correlations between new users), but other measures such as vector cosine similarity can be used as well.

[21] The system partitions users by newsgroup in order to improve correlation and reduce the number of relevant users.

an example of a User-User memory based approach to the problem as the system relies on user similarity. Memory based approaches use an aggregate of user ratings to perform prediction. This system showed the predictive power of collaborative filtering although it has some drawbacks for our task. The correlation table has to be updated every so often to take into account changing user preferences, and it relies on explicit data to perform the weighted sum. Furthermore the system does not take into account the tendency for some users to rate items higher on average than others which may harm the prediction process.

Item-Item collaborative filtering approaches are in some ways similar to content based recommender systems, in that we infer similarity of items in order to show a user new items that resemble previously highly rated items. The difference, which makes this a collaborative filtering approach, is that we do not use an item encoding based on content, but instead items are encoded by which users have rated them highly. This can often be preferable to User-User methods if a system has a larger number of users than items which would cause User-User systems to be less performant. An example of this which has been scaled to millions of items has been used by Amazon[22] to recommend products to users. The item-item based approach shows promise for our task as our data suggests that we have more users than items, however we suspect that the data is skewed by a high churn rate[23] due to the nature of an early stage startup. We predict therefore that in the future the platform will have many more polls than users as posting a poll is a common activity on the application. From this we focus our research on User-User approaches.

[22] Linden *et al.*, 'Amazon.com recommendations: item-to-item collaborative filtering', 2003.

[23] Wikipedia contributors, *Churn rate — Wikipedia, The Free Encyclopedia*, 2019.

Miyahara & M. J. Pazzani[24] Describe using a Bayesian model based on positive and negative feedback about items which leads to strong recommendation performance, we see that this approach could be useful to our problem given that we have engagements such as comments and tracking which can be associated with positive ratings, and skipping which can be associated with negative ratings. This approach is advantageous in it's simplicity however it appears to degrade in prediction accuracy after a certain number of ratings (peaking at 68.5% correctly predicted items on test data) and it also uses explicit ratings which we do not have available.

[24] Miyahara & M. J. Pazzani, 'Collaborative Filtering with the Simple Bayesian Classifier', 2000.

One common problem of collaborative filtering methods is that data is often sparse about which items users have rated which leads to difficulty in finding overlap between users to effectively infer similarity. Ungar & Foster[25] propose to address this problem by clustering similar users using statistical K-means clustering such that they can infer user similarity by how well users belong to different classes. They note however that while this approach goes some way to alleviating the sparsity problem, some data sets that they tested on were too sparse and the model degraded rapidly. We note that our training data contains on average 50 engagements for each user over 570

[25] Ungar & Foster, 'Clustering methods for collaborative filtering', 1998.

polls which, while moderately sparse does not seem to warrant too much attention paid to sparsity reduction measures.

The above models do not seem to scale well to many users due to inference being run over all users in the data sets, Chee *et al.*[26] try to overcome this problem by first clustering users in a similar way to Ungar & Foster and then using a divide and conquer decision tree method over these clusters to perform prediction inference over a smaller subset of users. They note that this improves accuracy of recommendations as the predictions are not biased at all by users who share very little commonality with the target user. This approach seems favourable to our problem as it could add some explainability to recommendations, however we do not see a simple route to optimise the model for implicit feedback.

[26] Chee *et al.*, 'RecTree: An Efficient Collaborative Filtering Method', 2001.

One approach popularised by the Netflix prize[27] is to use matrix factorisation and singular value decomposition on the $User \times Poll\_Utility$ matrix, Funk[28] proposed in a now famous blog post an approach to the netflix prize in which he factorized the $User \times Poll\_Utility$ matrix into two lower dimensional matrices, the first has a row for each user, and the second has a column for each item. These lower dimensional matrices represent latent factors of the information which can be recombined to provide predictions to a user. The original system was not designed to handle implicit ratings however Kumar *et al.*[29] have elaborated a solution to enable the same techniques to be applied to implicit ratings. While we believe that matrix factorisation would be a viable solution to our problem, we note there are other methods for discovering the latent factors required to perform accurate prediction. One problem with SVD is that it learns a linear matrix factorisation, this assumes that a users ratings are consistent over all polls however we believe this not to be the case as users may engage with polls in different ways depending on the environment in which they are using the app. We therefore look into autoencoders to learn a non-linear latent representation of user preferences.

[27] Bennett, Lanning *et al.*, 'The netflix prize', 2007.

[28] Funk, *https://sifter.org/ simon/journal/20061211.html*, 2006.

[29] Kumar *et al.*, 'Social popularity based SVD++ recommender system', 2014.

In 2015 Sedhain *et al.*[30] proposed a novel approach to recommendations which outperforms adapted matrix factorisation approaches such as the locally-low rank model proposed by Lee *et al.*.[31] The model also has the advantage of being fast at inference time and being able to learn a nonlinear latent representation due to the nonlinear activation function in the network whereas Funk's Matrix factorisation only learns a linear representation. The autoencoder network architecture works by having some number of encoder layers which become progressively smaller, followed by the same number of decoder layers which become progressively larger. The network is trained such that the input is the same as the expected output. From this the model must learn how to represent the input data in a lower dimension as it is squeezed through the middle layers such that it can be accurately reconstructed by the decoder. Autoerec uses an

[30] Sedhain *et al.*, 'Autorec: Autoencoders meet collaborative filtering', 2015.

[31] Lee *et al.*, 'Local Low-Rank Matrix Approximation', 2013.

Item-Item approach with one model per item. In order to best train these autoencoders Strub *et al.*[32] mask off a portion of the training data so that the model must learn to reconstruct data that doesn't exist, and therefore better predict unknown items. We see this approach as being most promising for our system due to the speed of prediction and adaptability.

[32] Strub *et al.,* 'Hybrid Collaborative Filtering with Neural Networks', 2016.

2.2.3 *Hybrid approaches*

Hybrid approaches are attractive because they promise some of the scalability benefits of collaborative filtering approaches while using content information to mitigate the new item problem. Adomavicius & Tuzhilin[33] break down the category of hybrid recommenders into 4 subcategories:

[33] Adomavicius & Tuzhilin, 'Toward the next generation of recommender systems: a survey of the state-of-the-art and possible extensions', 2005.

The first is to build one content-based system, and one collaborative filtering system then to merge the recommendations of the two based on some metric. Claypool *et al.*[34] achieve this through a linear combination of the ratings from collaborative and content-based methods. While this seems attractive in it's simplicity it may also be wasteful in terms of computational resources.

[34] Claypool *et al.,* 'Combing content-based and collaborative filters in an online newspaper', 1999.

The second approach is to add content information to collaborative methods, this is the approach taken by Strub *et al.*[35] in which content information is added to the autoencoder.

[35] Strub *et al.,* 'Hybrid Collaborative Filtering with Neural Networks', 2016.

Thirdly we can add collaborative data to content based methods, M. J. Pazzani,[36] address data sparseness by first removing items from the collaborative filtering input that do not correspond well on a content level to each user and then determining correlation between users using collaborative methods.

[36] M. J. Pazzani, 'A Framework for Collaborative, Content-Based and Demographic Filtering', 1999.

Finally, methods exist to create a unified model involving both content and collaborative information at the same time, Condli *et al.*[37] propose a bayesian model which is able to infer the probability that a user will like an item given all the previous items that a user has rated and those ratings. While we believe that adding content information to our model would improve prediction accuracy and help alleviate data sparsity problems that may arise, we also think that the potential problem of introducing bias (from content information as in 1.3) into the system is not one we want to face at this early proof of concept stage so we leave this for future work.

[37] Condli *et al.,* 'Bayesian mixed-E ects models for recommender systems'.

3

SOLUTION

3.1 Overview

After having reviewed current solutions and examining the data we have available to us we decide to settle on using an autoencoder architecture as a starting point for our system. This approach learns a generalised model of user preferences about polls which is able to adapt over time through occasionally continuing training.

The autoencoder architecture (see figure 3.1) is a network architecture that is trained using the same input as expected output. The neural network is made up of an encoder section and a decoder section, the encoder layer(s) take input and reduce that data into a lower latent dimension by using a 'squashing' layer that has a low number of neurons in it. The decoder's task is to try and reconstruct the original data from the squashed data. The error is given by how different the output is from the input and training proceeds from here. Over time the network learns to reduce the amount of information lost in the squashing layer such that reconstruction error diminishes.

In order to recommend items to users using this architecture we exploit the errors produced by the encoder-decoder operation. By training the network on data about many users it learns a generalised model of what user preferences look like, but is not necessarily able to entirely capture extremely specific information. For example, if we train the network on three users, two of which engaged with polls **A** and **B** and the third who engaged with polls **A** and **C** the network (given we constrain the squashing layer enough) should learn that all three users are similar due to their engagement with poll **A**. Now if the squashing layer is small enough then it won't be able to effectively encode enough information about all polls individually and will be forced to make mistakes, but these mistakes will be related to what the network has seen before ie. other user's engagements as the model tries to minimise loss over all outputs. In this way, given the input of a user's engagements, the output will be a mixture of the user's engagements and the engagements of similar

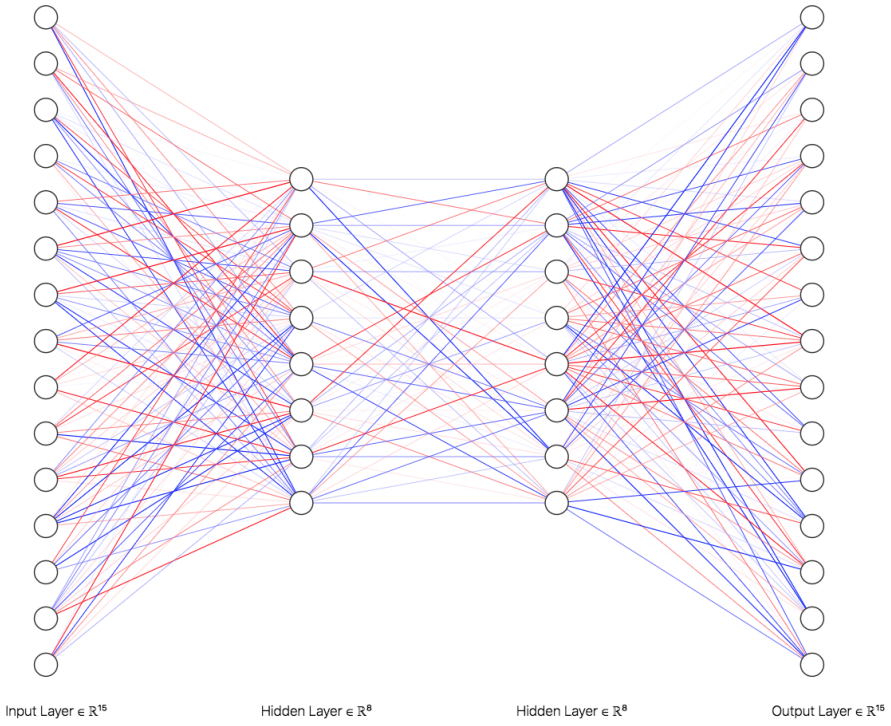

Figure 3.1: Autoencoder architecture overview - Diagram made using http://alexlenail.me/NN-SVG/LeNet.html due to Alex Lenail

users. We then just have to recommend the polls which the network expects the most for a given user and remove the polls that user has already engaged with.

This architecture also lends itself nicely to the task of user clustering. We can simply perform clustering on the output of the encoder layer which is a low dimensional latent representation of a users preferences. running each user's engagements through the encoder portion of the network gives us a vector representation of that user's preferences. We can then run clustering algorithms over all users in order to group users. Taking that one step further, we can take the average vector of each cluster and run it through the decoder portion of the network in order to see which polls the cluster most engages with which we can use to infer user interests.

The advantage of this system is that it is fast at filling in the $User \times Poll\_Utility$ matrix requiring a single forward propagation of a each users previous poll engagements through the network.

3.2 *Model architecture*

In our specific case we take the standard autoencoder recommender system which runs each user's ratings through the network during training and modify it such that instead of 1 rating per poll we encode the rating as a binary vector that includes whether a user owns the poll[1], voted, commented, tracked or skipped it - [Ownership,

[1] We believe that the polls a user creates themselves are able to provide a strong indication of preference

voted, commented, tracked, skipped] - which means our input size for each user is ($number\_of\_polls \times 5\_engagements$). This augments the network with more diverse implicit feedback about a user than simply whether they will engage with a poll, namely the network now has the ability to learn what interaction a user is likely to have with a poll. This also allows us to rank the importance of different engagements when we calculate network loss. By ranking engagements we are able to give more weight to a comment than a vote for example on the basis that it is much less likely for a user to comment on a poll than to vote and therefore we learn more about their preferences from comments. Furthermore in the recommendation step of the model we are able to bias recommendations towards different engagement types simply by giving more weight to an engagement when summing each poll's engagements to find the best polls to recommend a user.

In order to try to ensure that the recommender does not alter poll outcomes as detailed in 1.3 we do not differentiate between whether a user votes one way or another in our encoding, in this way we hope that the model can not learn user similarity based on a group of users all voting the same way on a poll and thus should be less prone to biased recommendations.

As the rankings of engagements affects how users will interact with the platform it is important to decide these carefully. In order to do so we created fictitious data about 20 polls which included how many people voted, commented, skipped, and tracked the poll. We then asked the team to each score the polls out of 100 on how engaged they thought the users on the platform were with each poll. From this we hoped that they would perhaps favour polls with one engagement type prevalent over the others for example those with a few comments are better than those with many votes but no comments. Using the average score for each poll from the 4 ThisThat team members who participated we created a linear regression model such that we can now decide how valuable different engagements are to ThisThat according to the team (We use this as a proxy for user enjoyment). We then use this model when calculating the autoencoder network loss during training such that we hopefully learn to predict the most valuable engagements best[2] and when producing recommendations by taking a weighted sum of the network output in order that polls which a user is most likely to engage with in the most valuable way are recommended first. This is simply an adaption of the standard RMSE function with bias towards certain engagements.

[2] The network should hopefully seek to minimise the outputs causing the greatest loss first and we are artificially forcing some interactions to have higher loss.

The neural network developed using the training data has only 2 hidden layers, one for encoding and one for decoding. each layer has 155 neurons and is initialised with random weights before training. We use a custom loss function based on the linear regression system detailed above which takes a weighted sum of the model error over

all polls compared to the network input. Our loss function can be seen below where **r** are the linear regression coefficients, **I** is the regression intercept and **N** is the number of outputs/total number of engagements.

$$\sqrt{\sum \frac{((y\_true \cdot r + I) - (y\_pred \cdot r + I))^2}{N}}$$

Further specifics of the model were decided using meta-learning techniques that are detailed in the Meta Learning section (3.3). We must be careful not to over train the network as we rely on the network output errors. Specific training regimes are tested in the Testing chapter (4) in order to find a balance between training time and model accuracy.

An interesting benefit of autoencoders is that we are theoretically able to add random noise to the network input while keeping the output as the original, by adding noise the network must learn what the true patterns in the data are without being distracted by the noise in order to accurately reproduce the data after the decoder operation.[3] In our case the benefit is clear, given the ease by which a user can vote on a poll or accidentally swipe in the wrong direction we believe that the data set is inherently noisy and not every interaction is representative of a user's true preferences. We therefore add random noise to the input in order that the network is better able to learn the underlying preferences of users. We must do this in a systematic way however as not all engagements are prone to noise. As we regard ownership of a poll as an interaction and we can see that it is almost impossible to post a poll by accident we must not add noise to the ownership portions of the input. Similarly it seems unlikely that a user will accidentally comment on a poll so we do the same in this case. We test whether adding systematic noise to the data set does indeed improve model accuracy in the Experiments section (4.2).

[3] Vincent *et al.*, 'Stacked Denoising Autoencoders: Learning Useful Representations in a Deep Network with a Local Denoising Criterion', 2010.

At this point the model has taken in an array of all polls on the platform and produced an output with expected engagement for all polls. From here we must incorporate the requirement from ThisThat by which we favour new polls and polls that have been 'boosted'. In order to do this we take the $Poll \times Poll\_Utility$ matrix and run it through a scoring function which takes into account poll age and poll boosting. The scoring function takes in the expectation of a given poll and outputs a score which is calculated as:

$$poll\_expectation * e^{time\_ago/x} + poll\_boost$$

Using this function we can see that the poll expectation decays as a poll gets older and poll boost can be taken into account (specific time decay (x) and boost parameters (poll_boost) are to be decided by the ThisThat team at a later date depending on how they wish to prioritise polls.) We can also allow users to control how much

they are affected by algorithmic curation (as per requirements - 1.3) by simply varying the affect that the time_ago parameter has. If we set this high then the poll expectation will have a minimal effect on what is recommended and instead users will just be presented with the newest polls, whereas if we set this low the user will see only polls that are curated for them with little regard to how long ago they were posted.

In order to meet the requirement that the system is able to adapt over time and work in an online manner we must allow for periodic retraining of the network. The network also needs to allow for new polls to be added to the platform. We solve these problems simultaneously. In order to add a new poll to the network we must add new weights to the encoder and decoder layers and biases to the decoder layer such that our network grows over time.

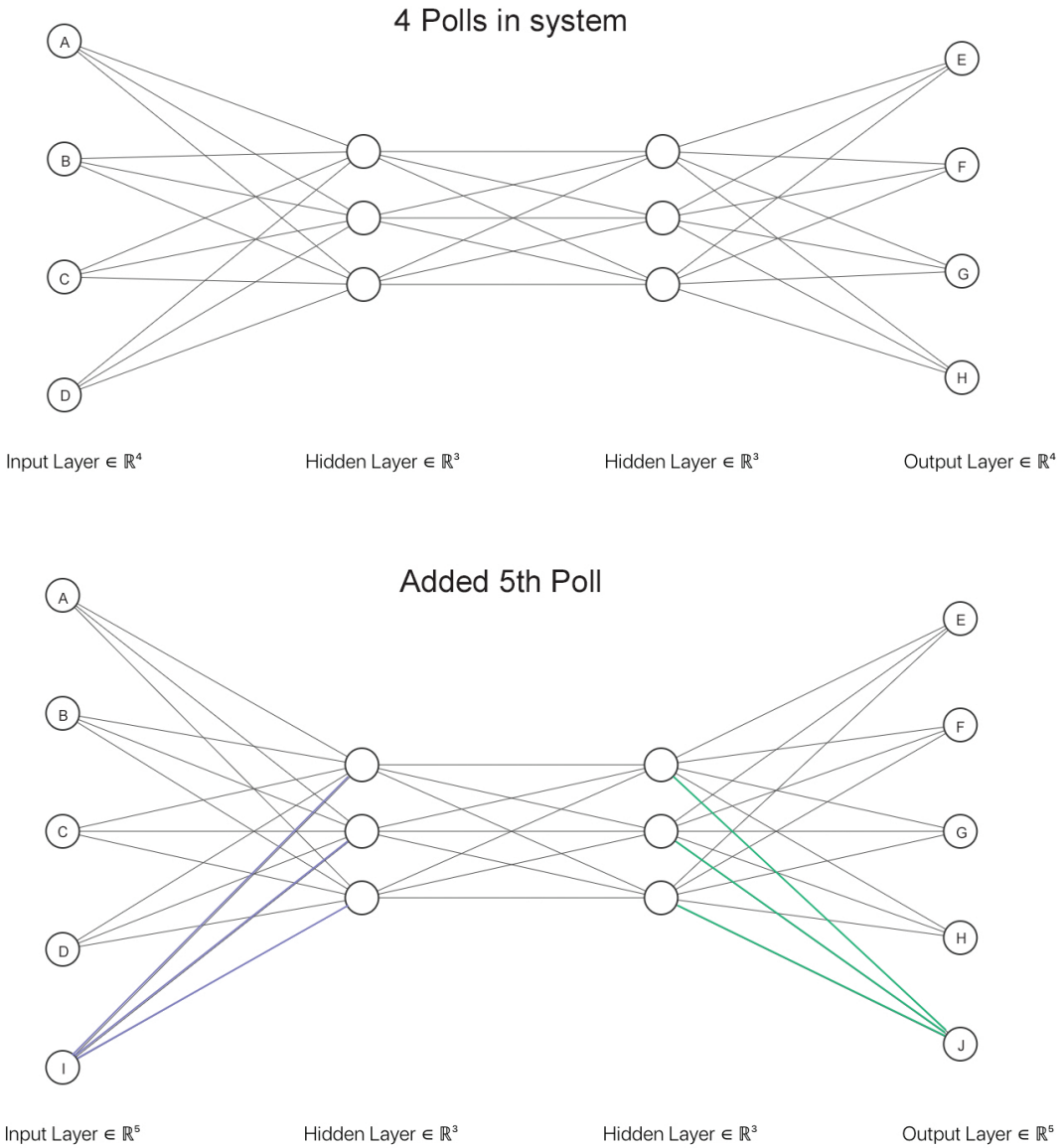

Figure 3.2: New poll addition

So that new polls do not negatively affect the predictions about other polls we set the new encoder weights to zero so that they have no affect on the forward propagation through the network. This would lead to no new polls ever being recommended at the beginning of their life however so we set the decoder weights to a small number such that new polls are automatically recommended to users. Periodic retraining then occurs after a certain number of interactions have been added to the system as a whole such that over time the new poll weights are learned and the new poll is recommended more accurately. This allows the system to be scalable as the number of interactions needed to trigger some training epochs can be set depending on the level of activity on the platform. We demonstrate how many interactions on a poll it takes to start accurately recommending new polls and investigate suitable weight initialisation in the Testing chapter. Figure 3.2 shows how this whole process works. Each input and output node corresponds to a poll (We do not show all the interactions for simplicity). When we add poll 'I' to the network we set the blue weights to zero such that I does not affect the rest of the network. We then set the green weights and biases to a whatever our weight initialisation parameter is in order that J is given some value and may be included in recommendations. Over time the network will learn better values for the blue and green weights as we train but we wait for some number of interactions to give us enough information to perform this training.

3.3 *Meta Learning*

There is growing interest in meta learning techniques used to learn network parameters for neural networks.[4] In this section we explore using a genetic algorithm to find suitable parameters for our autoencoder. Genetic algorithms work by creating a population of candidate solutions represented by vector 'genomes' which are then tested by a fitness function.[5] A new population of candidate solutions is then created by cross-breeding different genomes chosen based on fitness in some manner and then mutating the resulting children with some probability. This process is then repeated multiple times in the hope that the solutions produced continue to improve over time.

[4] Lam *et al.*, 'Tuning of the structure and parameters of neural network using an improved genetic algorithm', 2001.

[5] Castillo *et al.*, 'Artificial neural networks design using evolutionary algorithms', 2003.

In order to promote better solutions over time the algorithm selects viable candidates based on fitness in some way, for example tournament selection[6] or roulette wheel selection.[7] In our case we employ roulette wheel selection which is a fitness proportionate method to pick two 'parents'. Once we have a pair of solutions we then choose a random point in the genome to split on and cut the genomes at this point. We then splice the head of the first genome with the tail of the second and vice versa in order to create two new genomes. After this we mutate the genomes with a small probability in order to hopefully ensure population diversity and greater explora-

[6] Miller, Goldberg *et al.*, 'Genetic algorithms, tournament selection, and the effects of noise', 1995.

[7] Razali & Geraghty, 'Genetic Algorithm Performance with Different Selection Strategies in Solving TSP'.

tion of the solution space.

Fitness of each model is determined by the validation loss after 500 epochs of training, and the model complexity, exponents were discovered using trial and error which lead to good solutions:

$$validation\_loss^2 / number\_of\_layers^{0.5}$$

By this method we are able to trade off model complexity with accuracy.

In our case we are able to apply prior knowledge to creating the initial population of solutions and mutating their genomes. We know for example that we don't want our network to be too deep due to the regular training that will occur during production, we therefore limit our search in this dimension to the range [1-3]. The learning rate dimension of the genome was bounded by exploring learning rates used for neural networks with similar sized inputs in order to make sure we do not waste time evaluating the fitness of models which were not learning at all or learning too quickly and over-fitting significantly. For this we set the initial learning rate of each of our solutions to be either 1, 0.1 or 0.01. The rest of the genotype involves layer sizes, we have a gene which encodes each of the 3 possible layer sizes even if our models do not use 3 layers, we initialize the first layer with a random number between 100 and 400, the second between 60 and 200, and the third between 10 and 100 in order that the layers become progressively smaller to gradually reduce the dimensionality of the input data.

We use a population size of 8 and use prior knowledge when mutating the genes of each candidate solution. In order to hone in on an acceptable learning rate for our model we choose a learning rate of either 1, 0.1 or 0.01 for the first half of our generations in the hope that the population will settle on a rough learning rate for the task, then in the second half of the generations we mutate the learning rate by adding a small random number between -0.5 and 0.5 which is further reduced as the generations progress in order to gradually mutate the solutions less and less as time passes. In order to mutate the number of layers gene we simply choose one of the two other options for number of layers than is currently encoded ie. 1 or 3 if the current gene is 2. In order to mutate layer sizes we use the formula:

$$current\_layer\_size + random\_number\_between\_bounds / generation\_number^{2/3}$$

[8] By reducing the size of these mutations over time using the generation divider we are able to modify the search to become less explorative and more exploitative of the solution space over time.

[8] "random_number_between_bounds" refers to the bounds given in the population initialisation phase.

Finally we employ elitism in our search in order that the algorithm doesn't discard the best solutions found thus far. Elitism simply keeps the best solutions in the current population and inserts them into the new population without crossover or mutation. We keep the

two best solutions from each generation in this way such that every new population is comprised of six new solutions and the two best solutions from the previous generation.

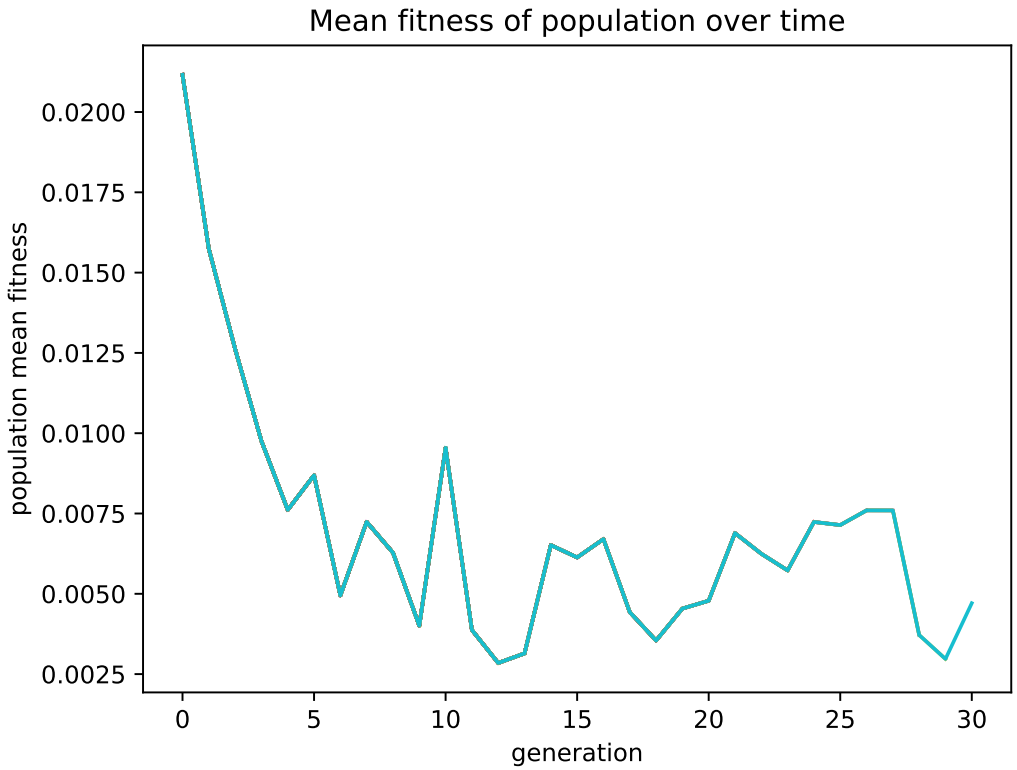

Figure 3.3: Model improvement over time as population of solutions improves

We can see in figure 3.3 that the mean population fitness (and therefore loss) is decreasing over time. the algorithm seems to find a good solution after around 8 generations then struggles to exploit the solution space enough to find a much better solution after that, we believe this to be because we did not employ enough mutation to make the algorithm sufficiently exploitative, but we feel the results we have gained are enough to use for our system on a test basis due to its proof of concept nature. Furthermore the data set which the system is being designed for may have different characteristics to the test data which we currently have access to, therefore new parameters may be needed if the system is deployed in production. After running the genetic algorithm for 30 generations we discover the best set of parameters for our model to be a single layer each for the encoder and decoder layers with 155 neurons and a learning rate of 0.1. which we shall use from now on.

4

TESTING

4.1 Setup

In order to test our system we split our data set into a training and test set, the training set contains 540 users and their interactions, the test set contains the other 231 users. We cannot directly test the system by integrating it with the ThisThat app at this stage so instead we must train our model on the training set and then hide some interactions in the test set from our model and test how well it predicts these interactions. We hide 20% of each test user's interactions except when testing how many interactions a new user needs for good recommendations. We choose our experiments such that we can independently test against the requirements set out for our model. Unless otherwise stated we are testing the model for top-50 recall against the hidden engagements as due to the ease of swiping away an answer we are not worried about relative ordering so much as making sure that a batch of recommendations is on the whole engaging. For experiments involving training the model for 5000 epochs we employ 5-fold cross validation in order to ensure that our results are less prone to chance.

In order to provide a baseline for our data we employ ThisThat's current approach of suggesting the 50 most recent polls to a user. We also compare against an SVD++[1] approach written by Nicolas Hug[2] due to its popularity and performance over many data sets. Thisthat's current approach is able to achieve 22% recall in our testing. SVD++ is able to achieve 69% recall in the same tests when hiding 20% of the testing data. We aim to achieve similar recall with our system overall while still being able to meet our other requirements.

[1] Kumar *et al.*, 'Social popularity based SVD++ recommender system', 2014.

[2] Hug, *Surprise: A Python scikit for building and analyzing recommender systems*, 2015–.

4.2 Experiments

4.2.1 Does masking training data help the model learn?

Strub *et al.*[3] propose an approach to training autoencoders for recommendation tasks by masking off a portion of the training ratings from the input but keeping them in the output in order that the model must learn to reconstruct new data, not just accurately reconstruct already seen ratings. In this experiment we test various amounts of input masking to investigate how it affects recommendation performance.

[3] Strub *et al.*, 'Hybrid Collaborative Filtering with Neural Networks', 2016.

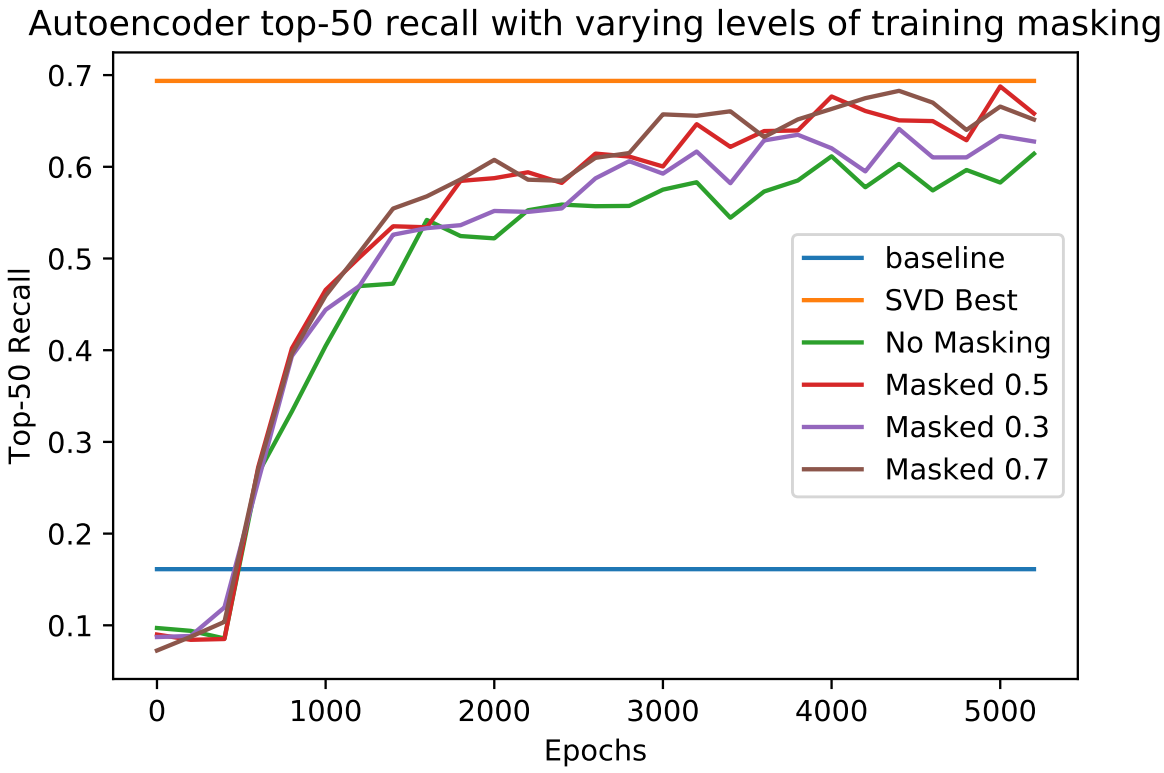

Figure 4.1: Autoencoder performance throughout training with different levels of input masking.

We can see that hiding as much as 70% of the input data produces notable improvement in the recall of the system. We understand this improvement to come from the system essentially having a greater amount of synthetic training data as the input is masked randomly each epoch meaning that for every user the model sees many similar users with less interactions that resemble that user.

4.2.2 Are results relevant?

The second experiment we must run is to discover how much initial training causes our model to be most accurate. As we are exploit-

ing the errors in the model it is important that we train the model enough that these errors are caused by confusing patterns in the data rather than randomness in the model weights. We therefore train our model for 5000 epochs while testing predictions every 100 epochs to find out how long we need to initially train our model to achieve high prediction accuracy. We remove any source of noise in this experiment by testing directly which polls are expected to be engaged with most without including any time decay or boost. We measure the top-50 recall of the model by counting how many of the hidden polls appear in the model's top 50 predictions. We test whether our custom loss function does indeed improve accuracy of predicting certain engagements by training a model with our loss function, and one with a standard RMSE loss function to compare against the ground truth for recall. Our results in figure 4.2 show that the model is able to learn to predict polls with **50%** recall after 1000 epochs. The model trained using a standard root mean squared error loss function is able to achieve **64%** recall after 5000 epochs whereas the model trained using our custom loss function which prioritises specific engagements is able to achieve **65%** recall after 5000 epochs, although we note that the custom loss autoencoder has consistently better recall throughout training.

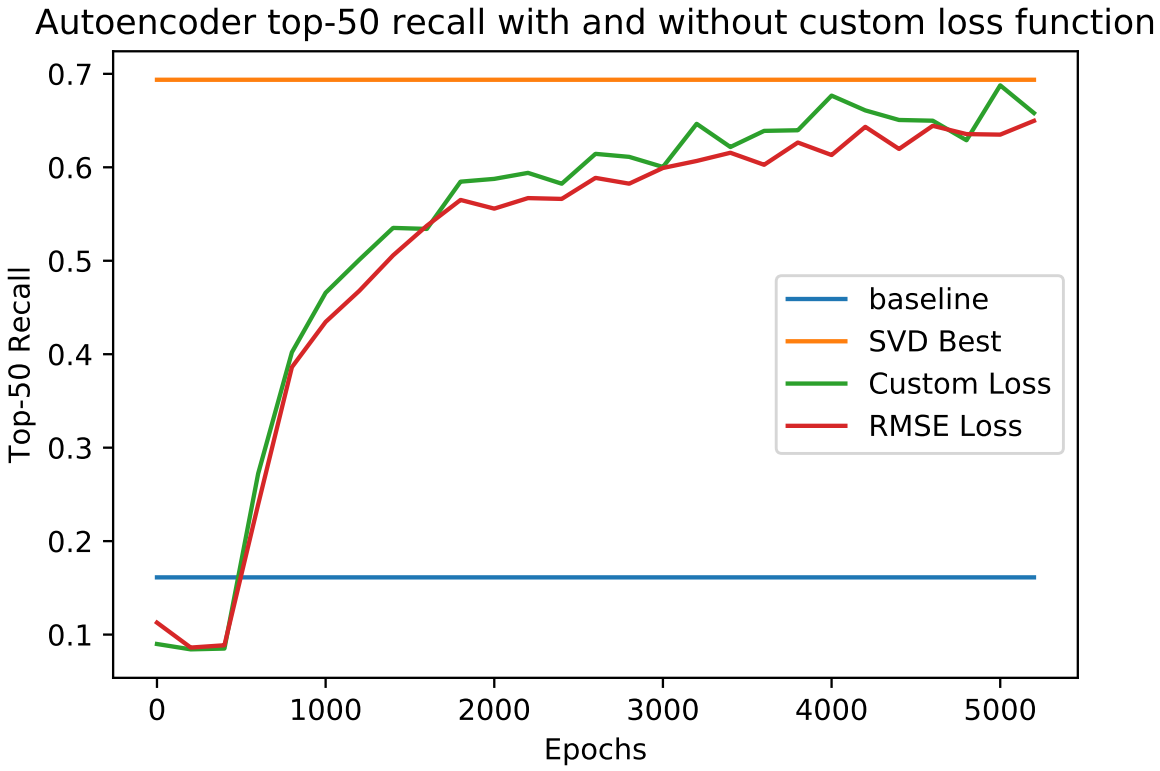

Figure 4.2: Custom Loss Autoencoder vs RMSE loss Autoencoder

We also investigate the effect of adding noise to our training data to see if this allows the model to better learn the underlying patterns.

For this purpose we randomly add interactions into the training data with some probability and then test recall on the test data to see if it has an effect on training time. In order to simulate random 'noisy' engagements we do not add noise to the 'ownership' dimension of the input as we do not believe it easy to accidentally create a poll. Results of this experiment are shown in figure 4.3.

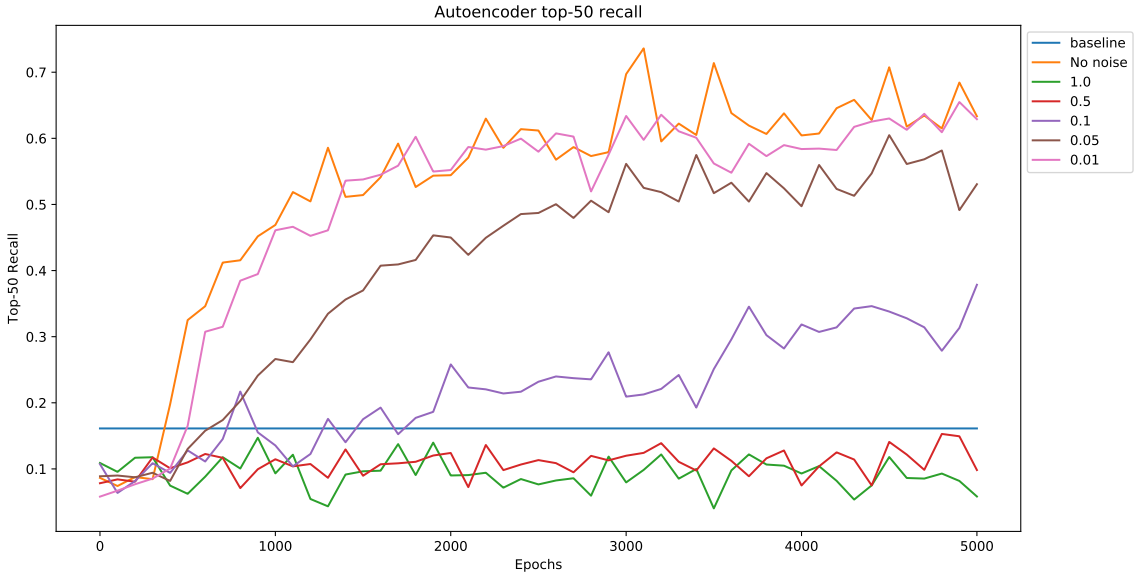

Figure 4.3: Effect of adding varying numbers of noisy engagements to the training data.

We can see that our model is unable to distinguish between the noise and the true data, this leads the model to rapidly degrade if the input is corrupted. The hypothesis that adding noise will in fact aid our model is proven incorrect.

Finally we try to investigate how many interactions a user must provide before the system starts giving accurate recommendations to that user. This is a test of how well our model is able to deal with the new user problem. Up until this point we have been testing our model by hiding 20% of our test users engagements from the forward pass through the network and investigating how many of the hidden engagements the model accurately predicts in its top 50 recommendations. For this test however we vary the number of hidden engagements in order to simulate a new user being added to the system. We use a model that has been trained in advance without seeing the test users to simulate a new user joining the platform. We note that the average test user has 64 engagements so when we hide 90% for example this simulates a user who has only interacted with the system 8 times. Figure 4.4 shows us that the system is able to learn about a users preferences after only a few interactions which is promising, furthermore, predictions continue to improve the more a user interacts unlike some systems. We note that the initial recall is

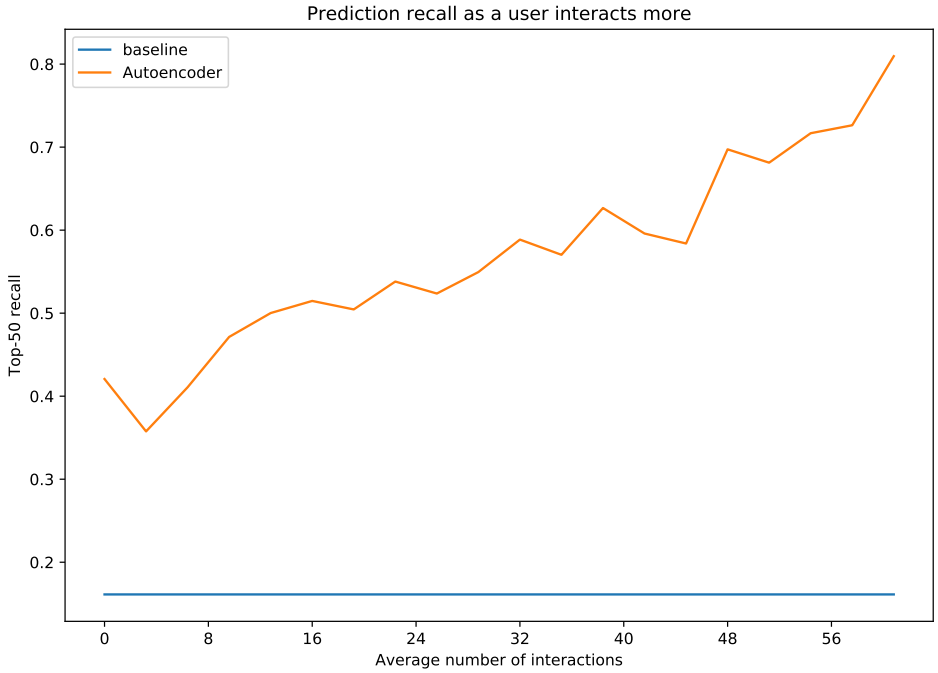

Figure 4.4: Average prediction recall for a new user over time as new interactions are registered

fairly high without any interactions because the system simply predicts the most popular polls as they are the most activated nodes in the network.

4.2.3 *Are new polls answered quickly?*

In this experiment we investigate how the small weights we add to a new poll in the decoder layer affect how quickly a poll is recommended. For this experiment we ignore time decay and boost and experiment with different weight initialisations. We add a new poll to the system and find the average position it is recommended at across all test users. As we count ownership as an engagement which affects poll expectation we must randomise which user submitted our test poll and take an average over a group of users for each weight initialisation trial. Figure 4.5 shows the effect of different decoder weight initialisations on when a new poll is recommended to a user. We can see that by varying the weights we are able to prioritise new polls to a greater or lesser extent.

Our next experiment is to investigate how polls are affected by the time decay parameter. For this we add test polls into the system with artificial timestamps in order to simulate them being submitted from 0 seconds ago to 1 month ago. Figure 4.6 shows how these test polls appear in a user's recommendations. We can see that we are able to vary the exponential decay to choose how much we prioritise new polls against those that the system believes are the most relevant to a

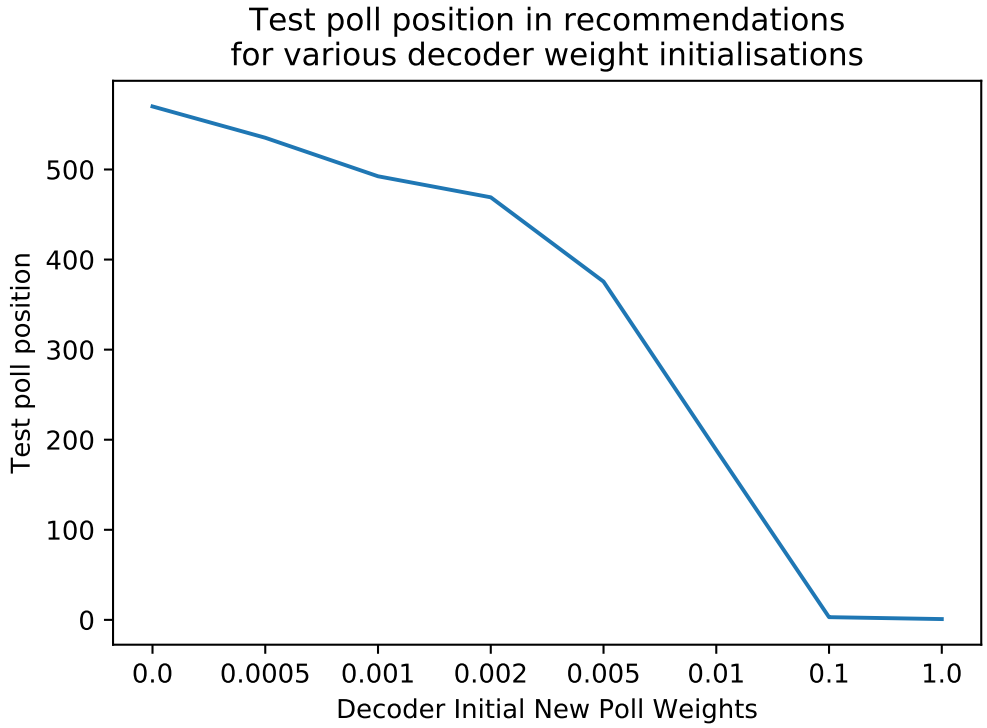

Figure 4.5: Testing how we should initialise weights in the decoder layer for new polls so that a new poll is quickly recommended

user. This also allows a user to decide the extent to which they wish to be affected by algorithmic curation. By changing the divisor of the **t** parameter the user is able to favour being shown new polls if it is large and favour algorithmically curated polls if it is set to a smaller number.

4.2.4 *How does the system adapt over time?*

In this experiment we seek to work out how many interactions a poll needs in order to be accurately recommended to a user. We choose a random selection of polls from our data which we hide from the model, then we insert the polls into the system while testing the users who we know interacted with the poll to find out where the poll appears in their recommendations. Over time we add more interactions to the poll and test how this affects the polls position in the user's suggestions. Figure 4.7 shows us that the system is able to adapt over time to new user interactions. We can see that the hidden poll is on average being recommended earlier to our relevant users as more interactions are being registered for the poll.

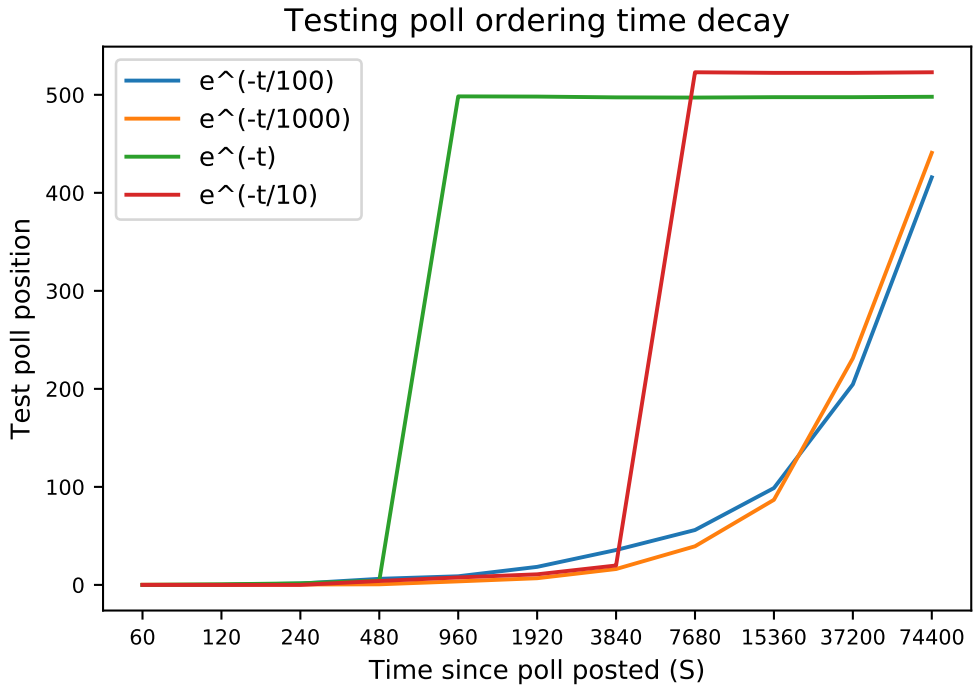

Figure 4.6: New poll recommendation position for different time decay over time

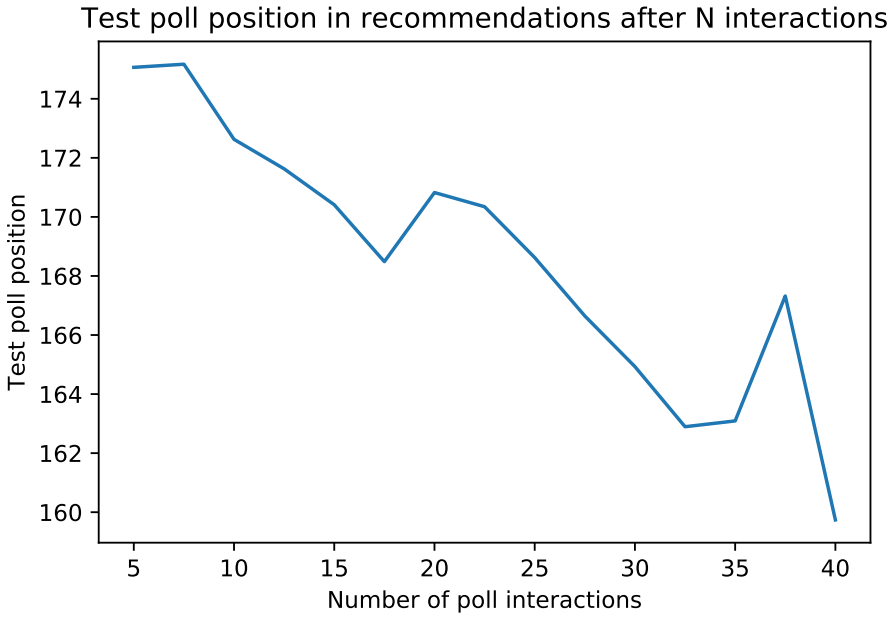

Figure 4.7: New poll positioning in user recommendations as more interactions are registered on the poll

5

EVALUATION

We can see from our experiments above that our model is able to perform well with respect to some of the requirements laid out by ThisThat. In this chapter we perform a detailed analysis of our model against each requirement.

5.0.1 Provide relevant recommendations

We can see that the approach used by Strub *et al.*[1] of masking input data when training autoencoders provides significant benefits for our task. We see in figure 4.1 that recall improves by as much as 6%. We also note that adding masking to the training data serves to better predict new items such that think the model using masking will in fact be more than 6% better than one without as it will also be more accurate on predicting polls the user has not seen in our data set. We can see from figure 4.2 that the model is indeed able to provide useful recommendations to users. The recall of the model is reasonably high on hidden data which shows promise for our system in a real world setting. The predictions of our model may actually be better than we believe as we do not know how many of the recommendations that were provided by the system but were not part of the hidden data would in fact be relevant to users. It is impossible to tell whether the user may have in fact engaged with the polls that our test discarded as irrelevant more than those which the user had already interacted with. We therefore believe that the recall figure of the model at 67% is a significant achievement. We note the improvement in recall gained from using our custom loss function which takes into account weighted engagements. This confirms our hypothesis that certain interactions are able to provide more information about user preferences than others.

[1] Strub *et al.*, 'Hybrid Collaborative Filtering with Neural Networks', 2016.

While our model is not quite able to achieve the same recall as The SVD++ approach, we believe that our results are close enough to be counted as a success in the overall accuracy criteria. We also note that our model may be better at making novel predictions compared

to the SVD++ approach but we are unable to test this.

One cause for concern is the sheer number of epochs needed in order to begin recommending polls precisely. This however is only the initial seed data needed to build an underlying model of similarity, the model will not need to be continuously trained for 5000 epochs every time a new user or poll is added. We see this in the quality of recommendations demonstrated in figure 4.7.

We also note from figure 4.3 that our hypothesis stating that added noise in the training data will lead the model to better learn underlying patterns in the data is entirely incorrect. We see that the model struggles to predict user interests with even the smallest amount of added random noise in the data set, we suspect this to be because the model may rely too heavily on predictions based on users who are incredibly similar rather than those who share some commonalities. This is concerning for the overall performance of the model as our data set is dense with engagements due to the relatively low number of polls in the set, we expect in future though that the number of polls will greatly outweigh the number of users on the system and therefore the model may struggle as the user similarity space will become wider and more diverse. Further testing of this hypothesis is needed.

Our experiment detailing how many interactions a new user needs in order to be accurately recommended polls shows particular promise as we can see that the new user problem is effectively solved by recommending the most popular polls to a user at the start and then the system is able to learn after as few as 10 interactions how best to recommend polls to a specific user (see figure 4.4). We see this as the system learning coarse user preferences at the start as user interactions are mapped into the hidden latent space, then the user's position in the latent space is updated as more interactions are registered and thus becomes more accurate. We also note that prediction recall can become as high as 80% for users who have many interactions with the system, we can't however confirm how this generalises to unseen polls as we cannot test unknown data.

5.0.2 *Poll recommendations should not lead to biased sampling of opinion.*

While this requirement is perhaps the most difficult to test for, we feel that by not taking into account which way a user voted when training our model we are able to help meet this criteria to some extent. We also feel that the decision to not use content information in our model is justified at this stage due to the bias in recommendations that would be introduced if the model has seen polls involving information about only one of the two possible answers before, although we discuss hybridisation approaches in the Further Work

chapter (7). Given access to a live testing environment we would like to perform split testing over a random group of users in which the first group of users is recommended a poll using the standard most recent approach and then the second group of users is recommended the same poll using our model to see whether or not the recommendations lead to a shift in how the user base answers polls.

5.0.3 Ensure new polls are answered quickly

This requirement is one in which any model is able to perform well simply by selecting the most recent polls posted, we therefore do not put much stock into the results obtained here. Our experiment shown in figure 4.5 is simply a parameter search in order to demonstrate that the logic behind our model is sound. We can see from the graph that varying the decoder layer weights does lead to a change in the relevance of new polls. Interestingly we note that weights lower than 0.002 seem to have no effect on the recommendations of the system, we assume this is because the weights of all other output nodes are higher than this number so the new poll is lost in a sea of noise. Furthermore, setting the weights too high simply leads to the poll being recommended first to every single user which is not a desired characteristic. It is important therefore to find a trade off between suitable weights and a time decay parameter such that new polls are recommended often to relevant users. We also see that the new poll is not recommended last to all users even when the decoder weights and biases are set to 0, we believe this to be because the single interaction of ownership by the submitting user which we count as an engagement causes enough disturbance in the network during the forward propagation step that the poll is immediately recommended somewhere above last place to users who are very similar to the owner of the poll in terms of interactions.

5.0.4 Allow users to control how they are affected by algorithmic curation

We can conclude from figure 4.6 that we are able to allow users to control the degree to which they are affected by algorithmic curation with some precision. From the graph we can see that we are able to vary the time decay parameters to such an extent that we can either wholly utilise algorithmic curation, or almost entirely rely on how old a poll is. If we were to tune the parameters further we could find a suitable trade off between poll novelty and curated recommendations which would serve as a middle ground. We also note that allowing users to choose to answer only new polls may need to be taken into account when updating the model over time to take into account the increased noise in the interactions that would surely follow as it is unlikely a user will enjoy every new poll.

5.0.5 *The system must adapt over time*

We can see from figure 4.7 that the system is able to learn about new polls to some degree. Testing of this was performed without taking into account a time decay however so the new poll would in fact be recommended earlier to users in that case, we therefore are content that the improvement in recommendation over time despite the lack of taking into account novelty does show promise as a proof of concept. The poll positioning in this test is compared to every other possible poll that a user hasn't seen (on average 500 polls) therefore recommending the poll moderately early on based solely on the single interaction of knowing who submitted the poll provides strong evidence that the model is accurately learning user preferences. The improvement shown over time further demonstrates the model's capability to adapt and improve as time goes on.

5.0.6 *Allow ThisThat to cluster users based on observed behaviour*

This requirement is fulfilled in the sense that we can cluster users based on their interactions. By running each users interactions through the encoder operation of our autoencoder we thereby find the latent representation of the user's preferences. From this we can then perform clustering over all users. This however lacks explainability, as beyond running cluster center defined vectors through the whole network and then manually examining which polls are predicted highest we cannot tell what each cluster of users enjoy. We find a potential solution to this problem in the work of Burgess *et al.*[2] using disentanglement which we discuss in Further Work (7)

[2] Burgess *et al.*, 'Understanding disentangling in $\beta - VAE$'■, 2018.

5.0.7 *Build a system that can scale*

This requirement is also difficult to evaluate given the size of our sample. We are able to conclude that our network is fast at inference time requiring a single forward propagation to compute expectation followed by post-processing steps involving simple matrix multiplication. However our model seems to perform poorly at initial training time, taking 1000 epochs to produce accurate recommendations on the test data. We believe from figure 4.7 that once the model has learned which users are similar it is able to update recommendations fairly quickly. We believe this criteria has been somewhat achieved given the speed of inference, but further testing would have to take place in order to fully justify the model's scalability.

6

CONCLUSION

The project as demonstrated shows promise for being well adapted to the requirements of the problem at hand. We can see that we are able to provide far better recommendations than the current approach taken by ThisThat. While we do not achieve results that are as accurate as Hug's[1] SVD++ we note that our predictions may be providing relevant polls from a broader range of polls due to the non-linearity of the model and that we are able to provide a flexible and fast solution to the problem. However the results given above are difficult to verify in a real life scenario as we are unable to accurately test how much a user enjoys a poll but must instead resort to a proxy for enjoyment given by our engagement metric. We are also unable to effectively test how good recommendations are for polls that the user has not yet interacted with as that would require predicting the future. We must conclude therefore that what we have produced is a viable proof of concept that seems well adapted to the problem and indeed represents a novel solution to the problem of recommendations given implicit data. However without further testing in a real life scenario it is difficult to suggest that our solution may be any better than other systems on a similar task. We believe that our solution could be applied to many recommendation problems in which implicit feedback is gathered, particularly those where the number of items is larger than the number of users, for example website recommenders and photo sharing sites such as Instagram.

We also note that the model may be too flexible given the number of different parameters that can be tuned, it might be impossible to find a set of parameters that allows for high recall, speed and scalability whilst also meeting the rest of the requirements.

[1] Hug, *Surprise: A Python scikit for building and analyzing recommender systems*, 2015–.

7

FURTHER WORK

In future we would like to explore hybridisation techniques for the system, namely encoding a content representation into the engagement array that the model is fed such as proposed by Strub *et al.*.[1] We believe that this would lead to an increase in recall of relevant new polls, going some way to solve the new item problem for our system. We also feel that added context for the system would benefit recommendation recall over all as the current encoding of five values per poll provides very little information about each poll. However work would have to be done to try and avoid the problem of biasing poll outcomes as detailed in 1.3, as adding content information to the system naturally introduces bias.

[1] Strub *et al.*, 'Hybrid Collaborative Filtering with Neural Networks', 2016.

There is also room to improve accuracy by using multiple autoencoders. We see from Sedhain *et al.*'s[2] work that they use an Item-Item based approach with multiple autoencoders and we think it would be interesting to perhaps develop our approach using different autoencoders for different groups of users in order to better learn preferences rather than using a one size fits all model.

[2] Sedhain *et al.*, 'Autorec: Autoencoders meet collaborative filtering', 2015.

We also see scope for improvement in how we rank user preferences from implicit data, currently our model assumes that all users are the same in the sense that they all have similar predisposition to each engagement type. In future we would like to research adapting our loss function and prediction post processing to take into account different user tendencies to comment etc. Jin & Si[3] perform a study of ratings normalisation for collaborative filtering techniques which would serve as a foundation for adapting our model in this way.

[3] Jin & Si, 'A study of methods for normalizing user ratings in collaborative filtering', 2004.

Further research is needed to improve explainability of recommendations which would also help explain clustering. We see in the work of Burgess *et al.*[4] that it is possible to disentangle the nodes of the autoencoder in such a way that we can explain the variance in output from a node based on changes in the input. We feel this would go some way to explaining why certain clusters of users are formed in any attempt to cluster users from the latent representation that the model learns.

[4] Burgess *et al.*, 'Understanding disentangling in $\beta - VAE$'■, 2018.

Finally we anticipate the growth of the model becoming untenable and therefore we would need to research mechanisms for removing polls from the recommendations after a certain time period as they become stale. There is also scope for modifying the loss function further to learn more from newer polls by scaling the loss in some way based on how many recent polls a user has answered such that we can ensure we are always learning about a users latest preferences.

BIBLIOGRAPHY

1. Linden, G., Smith, B. & York, J. Amazon.com recommendations: item-to-item collaborative filtering. *IEEE Internet Computing* **7,** 76–80. ISSN: 1089-7801 (Jan. 2003).
2. Kabiljo, M. & Ilic, A. *Recommending items to more than a billion people* Dec. 2018. `https://code.fb.com/core-data/recommending-items-to-more-than-a-billion-people/`.
3. Howe, M. Pandora's Music Recommender. *A Case Study, I,* 1–6 (2009).
4. Belkin, N. J. & Croft, W. B. Information Filtering and Information Retrieval: Two Sides of the Same Coin? *Commun. ACM* **35,** 29–38. ISSN: 0001-0782 (Dec. 1992).
5. Powell, M. J. D. *Approximation theory and methods* (Cambridge university press, 1981).
6. Adomavicius, G. & Tuzhilin, A. Toward the next generation of recommender systems: a survey of the state-of-the-art and possible extensions. *IEEE Transactions on Knowledge and Data Engineering* **17,** 734–749. ISSN: 1041-4347 (June 2005).
7. Rader, E. & Gray, R. *Understanding User Beliefs About Algorithmic Curation in the Facebook News Feed* in *Proceedings of the 33rd Annual ACM Conference on Human Factors in Computing Systems* (ACM, Seoul, Republic of Korea, 2015), 173–182. ISBN: 978-1-4503-3145-6. doi:10.1145/2702123.2702174. `http://doi.acm.org/10.1145/2702123.2702174`.
8. Friedkin, N. E. & Johnsen, E. C. Social influence and opinions. *The Journal of Mathematical Sociology* **15,** 193–206 (1990).
9. Kumar, R., Verma, B. & Rastogi, S. S. Social popularity based SVD++ recommender system. *International Journal of Computer Applications* **87** (2014).
10. Strub, F., Mary, J. & Gaudel, R. Hybrid Collaborative Filtering with Neural Networks. *CoRR* **abs/1603.00806.** arXiv: 1603.00806. `http://arxiv.org/abs/1603.00806` (2016).
11. Zhang, S., Yao, L. & Sun, A. Deep learning based recommender system: A survey and new perspectives. *arXiv preprint arXiv:1707.07435* (2017).

12. Hu, Y., Koren, Y. & Volinsky, C. *Collaborative filtering for implicit feedback datasets* in *2008 Eighth IEEE International Conference on Data Mining* (2008), 263–272.

13. Geurts, T. & Frasincar, F. *Addressing the cold user problem for model-based recommender systems* in *Proceedings of the International Conference on Web Intelligence* (2017), 745–752.

14. Aleksandrova, M., Brun, A., Boyer, A. & Chertov, O. Identifying representative users in matrix factorization-based recommender systems: application to solving the content-less new item cold-start problem. *Journal of Intelligent Information Systems* **48,** 365–397. ISSN: 1573-7675 (Apr. 2017).

15. Herlocker, J. L., Konstan, J. A., Terveen, L. G. & Riedl, J. T. Evaluating collaborative filtering recommender systems. *ACM Transactions on Information Systems (TOIS)* **22,** 5–53 (2004).

16. Van den Oord, A., Dieleman, S. & Schrauwen, B. in *Advances in Neural Information Processing Systems 26* (eds Burges, C. J. C., Bottou, L., Welling, M., Ghahramani, Z. & Weinberger, K. Q.) 2643–2651 (Curran Associates, Inc., 2013). http://papers.nips.cc/paper/5004-deep-content-based-music-recommendation.pdf.

17. Pazzani, M. & Billsus, D. Learning and Revising User Profiles: The Identification of Interesting Web Sites. *Machine Learning* **27,** 313–331. ISSN: 1573-0565 (June 1997).

18. Furnkranz *et al. A case study in using linguistic phrases for text categorization on the WWW* in (2014).

19. De Gemmis, M., Lops, P., Semeraro, G. & Basile, P. *Integrating Tags in a Semantic Content-based Recommender* in *Proceedings of the 2008 ACM Conference on Recommender Systems* (ACM, Lausanne, Switzerland, 2008), 163–170. ISBN: 978-1-60558-093-7. doi:10.1145/1454008.1454036. http://doi.acm.org/10.1145/1454008.1454036.

20. Iacobucci, D., Arabie, P. & Bodapati, A. Recommendation agents on the internet. *Journal of Interactive Marketing* **14,** 2–11 (2000).

21. Cho, Y. H., Kim, J. K. & Kim, S. H. A personalized recommender system based on web usage mining and decision tree induction. *Expert systems with Applications* **23,** 329–342 (2002).

22. Zheng, L., Noroozi, V. & Yu, P. S. *Joint Deep Modeling of Users and Items Using Reviews for Recommendation* in *Proceedings of the Tenth ACM International Conference on Web Search and Data Mining* (ACM, Cambridge, United Kingdom, 2017), 425–434. ISBN: 978-1-4503-4675-7. doi:10.1145/3018661.3018665. http://doi.acm.org/10.1145/3018661.3018665.

23. Egghe, L. Untangling Herdan's law and Heaps' Law: Mathematical and informetric arguments. *JASIST* **58,** 702–709 (Mar. 2007).

24. Konstan, J. A. *et al.* GroupLens: Applying Collaborative Filtering to Usenet News. *Commun. ACM* **40,** 77–87. ISSN: 0001-0782 (Mar. 1997).

25. Wikipedia contributors. *Usenet — Wikipedia, The Free Encyclopedia* https://en.wikipedia.org/w/index.php?title=Usenet&oldid=889940787. [Online; accessed 30-March-2019]. 2019.

26. Linden, G., Smith, B. & York, J. Amazon.com recommendations: item-to-item collaborative filtering. *IEEE Internet Computing* **7,** 76–80. ISSN: 1089-7801 (Jan. 2003).

27. Wikipedia contributors. *Churn rate — Wikipedia, The Free Encyclopedia* https://en.wikipedia.org/w/index.php?title=Churn_rate&oldid=885259360. [Online; accessed 30-March-2019]. 2019.

28. Miyahara, K. & Pazzani, M. J. *Collaborative Filtering with the Simple Bayesian Classifier* in *Proceedings of the 6th Pacific Rim International Conference on Artificial Intelligence* (Springer-Verlag, Melbourne, Australia, 2000), 679–689. ISBN: 3-540-67925-1. http://dl.acm.org/citation.cfm?id=1764967.1765055.

29. Ungar, L. H. & Foster, D. P. *Clustering methods for collaborative filtering* in (1998).

30. Chee, S. H. S., Han, J. & Wang, K. *RecTree: An Efficient Collaborative Filtering Method* in *Data Warehousing and Knowledge Discovery* (eds Kambayashi, Y., Winiwarter, W. & Arikawa, M.) (Springer Berlin Heidelberg, Berlin, Heidelberg, 2001), 141–151. ISBN: 978-3-540-44801-3.

31. Bennett, J., Lanning, S. *et al. The netflix prize* in *Proceedings of KDD cup and workshop* **2007** (2007), 35.

32. Funk, S. *https://sifter.org/simon/journal/20061211.html* 2006. https://sifter.org/~simon/journal/20061211.html.

33. Sedhain, S., Menon, A. K., Sanner, S. & Xie, L. *Autorec: Autoencoders meet collaborative filtering* in *Proceedings of the 24th International Conference on World Wide Web* (2015), 111–112.

34. Lee, J., Kim, S., Lebanon, G. & Singer, Y. *Local Low-Rank Matrix Approximation* in *Proceedings of the 30th International Conference on Machine Learning* (eds Dasgupta, S. & McAllester, D.) **28** (PMLR, Atlanta, Georgia, USA, 17–19 Jun 2013), 82–90. http://proceedings.mlr.press/v28/lee13.html.

35. Claypool, M. *et al.* Combing content-based and collaborative filters in an online newspaper (1999).

36. Pazzani, M. J. A Framework for Collaborative, Content-Based and Demographic Filtering. *Artificial Intelligence Review* **13,** 393–408. ISSN: 1573-7462 (Dec. 1999).

37. Condli, M. K., Lewis, D. D., Madigan, D. & Posse, C. *Bayesian mixed-E ects models for recommender systems* in ().

38. Vincent, P., Larochelle, H., Lajoie, I., Bengio, Y. & Manzagol, P.-A. Stacked Denoising Autoencoders: Learning Useful Representations in a Deep Network with a Local Denoising Criterion. *J. Mach. Learn. Res.* **11,** 3371–3408. ISSN: 1532-4435 (Dec. 2010).

39. Lam, H. K., Ling, S. H., Leung, F. H. F. & Tam, P. K. S. *Tuning of the structure and parameters of neural network using an improved genetic algorithm* in *IECON'01. 27th Annual Conference of the IEEE Industrial Electronics Society (Cat. No.37243)* **1** (Nov. 2001), 25–30 vol.1. doi:10.1109/IECON.2001.976448.

40. Castillo, P. *et al.* in *Advances in Soft Computing* 43–52 (Springer, 2003).

41. Miller, B. L., Goldberg, D. E. *et al.* Genetic algorithms, tournament selection, and the effects of noise. *Complex systems* **9,** 193–212 (1995).

42. Razali, N. M. & Geraghty, J. *Genetic Algorithm Performance with Different Selection Strategies in Solving TSP* in ().

43. Hug, N. *Surprise: A Python scikit for building and analyzing recommender systems* [Online; accessed <today>]. 2015–. http://surpriselib.com/.

44. Burgess, C. P. *et al.* Understanding disentangling in $\beta - VAE$. *arXiv e-prints,* arXiv:1804.03599 (Apr. 2018).

45. Jin, R. & Si, L. *A study of methods for normalizing user ratings in collaborative filtering* in *Proceedings of the 27th annual international ACM SIGIR conference on Research and development in information retrieval* (2004), 568–569.