# Autoencoder Based Recommender Systems for SciStarter

*Thomas James Cartwright*

Master of Science
School of Informatics
University of Edinburgh
2019

# Abstract

In this paper we researched autoencoder based recommender systems on the SciStarter dataset. Our research covers content based recommenders, collaborative filtering recommenders using shallow and deep autoencoders, and finally hybrid recommenders. Our results show that the SciStarter dataset is not ready to sufficiently train an autoencoder based recommender, since our experiments failed to match the benchmark. Our experiments did however show comparable results on the MovieLens dataset therefore showing that, with additional data, the SciStarter dataset could be suitable for an autoencoder based recommender systems in the future.

## Acknowledgements

I would like to thank my supervisors: Dr. Kobi Gal and Dr. Avi Segal. They provided guidance and feedback for the research from start to completion, as well as suggesting improvements and correcting errors in the actual dissertation.

I would also like to thank Naama Dayan for providing the code with which to run the benchmark algorithms.

Finally, I would like to thank my family, friends and partner. They inspire me with their constant support and dedication to helping me achieve my goals.

# Table of Contents

# Chapter 1

# Introduction

Recommender systems can be used to recommend products for a user to buy, films for a user to watch or be combined with information retrieval algorithms to rank items for a user's query. Making good recommendations to users can increase their engagement, reduce churn rates and increase revenue. Consequently, there has been extensive research into recommender systems [7].

## 1.1 Context

In this paper we will discuss recommender systems built using autoencoders on content based and collaborative filtering data from the SciStarter dataset.

Scistarter is a portal where scientists and researchers can outsource tasks and projects to members of the public. The tasks can be anything from identifying cancer cells to bird spotting and can be completed online or offline. Currently, users will arrive on the SciStarter website, find a project through the homepage or by searching, participate with it and then leave the site. The aim is to build a recommender system that uses a record of the users participations to recommend more relevant tasks or projects to a user, therefore keeping them engaged.

## 1.2 Motivation

SciStarter has a user engagement issue with a high user attrition rate. The negative affects of this are twofold. Firstly, researchers fail to get the engagement they require in order to complete their research. Secondly, the quality of completed tasks is low since users usually need some practice before they can complete tasks to a higher standard. Since Scistarter projects involve important work in science around the world we therefore believe that any work that increases engagement on the portal is a worthy area of research.

It has been proven that recommender systems can increase engagement [9] and we therefore believe that a recommender system can solve SciStarter's engagement issue. Promising results have been shown with autoencoders on other datasets [23] and therefore we believe that this is a good direction with which to push the research. Finally, we believe that the data that has already been collected by SciStarter on their user participations lends itself well to building a recommender system and therefore this is a reasonable next step in improving the SciStarter experience.

## 1.3 Objective of Project

Therefore the aim of our project is to build a reliable autoencoder based recommender systems on the SciStarter dataset. We aim to experiment with both the project content data and user participation data in order to accurately predict projects that a user will engage with in the future.

## 1.4 Results Achieved

In this paper our aim was to create several different types of recommender systems and successfully recommended relevant projects to users based on their previous interactions on the site. We experimented with various recommender models and autoencoder architectures that achieved promising results on the industry standard recommendation datasets. However, we failed to beat the benchmark models and provide reliable recommendations for users on the SciStarter dataset.

### 1.4.1 Benchmarks

The first aim of our research was to establish a benchmark and we achieved this through a standard popularity based model, user-to-user collaborative filtering and project-to-project collaborative filtering. These models achieved comparable results to the literature and therefore provided a reasonable baseline.

### 1.4.2 Content-Based Models

We developed a number of content based models through embedding of the projects content data using autoencoders. We expected our content based models to achieve comparable, or even higher, results than our benchmarks since they use project similarity to recommend users projects. However, these models achieved lower results overall than the benchmarks. In Section 4.2 we address this in more detail and hypothesize the reason for these results.

### 1.4.3 Collaborative Filtering Models

We experiment with various collaborative filtering models based on autencoders. Firstly, we developed a denoising autoencoder in an attempt to generate recommendations by encoding and then decoding our user participation data. The results from these experiments were not promising and we address this in Section 4.3.

We then developed a deep autoencoder architecture with the hope that it would capture patterns in the data and consequently make more accurate predictions. Though this was our best performing model, it still failed to beat the baseline. We therefore address this issue in Section 4.4.

Here we also evaluated our models on the MovieLens dataset to compare them to the current state-of-the-art.

### 1.4.4 Hybrid Recommender Models

Finally, we believed that we could combine our results from the content-based and collaborative filtering models in order to beat our baseline and generate accurate recommendations. However, these models performed similarly to the deep collaborative filtering architectures and again failed to beat the baseline.

We therefore failed to achieve the aim of our project and have come to the conclusion that autoencoders were not suitable for generating recommendations on the SciStarter dataset. We give more information on this in our conclusion.

## 1.5 Overview of the Paper

In this section we detail the structure of the report, outlining the highlights from each chapter.

### 1.5.1 Background

In this chapter we provide the necessary background on the Scistarter portal, Scistarter dataset, a brief history recommender systems and autoencoders and an explanation of our implementation of the recommender systems.

### 1.5.2 Methodology

The third chapter contains explanations of the evaluation metrics and models we developed. We begin by describing our data preprocessing before going on to describe the implementation of our content-based models, denoising autoencoders, deep autoencoders and hybrid autoencoders.

### 1.5.3 Results and Discussion

The fourth chapter contains the results we achieved from running our experiments as well as discussion about these results. We hypothesis why certain results were achieved and what could be done to improve them in the future.

### 1.5.4 Conclusion

In this final chapter we conclude our findings, explore the limitations of our work and suggest areas of future research.

# Chapter 2

# Background

In this paper we assume that the reader has a basic working knowledge of neural networks and machine learning techniques.

## 2.1 Scistarter

SciStarter is a portal containing thousands of projects that help forward scientific research. They attract volunteers through a number of channels to take part in tasks. However, SciStarter has an engagement problem since on average users who interact with the site participate in 3-5 projects before churning.

Currently the only way for users to discover projects is by directly searching for them, or finding them on the SciStarter homepage. Therefore a recommender system should be a viable way of retaining users on the site, contributing to projects.

## 2.2 Recommender Systems

In this paper we implement various architectures for recommender systems on the SciStarter dataset.

### 2.2.1 Content-Based Recommenders

Content based recommenders have been shown to achieve state-of-the-art results on recommendation tasks [18]. The core concept of content based recommendation systems is that they use information about the items and users profiles to build similarity models and recommend users a project that is most similar to their profile. A main advantage of this method is that content based recommenders do not suffer from the cold start problem and can therefore recommend new projects to users immediately. They are also often more simple to implement than a collaborative filtering model and therefore a good first step in developing a recommender system. However, they can create filter bubbles where users are only recommended projects that are very similar to previous projects, therefore not recommending more original content.

### 2.2.2 Collaborative Filtering Recommenders

Collaborative filtering methods are based on the assumption that if user A has interacted with the same item as user B, then user A is more likely to interact with a different item that user B has interacted with, as opposed to a random item. Collaborative filtering then falls into two categories, user-user methods and item-item methods.

The item-item methods take the following steps,

- Build an Item-Item matrix that stores the relationships between items i.e. whether a user has interact with both items
- Match the users previous items to the matrix in order to recommend any new items

The user-user methods take the following steps,

- Look for users that have interacted with similar items to the current user
- Use the ratings from these similar users to recommend new items

### 2.2.3 Hybrid Recommenders

In an attempt to overcome the limitations of content based and collaborative filtering recommenders, researchers have made several attempts at combining the two approaches into hybrid recommenders [6]. Hybrid recommenders don't suffer from the above problems and can also use all the available information to make recommendations to users. There are a number of ways to combine the collaborative filtering data and the content data in a hybrid recommender and we will therefore research and implement a number of these.

### 2.2.4 State-of-the-Art and Limitations

State-of-the-art recommenders have been shown to perform well on MovieLens and Netflix dataset [30]. However, they can often struggle with sparse data or content information that is not in a digestible form ( [26], [24]).

## 2.3 Autoencoders

Autoencoders are neural networks that learn to reconstruct the input [23]. What makes autoencoders special is that they contain at least one hidden layer with fewer variables than the input or output layers, these are called the latent variables. This hidden layer prevents the autoencoder from learning the identity function, therefore forcing it to learn hidden patterns in the data in order to embed the input into a smaller number of dimensions. Autoencoders can come in many forms, with a varying number of hidden layers, latent variables, loss functions and optimisation techniques.

Autoencoders have been successfully applied in image classification tasks [27], text classification tasks [31] and recommender systems. [25]. Autoencoders can also be used to handle sparse data as they reduce the dimensionality of data. Therefore they are a suitable candidate for application to the SciStarter dataset as they will be able to extract hidden patterns in the data and therefore make recommendations based on a users previous project interactions.

## 2.4 Datasets

### 2.4.1 SciStarter User Interactions Dataset

Alongside the project content information, SciStarter also provided us with a dataset containing 42,159 participations. Each participation record is an interaction between a profile and a project. Interactions can come in a number of forms and are distributed as shown in Table 2.1

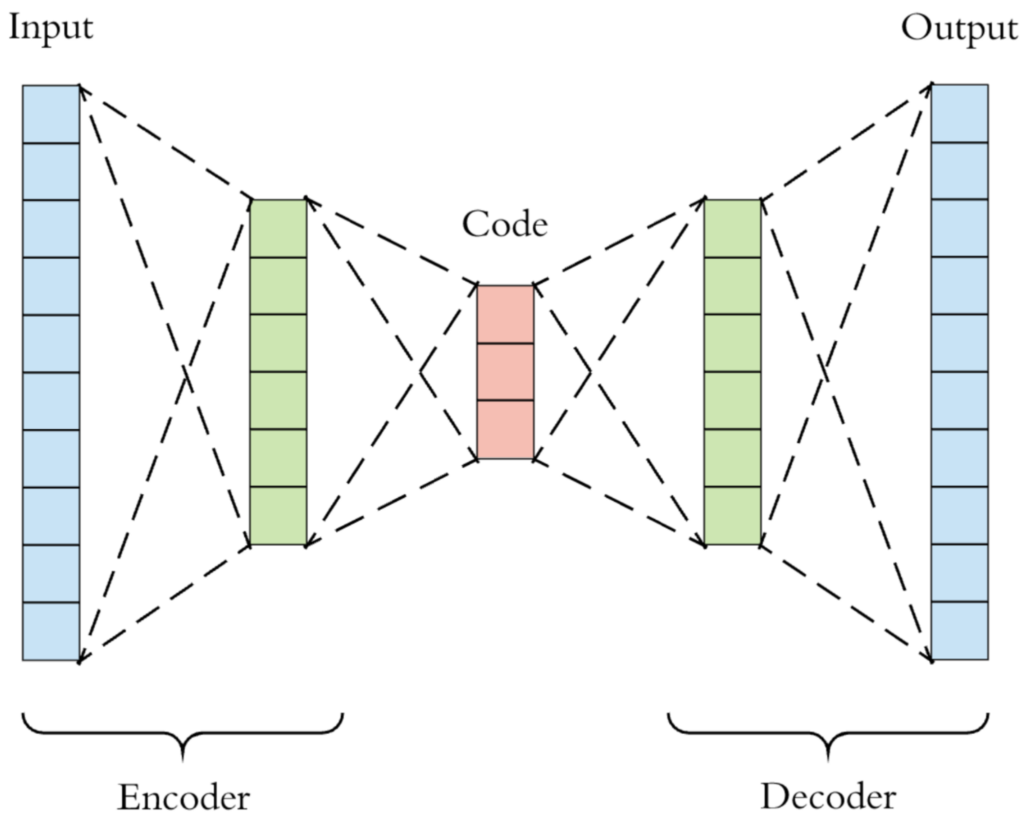

Figure 2.1: Autoencoder architecture [20]

| Interaction Type | Count |
|---|---|
| Classification / Transcription | 12074 |
| Data collection | 23717 |
| Joined the project | 3020 |
| Participated | 3348 |

Table 2.1: SciStarter Participation Dataset

This dataset contains 896 users and 1781 projects.

### 2.4.2 SciStarter Project Content Dataset

For the SciStarter content dataset, the majority of our features take the form of plain text. Projects are described using a number of fields including Title and Description fields for example,

*Nature's Notebook*

*Pay attention to the plants and animals in your yard, and you can contribute to scientific discovery! Observing life cycles of plants and animals with Nature's Notebook is easy and fun, and you will discover so much more about the plants and animals you see everyday. Sign up to observe one or more species in your yard or another place that you frequent. Use the Nature's Notebook smartphone app to send your observations directly to the National Phenology Database, or fill out paper datasheets and submit them online.*

### 2.4.3 MovieLens Dataset

In order to generate findings that build-on and are comparable to existing work in this space we evaluated our models on the MovieLens-100k dataset [10] since this dataset has been used in a number of papers where recommenders were being researched [7].

## 2.5 Evaluation Metrics

Though we are building recommender systems, there are two components to these models that we need to measure. We need to measure both the accuracy of our autoencoders, as well as the success and accuracy of our recommenders.

### 2.5.1 Metrics to Evaluate an Autoencoder

To measure the success of our autoencoders we will experiment with a number of loss functions. Firstly we will use *Mean Absolute Error* (MAE) [29],

$$MAE = \frac{\sum_{i=1}^{n} |e_i|}{n} \tag{2.1}$$

This is a common measurement used for evaluating neural networks and gives us a measure of the loss our autoencoders achieve when attempting to reconstruct their input.

However due to the sparsity of our data, the classes *interacted* and *not interacted* are severely unbalanced. We therefore have to ensure that our metrics correctly weight the results on both classes. Usually autoencoders are trained to reconstruct the input and would therefore struggle to handle corrupt or missing data. Extensive experimentation with loss functions [25] has focused on testing their ability to encourage predictions as well as reconstruction in autoencoders. And so we also experimented with a loss function provided by Strub et al. [25] which encourages the autoencoder to make predictions by only evalutating the loss on known ratings and ignoring the errors on unknown ratings.

### 2.5.2 Metrics to Evaluate a Recommender System

**Precision and Recall**

The most common metrics to evaluate the success of a recommender system are *precision* and *recall* [5]. Recall is the percentage of projects that the model correctly predicted the user would interact with, out of all the projects the user has interacted with. Precision is the percentage of projects that were correctly chosen by the model, out of all the projects that were chosen by the model.

However, precision and recall do not take into account the ranking of a recommendation (i.e. was a recommendation at the top or bottom of the recommendation list) and can also not discriminate between the relevance of a rejected recommendation (a recommendation that was not selected by the user). We believe recall is one of the most important metrics for our recommenders because it demonstrates how many accepted recommendations were in our top K projects.

**Refined Precision**

To provide a more complete evaluation of the success of our recommender systems we also used refined precision [4]. This metric was created to award value to rejected recommendations if these rejected recommendations are similar to an item that was actually selected. To measure the similarity between projects we calculated the cosine similarity of the TF-IDF vectors for each project. Refined Precision therefore gives us more meaningful measurements on the performance of our recommender system as it can give values to rejected recommendations that may be similar to accepted projects (i.e. projects that the user went on to accept). The equation for refined precision is as follows,

$$p_u^{sim} = p_u + \frac{\sum_{i \in r_u \ c_u} max_{j \in c_u : t(u,j) > t(u,i)} sim(i,j)}{|r_u|} \tag{2.2}$$

where $r_u$ is the set of items recommender to user $u$, $c_u$ is the set of items clicked on, $t(u,i)$ is the time user $u$ interacted with item $i$, $p_u$ is the precision for user $u$ and $sim(i,j)$ is the similarity between items $i$ and $j$ [4].

### Mean Average Precision

The final metric we used to measure the success of our models is Mean Average Precision (MAP) [28]. MAP considers the ranking of each recommendation and awards higher scores to highly ranked, accepted recommendations. This is given by,

$$MAP = \frac{\sum_{q=1}^{U} AveP(q)}{U} \tag{2.3}$$

where $U$ is the number of users.

Finally, we further experimented with different numbers of recommendations in order to find the optimal value that balances precision, recall, refined precision and mean average precision.

## 2.6 Implementation and Code

The code for this project will be submitted independently. All the code for the project was written in Python using a number of third party libraries.

- SkLearn was used for their implementation of TF-IDF and Cosine Similarity
- Keras was used to implement the autoencoders
- Gensim was used for their implementation of Doc2Vec
- Google Cloud Servers were used for training our autoencoders and testing our recommenders

# Chapter 3

# Methodology

In the following section we will discuss our methodology for achieving the aim of an accurate recommender on the SciStarter dataset. We will discuss preparing the data, building our models, making recommendations to users and then evaluating our recommendations.

## 3.1 Preprocessing our Data

The first step in training and testing our models is to preprocess the data. For the SciStarter dataset we were provided with two data sources, the content data and the participation data.

### 3.1.1 Preprocessing the Content Data

So that we can group projects in a meaningful way we want to make use of the information rich *Title* and *Description* fields for each project. We chose these fields as they are the most descriptive and varied of the SciStarter project data and should therefore be our best resource for measuring project similarity.

However, these fields are plain text fields and so in order to group the projects we decided to vectorise the text fields before then finding the similarities between these vectors. There are several methods with which to vectorise plain text however the most common are *Term Frequency–Inverse Document Frequency* (TF-IDF) and *Doc2Vec*.

TF-IDF, which is used in 83% of text-based recommender systems in digital libraries

[13], is a technique that reduces plain text into a vector representation. It is based on the intuition that words that appear in a high number of documents in the corpus are not useful for determining the similarity between documents [32]. For each word, the TF-IDF value changes proportionally to the frequency of that word in a document, and is inversely proportional to the number of documents in the corpus that contain the word. And so in order to group our projects we first generated TF-IDF vectors for the Title and Description fields of each project.

The next text vectorisation technique we used is known as Doc2Vec, which originated from Word2Vec [15] where the semantic relationship between words is extracted using proximity of the word to a target word. Similar to the famous Word2Vec model, Doc2Vec is a technique developed by [15] to represent large documents (of plain text) as vectors. Doc2Vec goes a step further than TF-IDF in that it can learn the relationships between words and understand the semantics of text and has therefore achieved impressive results on text classification tasks [12]. We therefore also vectorised our Title and Description fields using the Doc2Vec technique.

After representing the project's text fields as vectors our next task was to measure the similarity between these vectors in order to group them. To do this we used Cosine Similarity [11] given by,

$$similarity = \frac{\mathbf{A} \cdot \mathbf{B}}{||\mathbf{A}||||\mathbf{B}||} \tag{3.1}$$

Cosine similarity is commonly used to measure the similarity of vectors [11] and can therefore be used to group projects together based on their similarity.

### 3.1.2 Preprocessing Collaborative Filtering Data

After we had prepared our content data our next step was to preprocess the participations data by creating 3 adjacency matrices for the train, validation and test sets. An adjacency matrix is a matrix with # user columns and # project rows where a non-zero value in a cell represents an interaction occurring between a user and a project.

To construct these matrices we firstly found all users that had interacted with at least 3 projects as this would ensure that there were interactions in the train and val/test set for this user (which is required for training and evaluation of the model). We then grouped these interactions by profile and picked the earliest 80% of each user's interactions and set them aside as our training set. We then randomly split the remaining 20% into two datasets, the validation and test sets. We did this to maintain a form of time consistency as it has been shown that splitting in a time consistent manner outperforms a truly random split whilst also being a closer representation of the real world [14]. After doing this we found all profiles and projects with 0 interactions in the train set and removed them from the train, test and validation sets. This left us with a a train, test and validation set that contained user-project interactions, such as those shown in Table 3.1.

| Train | Validation | Test |
|---|---|---|
| Volunteer at The Marine Mammal Center | The Genographic Project | Perfect Pitch Test |
| Maine Amphibian Monitoring Program | - | Yellowhammer Dialects |
| Citizens and Remote Sensing Observational Network | - | - |
| Snow Tweets | - | - |

Table 3.1: Test Users Train, Validation and Test Projects

This preprocessing left us with a dataset with 896 users, 1021 projects with a total of 6000 interactions, giving us a data sparsity of 99%. We can already see here that our data sparsity is very high compared to standard datasets used in recommender system research, that usually have a data sparsity of 95% [25]. The issue of sparsity is a common problem in recommender system and can cause collaborative filtering methods to degrade as they struggle to learn latent factors of the models and therefore make accurate recommendations [8] . This issue can be addressed through the use of content data and specialised loss function ( [19] [22], [21]). The participations data also has a skewed distribution where a small number of projects have the majority of the participations. For example, the project ”The Twitter Earthquake Detection Program” has 332 interactions, whereas the median amount of interactions per project is 1. Compare this to the top profile, which has 17 interactions and we can see that there is a skew in the data that could cause issues with our future models.

## 3.2 MoveLens Dataset

As discussed in Section 2.4.3, we also evaluated our models on the industry standard MovieLens-100k dataset to compare it to models in the literature. The MovieLens-100k dataset consists of 100,000 interactions between 1,700 movies and 1000 users. The data sparsity on the dataset is 94.12% which is significantly lower than our SciStarter dataset. This suggests that our SciStarter dataset might be too sparse for our models to reliably beat the baseline, however the MovieLens dataset can be a reliable measure of whether our models compare to those in the literature.

Other differences in the MovieLens dataset are that it uses explicit ratings (e.g. a user has to actively rate a film) whereas in the SciStarter dataset the ”ratings” are implicitly derived from a users interactions with a project. This can cause many issues with our models, including a models ability to distinguish between a project that a user has actively avoided interacting with, and a project a user has just not seen yet.

## 3.3 Autoencoders

The main aim of this project is to evaluate the use of autoencoders in recommender systems and we therefore had to experiment with a number of different autoencoder architectures.

### 3.3.1 Denoising Autoencoders

Denoising autoencoders are commonly used in recommendation models [30]. They aim to stop the model from over learning the input data by corrupting it, therefore helping the model generalise well to incomplete data [27].

They are usually shallow architectures, having one hidden layer and are known as denoising architectures because their fully connected layers are interspersed with dropout layers. Dropout layers randomly set a small fraction (set by the *dropout probability*) of the inputs values in a layer to be 0. This simulates missing values in the input data and consequently prevents the models from overfitting on the training. This architecture therefore enables better generalisation on the test set and better performance in the real world [25].

### 3.3.2 Deep Autoencoders

Deep neural networks have been shown to achieve state-of-the-art results in various tasks, such as image classification and speech recognition [17]. Deep autoencoders have shown equally promising results and have therefore been applied to the recommendation problem [14]. The motivation for using deep autoencoders is that they can recognise more complex patterns and are therefore uniquely positioned to make better recommendations from incomplete or corrupted data. As the autoencoders learn more complex patterns in the data they can become more adept at predicting future projects the user will interact with.

## 3.4 Benchmarks

Before building our autoencoder models we first had to establish our benchmark results.

### 3.4.1 Popularity recommendation model

The first benchmark model we built was the popularity based model. This model simply recommends the most popular projects to every user regardless of their project interaction history. We used this as it is the simplest model to create yet can give a reliable benchmark.

### 3.4.2 Collaborative Filtering

The next benchmarks we implemented from previous research into the SciStarter dataset were basic collaborative filtering models. Our item-item method uses SciStarter's project-to-project relationships. This model examines the projects the user has already interacted with and then finds new projects that are most similar to these [22].

Our user-to-user relationships model calculates the similarity between users based on the items they have interacted with. By looking at users that have already interacted with a project and finding new users that have interacted with similar projects. It can then make new recommendations to users by looking at the projects they are most similar to.

We chose these benchmark models as they are simple to create whilst also providing a representative performance which we can use to benchmark our results.

## 3.5 Content based recommendations

In the SciStarter dataset we have content information for the projects but were required to generate our user profiles. We did this by concatenating the Title and Description fields of all projects they interacted with in the train set. As discussed above we first encode these fields using a text vectorisation technique and then use cosine similarity to provide a similarity measure between projects. For example, using cosine similarity on the TF-IDF of the projects description fields, the project ”North American Amphibian Monitoring Program” is marked as most similar to,

| Similar Projects |
|---|
| Spokane Area Amphibian Monitoring |
| FrogWatch USA |
| Frog Listening Network |

As we can see from the project titles, the similarity model appears to be finding relevant projects, in this case it is finding projects that are about amphibians.

To then recommend projects to users we calculate the similarity between a user's profile and all projects that they have yet to interact with and use these to generate a list of recommendations, with the most similar being the highest recommended [3].

One problem with TF-IDF or Doc2Vec vectors is that they can have a large dimensionality which causes computational inefficiencies. We therefore used denoising autoencoders to learn embeddings of the vectors, which we then used to calculate the cosine similarity between a users profile and the projects they are yet to interact with. The advantages of using the autoencoder embeddings is that they make the recommender more computationally efficient while they also remove noise from the vectors in an attempt to calculate similarity on the more fundamental aspects of the projects.

## 3.6 Autoencoder based Collaborative Filtering

After creating our content based models we then created autoencoder based collaborative filtering models as these have been shown to perform extremely well in recommendation tasks [14].

Aside from reducing the dimensionality of text vectors, autoencoders can also be used to make recommendations using collaborative filtering data. The data is split as described in Section 3.1.1 and the autoencoder is trained by inputting the training set and then trying to reconstruct the same training set but with additional interactions from the validation set. This technique was detailed by [14] and showed promising results on the MovieLens dataset. To then make recommendations we can feed the train set into our autoencoders and then review the output. From the output, we remove the projects that were already done in the train set and then pick the projects that have the highest value in the output vector. We can then recommend these projects to the user.

An example input adjacency matrix for one user would be,

$$\begin{bmatrix} 0 & 1 & 0 & 1 & 0 & 0 \end{bmatrix} \tag{3.2}$$

which shows that the user has interacted with project number 2 and 4. After this is fed into the autoencoder it will produce a vector that couple look like so,

$$\begin{bmatrix} 0 & 1 & 0 & 1 & 0.3 & 0.8 \end{bmatrix} \tag{3.3}$$

As we can see, we now have two new predictions that we can use as recommendations.

In order to find the best autoencoder architecture for recommendations we experimented with a number of different types, including shallow denoising autoencoders Figure 3.1, deep architectures and hybrid architectures.

## 3.7 Hybrid Recommenders

After developing content-based and collaborative filtering models, we proceeded to combine these models into hybrid recommenders since these have been shown to achieve higher accuracy than their constituent parts.

### 3.7.1 Basic Hybrid Recommender

The simplest version of a hybrid recommender simply averages the predictions from the collaborative filtering model and the content based model. The output from the

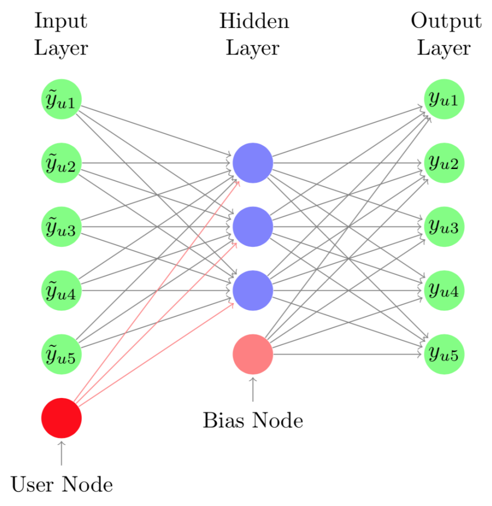

Figure 3.1: Denoising autoencoder architecture [30]

models gives each project a probability of being interacted with. These rankings can then be normalised and averaged before the recommendations are then picked from these combined rankings. For example, a profile might have a similarity of 0.7 with project 1 from the content based system while the collaborative filtering autoencoder generates a probability of 0.3 for project 1. We therefore average these two values $(0.7+0.3)/2 = 0.5$ which gives us a probability that a user will select project 1. We do this for every project, for every use to generate our list of recommendations.

The advantages of this model are that it is simple to implement and doesn't require any additional training. However this model is quite haphazard and can fail to correctly weight the rankings given by either model resulting in confident recommendations from one model being lost.

### 3.7.2 Putting Content Information into the Networks at the Start

Another method for creating hybrid recommenders is to feed content information into the collaborative filtering autoencoders in the first hidden layer of the neural network. This method has been tried by Strub et al. [25] and achieved state of the art results.

To insert content data into the first hidden layer we began by examining the results from our content based methods and chose our best performing word vectorization technique. We used this to embed our word vectors to be the same size as the first hidden layer of the autoencoder network. We then element-wise summed the collaborative filtering adjacency matrix and the word vector together. From here the autoencoder was trained as before [25]. For example,

$$cf = \begin{bmatrix} 0 & 1 & 0 & 1 & 0.3 & 0.8 \end{bmatrix} \tag{3.4}$$

$$content = \begin{bmatrix} 0.01 & 0.5 & 0.8 & 0.3 & 0.1 & 0 \end{bmatrix} \tag{3.5}$$

would result in the first layer,

$$content = \begin{bmatrix} 0.01 & 1.5 & 0.8 & 1.3 & 0.4 & 0.8 \end{bmatrix} \tag{3.6}$$

Using the output from the autoencoders we made our predictions using the same technique as for the collaborative filtering models (described in Section 3.6). The advantages of this model are that it considers both content and collaborative data and therefore avoids the cold start problem.

### 3.7.3 Putting Content Information into the Networks at Every Encoding Layer

Finally, there has been research into inserting content information into the models at every layer [25]. The benefit of this is that side information can influence the recommendations at every learning step and guide the neural network into learning the patterns beneath users project choices, alongside similarities in the projects.

To implement this model we first embedded the content information into a number of small vectors whose size matched our models hidden layers (for example we embed the content data to have size 1024, 512 etc.). From there we added the content information to each hidden layer using the same process as described in Section 3.7.2 for the first layer. An example architecture for our model can be seen in Figure 3.2. After this structure had been implemented, it was trained and evaluated in the same manner as described in Section 3.7.2.

## 3.8 Finally

And so using all the above techniques we aim to find a recommendation model that performs well on the SciStarter dataset. This recommendation model should therefore

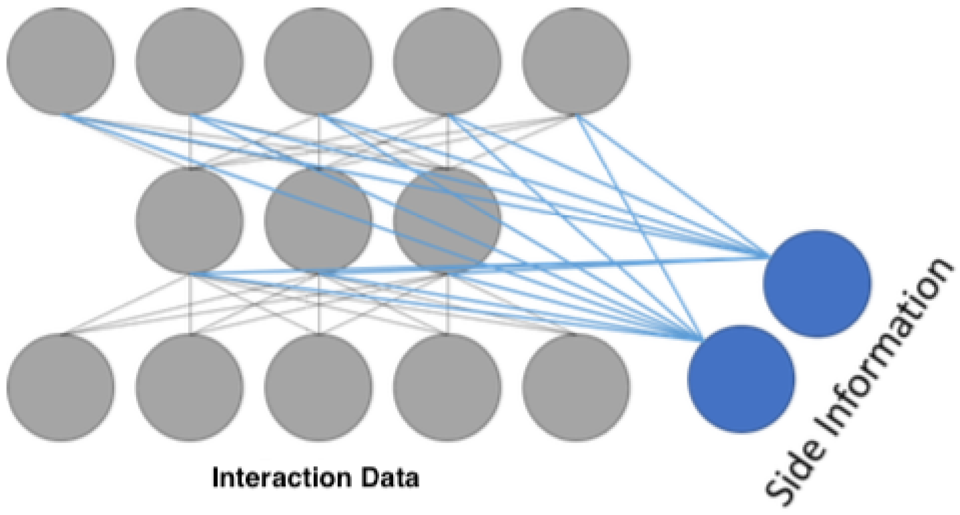

Figure 3.2: Autoencoder with content data inserted into every layer [25]

provide the SciStarter website with the ability to recommend projects to their users and increase user engagement. We also aim to produce a model that is easily trained, can be improved online and is computationally efficient for it to be hosted on the SciStarter website. To find this model we performed the experiments described in the next section.

# Chapter 4

# Results and Discussion

In order to find the best recommender model for the SciStarter dataset we had to train, evaluate and improve the models described in our methodology section. Below we will describe the experiments we ran and discuss their results. In all of our experiments we quote the results for 1, 5 and 10 recommendations and we only recommend projects that a user has not already interacted with.

## 4.1 Benchmark Results

The benchmark models described in our methodology section were picked since they were simple to implement and have already been implemented and optimised on the Scistarter dataset.

### 4.1.1 Popularity Based

We began by implementing a popularity based model. To find the most popular projects we grouped all the interactions by project and then ranked the projects in descending order by number of interactions. From this list we then recommended the most popular projects to every user. The results for the following model, evaluated on our test set are as follows,

| Recommendations | Precision | Recall | Refined Precision |
|---|---|---|---|
| 1 | 0.0067 | 0.0059 | 0.2331 |
| **5** | 0.0163 | 0.0719 | 0.9248 |

Table 4.1: Popularity Recommender Results

For our model the most popular projects can be seen in Table 4.2.

| Recommended Projects |
|---|
| The Twitter Earthquake Detection Program |
| The Genographic Project |
| Volunteer at The Marine Mammal Center |
| The Smell Experience Project |
| Perfect Pitch Test |
| NASA Top Stars |
| What on Earth |
| Snow Tweets |
| Seward Park Hemlock Tree Monitoring |
| Zero Robotics Autonomous Space Capture Challenge |

Table 4.2: Popularity Model Recommended Projects

For standard recommender systems on the MovieLens dataset the recall for 10 recommendations, 0.0922, is comparable [30]. We can also see that our refined precision for the 5 and 10 recommendations is high compared to the literature however the precision of our model is low. We believe this is most likely due to the data sparsity in the SciStarter dataset and the imbalanced distribution of project interactions. This meaning that the majority of the interactions in the dataset are with the 5 most popular projects and so high recall can easily by achieved by always recommending these projects.

### 4.1.2 Collaborative Filtering: User-User

The next benchmark we implemented was a collaborative filtering user-to-user model. This model uses a neighborhood model size of 10 and achieved the following results on the new SciStarter dataset,

| Recommendations | Precision | Recall | Refined Precision |
|---|---|---|---|
| 1 | 0.0944 | 0.0811 | 0.1826 |
| **5** | 0.0688 | 0.2933 | 0.1586 |

Table 4.3: Collaborative Filtering User-User Recommender Results

As we can see, this model's precision and recall are significantly better than the popularity model. This demonstrates that personalising recommendations to each user can yield promising results. Intuitively this is expected as different users will have different preferences in projects and any model that can learn these preferences should perform well.

### 4.1.3 Collaborative Filtering: Item-Item

The final benchmark we implemented was a collaborative filtering item-to-item model [22]. This model uses a neighborhood size of 10 and achieved the following results on the Scistarter dataset.

| Recommendations | Precision | Recall | Refined Precision |
|---|---|---|---|
| 1 | 0.0157 | 0.0148 | 0.0844 |
| **5** | 0.0215 | 0.1059 | 0.0878 |

Table 4.4: Collaborative Filtering Item-Item Recommender Results

As we can see, this shows us that the item-to-item models perform worse than the user-to-user models. We believe this is due to the profile interactions being more evenly distributed in the dataset compared to the project interactions. Due to a small number of projects having the majority of the interactions, the item-to-item model will struggle to learn user preferences as well as the user-to-user model.

Reviewing the performance of the above models demonstrates that they provide a reasonable benchmark as their results are comparable to models from the literature that were implemented on the MovieLens and Netflix datasets [14].

## 4.2 Content Based Recommendation

Now we have established our benchmarks we will proceed by implementing and training our content based recommender systems.

### 4.2.1 Embedding the Input

The first step in creating our content based recommender systems is to prepare our input by vectorising the projects Description and Title fields. We create these TF-IDF and Doc2Vec vectors using standard python libraries, SkLearn [2] and Gensim [1].

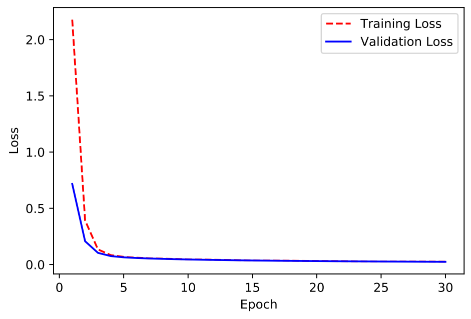

Figure 4.1: Training our autoencoder with embedding size 128, to reduce our TF-IDF vectors of the description field

Once we have created our text vectors we then use our autoencoders to reduce the vector's dimensionality. This technique is used to learn hidden patterns in the data, perform a form of clustering on the vectors and ultimately embed them into a smaller space to improve computational efficiencies. Training our autoencoders with a learning rate of 0.001 and dropout rate of 0.5 over 30 epochs yielded a loss of 0.0240 on the train and validation sets, as shown in Figure 4.1. We can see that our training quickly achieved a low loss, showing us that our autoencoders were able to embed and then reconstruct our vectors successfully.

Once we have obtained these embeddings we can calculate the similarity between the embeddings. For our TF-IDF vectors the max similarity is 0.9871, the lowest cosine similarity is 0.0009 and the standard deviation was 0.0455. Here we can see that the standard deviation is relatively low which could cause concern since we need a large spread of similarity values in order to make more confident recommendations.

### 4.2.2 Recommendations

Once the similarities had been calculated we could then generate our recommendations by ranking the projects by similarity for every user. This technique led to the following results shown in Table 4.5, Table 4.6, Table 4.7 and Table 4.8.

| Embed Size | K | Precision | Recall | Refined Prec. |
|---|---|---|---|---|
| | 1 | 0.0012 | 0.0005 | 0.7215 |
| 64 | 5 | 0.0016 | 0.0029 | 0.7242 |
| | 10 | 0.0015 | 0.005 | 0.726 |
| | 1 | 0.0012 | 0.0004 | 0.6517 |
| **128** | 5 | 0.0019 | 0.0041 | 0.6555 |
| | **10** | 0.0018 | 0.0083 | 0.6565 |

Table 4.5: Content Based TF-IDF on Description Recommender Results

| Embed Size | K | Precision | Recall | Refined Prec. |
|---|---|---|---|---|
| | 1 | 0 | 0 | 0.3282 |
| 64 | 5 | 0.0009 | 0.0025 | 0.3356 |
| | 10 | 0.0014 | 0.0059 | 0.3387 |
| | 1 | 0 | 0 | 0.2501 |
| 128 | 5 | 0.0011 | 0.0030 | 0.2575 |
| | 10 | 0.0014 | 0.0079 | 0.2609 |

Table 4.6: Content Based Doc2Vec on Description Recommender Results

| Embed Size | K | Precision | Recall | Refined Prec. |
|---|---|---|---|---|
| | 1 | 0.0019 | 0.0004 | 0.8166 |
| 64 | 5 | 0.0022 | 0.0049 | 0.8194 |
| | 10 | 0.0017 | 0.0083 | 0.8201 |
| | 1 | 0.0006 | 0 | 0.6739 |
| 128 | 5 | 0.0014 | 0.0036 | 0.6760 |
| | 10 | 0.0012 | 0.0058 | 0.6768 |

Table 4.7: Content Based TF-IDF on Title Recommender Results

| Embed Size | K | Precision | Recall | Refined Prec. |
|---|---|---|---|---|
| | 1 | 0.0006 | 0 | 0.7623 |
| 64 | 5 | 0.0019 | 0.0041 | 0.7644 |
| | 10 | 0.0016 | 0.0071 | 0.7654 |
| | 1 | 0 | 0 | 0.6492 |
| 128 | 5 | 0.0011 | 0.0026 | 0.6555 |
| | 10 | 0.0011 | 0.0056 | 0.6567 |

Table 4.8: Content Based Doc2Vec on Title Recommender Results

We can see from the results that even our best performing model, which is our TF-IDF vectorisation on the Description Field with embedding size 128, for precision and recall we do not beat the benchmark results. This is most likely due to two factors. Firstly, the uneven distribution of interactions with projects (i.e. a small number of projects having the majority of the interactions) means that our benchmark results were artificially inflated. Secondly, the low standard deviation in our similarity values means that our model could not be confident about it's recommendations, therefore causing the precision and recall to be low.

However, we can see that the refined precision is comparable to the popularity benchmark model. This is encouraging as it suggests that the users are selecting projects that are similar to their profile, therefore meaning that a content based system could be an important element of any recommender system on the SciStarter dataset.

In order to give an example of our recommender systems in action, we will show their recommendations for the same user at each stage. We can see this user's train, val and test projects in Section 3.1.2. For each recommender we will compare the recommended projects to the *true* projects (as shown in Table 4.9). As we can see from our best content recommender (TF-IDF on the Description Field with embedding size 128), the recommended projects, as shown in Table 4.10, are similar to the *true* projects. The recommended projects seem to follow two themes which are water/animals and health, one of which is very similar to the *true* projects. This is a promising result as it shows that are recommender system is forming groups of similar projects and making recommendations within them.

| True Projects |
|---|
| The Genographic Project |
| Maine Amphibian Monitoring Program |
| Perfect Pitch Test |
| Citizens and Remote Sensing Observational Network |
| Yellowhammer Dialects |

Table 4.9: Projects in test set for our test user

| Recommended Projects |
|---|
| Salida Trail Project Ditch Creek Water Quality Testing |
| Testing the waters in Negril, Jamaica |
| UK Ladybird Survey |
| Makers Local 256 |
| The Microbiome and Oral Health |
| BioCurious |
| Track a Tree |
| Connecticut Turtle Atlas |
| Counter Culture Labs |
| $79 Dental Genome Kit |

Table 4.10: Content Based with TF-IDF on Description and Embedding size 128, Recommended Projects

## 4.3 Collaborative Filtering with Autoencoders

### 4.3.1 CDAE

Due to the lack of performance with our content based method we next decided to experiment with denoising autoencoders as they have been proven to perform well on tasks with incomplete or corrupted data, such as the SciStarter dataset [30]. Using MAE as our loss function, we experimented with various numbers of latent variables, 32, 64 and 128 with a learning rate of 0.001 and dropout rate of 0.5 [16] which yielded the following results shown, with our best model with embedding size 128 achieving a loss of 0.0746 in Figure 4.2

Again here we can see that our autoencoders are quickly achieving a low loss, which suggests that our adjacency vectors can be successfully embedded into a smaller state space and then reconstructed. To test the model we then fed our training data into the autoencoder, took the reconstructed vector and picked the K largest items from the vector giving us our recommendations, seen in Table 4.11.

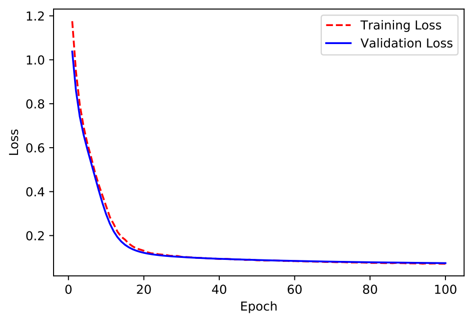

Figure 4.2: Denoising autoencoder trained on the collaborative filtering data with embedding size 128.

| Embed Size | K | Precision | Recall | MAP | RMSE |
|---|---|---|---|---|---|
| 32 | 1 | 0.0116 | 0.0071 | 0.0249 | 0.0884 |
| | 5 | 0.0081 | 0.0318 | 0.0249 | 0.0884 |
| | 10 | 0.0047 | 0.0353 | 0.0249 | 0.0884 |
| 64 | 1 | 0.0116 | 0.0071 | 0.0234 | 0.0703 |
| | 5 | 0.0052 | 0.0188 | 0.0234 | 0.0703 |
| | 10 | 0.0044 | 0.0315 | 0.0234 | 0.0703 |
| **128** | 1 | 0.0116 | 0.0071 | 0.0238 | 0.0660 |
| | 5 | 0.0058 | 0.0222 | 0.0238 | 0.0660 |
| | **10** | 0.0047 | 0.0353 | 0.0238 | 0.0660 |

Table 4.11: Collaborative Filtering Denoising Autoencoder Recommender Results

The results we got from the recommenders show us that the denoising autoencoders are still not performing well on the SciStarter dataset. The results from the autoencoders show that it has a significantly low validation loss however, the precision and recall from our results still fail to beat the baseline. We can also see that using an embedding size of 128 performs marginally better which may suggest that the embedding sizes 32 and 64 are too small to accurately capture patterns within the data.

We can see from the recommended projects in Table 4.12 that our user are being recommended projects in a seemingly random order. These results could be caused by a number of issues. The data sparsity in our dataset likely means that the autoencoders are not being provided enough examples with which to detect patterns and make reliable recommendations. Alongside this, we are also no longer considering the content data and therefore reducing the refined precision.

| Recommended Projects |
|---|
| The Twitter Earthquake Detection Program |
| The Genographic Project |
| Habitat Steward |
| The Smell Experience Project |
| Shermans Creek Watershed Monitoring Program |
| Perfect Pitch Test |
| NASA Top Stars |
| What on Earth |
| Seward Park Hemlock Tree Monitoring |
| PhotosynQ |

Table 4.12: Collaborative Filtering Denoising Autoencoder with Embedding Size 128 Recommended Projects

## 4.4 Deep Autoencoders and Recommenders

By observing that our denoising model with embedding size 128 performed best, we consequently hypothesised that deeper architectures may achieve a higher performance due to their ability to learn more complex patterns within the data. For our deep autoencoders we experimented with a number of different architectures.

### 4.4.1 Architectures

In order to experiment with deep autoencoder algorithms we tried the embedding structures shown in Table 4.13.

| Name | Embedding Structure | Source |
|---|---|---|
| Deep1 | $32 \to 10 \to 32$ | Liu et. al. [16] |
| Deep2 | $128 \to 128$ | Kuchaiev et. al. [14] |
| Deep3 | $1024 \to 512 \to 512 \to 512 \to 512 \to 1024$ | Kuchaiev et. al. [14] |
| Deep4 | $500 \to 250 \to 500$ | Sedhain et. al. [23] |

Table 4.13: Deep Autoencoder Architectures

We hoped that increasing the hidden layers would improve the models ability to learn the underlying patterns in the data and therefore gain the ability to accurately recommend new projects to users. Our motivation for this came from a number of papers that had achieved state-of-the-art results using deep autoencoders ( [16], [14], [23]). Here we also introduced our specialised loss function to encourage the autoencoder to create new recommendations alongside reconstructing the input as described in Section 2.5.1. [25].

### 4.4.2 Training the Autoencoder

We trained all of our autoencoders with a learning rate of 0.001 on the SciStarter dataset and you can see the results achieved by the Deep3 architecture, achieving loss of 0.0531, in Figure 4.3.

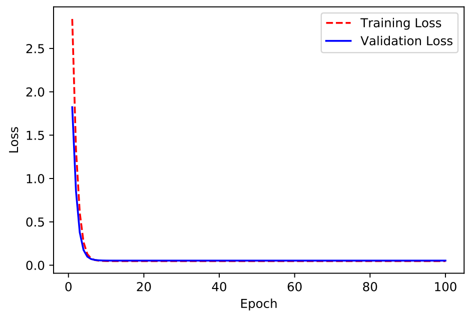

Figure 4.3: Deep3 Architecture trained on the SciStarter Interactions Dataset

As we can see the new loss function performs better than our previous autoencoders by achieving a lower loss. This is encouraging as it suggests that deeper autoencoders are more capable of learning the hidden patterns in the data. We also trained with a number of different dropout probabilities and found that $p = 0.8$ yielded the best recommendation results, shown in Table 4.14. This supports the claims made by Kuchaiev et al. [14] and shows that higher dropout probabilities are better at simulating real world recommendation scenarios. The next step is to evaluate how well our models perform on our recommendation task.

### 4.4.3 Test the Recommender

We ran our deep autoencoders on the SciStarter dataset and achieved the results shown in Table 4.14.

| Architecture | K | Precision | Recall | MAP | RMSE |
|---|---|---|---|---|---|
| Deep1 | 1 | 0.0087 | 0.0064 | 0.0220 | 0.0407 |
|  | 5 | 0.006977 | 0.025145 | 0.021950 | 0.0407 |
|  | 10 | 0.0038 | 0.0281 | 0.0220 | 0.0407 |
| Deep2 | 1 | 0.0029 | 0.0006 | 0.0165 | 0.0407 |
|  | 5 | 0.0041 | 0.0128 | 0.0165 | 0.0407 |
|  | 10 | 0.0035 | 0.0228 | 0.0165 | 0.0407 |
| **Deep3** | 1 | 0.0029 | 0.0006 | 0.0105 | 0.0407 |
|  | 5 | 0.0302 | 0.1328 | 0.0105 | 0.0407 |
|  | **10** | 0.0180 | 0.1545 | 0.0105 | 0.0407 |
| Deep4 | 1 | 0 | 0 | 0.01391 | 0.0471 |
|  | 5 | 0.0017 | 0.0065 | 0.0139 | 0.0471 |
|  | 10 | 0.0020 | 0.0158 | 0.0139 | 0.0471 |

Table 4.14: Deep Recommender Results

As we can see the deep architectures perform better than the shallow denoising autoencoders. This shows that the deeper architectures are better at capturing the hidden patterns in the data and the loss function encourages the autoencoders to make predictions as well as reconstruct the data. This observation follows the results in the literature detailing how deeper architectures are better at learning hidden patterns in the data. We can see this by looking at some example recommendations in Table 4.15 where the project titles seem to be more similar to each other.

| Recommended Projects |
|---|
| The Genographic Project |
| Bay Area Ant Survey |
| Volunteer at The Marine Mammal Center |
| Habitat Steward |
| Global Warming Ambassador |
| Maine Amphibian Monitoring Program |
| The Smell Experience Project |
| Shermans Creek Watershed Monitoring Program |
| What on Earth |
| NatureWatch |

Table 4.15: Deep Autoencoder Recommendations

However, though this model beat the denoising autoencoders it still fails to beat the benchmarks. We believe this is due to the data sparsity and size of the dataset. Deep architectures often need very large datasets in order to learn hidden patterns in the dataset and it appears that the SciStarter dataset does not contain an adequate level of examples.

As a means to verify our models performance we also ran our recommenders on the MovieLens-100k dataset, achieving the results shown in Figure 4.4 and Table 4.16.

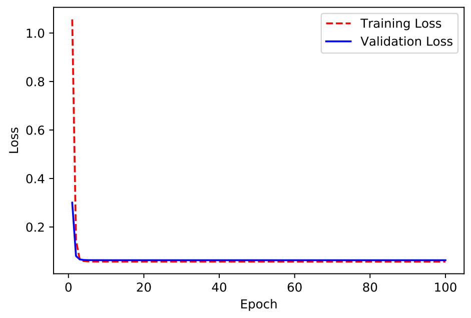

Figure 4.4: Deep3 Architecture trained on the MovieLens Dataset

| Architecture | K | Precision | Recall | MAP | RMSE |
|---|---|---|---|---|---|
| Deep3 | 1 | 0.1771 | 0.0177 | 0.0675 | 0.08 |
| | 5 | 0.1122 | 0.0561 | 0.0675 | 0.08 |
| | 10 | 0.0857 | 0.0857 | 0.0675 | 0.08 |

Table 4.16: Deep Recommender Results on MovieLens

As we can see from this, our results are comparable to the literature [30] and so this justifies our belief that it is the Scistarter dataset's quality and size that is causing the lower performance values. In order to combat this we may be required to collect more data, or artificially populate the data.

## 4.5 Hybrid Recommenders

Our final attempt at creating a viable recommender system for the SciStarter dataset is a hybrid recommender system. We believe that by combining our best content based model and our best collaborative filtering model, we can find a way to significantly increase the performance of our recommender systems. We believe that the content data will help guide the autoencoder to learn hidden patterns in the participation data whilst also taking into account project similarity. To create our hybrid recommender we will experiment with a number of different models.

### 4.5.1 Basic Hybrid Recommender

For our first hybrid recommender we averaged the results output from our content based TF-IDF model with embedding size 128 (our best content model) and our Deep3 Architecture (our best collaborative filtering model). This model therefore required no more training and could be evaluated immediately, giving the results shown in Table 4.17

| Recommendations | Precision | Recall | Refined Precision |
|---|---|---|---|
| 1 | 0 | 0 | 0.1469 |
| **5** | 0.0074 | 0.037 | 0.332 |
| 10 | 0.0037 | 0.037 | 0.225 |

Table 4.17: Averaging Hybrid Recommender Results

We can see that this model performs worse than the Deep3 model but better than the TF-IDF, embedding size 128 model. We can therefore see that though the deep model has helped improve the TF-IDF models, the inclusion of content data has overall decreased the accuracy of the model. We believe this is due to the low standard deviation in our cosine similarity matrix meaning that the content data is not confident enough on project similarities and is therefore confusing the results.

### 4.5.2 Injecting Content Data into the First Layer

The next hybrid recommender model involved injecting content data into the first hidden layer of the autoencoder as described in Section 3.7.2. We chose the Deep3 architecture because this is the architecture that performed best in our experiments. First we had to train our autoencoder with a learning rate of 0.001 and dropout rate of 0.8 which achieved a loss of 0.0725 as shown in Figure 4.5.

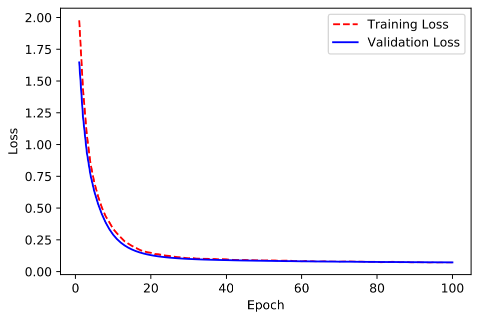

Figure 4.5: Hybrid Recommender inserting content data into the first layer

Next we had to test our autoencoder which achieved the results shown in Table 4.18.

| K | Precision | Recall | MAP | RMSE | Refined Prec. |
|---|---|---|---|---|---|
| 1 | 0.0116 | 0.0071 | 0.0227 | 0.0506 | 0.2166 |
| 5 | 0.0076 | 0.0281 | 0.0244 | 0.0588 | 0.3084 |
| **10** | 0.0041 | 0.0310 | 0.0244 | 0.0588 | 0.3226 |

Table 4.18: Hybrid Recommender with First Layer Content Results

We can see that this model performed marginally better than the individual collaborative filtering approach in precision and recall. We believe this is because the content data is helping the network learn patterns in the sparse data and therefore increasing its performance in generating recommendations. We also believe that the inclusion of the content data is guiding the network towards projects that are similar to the accepted recommendations and therefore increasing the refined precision values.

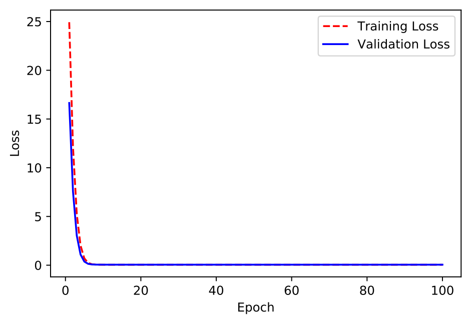

Figure 4.6: Training our hybrid recommender with content data injected into every layer

### 4.5.3 Injecting Content data into Every Layer

The final model we developed injected content data into every layer of the autoencoder. We believed this would perform well because the content data would nudge the autoencoder to select similar projects at every layer, therefore guiding the network towards relevant choices at every learning step. First we had to train our autoencoder with a learning rate of 0.001 and dropout rate of 0.8, achieving a loss of 0.0532 as shown in Figure 4.6.

The autoencoder architecture was our best architecture from the collaborative filtering results, Deep3. We also used the loss function that promoted the autoencoder to make new predictions (as explained in Section 2.5.1) as this has been shown to increase the performance of recommender systems.

We then ran our autoencoder to gain the results shown in Table 4.19.

| Architecture | K | Precision | Recall | MAP | RMSE | Refined Prec. |
|---|---|---|---|---|---|---|
| **Hyb3** | 1 | 0.0 | 0.0 | 0.010013 | 0.047 | 0.1210 |
| | 5 | 0.0285 | 0.1249 | 0.0100 | 0.0470 | 0.7383 |
| | **10** | 0.0180 | 0.1501 | 0.0100 | 0.0470 | 0.7825 |

Table 4.19: Hybrid Recommender with Content in Every Layer Results

We can also see the recommended projects in Table 4.20. These recommendations are similar to our previous models and achieve a recall of 0.4. We can see however that the projects that we correctly predicted are ”The Genographic Project” and ”Maine Amphibian Monitoring Program” which again shows that the Amphibian/Biology grouping is still present in our recommendations, showing that the content information has provided some guidance to our recommendations.

| Recommended Projects |
|---|
| The Genographic Project |
| Bay Area Ant Survey |
| Volunteer at The Marine Mammal Center |
| Habitat Steward |
| Global Warming Ambassador |
| Maine Amphibian Monitoring Program |
| NASA Top Stars |
| moo-Q |
| Mark My Bird |
| NatureWatch |

Table 4.20: Hybrid Recommendations for Hyb3: the best hybrid architecture results

## 4.6 Findings and Parameter Influences

The results show us that our best performing mode was the Deep3 collaborative filtering architecture with no content data. We believe this is because the content data was not providing sufficient information to form groups of projects, therefore meaning that the introduction of content data to the autoencoder caused the neural network to learn suboptimal recommendations.

However, we can still see that none of our results have beaten the baseline. We believe this is caused by three issues. Firstly, the data quality is not high enough (i.e. it is too sparse) and so our autoencoders do not have sufficient information with which to learn patterns in the data and form models of user behaviour. The second issue is that the distribution of interactions among the projects is extremely uneven, likely meaning that the popularity results are artificially higher than they would be if the interactions were more evenly distributed. Finally, we believe that the dataset is not large enough for the deeper architectures to learn patterns in the data with which to make recommendations.

In this section we also experimented with different values of dropout probabilities which resulted in us finding that higher probabilities help the autoencoder's recommendation recall. This is consistent with the literature and should therefore be the strategy for future studies.

Finally, we can see that our models performed comparably on the MovieLens dataset and so there is definitely some progress that can be made in these areas on industry standard datasets.

# Chapter 5

# Conclusion

In our research we had the objective of creating an autoencoder based recommender system for the SciStarter dataset. We will now briefly summarise our findings.

## 5.1 Summary of Findings

Our autoencoder based recommenders took the form of content based recommenders, shallow autoencoder recommenders, deep autoencoder recommenders and hybrid recommenders. Our findings were ultimately that none of these approaches matched, or beat, the benchmark of a standard user-user collaborative filtering method.

We found that our content based models appeared to form some groupings of projects but failed to adequately discern between project groups, leading to confusion and inaccurate recommendations.

Upon studying our collaborative filtering autoencoder based approaches we found that the deeper the autoencoder architecture, the better the results. This supported our hypothesis that in order to provide reliable recommendations, an autoencoder needs to understand complex patterns hidden in the data and personalise it's recommendations to each users preferences.

Finally, our hybrid recommender results performed marginally worse than their constituent parts showing us that the content data was causing the hybrid model to make worse recommendations.

We believe our models achieved these low results due to the sparsity and quality of the SciStarter dataset.

## 5.2 Future work

Though we have shown that our models perform well on the MovieLens dataset, our main aim of the project was not achieved. However, this does not mean that the SciStarter dataset is not suitable for building a recommender system. Our research was almost solely on autoencoder based models, and our benchmarks showed that simple collaborative filtering methods could achieve good results. Therefore future work in this area could be further researching other types of recommender systems, possibly those that rely more on content data as this is more rich in the SciStarter dataset.

Finally if more data is collected for SciStarter, autoencoders (especially deep autoencoders) may perform significantly better. Research has also shown that any negative implicit data can greatly increase accuracy. And so if data is collected on when a user has seen a project but not interacted with it, this could greatly improve the performance of a recommender system. Therefore, the research into autoencoder based recommender systems on the SciStarter dataset should not be abandoned.

# Bibliography

[1] *Gensim.*

[2] *SciKit Learn.*

[3] X Amatriain, A Jaimes, N Oliver, and JM Pujol. *Data Mining Methods for Recommender Systems - Recommender Systems Handbook.* Springer, 2011.

[4] Anonymous Author(s). *Are All Rejected Recommendations Equally Bad? Towards Analysing Rejected Recommendations.* Proceedings of UMAP, 2019.

[5] Michael Buckland and Fredric Gey. *The Relationship between Recall and Precision.* 1994.

[6] ROBIN BURKE. *Hybrid Recommender Systems- Survey and Experiments.* User Modeling and User-Adapted Interaction, 2001.

[7] Tiago Cunhaa, Carlos Soaresa, and Andre C.P.L.F. de Carvalho. *Metalearning and Recommender Systems: A literature review and empirical study on the algorithm selection problem for Collaborative Filtering.* Elsevier, 2017.

[8] Xin Dong, Lei Yu, and Zhonghuo Wu. *A Hybrid Collaborative Filtering Model with Deep Structure for Recommender Systems.* AAAI Conference on Artificial Intelligence, 2017.

[9] Jill Freyne, Michal Jacovi, Ido Gu, and Werner Geyer. *Increasing Engagement through Early Recommender Intervention.* RecSys'09, 2009.

[10] F. MAXWELL HARPER and JOSEPH A. KONSTAN. *The MovieLens Datasets: History and Context.* 2015.

[11] Anna Huang. *Similarity Measures for Text Document Clustering.* 2007.

[12] Mark HUGHES, Irene LI, Spyros KOTOULAS, and Toyotaro SUZUMURA. *Medical Text Classification using Convolutional Neural Networks*. IBM Research Lab, 2017.

[13] Beel Joeran, Gipp Bela, Langer Stefan, and Breitinger Corinna. *Research-paper recommender systems : a literature survey*. International Journal on Digital Libraries, 2016.

[14] Oleksii Kuchaiev and Boris Ginsburg. *Training Deep AutoEncoders for Collaborative Filtering*. 2017.

[15] Quoc Le and Tomas Mikolov. *Distributed Representations of Sentences and Documents*. International Conference on Machine Learning, 2014.

[16] Yu Liu, Shuai Wang, M. Shahrukh Khan, and Jieyu He. *A Novel Deep Hybrid Recommender System Based on Auto-encoder with Neural Collaborative Filtering*. BIG DATA MINING AND ANALYTICS, 2018.

[17] Weibo Liua, Zidong Wanga., Xiaohui Liua, Nianyin Zengb, Yurong Liuc, , and Fuad E. Alsaadid. *A Survey of Deep Neural Network Architectures and Their Applications*. Elsevier, 2017.

[18] Pasquale Lops, Marco de Gemmis, and Giovanni Semeraro. *Content-based Recommender Systems - State of the Art and Trends*. Springer Science+Business Media, 2011.

[19] Alexandrin Popescu, Lyle H. Ungar, David M. Pennock, and Steve Lawrence. *Probabilistic Models for Unified Collaborative and Content-Based Recommendation in Sparse-Data Environments*. 2001.

[20] Christoph Quix and Jorge Bernardino. *Data Management Technologies and Applications*. Communications in Computer and Information Science, 2018.

[21] Badrul Sarwar, George Karypis, Joseph Konstan, and John Riedl. *Application of Dimensionality Reduction in Recommender System - A Case Study*. 2000.

[22] Badrul Sarwar, George Karypis, Joseph Konstan, and John Riedl. *Item-Based Collaborative Filtering Recommendation Algorithms*. WWW10, 2001.

[23] Suvash Sedhain, Aditya Krishna Menon, Scott Sanner, and Lexing Xie. *AutoRec: Autoencoders Meet Collaborative Filtering*. 2015.

[24] Meenakshi Sharma and Sandeep Mann. *A Survey of Recommender Systems: Approaches and Limitations*. International Journal of Innovations in Engineering and Technology, 2013.

[25] Florian Strub, Jeremie Mary, and Romaric Gaudel. *Hybrid Recommender System based on Autoencoders*. 2017.

[26] Xiaoyuan Su and Taghi M. Khoshgoftaar. *A Survey of Collaborative Filtering Techniques*. Advances in Artificial Intelligence, 2009.

[27] Pascal Vincent, Hugo Larochelle, Yoshua Bengio, and Pierre-Antoine Manzagol. *Extracting and Composing Robust Features with Denoising Autoencoders*. 2008.

[28] Ellen M. Voorhees. *The Philosophy of Information Retrieval Evaluation*. CLEF, 2002.

[29] Cort J. Willmott and Kenji Matsuura. *Advantages of the mean absolute error (MAE) over the root mean square error (RMSE) in assessing average model performance*. CLIMATE RESEARCH, 2005.

[30] Yao Wu, Christopher DuBois, Alice X. Zheng, and Martin Ester. *Collaborative Denoising Auto-Encoders for Top-N Recommender Systems*. WSDM, 2016.

[31] Weidi Xu, Haoze Sun, Chao Deng, and Ying Tan. *Variational Autoencoder for Semi-Supervised Text Classification*. AAAI Conference on Artificial Intelligence, 2017.

[32] Wen Zhang, Taketoshi Yoshida, and Xijin Tang. *A comparative study of TF-IDF, LSI and multi-words for text classification*. Elsevier, 2011.